Energy–Delay Optimization of 64-Bit Carry-Lookahead Adders With a 240 ps 90 nm CMOS Design Example

Radu Zlatanovici, *Member, IEEE*, Sean Kao, and Borivoje Nikolić, *Senior Member, IEEE*

***Abstract*—A methodology for energy–delay optimization of digital circuits is presented. This methodology is applied to minimizing the delay of representative carry-lookahead adders under energy constraints. Impact of various design choices, including the carry-lookahead tree structure and logic style, are analyzed in the energy–delay space and verified through optimization. The result of the optimization is demonstrated on a design of the fastest adder found, a 240-ps Ling sparse domino adder in 1 V, 90 nm CMOS. The optimality of the results is assessed against the impact of technology scaling.**

***Index Terms*—Adder, carry-lookahead, CMOS, high performance, low power, power–performance optimization.**

I. Introduction

FAST and energy-efficient single-cycle 64-bit addition is essential for today's high-performance microprocessor execution cores. Wide adders are a part of the highest power-density processor blocks, creating thermal hotspots and sharp temperature gradients [1]. The presence of multiple ALUs in modern superscalar processors [2], [3] and of multiple execution cores on the same chip [3]–[5] further aggravates the problem, impacting circuit reliability and increasing cooling costs. At the same time, wide adders are also critical for performance, and appear inside the ALUs, AGUs and FPUs of microprocessor datapaths. Ideally, a datapath adder would achieve the highest performance using the least amount of power and have a small layout footprint in order to minimize interconnect delays in the core [1]. These contradictory requirements pose a challenging problem in choosing the optimal adder architecture and circuit implementation. Designers have several degrees of freedom to optimize the adder for performance and power. Fast adders are commonly implemented as carry-lookahead. Within the carry-lookahead family there is a wide array of choices that include: tree topologies, full or sparse implementation of the trees, conventional or Ling's carry-lookahead equations, and the circuit design style.

Although there are many publications written about adder design, fundamental understanding of the impact of the various design choices on the performance and power of a particular design is still incomplete. Traditionally, Kogge–Stone parallel prefix trees [6], characterized by their minimum logic depth, regular structure, and uniform fanout are used when very high performance is needed. Their main disadvantage is the large number of gates and wires, which leads to high power consumption. An implementation of a 64-bit adder using a Kogge–Stone tree has been reported in [7]. The number of nodes and connections in the tree can be reduced by trading it off for increased logic depth, such as the sparse Han-Carlson tree [8]. Many sparse tree implementations have been reported in recent years, with sparseness of two [9], [10], four [11], [12], or variable [1].

An alternate logic design investigates different implementations of the carry equations. Ling's equations can lead to a simplification of parallel prefix nodes and reduced transistor stack heights [13], [14].

To fairly compare different adder designs it is necessary to establish a common baseline. Existing adder comparisons can be classified into two categories:

- Comparisons without using optimization:
  - — simulation-based study of the impact of wires on the adder delay with fixed gate sizes [15];
  - — logic manipulation-based study of the impact of carry tree topology on the logic depth [16];
- Comparisons using optimization:
  - — optimal transistor sizing for minimum delay using logical effort [17];
  - — optimal transistor sizing in the energy–delay space using a combination of logical effort and net load assignments for static adders [18];
  - — optimal transistor sizing in the energy–delay space using logical effort on adders implemented in various logic families [29], [30].
  - — optimal transistor sizing in the energy–delay space using continuous transistor sizing, supply and threshold voltages [27], [32].

Manuscript received December 21, 2007; revised September 11, 2008. Current version published January 27, 2009. This work was supported in part by NSF Award #0238572, NSF Research Infrastructure Grant 0403427, Intel Corporation through a UC Micro grant. ST Microelectronics donated the test chip fabrication. This work was performed while the authors were with the University of California at Berkeley.

R. Zlatanovici was with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. He is now with the Cadence Research Laboratories, Berkeley, CA 94704 USA (e-mail: zradu@cadence.com).

S. Kao was with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. He is now with Newport Media, Inc., Lake Forest, CA 92630 USA.

B. Nikolić is with the Department of Electrical Engineering and Computer Sciences, University of California Berkeley, CA 94720 USA (e-mail: bora@eecs.berkeley.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2008.2010795

This paper presents a thorough analysis of the design tradeoffs for 64-bit carry-lookahead adders in a typical high-performance microprocessor environment and a practical design of an optimal adder in a general purpose 90 nm CMOS process. It builds upon our previous work on optimal transistor sizing in the energy–delay space using convex and tabulated models [19]. The impact in the power–performance space is analyzed for design choices in six categories: 1) set of equations; 2) logic family; 3) radix of the carry tree; 4) lateral fanout of the carry tree; 5) sparseness of the carry tree; and 6) sizing strategy. We generalize the results by normalizing them against general parameters that characterize a technology. The detailed implementation example is a design of an energy-optimized 64-bit carry lookahead adder using Ling's equations, that achieves 240 ps delay [10].

Section II presents a brief description of the circuit optimization framework used to perform the analysis and to design the example adder. Section III analyzes each of the various adder design choices and their impact in the energy–delay space for a 90 nm CMOS process. Section III-A defines the design choices and establishes a common set of notations. Section III-B shows the actual analysis, highlighting the resulting design rules. Section III-C extends the analysis for different processes and environments. Section IV presents the design of the optimal 64-bit adder in a general purpose 90 nm CMOS with implementation details and measured results. Section V concludes the paper.

II. Circuit Optimization Framework

This section briefly describes the circuit optimization used to size the gates in all the 64-bit adders discussed in this paper; a more detailed description can be found in [20].

The framework is built around a versatile optimization core consisting of a static timer in the loop of a mathematical optimizer. The circuit is defined using a SPICE-like netlist and the static timer employs user-specified models in order to compute delays, cycle times, power, signal slopes and node loads.

In this work, the optimization framework is configured to solve the following optimization problem:

$$\begin{aligned} \min_{W_i} \quad & t_D \\ \text{subject to}: \quad & E \le E_{\max}, \quad t_{s,j} \le t_{s,\max}, \\ & C_{in,k} \le C_{in,k,\max}, \quad W_i \ge 1 \end{aligned} \tag{1}$$

where t_D is the delay of the circuit, W_i is the size of gate i, E is the total energy of the circuit, $t_{s,j}$ is the signal slope at node j and $C_{in,k}$ is the input capacitance at primary input k.

By solving this problem for different values of the energy constraint, the optimal energy–delay tradeoff curve for that circuit is obtained. Critical 64-bit adders in microprocessor cores are usually designed close to the minimum achievable delay point, and their power consumption can be reduced with a small performance penalty because the energy–delay tradeoff curve is very steep in that region [31].

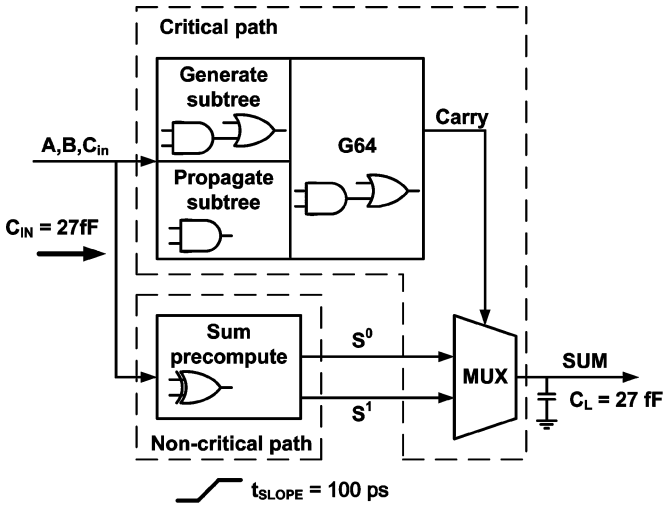

Fig. 1. Generic 64-bit adder block diagram and optimization setup.

III. Optimization of Adders in the Energy-Delay Space

In order to determine an optimal design and make a fair comparison between various implementations, a generic 64-bit adder structure is constructed, as shown in Fig. 1. It is a conventional architecture, featuring a carry tree, a sum-precompute block and a sum-select multiplexer.

The carry tree is composed of two sub-trees: a generate sub-tree, implementing the AND-OR equations of the generate terms (G) and a propagate sub-tree implementing the AND equations of the propagate terms (P). The sum-precompute block precomputes the sums at each bit assuming incoming carries of 0 and 1 for the two conditional sums, S^0 and S^1, respectively. The final multiplexer selects the appropriate sums based on the carry signals computed by the carry block. In most cases, the carry tree and the sum-select multiplexer are in the critical path, while the sum-precompute block is non-critical.

In the subsequent Sections III-A–C, this generic architecture is implemented in a general-purpose 90 nm CMOS process by varying the set of equations, logic family, carry tree architecture, sizing strategy and technology parameters. The optimization problem (1) is solved for different energy constraints for each architecture and a corresponding optimal energy–delay tradeoff curve is plotted in order to analyze the impact of these choices in the energy–delay space. The surrounding environment, as shown in Fig. 1, is the same for all the adders optimized in the subsequent subsections. In a high-performance integer execution unit, the microarchitecture sets the constraints for the adder design. The selected bitslice height of 24 metal tracks accommodates a wide-issue architecture. The bitslice height and adder topology determine the length of the long carry wires in the tree. Unless specified otherwise, the long wires are routed with double-width and double-spacing. In this study, the input capacitance per bitslice is limited to 27 fF, which is approximately equivalent to the capacitance of 25 minimum-size inverters. The output loading capacitance of the adder is chosen to equal its input capacitance, assuming that a buffer would be used to efficiently drive the local loop-back bus. The output capacitance and bitslice height are changed only in the analysis in Section III-C to reflect different adder environments. The

10%–90% rise/fall times in the circuit are constrained to 100 ps, to maintain signal integrity.

A. Design Choices

1) Set of Logic Equations: The conventional implementation of the carry-lookahead adder uses the traditional generate-propagate equations [22]. If a_i and b_i are the input operands, then propagate and generate signals, p_i and g_i, and the sum bits, S_i, can be computed using the following recursive equations:

$$\begin{aligned} g_i &= a_i b_i; \quad p_i = a_i + b_i \\ G_{i:0} &= g_i + p_i G_{i-1:0}; \quad S_i = a_i \oplus b_i \oplus G_{i-1:0}, \end{aligned} \tag{2}$$

where $G_{i:0}$ denotes the generate signal for a group of bits from position 0 to i.

The quantity that is propagated to the next stage is the carry-out at bit i. Ling's equations [13] are an alternative to the classical CLA. By identifying that $p_i g_i = g_i$, the generate term $G_{i:0}$, can be reformulated as

$$G_{i:0} = p_i \left(g_i + G_{i-1:0}\right) = p_i H_{i-1:0}. \tag{3}$$

In Ling's adder, the pseudo-carry H_i is propagated, and combined with the remaining terms in the final sum:

$$H_{i:0} = g_i + p_{i-1} H_{i-1:0}; \quad S_i = p_i \oplus H_{i:0} + g_i p_{i-1} H_{i-1:0}. \tag{4}$$

The advantage of using Ling's equations comes after unrolling the recursions [14]. For instance, unrolling the recursions (2) and (3) for a group of 4 bits results in

$$\begin{aligned} G_3 &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 \\ H_3 &= g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0. \end{aligned} \tag{5}$$

The H_3 term has fewer factors than the G_3, which in CMOS requires fewer transistors in the stack of the first gate. However, the sum computation when using Ling's pseudo-carry equations is more complex. For a conventional CLA, the sum-precompute block from Fig. 1 must implement

$$S_i^0 = a_i \oplus b_i; \quad S_i^1 = (a_i \oplus b_i)' \tag{6}$$

where S_i^0 is the precomputed value of the sum for an incoming carry (G_i) of zero and S_i^1 for an incoming carry of one. If a Ling-CLA scheme is used, S_i^1 changes to

$$S_i^1 = a_i \oplus b_i \oplus (a_{i-1} + b_{i-1}) \tag{7}$$

which is more complex to implement. Ling's equations effectively move complexity from the carry tree into the sum-precompute block.

2) Logic Family: A set of logic family choices that we examine include static CMOS, domino, compound domino and compound domino with stack height reduction. In this context, the term "domino logic" is used for a family in which a dynamic gate is always followed by a static inverter. By contrast, in "compound domino logic" a dynamic gate can be followed by an arbitrary static gate, such as an AND-OR-INV that can merge several carry signals. In both cases, the sizing of static gates is skewed, having the pull-up strength larger than the pull-down. In this analysis, we are using footless domino logic in every stage, which is enabled by the fact that all inputs are monotonic. Since the monotonicity of the input signals cannot be guaranteed in general, the first stage is implemented using footed domino.

It is possible to reduce the transistor stack height by two in the gates by reformulating the logic equations using the absorption property [7], which transforms the unrolled carry lookahead equation (4) into

$$\begin{aligned} G_3 &= \left(g_3 + g_2 + g_1 + p_1 g_0\right)\left(g_3 + p_3 p_2\right) \\ H_3 &= \left(g_3 + g_2 + g_1 + g_0\right)\left(g_3 + p_2 p_1\right). \end{aligned} \tag{8}$$

These equations can be implemented by two parallel dynamic gates followed by a (skewed) static NOR2 gate. Such an implementation is further referred to as "compound domino with stack height reduction".

3) Carry Tree Radix: The focus of this paper is on 64-bit adders; however, all carry trees in Figs. 2 and 3 are for 16-bit trees, for simplicity. Carry-in and carry-out signals are omitted from the figures for the same reason, although they are included in the optimization. The common legend for all carry tree drawings (similar to [15]) is:

- white square: generate/propagate gate;
- black circle: carry merge gate;
- white rhomboid: sum select multiplexer.

The radix (also known as the valency, [26]) of a carry tree is defined as the number of carries merged in each step. A radix-2 (R2) carry tree is shown in Fig. 2(a) and a radix-4 (R4) tree in Fig. 2(e). The radix determines the number of stages needed in a tree in order to compute all the required carries. A 64-bit adder requires three R4 stages or six R2 stages. Mixed-radix trees can also be used; for instance a 64-bit carry tree can be implemented in four stages using a radix 4-3-3-2 scheme, where radix decreases from the output toward the input. In addition to the radix, lateral fanout inside the tree fully determines its topology.

4) Lateral Fanout: Ladner and Fischer introduced the minimum-depth prefix graph [24] based on earlier theoretical work [25]. The lateral fanout of a stage is defined as the maximum number of nodes driven by a node in the current stage. The longest lateral spanning wires in a Ladner–Fischer tree go from one node in the tree to $N/2$ other nodes. Branching loads become particularly large for later stages in the graph, as increasing logical fanout combines with the increasing span of the wires. Kogge and Stone [6] addressed this fanout issue by introducing the "recursive doubling" algorithm. Using the idempotency property, the lateral fanout at each node is limited to one, at the cost of an increase in the number of lateral wires and logic gates at each level. Knowles, by indicating lateral fanouts, has introduced a unique notation for the minimum-depth adders [23] that spans all the range from Ladner–Fischer to Kogge–Stone. Several terms have been used for "lateral fanout" in the literature, among which are "branching" [16] and simply "fanout" [26], [28]. In this paper the term "lateral fanout" is used with the same meaning as in [23], in order to avoid confusion with the electrical fanout of the gate.

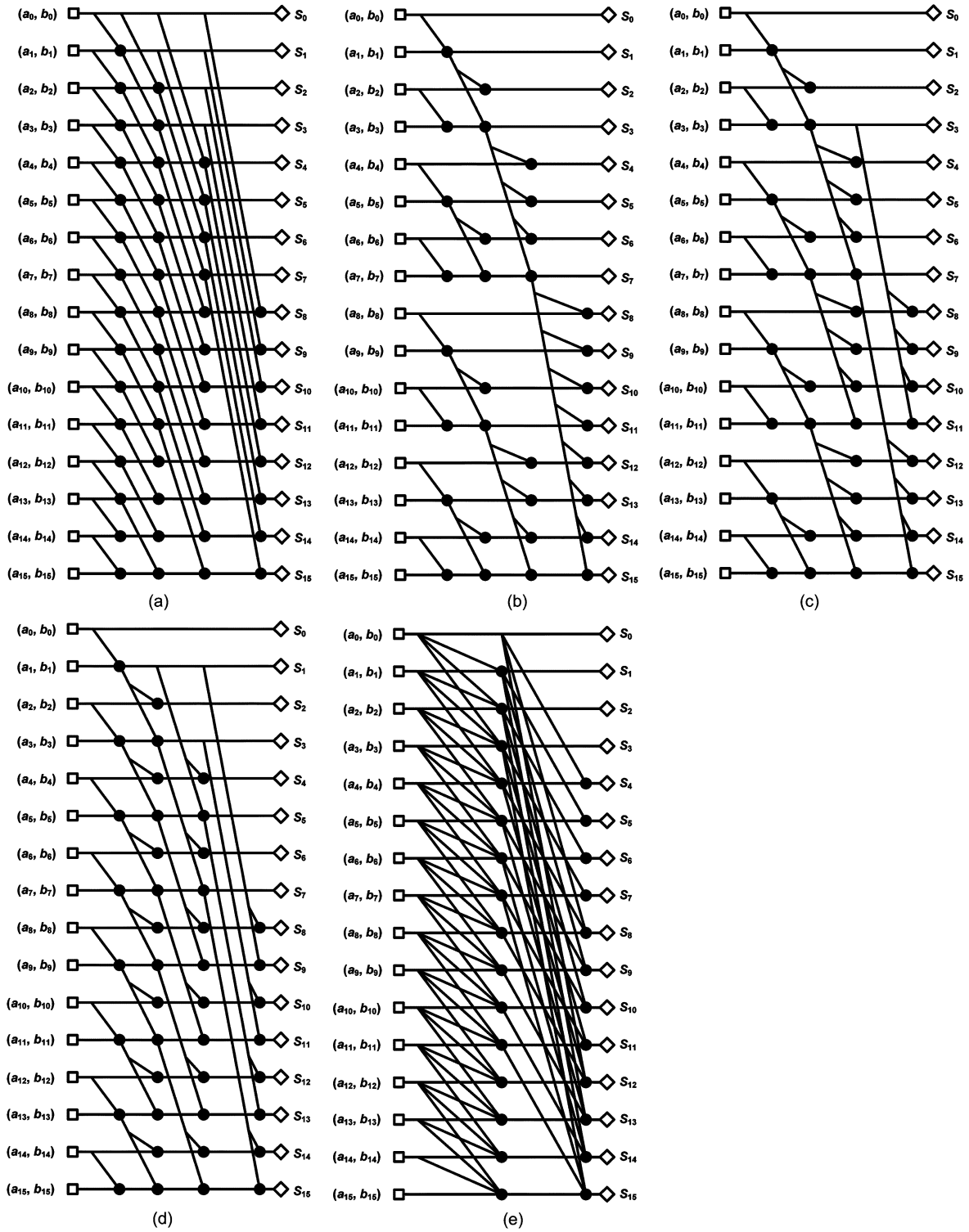

Fig. 2. 16-bit full trees: (a) radix-2 1-1-1-1 (Kogge–Stone); (b) radix-2 8-4-2-1 (Ladner–Fischer); (c) radix-2 4-4-2-1; (d) radix-2 2-2-2-1; (e) radix-4 1-1 (Kogge–Stone).

A sample of the Knowles's family of R2 minimum depth trees are shown in Fig. 2(b)–(d) (R2: 8-4-2-1 -the original Ladner–Fischer tree, R2: 4-4-2-1, and R2: 2-2-2-1). The lateral fanouts are indicated from the stage nearest to the outputs, back toward the inputs. The tree in Fig. 2(a) is a R2: 1-1-1-1 tree (Kogge–Stone). Knowles's labeling can be extended to higher radix trees as well—for instance the R4 tree from Fig. 2(e) is an R4: 1-1 tree, hence a Kogge–Stone tree.

5) Tree Sparseness: All the carry trees discussed so far are "full" trees because they compute the final carry at every bit. It is possible to compute only some of the carries and select the sum based only on the available carry bits. For instance, one can compute only the even carries $(H_0, H_2, H_4, \ldots, H_{62})$ in the CLA block and use them to select the multiplexers in the sum-select stage. The gates and wires corresponding to the eliminated carries are pruned down, dramatically reducing the com-

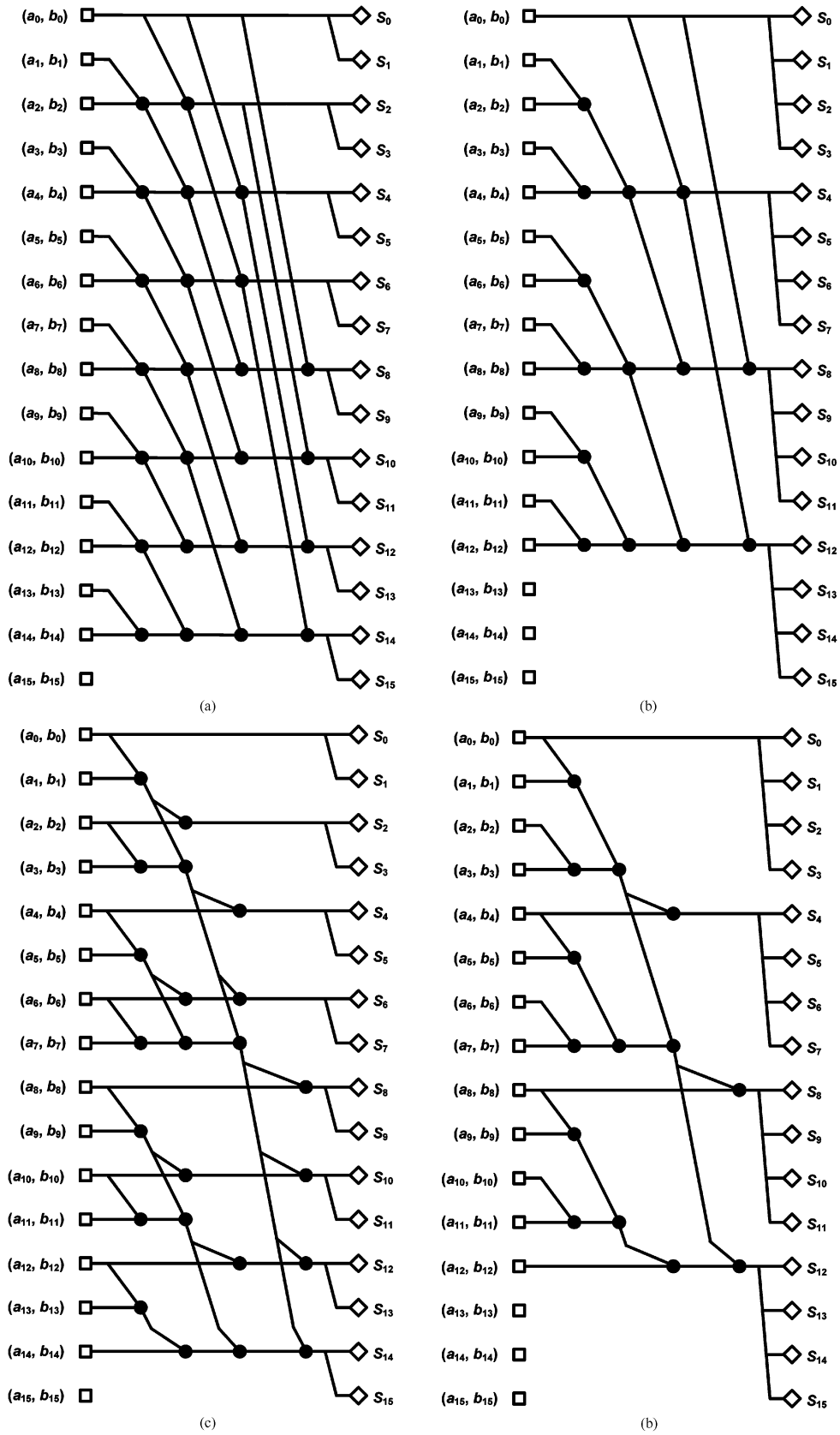

Fig. 3. 16-bit radix-2 sparse trees: (a) 1-1-1-1 sparse-2; (b) 1-1-1-1 sparse-4; (c) 8-4-2-1 sparse-2; (d) 8-4-2-1 sparse-4.

plexity of the tree. The resulting tree is sparse, with a sparseness of 2, as exemplified in Fig. 3(a).

The sum-precompute block is more complex in sparse trees, but still can be removed from the critical path. Even-order pre-

computed values for the sum are given by (6) for Ling's carry scheme, but odd-order sums must be pre-computed by unrolling the carry recursion (3) once:

$$S_i^0 = a_i \oplus b_i \oplus (a_{i-1}b_{i-1})$$
$$S_i^1 = a_i \oplus b_i \oplus [a_{i-1}b_{i-1} + (a_{i-1} + b_{i-1})(a_{i-2} + b_{i-2})]. \quad (9)$$

Sparseness can be larger than two and can be applied to any carry tree. A sparse-4 version of the tree in Fig. 2(a) is shown in Fig. 3(b). In this case, the carry recursion must be unrolled once, twice and three times repeatedly for every four bitslices in the sum precompute block. In this case, the logic depth of the sum-precompute block matches the logic depth of the carry tree.

The sparse-2 (SP2) and sparse-4 (SP4) versions of the Ladner–Fischer tree are shown in Fig. 3(c)–(d). It should be noted that sparseness reduces the actual lateral branching in the tree: the third stage has a branching of 8 in the full tree, 4 in the SP2 tree, and 2 in the SP4 tree. However, in order to uniquely identify the tree and keep sparseness as an independent variable, the lateral fanout notation corresponding to the full tree is used for all its sparse versions. Therefore, although the tree from Fig. 3(d) has lateral branching of 2, 2, 1, and 1, it is labeled as "R2: 8-4-2-1 SP4" because it is a sparse-4 version of the full R2: 8-4-2-1 tree (from Fig. 2). The design from [1] is an implementation of a 64-bit version of this tree (Fig. 3(d)).

By using this notation, the space of minimum-depth carry trees can be represented in a space with three independent dimensions: radix, lateral fanout and sparseness [26].

6) Sizing Strategy: An adder is a regular structure that is suited for bit-sliced implementations. A common sizing strategy is to group all identical gates in a stage such that they have the same size. Gate grouping speeds up the design process by reducing the number of variables in the optimization and by allowing layout reuse. The size and distribution of groups can be varied. For example, to reduce the timing window in footless domino implementation, some of the lower bits in higher stages can be downsized or footed. Ultimately, each gate could be individually sized (flat sizing).

B. Analysis of Design Choices

1) Set of Logic Equations: Fig. 4 shows the energy–delay tradeoff curves for R4 and R2 domino adders using conventional CLA and Ling equations. For high speeds, where the carry tree is in the critical path, using Ling's equations is advantageous: by lowering the stack height in the first stage, Ling's equations allow the first gate in the carry tree to be larger for the same input capacitance. When driving the same load (next stage and the subsequent interconnect), the delay decreases. At longer delays with most gates minimum sized, the sum-precompute block appears in the critical path and the conventional CLA equations offer lower energy.

2) Logic Style: Fig. 5 presents the optimal energy–delay tradeoff curves for 64-bit adders for the architecture from Fig. 1 in the four logic families from Section III-A2: R2 and R4 domino. For long cycle times a static implementation is preferred due to its low power consumption. Due to their high

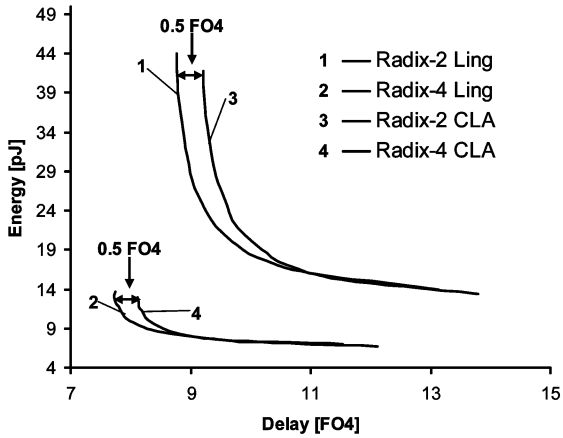

Fig. 4. Energy–delay tradeoff curves for domino adders implemented using classical CLA and Ling equations, with Kogge–Stone trees and grouped sizing strategy.

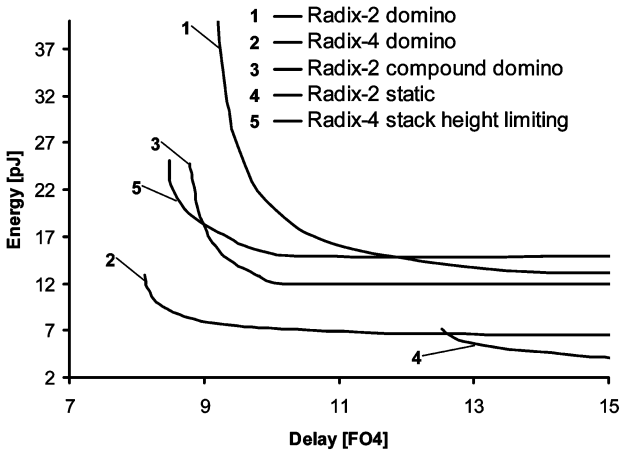

Fig. 5. Energy–delay tradeoff curves for adders implemented in various logic styles, Kogge–Stone trees, grouped sizing strategy.

activity factors, dynamic circuits cannot translate the extra slack into power savings for cycle times beyond 9 FO4. However, static R2: 1-1-1-1-1-1 adders can only achieve the minimum of 12.5 FO4 delays. If the required delay is shorter than 12.5 FO4, a dynamic design is required. We did not explore further optimization of static adders.

The R4 domino design has the best energy–delay performance at high speeds, in the typical design environment selected for this exploration, Fig. 5. R2 compound domino design approaches the R4 domino design in speed, but has higher energy. R2 compound domino adders can be implemented in the same number of stages as a R4 domino. However, they suffer from an increased impact of the wires: compound domino adders have dynamic wires that traverse multiple bitslices and prudent design practices require that such wires be shielded. In this example, all dynamic wires longer than the height of a bitslice are shielded on both sides with power and ground tracks at the minimum spacing allowed in the routing layer. All adders are optimized for the exact same conditions, including wires; however, the shields required by compound domino consume routing resources, increasing the actual length of the wires. This added capacitance is in the critical path, leading

to an increase in delay and power consumption. This reduces the performance of compound domino adders and is taken into account in Fig. 5. The same factors limit the performance of higher-radix compound domino adders.

If the extra loading due to shielding is ignored and all dynamic wires are routed in the same "double-width, double-space" style, the delay of compound domino adders can be reduced by approximately 1 FO4, making them slightly faster than their domino counterparts. A similar result is obtained by the authors of [29] using logical effort minimum-delay sizing with an Elmore-based delay formula for the wires in a 130 nm technology with consistent wire loading across all logic families.

The logic design of the adder that uses stack height limiting, implemented in compound domino, recuperates the speed loss due to extra wiring capacitance because all long wires are driven by static gates. However, these long wires must be driven by a stack of 2 PMOS transistors in the NOR2 gates characteristic for this logic implementation. While providing relatively short delays due to a small number of stages and reduced stack height, the large PMOS transistors in the NOR2 gates and the high count of transistors and clocked gates increase the power consumption of this logic family.

3) Radix of the Carry Tree: Fig. 6 shows the optimal energy–delay tradeoff curves for 64-bit domino adders implemented with carry trees with different radices. For the chosen loading conditions, R4 trees are closer to the optimal number of stages [21] and achieve the best performance and lowest power. Using the logical effort formalism from [21] it can be easily shown that if wire loads and stack effects are ignored, all minimum depth carry trees have the same branching effort and the same logical effort. For the same external loading conditions (i.e., the same electrical effort), the overall effort of the carry tree is a constant regardless of the chosen architecture; in this case the overall fanout is 1, branching factor is 64 and the optimal stage effort is 3 for a domino stage (composed of a dynamic gate followed by a static gate). Therefore, the optimal number of stages is $\log_3(64) = 3.8$. With wires included, the optimal number of stages increases to between 4 and 5. The R4 trees have 4 domino stages—three in the actual carry tree and one in the sum select multiplexer. As shown in Fig. 6, R2 adders (6 stages) are the slowest, mixed-radix 4-3-3-2 adders (4 stages) have significantly better performance and R4 adders (3 stages) are the fastest. The result holds for larger loads as well: if the optimal number of stages increases, it is always better to drive high loads with buffers (inverters) rather than with the last gates of the adder (complex AOI gates or multiplexers).

The assumptions that limit the validity of the effort analysis are: the effect of the wire loads and correct accounting for transistor stacks.

- Higher radix trees will have longer wires closer to the inputs: the first stage of a R4 tree needs to drive wires spanning 12 bitslices; the first stage of a mixed radix 4-3-3-2 tree drives wires spanning 8 bitslices; the first stage of a R2 tree drives wires spanning just 2 bitslices. The advantage of a high-radix tree is eroded in processes with increased relative wire capacitance and resistance.

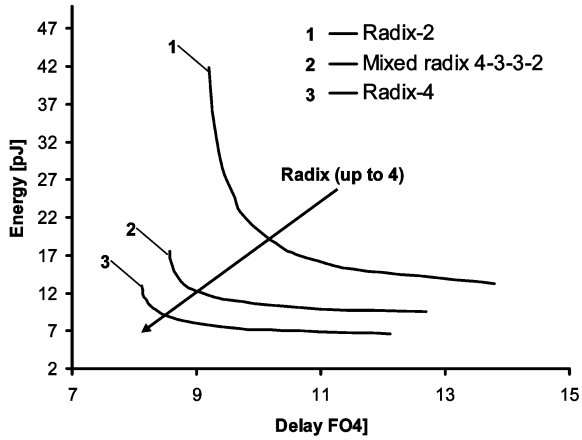

Fig. 6. Energy–delay tradeoff curves for adders implemented with Kogge–Stone trees of various radices, grouped sizing strategy.

- The maximum transistor stack height in a carry tree is usually equal to the radix of the stage. However, the tallest stack that can be used is effectively limited by the reduced V_{DD}/V_{TH} ratio of deep submicron technologies. A tall stack can cause a slowdown beyond the simple logical effort calculations, along with significant signal slope degradation, as in the case of higher radix trees (R6 and R8). In such situations, stack height limiting in compound domino can help maintain an adequate performance for the adder.

4) Lateral Fanout: The optimal energy–delay tradeoff curves for R2 trees with increasing lateral fanouts are presented in Fig. 7(a)–(c), for full, SP2 and SP4 trees. Trees with high fanout have low degrees of redundancy, with the Ladner–Fischer tree computing the minimum number of carries and having the highest loading along the critical path. At the other extreme, the Kogge–Stone tree computes all the carries needed to reduce the fanout, offering the lowest loading along the critical path. Consequently, the tradeoff along the fanout axis in the carry tree space is between the length of wires and gate fanout versus number of gates. Fig. 7(a) shows that for full trees, the lower the fanout, the higher the maximum speed. At the other end of the curves, the higher the fanout, the lower the minimum achievable energy. Figs. 7(b) and (c) show the same effect for SP2 and SP4 trees, but with progressively smaller differences. Although the order is maintained throughout the whole spectrum of designs, the differences in performance and power among SP4 trees are negligible across the fanout range. The reduced impact of lateral fanout on very sparse trees is due to the sparseness, which reduces the effective fanout of the tree. As the trees are pruned to SP4, their critical paths become similar, with differences only in the branching factors at the input (lower) and output (higher), hence the similar delays. Although the number of gates is reduced, the increased output branching requires the remaining gates to be upsized by roughly an equal factor, resulting in similar energy consumption as well.

Wire loading is kept at the same capacitance per unit length for all the experiments in Fig. 7(a)–(c). Higher lateral fanout trees, however, can have less capacitance because fewer parallel

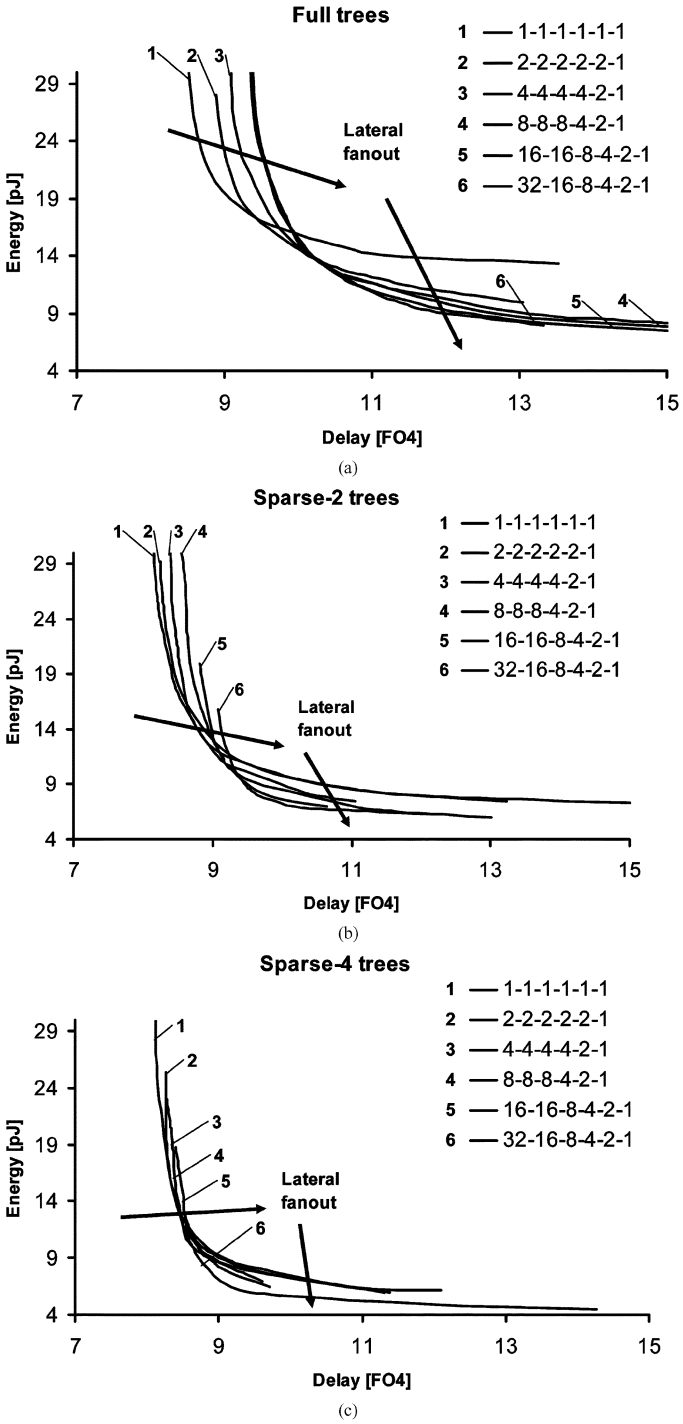

Fig. 7. Energy–delay tradeoff curves for Ling domino adders implemented using radix-2 trees with different lateral fanouts and grouped sizing strategy: (a) full trees; (b) sparse-2 trees; (c) sparse-4 trees.

wiring tracks are required. This allows the remaining carry signals to be routed with increased spacing between the wires, thus reducing the overall capacitive load. In an experiment with the exact same conditions as in Fig. 7(a)–(c), except with wire loadings reduced by 25%, Ladner–Fischer trees gain approximately 0.2 FO4 in speed. This change does not modify the ordering of trees in Fig. 7(a) and (b); however, tradeoff curves in Fig. 7(c) get even closer together, and their differences fall within the model tolerances.

Ordering of optimal designs is different for shorter wordlengths. Recent work [27] shows that for 32-bit adders, where the lateral fanouts are smaller, Ladner–Fischer trees are the best choice for both performance and power. Ladner–Fischer trees have also been the choice in recent practical implementations of adder designs in high-performance microprocessor datapaths that support 16- and 32-bit operations [33].

5) Tree Sparseness: Fig. 8(a)–(d), show the impact of tree sparseness along the fanout dimension of the carry tree space (for lateral fanouts of 1-1-1-1-1-1, 4-4-4-4-2-1, and 32-16-8-4-2-1) for domino and compound domino adders. Fig. 9 shows the impact of sparseness along the radix dimension of the space (for R2 and R4 trees).

Adders with sparse trees differ from full trees in three ways:

- the gates and connections in the carry tree are pruned down, reducing the size of the tree;
- the sum-precompute block becomes more complex because of the unrolling of the carry iteration at bit indexes where carries are not directly available;
- the branching at the output of the carry tree increases.

These differences have several consequences in the power–performance space:

1) a smaller carry tree with less gates and less wires can be upsized for better performance with the same power budget;
2) reduced input loading for the carry block: a sparse tree has fewer gates in the first stage and therefore the load on the input is smaller; thus, for the same input capacitance, the input gates on the critical path can be made larger, resulting in a faster adder; also, as a result, larger gates will drive the internal wiring;
3) larger output load for the carry tree: one carry output must drive a number of gates equal to the sparseness factor of the tree, thus slowing down the circuit. Optimal delay is obtained through upsizing the critical path;
4) reduced internal branching due to gate pruning speeds up the adder; the effect is more pronounced in high fanout trees and nonexistent in Kogge–Stone derivatives (which have a constant fanout);
5) more complex sum-precompute blocks slow down the critical path through additional branching from the input and extra power consumption.

The overall result is a balance of all of the above factors, depending on the topology of the original tree and the technology. Factor 1 is dominant for large trees, with many gates and many wires, such as R2 but less important for the smaller R4 trees, as shown by Fig. 9. Factor 2 is generally small and benefits mostly higher radix trees, where long wires start to appear already after the first stage. Factor 3 mostly affects higher-radix trees with fewer stages available to drive the extra load, such as R4 or compound domino R2, as shown in Figs. 8(d) and 9. Pruning these trees down to SP4 tips the balance of the above factors, resulting in a slower and less energy-efficient adder. Factor 4 is dominant for high-fanout trees, where sparseness provides dramatic speed improvements and power reductions, as shown in Fig. 8(b)–(c). Factor 5 has influence on all trees, but its impact is particularly pronounced for highly sparse trees. All the figures in this subsection reflect a decrease of the power/performance gain at the sparse-2 to sparse-4 transition when compared to

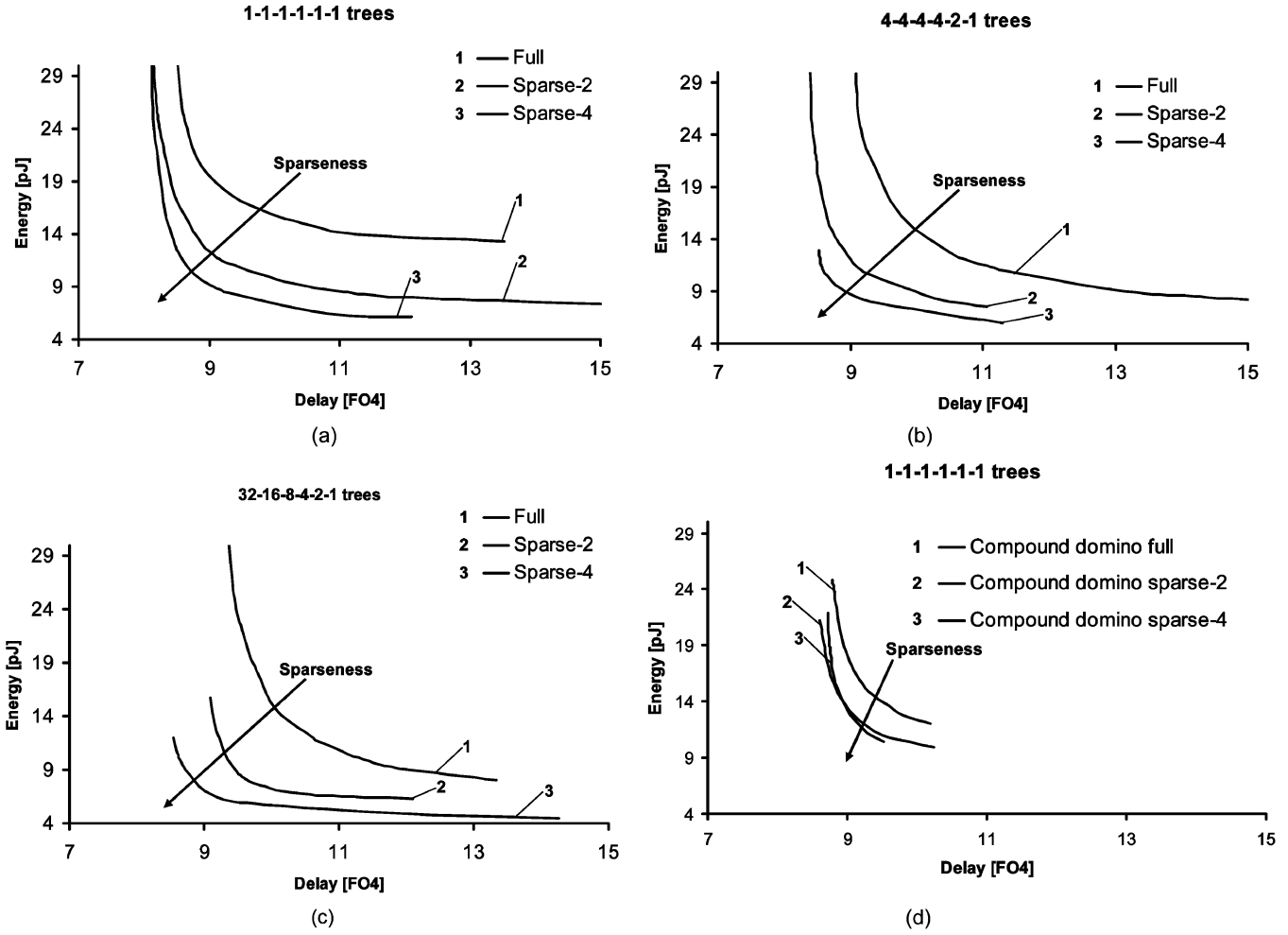

Fig. 8. Energy–delay tradeoff curves for radix-2 Ling adders with different sparseness factors and grouped sizing strategy: (a) domino 1-1-1-1-1-1-1; (b) domino 4-4-4-4-2-1; (c) domino 32-16-8-4-2-1; (d) compound domino 1-1-1-1-1-1-1.

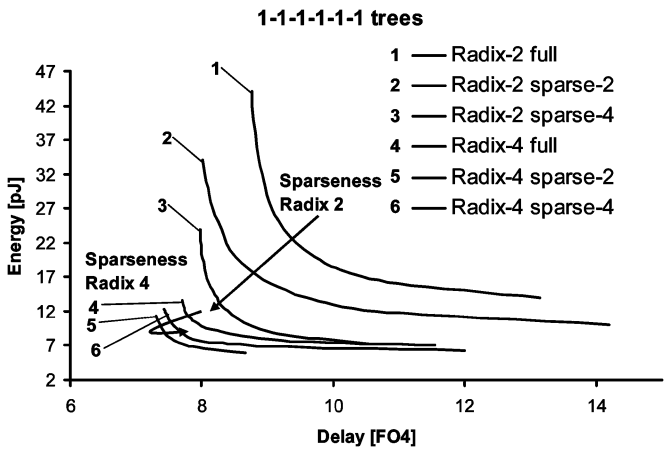

Fig. 9. Energy—delay tradeoff curves for Ling domino adders implemented using 1-1-1-1-1-1-1 trees with different sparseness factors and grouped sizing strategy.

the full to sparse-2 transition. With Ling's equations, pruning a carry tree in order to make it sparse effectively moves complexity from the carry tree to the sum-precompute block. As long as the carry tree remains in the critical path, this move is advantageous.

It should be noted that the complexities in the carry tree and sum-precompute block scale differently with the sparseness factor: for a sparseness factor of N, the carry tree is generally N times smaller than the full tree. However, since the simple ripple-carry design is not fast enough, the sum-precompute block has to repeatedly unroll the carry iteration for each bitslice: if G_i and G_{i+N} are available, the carry iteration needs to be unrolled once at bit index $i+1$, twice at $i+2$, and up to $N-1$ times at $i+N-1$. In N bitslices of a sparse-N tree, the carry iteration needs to be unrolled $1+2+\cdots+N-1 = N*(N-1)/2$ times. Therefore, a $O(N)$ decrease in the complexity of the carry tree leads to a $O(N^2)$ increase in the complexity of the sum-precompute block. Ling adders have more complex sum precompute blocks, with a larger constant for the $O(N^2)$ growth. Thus, the effectiveness of using Ling's equations decreases with higher sparseness factors, compared to the conventional CLA.

6) Sizing Strategy: All the energy–delay tradeoff curves presented in this paper so far use the grouped sizing strategy. While convenient for the implementation, gate grouping negatively impacts the performance and power consumption of the adder.

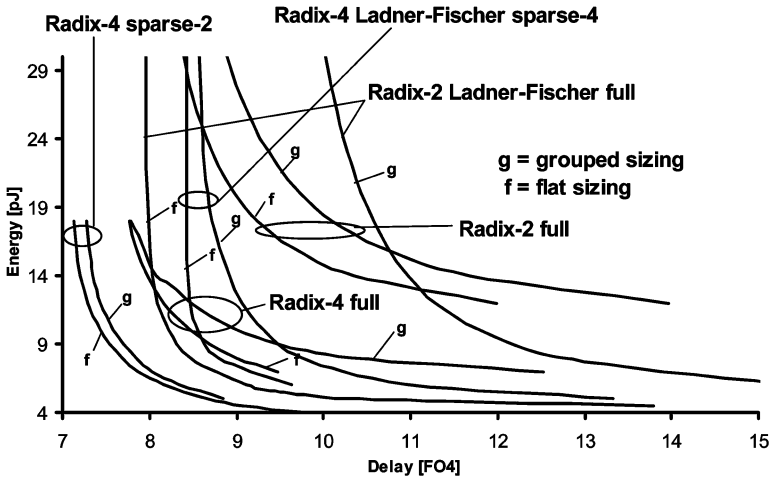

Fig. 10. Energy–delay tradeoffs for various adder trees when using a flat versus grouped sizing strategies.

Fig. 10 shows the impact in the energy–delay space of using a flat sizing strategy as opposed to a grouped sizing strategy for several representative adder topologies. Structures with regular fanout (such as the radix-2 and radix-4 Kogge–Stone trees depicted in Fig. 10) marginally benefit from a flat sizing strategy (3.6% delay improvement). On the other hand, structures with irregular fanout (such as the Ladner–Fischer tree) can be significantly sped up by ungrouping the gates. In such a situation, high fanout gates can be upsized without increasing the power consumption and input loading of lower fanout gates, resulting in an 18% speed increase for the radix-2 Ladner–Fischer tree. The radix-4 Ladner–Fischer tree requires additional buffers in the critical path in order to meet the slope constraint even with the flat sizing strategy.

On the low-power end of the curves, flat sizing offers power savings in similar percentages because ungrouping allows more non-critical gates to be downsized without violating the maximum slew constraints.

Flattening the design and customizing gate sizes will increase the design effort considerably, but will always help the timing in the circuit by aligning the arrival times at the outputs of each stage. However, this makes the circuit more susceptible to process variations due to the degradation of the statistics of the arrival times [28]. The increase in design effort is less pronounced in irregular structures such as the Ladner–Fischer trees where similar fanout gates in the same stage can be grouped together.

C. Fastest Adder Across Different Technologies and Environments

All the adders optimized so far use a reference 90 nm general purpose bulk CMOS process under the environment specified in the beginning of this section. While the design guidelines formulated in the previous subsections are general, the conclusions on which adder architecture is fastest in a given environment are dependent on the parameters of the particular process used for the analysis and the particular environment of the adder.

The optimization framework from Section II can be used to investigate the influence of certain technology parameters on the behavior of digital circuits in the power–performance space. In the case of 64-bit adders, two technology parameters can significantly influence the design choices:

- Wire capacitance ratio, defined as the ratio of the lumped capacitance of 1 μm of carry wires to the capacitance of 1 μm of minimum length transistor gate;
- Self-loading ratio, defined as the ratio of the drain capacitance, including Miller-multiplication, of a 1 μm minimum length transistor to its gate capacitance. The self-loading ratio is directly linked to the normalized intrinsic delay p in the logical effort formalism [21].

The environment of the adder is usually reflected in the height of the bitslice. This, in turn, determines the wire capacitances as well as the output capacitance of the adder. A taller bitslice will increase the wire loads on the internal nodes of the adder but will decrease the output load because the layout will be narrower. The effect of the bitslice height can be presented in the same coordinate space as the technology (wire capacitance ratio, self-loading ratio) by simple scaling operations:

- a taller bitslice is equivalent to a technology with a proportionally higher wire capacitance ratio;
- a smaller output load is equivalent to a technology with a smaller self-loading ratio that will yield the same delay when resized for the new load.

Fig. 11 shows a partition of the wire capacitance ratio, the self-loading ratio space, highlighting the architecture of the fastest adder in each region. In this figure, the delays corresponding to the minimum adder delay are compared across processes with different wire capacitance and self-loading ratios and for different bitslice heights using the above equivalency. The graph in Fig. 11 is normalized to the parameters of the 90 nm process used in this analysis, such that its corresponding normalized wire capacitance and self-loading ratios are both equal to 1. All adders in Fig. 11 use Ling's equations and are implemented in domino logic. For the 90 nm process used in this analysis, the fastest architecture is R4 SP2. This adder has been built in 90 nm CMOS and Section IV presents the design details and measurement results.

Fig. 11 highlights the historical trend for the four bulk CMOS processes from the same foundry. While the optimal architecture is the same in the older three processes, scaling trend points

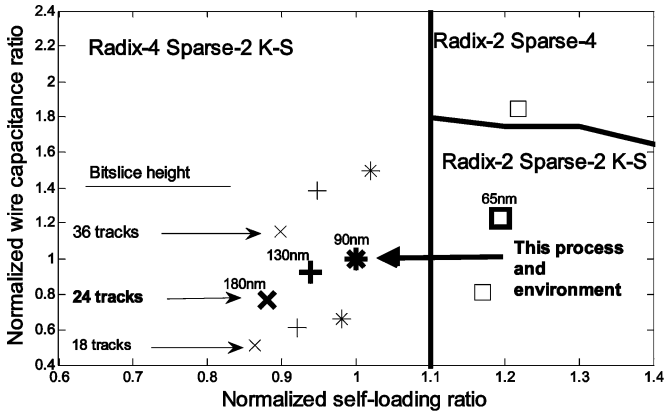

Fig. 11. Fastest adder across different processes and environments.

towards a change in 65 nm technology and beyond. An increased self-loading ratio decreases the driving capability of a gate, degrades the rise/fall times in the circuit and reduces the tallest acceptable transistor stack. Consequently, the highest feasible radix of the carry tree is also reduced, and radix-2 architecture becomes optimal. With a taller stack, the increased self-loading of the gates penalizes architectures with higher radix due to their longer wires and higher branching. The impact of this parameter on 64-bit adder performance is predicted by the analysis from Section III-B3: as shown in Fig. 11, for processes with self-loading ratios 10% greater than the reference process, R2 architectures become faster than R4.

Interconnect parameters can also influence the architecture of the optimal 64-bit adder. If wire capacitance is significant, factor 1 in Section III-B5 becomes dominant and tips the balance toward sparse architectures. Taller datapaths (exemplified by the 36 track example in Fig. 11), or lower input capacitance limits will tend to favor designs with high sparseness and many stages for appropriate tapering (R2). Fig. 11 shows that R2 SP4 adders offer the best performance in very aggressive processes with high wire capacitance and self-loading ratios. At that point, adders with good energy efficiency can be obtained by using designs with high lateral fanout: as shown in Section III-B4 sparse trees with high fanout achieve speeds very close to the low fanout trees but with lower power consumption. By analyzing the self-loading and wire capacitance ratios it should be possible to determine an optimal adder for a given technology foundry.

Finally, the presented analysis is based on conservative and robust design practices for dynamic logic. It is possible to trade off robustness for improvements in energy–delay. For example, reduced shielding requirements favor compound domino logic styles, while the relaxed transition time constraints favor higher fanout trees, like Ladner–Fischer, which may alter Fig. 11.

IV. Design Example in 90 nm CMOS

A test chip implementing the adder with the fastest architecture has been fabricated in a general purpose 90 nm bulk CMOS process using standard V_{TH} transistors. The chip contains eight 64-bit adder cores and the corresponding testing circuitry (Fig. 12), and is 1.6×1.7 mm^2 in size. The size of an

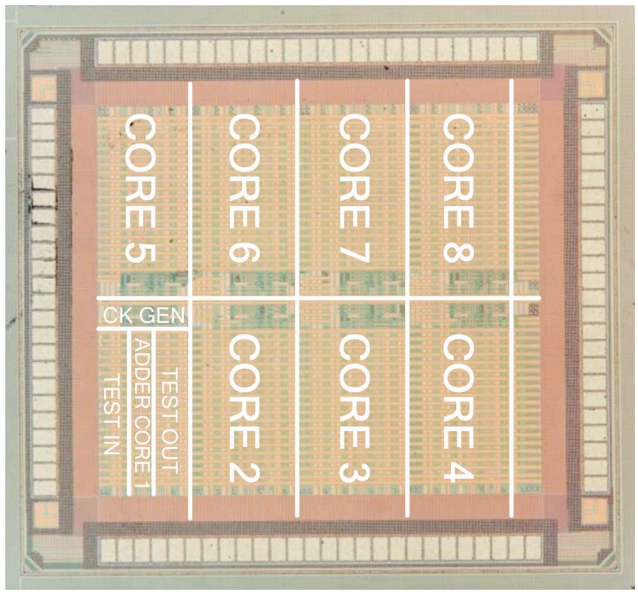

Fig. 12. Test chip micrograph.

actual adder core is 417×75 $\mu\mathrm{m}^2$. The chip is fully custom designed (the only standard cells used are the pin pads) and uses only standard threshold (SVT) devices.

A. Implementation Details

As concluded in Section III, for this process the fastest 64-bit adder architecture uses a R4: 1-1-1-1-1-1 SP2 carry tree implementing Ling's equations in domino logic and the sum-precompute block in static CMOS. A sizing strategy with gate grouping has been used for this adder. Fig. 13(a) shows the block diagram of the adder and on-chip testing circuitry, highlighting the logic families used in the core, and the corresponding timing diagram is shown in Fig. 13(b).

Delayed-precharge domino logic is used in the carry tree in order to hide the precharge phase from the overall cycle time. Most stages in the critical path use footless domino logic with stack node precharging when needed. Since the monotonicity of the global inputs $A[63:0]$ and $B[63:0]$ cannot be guaranteed, the first stage is implemented using footed domino logic. The inputs of the sum-select mux, $S0[63:0]$, $S1[63:0]$, are outputs of a static block and non-monotonic thus *psel* must be a hard clock edge (Fig. 13(b)). Critical timing edge arrivals can be fine tuned through the chip's scan chain in order to ensure correct functionality and best performance.

Using footless domino logic where possible increases the speed of the adder and reduces stack heights and transistors counts. All the gates are sized using the optimization framework described in Section II. Dynamic gates have a minimum sized PMOS keeper.

The challenge of using footless domino with large grouping can be seen in Fig. 13, where the precharge window shrinks toward the last stage. Any dynamic gate (footless or footed) must be in evaluation when its latest input arrives, in order to ensure correct operation of the circuit. In contrast to a footed gate, a footless gate must also be in evaluation when its earliest inputs arrive. When moving further away from the inputs, the

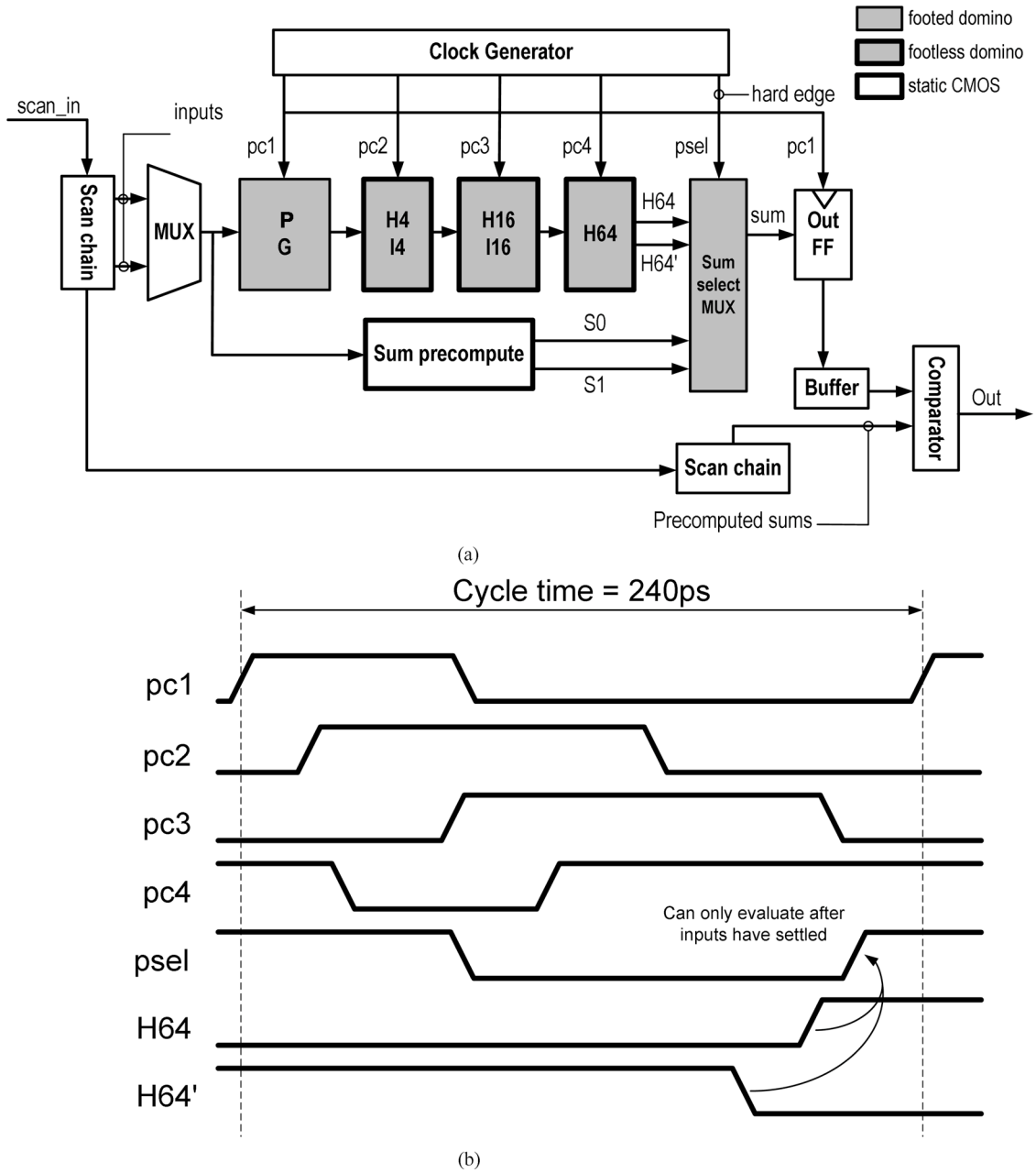

Fig. 13. Adder and testing circuitry: (a) block diagram and (b) corresponding timing diagram.

spread between the fastest and slowest path in the circuit increases, thus increasing the time a footless domino gate must stay in evaluation. Consequently, the precharge phase becomes critical and requires large clock drivers. Footless domino gates require bigger precharge transistors that slow down the evaluation path through their drain capacitance and increase the power consumption on the clock lines and in the clock distribution network.

The sparse 2 carry tree computes only even-order carries and each signal selects two sums (Fig. 14). The non-critical sum precompute block has two types of paths: odd-order sums are precomputed using (6); even-order sums are a function of the carry into the previous bit and are computed using (7). The layout of critical and non-critical paths are interleaved such that the more complex even-order sum-precompute gates fit in the space opened up by the eliminated carry gates of the sparse tree, resulting in a compact bit-sliced layout and straight clock lines, as shown in Fig. 15.

B. Measured Results

Measured results are presented in Fig. 16(a) and (b). At the nominal supply of 1 V the adder core runs at an average speed of 4.2 GHz (240 ps, approx. 7.7 FO4) for the slowest input vector and consumes 260 mW for the maximum power input vector, with 2.3 mW of leakage at room temperature. The power measurements include the adder core and clock generation circuitry but not the on-chip testing circuitry. By increasing the supply voltage to 1.3 V the delay decreases to 180 ps with 606 mW of active power and 4.9 mW of leakage.

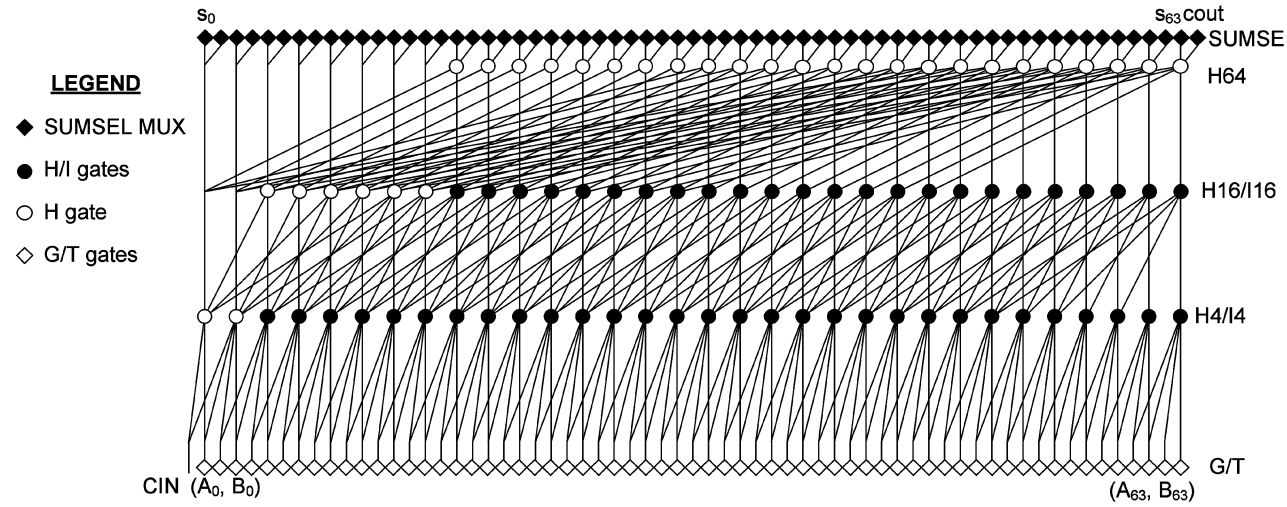

Fig. 14. 64-bit radix-4 1-1-1 sparse-2 carry tree.

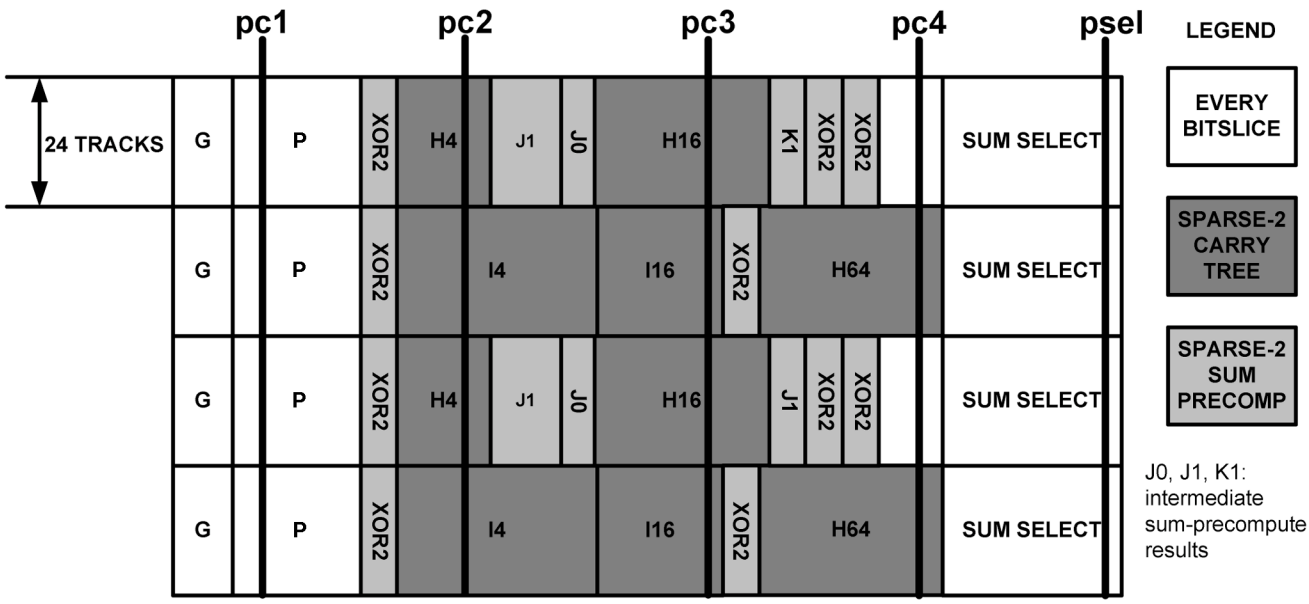

Fig. 15. Sparse-2 adder floorplan for four bit slices.

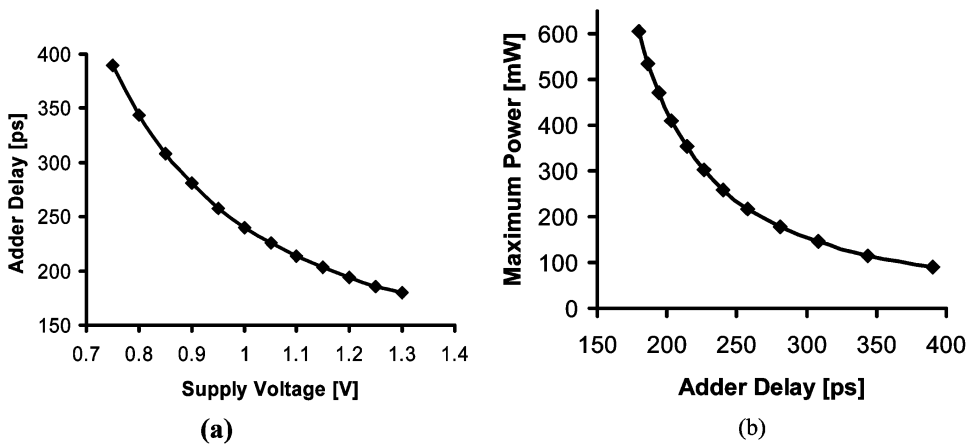

Fig. 16. Measured results for the 90 nm testchip: (a) delay; (b) maximum power.

V. Conclusion

A circuit optimization framework is used to size the gates in various 64-bit adders in a typical multi-issue high-performance microprocessor environment. By analyzing the impact of the main design choices on adder behavior in the energy–delay space, a set of guidelines can be established to guide the designer when choosing the architecture of a 64-bit adder:

- Ling's equations have a speed advantage over conventional CLA at very high speeds;
- for delay requirements longer than 12.5 FO4, static CMOS consumes less power than domino;
- the highest feasible radix for the carry tree has the lowest delay and energy;
- the lowest lateral fanout (Kogge–Stone trees) results in the highest speed; the highest lateral fanout (Ladner–Fischer trees) has lowest power;
- sparseness reduces the impact of lateral fanout;
- sparseness is most beneficial for adders with large carry trees, high lateral fanout and relatively small sum-precompute blocks;
- individual gate sizing is most beneficial for adders with irregular structure and high fanout. Gate grouping has low impact for regular adders.

The presented analysis is comprehensive, but by no means complete as the explored space is virtually infinite. The optimal adder depends on the design constraints, the environment, and the parameters of the process. Several architectures are often close to the optimum performance for the given design environment. Process trends show a change in the optimal architecture toward adders with a lower radix and a higher sparsness factor. A testchip has been fabricated in a general purpose 90 nm bulk CMOS process to test the optimal adder for that particular process. Measurement results for the domino Ling R4 SP2 Kogge–Stone adder show a delay of 240 ps for 260 mW power consumption at the nominal supply voltage of 1 V.

Acknowledgment

The authors acknowledge the contributions of the students, faculty and sponsors of the Berkeley Wireless Research Center and would like to particularly thank Young Yang for help in the design of test circuitry, Ryan Roberts for the board design and Brian Richards for tools support. Sanu Mathew and Ram Krishnamurthy helped motivate this project through numerous discussions. The authors also thank the anonymous reviewers for their thoughtful comments that helped improve this paper.

References

[1] S. Mathew, M. A. Anders, B. Bloechel, T. Nguyen, R. K. Krishnamurthy, and S. Borkar, "A 4 GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 44–51, Jan. 2005.

[2] E. S. Fetzer, M. Gibson, A. Klein, N. Calick, Z. Chengyu, E. Busta, and B. Mohammad, "A fully bypassed six-issue integer datapath and register file on the Itanium-2 microprocessor," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1433–1430, Nov. 2002.

[3] S. Naffziger, B. Stackhouse, T. Grutkowski, D. Josephson, J. Desai, E. Alon, and M. Horowitz, "The implementation of a 2-core multi-threaded itanium family processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 197–209, Jan. 2006.

[4] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang, "A dual-core multi-threaded Xeon processor with 16 MB L3 cache," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2006, pp. 102–103.

[5] M. Golden, S. Arekapudi, G. Dabney, M. Haertel, S. Hale, L. Herlinger, Y. Kim, K. McGrath, V. Palisetti, and M. Singh, "A 2.6 GHz dual-core 64b × 86 microprocessor with DDR2 memory support," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2006, pp. 104–105.

[6] P. M. Kogge and H. S. Stone, "A parallel algorithm for efficient solution of a general class of recursive equations," *IEEE Trans. Computers*, vol. 22, no. 8, pp. 786–793, Aug. 1973.

[7] J. Park, H. C. Ngo, J. A. Silberman, and S. H. Dong, "470 ps 64 bit parallel binary adder," in *Symp. VLSI Circuits*, Jun. 2000, pp. 192–193.

[8] T. Han and D. A. Carlson, "Fast area efficient VLSI adders," in *Proc. 8th Symp. Computer Arithmetic*, May 1987, pp. 49–56.

[9] S. Mathew, R. Krishnamurthy, M. Anders, R. Rios, K. Mistry, and K. Soumyanath, "Sub-500 ps 64b ALUs in 0.18 μm SOI/bulk CMOS: Design and scaling trends," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2001, pp. 318–319.

[10] S. Kao, R. Zlatanovici, and B. Nikolić, "A 240 ps 64b carry-lookahead adder in 90 nm CMOS," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2006, pp. 438–439.

[11] S. Naffziger, "A sub-nanosecond 0.5 μm 64b adder design," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 1996, pp. 210–211.

[12] Y. Shimazaki, R. Zlatanovici, and B. Nikolić, "A shared-well dual-supply-voltage 64-bit ALU," *IEEE J. Solid-State Circuits*, vol. 39, pp. 494–500, Mar. 2004.

[13] H. Ling, "High speed binary adder," *IBM J. Res. Develop.*, vol. 25, no. 3, pp. 156–166, May 1981.

[14] R. W. Doran, "Variants of an improved carry-lookahead adder," *IEEE Trans. Computers*, vol. 37, pp. 1110–1113, Sep. 1988.

[15] Z. Huang and M. D. Ercegovac, "Effect of wire delay on the design of prefix adders in deep-submicron technology," in *Proc. 34th Asilomar Conf. Signals, Systems and Computers*, Oct. 2000, vol. 2, pp. 1713–1717.

[16] A. Beaumont-Smith and C. C. Lim, "Parallel prefix adder design," in *Proc. 15th Symp. Computer Arithmetic*, Jun. 2001, pp. 218–225.

[17] H. Q. Dao and V. Oklobdzija, "Application of logical effort techniques for speed optimisation and analysis of representative adders," in *Proc. 35th Asilomar Conf. Signals, Systems and Computers*, Nov. 2001, vol. 2, pp. 1666–1669.

[18] H. Q. Dao, B. R. Zeydel, and V. G. Oklobdzija, "Energy minimization method for optimal energy—Delay extraction," in *Proc. 29th European Solid State Circuits Conf.*, Sep. 2003, pp. 177–180.

[19] R. Zlatanovici and B. Nikolić, "Power—Performance optimal 64-bit carry-lookahead adders," in *Proc. 29th European Solid State Circuits Conf.*, Sep. 2003, pp. 321–324.

[20] R. Zlatanovici and B. Nikolić, "Power–performance optimization for custom digital circuits," *J. Low Power Electron.*, vol. 2, no. 1, pp. 1–8, Apr. 2006.

[21] I. Sutherland, R. Sproul, and D. Harris, *Logical Effort*. New York: Morgan-Kaufmann, 1999.

[22] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[23] S. Knowles, "A family of adders," in *Proc. 15th Symp. Computer Arithmetic*, Jun. 2001, pp. 277–281.

[24] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[25] Y. Ofman, "On the algorithmic complexity of discrete functions," *Soviet Physics—Doklady*, vol. 7, no. 7, pp. 589–591, Jan. 1963.

[26] D. Harris, "A taxonomy of parallel prefix networks," in *Proc. 37th Asilomar Conf. Computing, Circuits and Systems*, Nov. 2003, pp. 2213–2217.

[27] D. Patil, O. Azizi, M. Horowitz, R. Ho, and R. Ananthraman, "Robust energy-efficient adder topologies," in *Proc. 18th IEEE Symp. Computing Arithmetic*, Jun. 2007, pp. 16–28.

[28] D. Patil, S. Yun, S. J. Kim, A. Cheung, M. Horowitz, and S. Boyd, "A new method for design of robust digital circuits," *Proc. ISQED*, pp. 676–681, Mar. 2005.

[29] V. G. Oklobdzija, B. R. Zeydel, H. Q. Dao, S. Mathew, and R. Krishnamurthy, "Energy-delay estimation techniques for high-performance microprocessor VLSI adders," in *16th IEEE Symp. Computing Arithmetic*, Jun. 2003, pp. 272–279.

[30] V. G. Oklobdzija, B. R. Zeydel, H. Q. Dao, S. Mathew, and R. Krishnamurthy, "Comparison of high-performance VLSI adders in energy-delay space," *IEEE Trans. VLSI Syst.*, vol. 13, no. 6, pp. 754–758, Jun. 2005.

[31] D. Marković, V. Stojanović, B. Nikolić, M. Horowitz, and R. W. Brodersen, "Methods for true energy performance optimization," *IEEE J. Solid-State Circuits*, vol. 39, no. 8, pp. 1282–1293, Aug. 2004.
[32] R. Zlatanovici, "Power–performance tradeoffs in datapaths," M.S. thesis, University of California, Berkeley, 2002.
[33] S. Wijeratne, "A 9 GHz 65 nm Intel Pentium 4 processor integer execution core," in *IEEE ISSCC. Dig. Tech Papers*, Feb. 2006, pp. 353–365.

Radu Zlatanovici (S'00–M'06) received the B.S. and M.S. degrees in electronics from Politehnica University Bucharest, Romania, in 1999 and 2000 respectively, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from University of California at Berkeley in 2002 and 2006, respectively.

He was on the faculty of Politehnica University Bucharest from 1999 to 2000. In 2002 and 2003, he interned at IBM T. J. Watson Research Center, Yorktown Heights, NY, working on power–performance tradeoffs for pipelined digital circuits. Upon graduation in 2006, he joined Cadence Research Laboratories, Berkeley, CA, as a Research Scientist. His research interests include low-power techniques for digital circuits, arithmetic circuits, and clock distribution.

Sean Kao received the B.Sc. degree in engineering from Harvey Mudd College, Claremont, CA, in 2002, and the M.Sc. degree from the University of California at Berkeley in 2004, where his research interests included high-performance digitial circuits, arithmetic archietectures, and energy–delay tradeoff analysis.

He worked as a research engineer at Xilinx from 2004 to 2006, where he was an inventor on over a dozen issued and pending U.S. patents. He is currently with Newport Media Inc., where he leads designs on mobile digital television baseband receivers.

Borivoje Nikolić (S'93–M'99–SM'05) received the Dipl.Ing. and M.Sc. degrees in electrical engineering from the University of Belgrade, Serbia, in 1992 and 1994, respectively, and the Ph.D. degree from the University of California at Davis in 1999.

He lectured electronics courses at the University of Belgrade from 1992 to 1996. He spent two years with Silicon Systems, Inc., Texas Instruments Storage Products Group, San Jose, CA, working on disk-drive signal processing electronics. In 1999, he joined the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, where he is now a Professor. His research activities include digital and analog integrated circuit design and VLSI implementation of communications and signal processing algorithms. He is co-author of the book *Digital Integrated Circuits: A Design Perspective*, 2nd ed. (Prentice-Hall, 2003).

Dr. Nikolić received the NSF CAREER award in 2003, College of Engineering Best Doctoral Dissertation Prize and Anil K. Jain Prize for the Best Doctoral Dissertation in Electrical and Computer Engineering at University of California at Davis in 1999, as well as the City of Belgrade Award for the Best Diploma Thesis in 1992. For work with his students and colleagues, he received the Best Paper Award at the ACM/IEEE International Symposium of Low-Power Electronics in 2005, and the 2004 Jack Kilby Award for the Outstanding Student Paper at the IEEE International Solid-State Circuits Conference.