Logical Effort of Carry Propagate Adders

David Harris and Ivan Sutherland
Harvey Mudd College / Sun Microsystems Laboratories
301 E. Twelfth St. Claremont, CA 91711
David_Harris@hmc.edu / Ivan.Sutherland@sun.com

Abstract - A wide assortment of carry propagate adders offer varying area-delay tradeoffs. Wiring and choice of circuit family also affect the size and performance. This paper uses the method of Logical Effort to characterize the effects of architecture, circuit family, and wire capacitance on adder delay. Domino logic offers about a 30% speedup on most valency-2 adders. Although Kogge-Stone adders are fastest in the absence of wire, other architectures such as variants on the Sklansky adder offer regular layouts and better delay in the presence of wiring capacitance.

I. Introduction

Fast adders are widely used in CMOS circuit design. The literature describes many adders including ripple carry, carry lookahead, carry select [2], carry skip [12], carry increment [16, 18], Sklansky (conditional sum) [14], Brent-Kung [3], Kogge-Stone [10], Ladner-Fischer [11], Han-Carlson [7], and Knowles [9]. Each architecture offers different tradeoffs between delay, area, and wiring complexity. Analytical delay models help designers evaluate these tradeoffs, but simply counting logic levels is inadequate because circuit delay also depends on fanout and wire capacitance.

Huang and Ercegovac [8] used an RC delay model to evaluate the effect of architecture and wiring capacitance on the Sklansky, Kogge-Stone, and Knowles adder architectures. The method of Logical Effort [15] builds on the RC delay model to offer a convenient shorthand for understanding the effects of fanout and gate sizing on delay. Dao and Oklobdzija [5, 6] applied this method to a few adders and concluded that logical effort predicted absolute delays within 5-20% of HSPICE.

This paper applies logical effort to understand the delay of eight different adder architectures that can be expressed as prefix computations according to the notation of [17]. The results show how adder delay depends on the number of inputs, the adder architectures, the cost of interconnect, and the circuit style. The model shows that most adder architectures can use uniform gate sizes to achieve regular layout with negligible performance loss. An exception is the Sklansky architecture that has highly irregular fanouts. This leads to a proposal for "helper" gates to construct very fast adders with regular layouts and low wiring cost.

II. Logical Effort of Circuit Building Blocks

The three basic building blocks for an adder are the bitwise Propagate/Generate (PG) cells, the group PG cells, and the sum XORs. High performance datapath adders often build these cells from domino gates while static CMOS is preferable when design simplicity and power consumption take precedence over utmost performance.

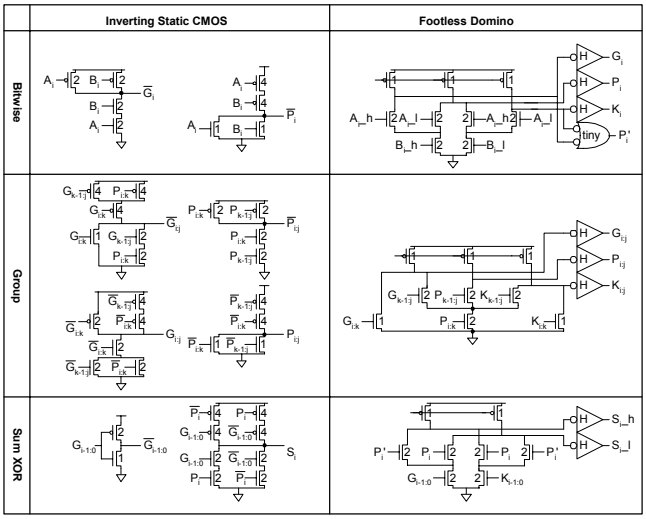
Static CMOS bitwise gates will compute generate as $G_i = A_i \bullet B_i$ and propagate as $P_i = A_i + B_i$. The sum is computed as $S_i = (A_i \quad B_i) \quad G_{i\ 1:0}$. Domino designs require monotonic inputs to the sum XOR. This is best done by calculating bitwise and group kill signals (K) and using XOR for propagate so that P, G, and K are 1-of-3 hot.

Define the group PG cell input coming from bits i:k as the upper input and that from k-1:j as the lower input. There are two types of group PG cells. Following the notation of [4], we call the cells black cells and gray cells. Black cells compute both $G_{i:j}$ and $P_{i:j}$ as defined in EQ (3). Gray cells compute only $G_{i:j}$. Black cells are required when the cell output drives the upper input of another group PG cell. The simpler gray cell may be used when the output drives only lower inputs or sum logic.

Consider four circuit styles: noninverting static CMOS, inverting static CMOS, footless domino, and footed domino. Fig 1 shows the basic cell designs. Inverting static CMOS gates consist of a single stage of logic for each cell (except that the final XOR requires an input inverter). Alternating stages use alternating polarities of inputs and outputs. Black cells contain both the group G and P gates while gray cells have only the G gate. Noninverting static CMOS gates add an output inverter to the bitwise and group static gates. Therefore, only the AND-OR and AND functions are required for group G and P, respectively. Footless domino gates computing 1-of-3 hot P, G, and K signals are shown in the second column. Each consists of a dynamic gate followed by an HI-skew inverter. Keepers and secondary precharge transistors are not shown. The group logic is shown for a black cell; a gray cell omits the P output. In the domino design, $K_{i\ 1:0} = \overline{G_{i\ 1:0}}$ so monotonic true and complementary versions of the carry signals are available at each final XOR. Footed domino gates are identical except for an extra series clocked evaluation transistor and greater transistor widths to compensate.

Transistors are annotated with widths measured in arbitrary units so that each pulldown stack has unit effective resistance. Table 1 lists the logical effort and parasitic delay of each cell input for each circuit family. The logical effort LE is the ratio of the input capacitance of the gate input to the input capacitance (3 units) of an inverter with the same unit effective resistance. The parasitic delay PD is estimated by counting the total transistor width on the output node,

Fig 1 Adder circuit building blocks



assuming diffusion and gate capacitance are approximately equal. In domino and noninverting static CMOS circuits, the output inverter also contributes parasitic delay. *LE* and *PD* are used in place of the usual symbols *g* and *p* to avoid confusion with *generate* and *propagate*.

Notice that the black cell has four inputs: $G_{i:k}$, $G_{k-1:j}$, $P_{i:k}$, and $P_{k-1:j}$. These are denoted as the upper and lower generate and propagate signals, *gu*, *gl*, *pu*, and *pl*, respectively, and each has a different logical effort. For inverting static CMOS circuits, the logical effort and parasitic delay are the average of the two polarities[1].

Some paths through the static XOR gate involve only a single A22OI stage while others also involve the inverter. A conservative estimate calculates the logical effort for the single stage path based on the 9 units of input capacitance on the *G* input. The parasitic delay is largest for the two-stage path, consisting of 9/3 for the inverter to drive its own diffusion parasitics and gate capacitance of the second stage plus 12/3 for the diffusion parasitics on the second stage.

In certain cases, buffers reduce the capacitance presented by noncritical forks of the circuit. Assume these buffers have half the drive (twice the resistance) of an ordinary gate and hence half the input capacitance. For the purpose of branching, the buffers therefore contribute only half the capacitance of a gate with comparable logical effort.

[1] If all cell sizes are chosen to provide unit drive as will be done in Section 4, this gives the correct delay through the path. If some cell sizes are selected for minimum delay, the logical efforts should be the geometric means of the efforts of the two polarities. In this case, the average and geometric mean are nearly identical, so the distinction is unimportant.

III. Adder Architectures

Adders are distinguished by the arrangement of cells in the group PG logic. Fig 2 shows eight such architectures for *N*=16. The upper box contains the bitwise PG logic and the lower box contains the sum logic. In the middle, the prefix tree is built from black cells, gray cells, and white buffers. The vertical axis indicates logic level and the critical path is indicated with a heavy line. For example, the ripple carry adder in Fig 2a is slow for long additions because the critical path propagates through *N*-1 gray cells.

Table 1 Logical effort and parasitic delay of adder circuit blocks

Cell	Term	Noninverting CMOS	Inverting CMOS	Footed Domino	Footless Domino
Bitwise	LE_{bit}	9/3	9/3	6/3 * 5/6	4/3 * 5/6
	PD_{bit}	6/3 + 1	6/3	7/3 + 5/6	5/3 + 5/6
Black Cell	$LE_{blackgu}$	5/3	4.5 / 3	1.5/3 * 5/6	1/3 * 5/6
	$LE_{blackgl}$	6/3	6/3	3/3 * 5/6	2/3 * 5/6
	$LE_{blackpu}$	10/3	10.5 / 3	3/3 * 5/6	2/3 * 5/6
	$LE_{blackpl}$	4/3	4.5 / 3	3/3 * 5/6	2/3 * 5/6
	PD_{blackg}	7/3 + 1	7.5 / 3	6/3 + 5/6	4/3 + 5/6
	PD_{blackp}	6/3 + 1	6/3	4/3 + 5/6	3/3 + 5/6
Gray Cell	LE_{graygu}	5/3	4.5 / 3	1.5/3 * 5/6	1/3 * 5/6
	LE_{graygl}	6/3	6/3	3/3 * 5/6	2/3 * 5/6
	LE_{graypu}	6/3	6/3	3/3 * 5/6	2/3 * 5/6
	PD_{gray}	7/3 + 1	7.5/3	6/3 + 5/6	4/3 + 5/6
Buffer	LE_{buf}	1 * 1/2	1 * 1/2	2/3 * 5/6 * 1/2	1/3 * 5/6 * 1/2
Sum XOR	LE_{xor}	9/3	9/3	3/3 * 5/6	2/3 * 5/6
	PD_{xor}	9/3 + 12/3	9/3 + 12/3	7/3 + 5/6	5/3 + 5/6

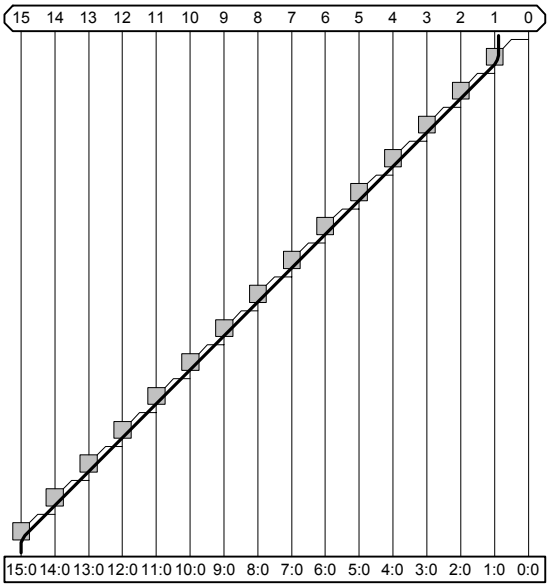
Several simplifying assumptions have been made:

- All inputs arrive at the same time with equal drive.
- Only horizontal wires are counted in the wire load. Vertical wires are assumed to be short enough to neglect (or lump into the parasitic gate delay).
- The A_i B_i term used to compute the final sum is not explicitly shown and may use buffered versions of the inputs to contribute negligible loading.
- Wires are assumed to be short enough that only capacitance must be considered, not wire RC delay. This assumption is supported by [8].

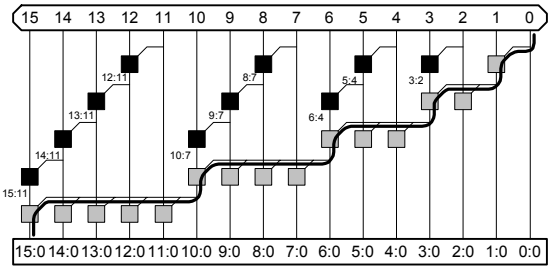
Note that in the Brent-Kung and Han-Carlson architectures there is never more than one black or gray cell per pair of bits in any given row. If pipelining is not required, the adder may be condensed to half the width, shortening the lateral wires as indicated in the table.

Fig 2 Adder architectures

(a) Ripple Carry



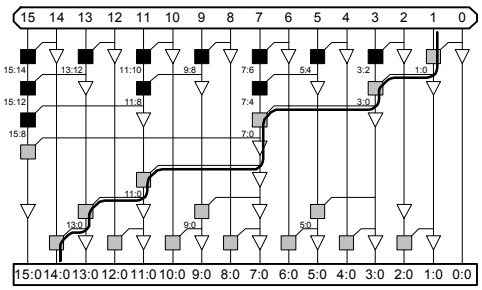
(b) Carry Increment



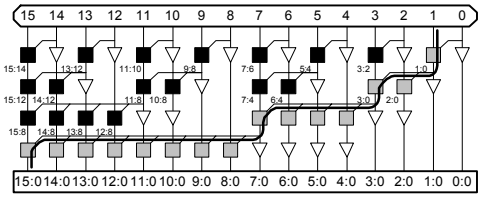
In most of the adder architectures, the stage effort is fairly constant throughout the adder if wire capacitance is neglected. We will see that this means uniform gate sizes may be used throughout with very little loss in performance. In the Sklansky graph, the fanout increase exponentially along the critical path. This leads to very poor performance unless cells have greater drive. One means to provide greater drive is to use larger gates in specific locations, but this increases the number of cells to design and verify and leads to irregular layout.

When trains must climb a steep grade with a heavy load, multiple locomotives are linked together. The extra locomotives are called *helpers*. In the Sklansky graph, multiple cells may be linked together to provide more current to drive the large fanouts and long wires. Four such adders with helpers are shown in Fig 3. Each is based on the Sklansky architecture. They differ in the number of columns required and the space available for buffers in pipelined adders.

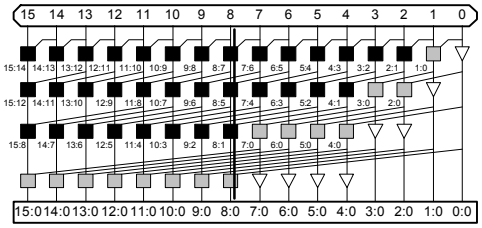
(c) Brent-Kung



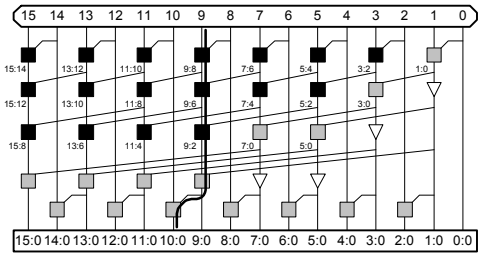
(d) Sklansky



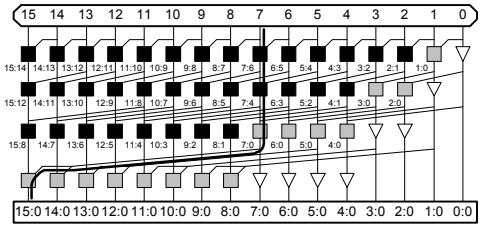
(e) Kogge-Stone



(f) Han-Carlson



(g) Knowles [2,1,1,1]



(h) Ladner-Fischer

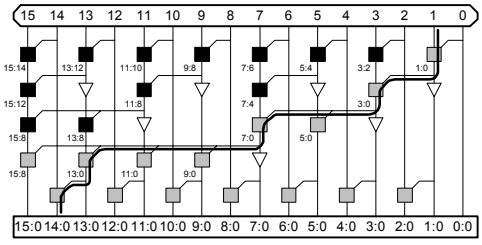
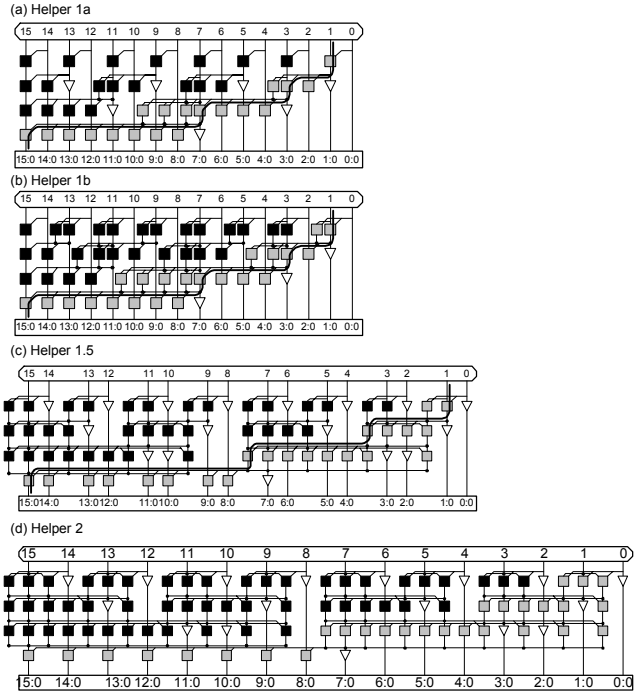


Table 2 Adder critical paths

	F	P	wire	repeats	notes
Ripple Carry	LE_{bit}	n/a	n/a		input -> bit 1
	LE_{graypu}	PD_{bit}	0		bit 1 -> gray pu
	$LE_{graygl}+LE_{xor}$	PD_{gray}	1	k = 1 … N-1	gray -> gray gl + xor
	n/a	PD_{xor}	n/a		xor -> output
Carry Increment	LE_{bit}	n/a	n/a		input -> bit 1
	$LE_{graygl}+LE_{xor}$	PD_{bit}	1		bit 1 -> gray gl and xor
	$k \bullet LE_{graygl}+LE_{xor}$	PD_{gray}	k	$k = 1 .. \sim \sqrt{2N}$	gray -> many gray gl and xor
	LE_{xor}	PD_{gray}	0		gray -> xor
	n/a	PD_{xor}	n/a		xor -> output
Brent-Kung	LE_{bit}	n/a	n/a		input -> bit 1
	LE_{graypu}	PD_{bit}	0		bit 1 -> gray pu
	$LE_{graygl}+LE_{buf}$	PD_{gray}	2^{k-1}	k = 1 … M - 1	gray -> gray gl + buf
	$LE_{graygl}+LE_{buf}$	PD_{buf}	2^{M-3}		buf -> gray gl + buf
	$LE_{graygl}+LE_{buf}$	PD_{gray}	2^{M+k-3}	k = 1 … M - 2	gray -> gray gl + buf
	LE_{xor}	PD_{gray}	0		gray -> xor
	n/a	PD_{xor}	n/a		xor -> output
Ladner-Fischer	LE_{bit}	n/a	n/a		input -> bit 1
	LE_{graypu}	PD_{bit}	0		bit 1 -> gray pu
	$LE_{graygl}+LE_{buf}$	PD_{gray}	2^{k-1}	k = 1 … M - 2	gray -> gray gl + buf
	$2LE_{graygl}+LE_{buf}$	PD_{gray}	2^{M-2}		gray -> 2 gray gl + buf
	$LE_{graygl}+LE_{buf}$	PD_{gray}	2^{M+k-3}	k = 1 … M - 2	gray -> gray gl + buf
	LE_{xor}	PD_{gray}	0		gray -> xor
	n/a	PD_{xor}	n/a		xor -> output
Sklansky	LE_{bit}	n/a	n/a		input -> bit 1
	LE_{graypu}	PD_{bit}	0		bit 1 -> gray pu
	$2^k \bullet LE_{graygl}+LE_{buf}$	PD_{gray}	2^k	k = 1 … M - 1	gray -> many gray gl + buf (-> 2^k)
	$LE_{xor}+LE_{graygl}$	PD_{gray}	1		gray -> xor + cout
	n/a	PD_{xor}	n/a		xor -> output
Kogge-Stone	LE_{bit}	n/a	n/a		input -> bit N/2
	$LE_{blackpl}+LE_{blackpu}$	PD_{bit}	1		bit N/2 -> black pu and pl
	$LE_{blackpl}+LE_{blackpu}$	PD_{blackp}	2^k	k = 1 … M - 2	black p -> black pu and pl
	LE_{graypu}	PD_{blackp}	2^{M-1}		black p -> gray pu
	LE_{xor}	PD_{gray}	0		gray -> xor
	n/a	PD_{xor}	n/a		xor -> output
Han-Carlson	LE_{bit}	n/a	n/a		input -> bit N/2
	$LE_{blackpl}+LE_{blackpu}$	PD_{bit}	1		bit N/2 -> black pu and pl
	$LE_{blackpl}+LE_{blackpu}$	PD_{blackp}	2^{k-1}	k = 1 … M - 2	black p -> black pu and pl
	LE_{graypu}	PD_{blackp}	2^{M-2}		black p -> gray pu
	LE_{xor}	PD_{gray}	0		gray -> xor
	n/a	PD_{xor}	n/a		xor -> output
Knowles [2,1,…,1]	LE_{bit}	n/a	n/a		input -> bit N/2
	$LE_{blackpl}+LE_{blackpu}$	PD_{bit}	1		bit N/2 -> black pu and pl
	$LE_{blackpl}+LE_{blackpu}$	PD_{blackp}	2^k	k = 1 … M - 3	black p -> black pu and pl
	$LE_{blackpl}+LE_{graypu}$	PD_{blackp}	2^{M-2}		black p -> gray pu and black pl
	$2LE_{graygl}+LE_{buf}$	PD_{gray}	2^{M-1}		gray -> 2gray gl and buf
	LE_{xor}	PD_{gray}	0		gray -> xor
	n/a	PD_{xor}	n/a		xor -> output

Fig 3 Helper adders



IV. Logical Effort Delay Model

The method of Logical Effort provides a simple method for determining a lower bound on critical path delay in circuits with negligible wire capacitance. If the path has M stages, a path effort of F, and a parasitic delay of PD, the delay (in) achieved with best transistor sizes is

$$D = D_F + PD = MF^{1/M} + PD \quad \textbf{(1)}$$

where D is measured in units of , the delay of an ideal inverter with no parasitic capacitance driving an identical inverter. Delay is often normalized to that of a fanout-of-4 inverter with the conversion 1 FO4 ˜ 5 . In a 180 nm process, 1 FO4 ˜ 60 ps. To illustrate the delay model, consider an N=4-bit ripple carry adder. According to the data from the previous sections the least delay is given below. Note that the inverting design is faster because the extra inverters in the noninverting CMOS version.

Inverting CMOS:

M $= N+1 = 5$

F $= (LE_{bit})(LE_{graypu})(LE_{graygl} + LE_{xor})^3$
$= (9/3)(6/3)(6/3 + 9/3)^3 = 750$

D_F $= 5(750)^{1/5} = 18.8$

PD $= PD_{bit} + 3PD_{gray} + PD_{xor}$
$= (6/3) + 3(7.5/3) + (9/3 + 12/3) = 16.5$

D $= 18.8 + 16.5 = 35.3 \; = 7.1$ FO4

Noninverting CMOS:

M $= 2(N+1) = 10$

F $= (LE_{bit})(LE_{graypu})(LE_{graygl} + LE_{xor})^3$
$= (9/3)(6/3)(6/3 + 9/3)^3 = 750$

D_F $= 10(750)^{1/10} = 19.4$

PD $= PD_{bit} + 3PD_{gray} + PD_{xor}$
$= (6/3 +1) + 3(7/3 + 1) + (9/3 + 12/3) = 20$

D $= 19.4 + 20 = 39.4 \; = 7.9$ FO4

In general, achieving least delay requires using different transistor sizes in each gate (although this delay model has assumed that all transistors in a branch scale uniformly). A regular layout with consistent transistor sizes in each type of cell is easier to build but may sacrifice performance. Consider designing all cells to have an arbitrary unit drive (i.e. output conductance). Define an inverter with unit drive to have unit input capacitance. For circuits with a single stage per cell (e.g. inverting static CMOS), the path effort delay is simply the sum of the effort delays of each stage:

$$D_F = \sum_{i=1}^{M} f_i \quad \textbf{(2)}$$

The total delay is still the sum of the path effort and parasitic delays. In a 4-bit ripple carry adder built from inverting static CMOS gates the delay is

Inverting CMOS:

D_F = LE_{bit} + LE_{graypu} + 3(LE_{graygl} + LE_{xor})
= 9/3 + 6/3 + 3(6/3 + 9/3) = 20
D = 20 + 16.5 = 36.5 = 7.3 FO4

In a circuit with two stages per cell (e.g. noninverting static CMOS or domino), let us design the first stage to have unit drive. Choose the size of the second stage for least delay. If the path has $C = M/2$ cells and the effort of the ith cell is F_i, the path effort delay is

$$D_F = \sum_{i=1}^{C} 2\sqrt{F_i} \quad \textbf{(3)}$$

In a 4-bit ripple carry adder built from noninverting static CMOS gates the delay is

Inverting CMOS:

$$D_F = 2\sqrt{LE_{bit}} + 2\sqrt{LE_{graypu}} + 3 \bullet 2\sqrt{LE_{graygl} + LE_{xor}}$$
$$= 2\sqrt{9/3} + 2\sqrt{6/3} + 3 \bullet 2\sqrt{6/3 + 9/3}$$
$$= 19.7\tau$$

D = 19.7 + 20 = 39.7 = 7.9 FO4

These delays are only slightly slower than ideal, justifying the use of a regular layout. The two-stage cell delay estimate is optimistic because in a regular design the second stage size will be fixed for each cell. However, the results from the single-stage cell estimate suggest the penalty is not large.

Horizontal wires add capacitance to the load of each stage. Let the wire capacitance be w units per column spanned. w depends on the width of each column, the width and spacing between wires, and the size of a unit transistor; in a trial layout in a 180 nm process, $w \sim 0.5$. While there is no closed-form solution for the minimum-delay problem with wire capacitance, the delay assuming fixed cell sizes is readily calculated by adding the wire capacitance to the stage effort f_i or F_i in EQ (2) or (3).

V. Results

The adder delays were evaluated using a MATLAB script. Table 3 lists delay (in FO4 inverter delays) for various adder architectures and widths assuming no wire capacitance and inverting static CMOS cells. It compares the delay achieved using best transistor sizes with the delay using uniform cell sizes. Observe that the penalty for uniform cell sizes is small in all cases except carry increment and Sklansky (where the fanouts vary wildly from one stage to another). This justifies using uniform cell sizes for most adders and for employing helpers on the Sklansky architecture to drive the high fanouts.

The remaining results are based on uniform cell sizes. Table 4 evaluates the effect of adder size by listing the delay of inverting static CMOS and footed domino adders assuming wiring capacitance w=0.5. Table 5 evaluates the impact of effect of circuit family, again assuming w=0.5. Table 6 evaluates the impact of wire capacitance on inverting static CMOS adders.

The Kogge-Stone, Han-Carlson, and Knowles adders require a large number of parallel wiring tracks for wide adders. This generally entails packing the wires close together, increasing the coupling capacitance on each wire. Huang and Ercegovac [8] found this nearly doubles the wire capacitance; therefore these architectures may be evaluated using the w=1.0 column of Table 6 compared against the w=0.5 column for adders with fewer wires.

The critical paths of most architectures (excluding Kogge-Stone, Han-Carlson, and Knowles) pass through a series of gray cell lower generate inputs. These adders may be sped up with asymmetric gray cells that reduce the logical effort LE_{graygl} at the expense of the other inputs [15]. This provides on average 9% speedup on the footed domino circuits, but almost none on the static CMOS circuits where noncritical transistors must be enlarged to preserve unit drive and thus increase parasitic delay.

VI. Conclusions

The logical effort model facilitates rapid comparison of a wide variety of adder architectures using multiple circuit families while accounting for the costs of fanout and interconnect.

The Sklansky architecture is slowed by its high fanout along the critical path. This may be addressed at the expense of regularity by using larger gates along the path. The helper architectures proposed in this paper gang together multiple cells to drive the high fanout nodes while maintaining regularity. Regular designs with unit drive work well in architectures with relatively constant stage efforts, i.e. all except Sklansky and carry increment.

In the absence of wiring capacitance, the Kogge-Stone adder is fastest because of its low number of stages and low fanout. When interconnect is considered, the Han-Carlson and helper adders become most attractive. Han-Carlson requires only half the number of columns, while helper adders are slightly faster at driving the long wires, especially when coupling capacitance is considered.

Fast static CMOS adders have a delay of about 10, 12, 14.5, and 17 FO4 for 16, 32, 64, and 128-bit widths, respectively. Most adders have a relatively low stage effort so the footed domino designs are only about 30% faster than the inverting static CMOS architectures because the high drive capability of domino is not fully exploited. This supports the use of higher-valency [1] domino designs. Asymmetric domino gates achieve another 9% speedup. Inverting static CMOS gates are also slightly faster than their noninverting counterparts except where high fanout capability is needed; however, the difference is much smaller than a method of "counting logic levels" would predict.

The delays estimated from logical effort are in good agreement with the HSPICE results of [1], [5], and [9]. However, the best 64-bit footless domino adder delays of 9-10

FO4 are still distinctly longer than the 7 FO4 delays achieved by the Naffziger domino Ling adder [13]. The differences may be attributed to the fact that velocity saturation makes tall domino gates slightly faster than simple logical effort models predict, the use of valency-4 cells and asymmetric gates biased to favor the critical path, and the logic level saved with the Ling algorithm. The fraction of the delay attributed to wires is important but significantly less than in [8] because this study assumed layouts with larger input transistors and a narrower column pitch to reduce the impact of wire capacitance.

REFERENCES

1 A. Beaumont-Smith and C. Lim, "Parallel prefix adder design," *Proc. 15th IEEE Symp. Comp. Arith*, pp. 218-225, June 2001.

2 O. Bedrij, "Carry-select adder," *IRE Trans. Electronic Computers*, vol. EC-11, June 1962, pp. 340-346.

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

4 N. Burgess, "Accelerated carry-skip adders with low hardware cost," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, vol. 1, pp. 852-856, 2001.

5 H. Dao and V. Oklobdzija, "Application of logical effort on delay analysis of 64-bit static carry-lookahead adder," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, vol. 2, pp. 1322-1324, 2001.

6 H. Dao and V. Oklobdzija, "Application of logical effort techniques for speed optimization and analysis of representative adders," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, vol. 2, pp. 1666-1669, 2001.

7 T. Han and D. Carlson, "Fast area-efficient VLSI adders," *Proc. 8th Symp. Comp. Arith.*, pp. 49-56, Sept. 1987.

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

9 S. Knowles, "A family of adders," *Proc 14th IEEE Symp. Comp. Arch.,* 1999 reprinted with corrections in *Proc. 15th IEEE Symp. Comp. Arith.*, pp. 277-281, June 2001.

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

11 R. Ladner and M. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831-838, Oct. 1980.

12 M. Lehman and N. Burla, "Skip techniques for high-speed carry propagation in binary arithmetic units," *IRE Trans. Electron Computers*, EC-10, Dec. 1962, pp. 691-698.

13 S. Naffziger, "A subnanosecond 0.5 m 64b adder design," *Intl. Solid-state Circuits Conf*., 1996, pp. 362-363.

14 J. Sklansky, "Conditional-sum addition logic," *IRE Trans. Electronic Computing*, vol. EC-9, June 1960, pp. 226-231.

15 I. Sutherland, R. Sproull, and D. Harris, *Logical Effort*, San Francisco: Morgan Kaufmann Publishers, 1999.

16 A. Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Trans. Computers*, vol. 42, no. 10, pp. 1162-1170, Oct. 1993.

17 N. Weste and D. Harris, *CMOS VLSI Design*, Addison-Wesley, 2004.

18 R. Zimmermann, "Non-heuristic optimization and synthesis of parallel-prefix adders," *Proc. Intl. Workshop on Logic and Architecture Synthesis*, pp. 123-132, Grenoble, France, Dec. 1996.

Table 3 Adder delays: w=0; inverting static CMOS

	Minimum Delay				Uniform Cell Size Delay			
	N=16	N=32	N=64	N=128	N=16	N=32	N=64	N=128
Ripple	24.7	48.7	96.7	192.7	25.1	49.1	97.1	193.1
Increment	13.0	21.7	34.5	59.5	14.3	24.4	40.5	71.6
B-K	9.4	11.4	13.4	15.4	9.4	11.4	13.4	15.4
L-F	9.0	11.0	13.0	14.0	9.1	11.1	13.1	15.1
Sklansky	9.8	13.5	18.9	26.7	11.5	18.5	31.9	58.1
K-S	7.6	9.0	10.4	11.8	7.9	9.3	10.7	12.1
H-C	8.8	10.2	11.6	13.0	9.1	10.5	11.9	13.3
Knowles	8.1	9.5	10.8	12.2	8.2	9.6	11.0	12.4
Helper 1a	8.7	10.5	12.5	14.4	9.2	11.2	13.2	15.3
Helper 1b	8.5	10.3	12.2	14.1	8.6	10.5	12.4	14.5
Helper 1.5	8.5	10.3	12.2	14.1	8.6	10.5	12.4	14.5
Helper 2	8.2	9.8	11.4	13.1	8.6	10.2	11.8	13.5

Table 4 Adder delays: w=0.5; uniform cell size

	Inverting Static CMOS				Footed Domino			
	N=16	N=32	N=64	N=128	N=16	N=32	N=64	N=128
Ripple	26.6	52.2	103.4	205.8	19.9	38.8	76.7	152.4
Increment	15.7	27.5	46.8	84.3	10.0	16.2	24.9	40.9
B-K	10.4	13.7	18.1	24.9	9.9	13.0	17.4	24.2
L-F	9.9	12.9	16.9	22.9	9.0	12.0	16.0	21.9
Sklansky	13.0	21.6	38.2	70.8	8.8	12.4	18.3	28.2
K-S	9.4	12.4	17.0	24.8	7.4	10.0	14.1	21.5
H-C	9.9	12.1	15.1	19.7	7.7	9.4	12.0	16.1
Knowles	9.7	12.7	17.3	25.1	7.8	10.3	14.5	21.8
Helper 1a	10.1	12.5	15.0	17.4	7.8	9.4	11.1	12.8
Helper 1b	9.4	11.6	14.0	16.4	7.5	9.1	10.7	12.4
Helper 1.5	9.7	12.0	14.6	17.2	7.8	9.5	11.3	13.2
Helper 2	9.7	11.7	13.7	15.7	7.8	9.3	10.9	12.5

Table 5 Adder delays: w=0.5; uniform cell size
(1) Inverting CMOS, (2) Noninverting CMOS,
(3) Footed Domino, (4) Footless Domino

	N=32				N=64			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
Ripple	52.2	54.6	38.8	31.3	103.4	107.7	76.7	61.9
Increment	27.5	22.1	16.2	13.5	46.8	33.3	24.9	21.1
B-K	13.7	16.8	13.0	10.7	18.1	21.8	17.4	14.6
L-F	12.9	15.6	12.0	9.8	16.9	20.2	16.0	13.3
Sklansky	21.6	16.3	12.4	10.5	38.2	23.4	18.3	15.9
K-S	12.4	13.4	10.0	8.7	17.0	18.0	14.1	12.7
H-C	12.1	13.3	9.4	7.9	15.1	16.4	12.0	10.3
Knowles	12.7	13.6	10.3	8.9	17.3	18.3	14.5	12.9
Helper 1a	12.5	12.6	9.4	7.7	15.0	14.8	11.1	9.2
Helper 1b	11.6	12.2	9.1	7.5	14.0	14.4	10.7	8.8
Helper 1.5	12.0	12.6	9.5	7.9	14.6	14.9	11.3	9.4
Helper 2	11.7	12.4	9.3	7.7	13.7	14.4	10.9	9.0

Table 6 Adder delays: inverting static CMOS; uniform cell size

	N=64				N = 32			
	w = 1/4	w = 1/2	w = 3/4	w = 1	w = 1/4	w = 1/2	w = 3/4	w = 1
Ripple	50.6	52.2	53.8	55.3	100.3	103.4	106.6	109.7
Increment	25.9	27.5	29.1	30.6	43.6	46.8	50.0	53.1
B-K	12.5	13.7	14.8	15.9	15.7	18.1	20.4	22.7
L-F	12.0	12.9	13.9	14.8	15.0	16.9	18.9	20.8
Sklansky	20.1	21.6	23.1	24.7	35.0	38.2	41.4	44.5
K-S	10.9	12.4	13.9	15.5	13.9	17.0	20.1	23.3
H-C	11.3	12.1	12.9	13.7	13.5	15.1	16.7	18.3
Knowles	11.2	12.7	14.3	15.8	14.2	17.3	20.4	23.6
Helper 1a	11.9	12.5	13.2	13.8	14.1	15.0	15.8	16.7
Helper 1b	11.1	11.6	12.2	12.8	13.2	14.0	14.7	15.5
Helper 1.5	11.3	12.0	12.8	13.6	13.5	14.6	15.6	16.7
Helper 2	10.9	11.7	12.4	13.1	12.7	13.7	14.6	15.5