Minimizing Energy by Achieving Optimal Sparseness in Parallel Adders

Mustafa Aktan
Tera-Hz Mikroelektronik Ltd.
Ankara, Turkey
e-mail: mustafa.aktan@tera-micro.com

Dursun Baran
TUBITAK
Kocaeli, Turkey
e-mail: dursun.baran@tubitak.gov.tr

Vojin G. Oklobdzija
University of California
Davis, USA
e-mail: vojin@acsel-lab.com

***Abstract*— Carry tree sparseness is used in high-performance binary adders to achieve better energy-delay trade-off. To determine the energy optimal degree of sparseness, a detailed analysis is performed in this work. An analytical expression for the upper bound of sparseness is derived. The effect of increased sparseness on partial sum block and total energy is explored on 32-, 64-, 128-, and 256-bit adders. Higher degrees of sparseness in the carry generation block is achieved by employing parallel adders in the sum block instead of serial ripple carry adders. 64-bit adders with various sparseness degrees using leading addition algorithms are synthesized and optimized with a standard cell library in 45nm CMOS technology. Post layout simulations revealed that the optimal sparse carry tree adders provide up to 50% and 22% improvement in energy at same performance over full carry tree Kogge-Stone and Ladner-Fischer adder designs, respectively.**

Keywords- Arithmetic and Logic Structures, Binary Adders, Low-Power Design, Sparseness, VLSI.

I. INTRODUCTION

Binary adders are the core of arithmetic operations in high-performance microprocessors [12]. In literature, various implementations have been proposed to speed up binary addition operation. They introduce different trade-offs in terms of algorithm, logic depth, wiring complexity and the maximum fan-out of the design [1-8][24].

The adder consists of a carry block, sum block and sum selection block. Sum block generates conditional sum signals pre-computed for the two cases where the carry from the previous digit (or group of digits) is '0' and '1'. The sum selection block selects the actual sum from the pre-computed values depending on the value of the carry signals. The carry block can generate all carry signals (full tree) or some of the signals (sparse tree). An example of *N*-bit adder structure with a sparse *M* carry tree (only the carry signals with an index of integer multiples of *M* are generated) is shown in Fig. 1.

Sparse carry trees have been shown to reduce energy in parallel adders [14][16][18]. Energy improvement is due to the complexity reduction of the carry path by reduced wiring and number of gates. On the other hand, a certain amount of complexity is moved to the sum path incurring a limit on the sparseness of the carry tree. Benefits of using a sparse carry tree become more apparent for dynamic designs where the carry generation block is implemented in dynamic logic (high performance) and sum path in static CMOS (low power). Dynamic logic suffers from high switching activity penalty. By moving the complexity from carry path to sum path, which has less switching activity due to static CMOS

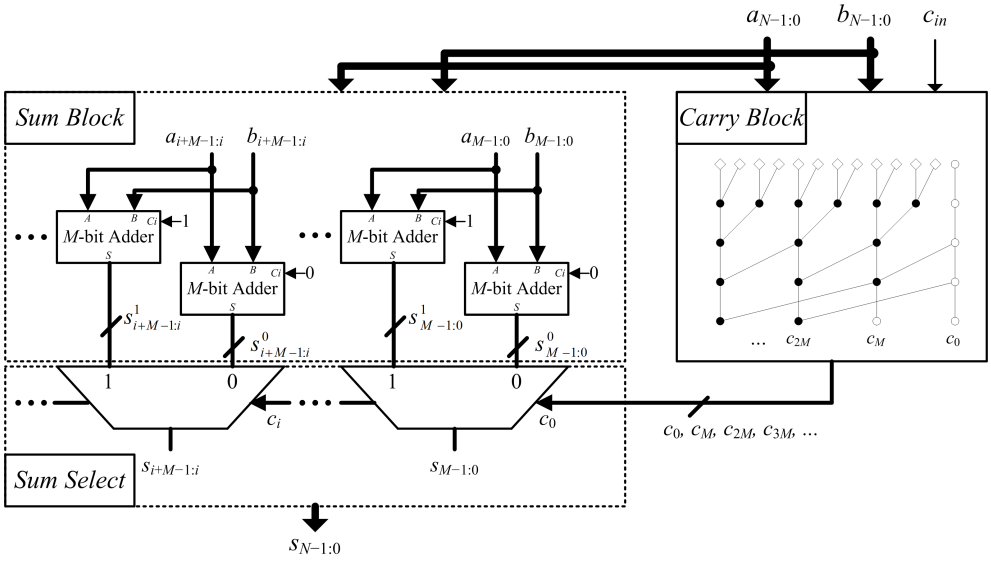


Figure 1. The structure of an *N*-bit adder with a sparse *M* carry tree.

1063-6889/15 $31.00 © 2015 IEEE
DOI 10.1109/ARITH.2015.13

implementation, a considerable reduction in energy is achieved [12].

There is a trade-off between carry tree sparseness and sum block complexity [16]. Hence, a limit on the energy optimal degree of sparseness for the carry tree is expected. In this work, we investigate the energy optimal sparseness degree in static CMOS adders. Static CMOS is known for its robustness compared to dynamic CMOS and is going to be the prevailing circuit implementation technique as the technology scales down. The energy-delay space of parallel adders with various sparseness degrees are explored using a 45nm standard cell library in synthesis-layout optimization flow. In Section II structural features of adder topologies are classified. Section III gives a brief survey of sparse adders appeared in the literature. In Section IV we derive a theoretical upper bound for the energy optimal degree of sparseness in parallel adders. A hardware efficient sum path adder that enables to reach higher levels of sparseness is also introduced. Section V gives the results for two high performance prefix adder structures which are analyzed to determine the energy optimal sparseness. Section VI concludes the discussion.

II. Adder Circuit Topologies

Carry computation is a prefix problem [17][21]. Every output in a prefix problem depends on all inputs having an equal or lower index. For example the i'th carry (the carry signal generated at digit i) is a function of all inputs having index i or less ($c_i = f(a_i,b_i,a_{i-1},b_{i-1},\ldots,a_0,b_0,c_{in})$). There have been many prefix algorithms proposed in the literature for the efficient computation of the carry signals [6-8, 20-21].

Weinberger introduced generate and propagate signals to parallelize the carry propagation for fast operation [1]. His generate-propagate based formulation inspired many adder architectures [6-8][20]. Ling [2] presented a transformation to reduce the fan-in of adder to comply with fan-in limitation of IBM ECL technology. Instead of propagating the carry signals the less complicated pseudo carry signals are propagated. By using Ling's equations the first stage of the carry tree and bitwise generate and propagate signals can be merged into one stage by using gates with a stack height of 2 [13]. This reduces the number of stages in the adder hence translates into increased performance at same energy [13]. In [3] it is shown that the only worth signals to use in prefix computation are the ones introduced by Weinberger [1] or Ling [2]. Adders using generate/propagate signals are named here as Weinberger adder and those using Ling's signals (pseudo carry, transmit) are named as Ling adders.

Carry block of prefix adders are shown using prefix graphs. An example prefix graph based on Ling recurrence is shown in Fig. 2. Black circles (●) in the prefix graph represent group generate ($H_{i:k} = H_{i:j} + T_{i-1:j-1}H_{j-1:k}$) and group propagate ($T_{i-1:k-1} = T_{i-1:j-1}T_{j-1:k-1}$) operations. White circles (○) are buffers. Rhomboids (◊) refer to bitwise generate ($g_i = a_ib_i$) and propagate ($t_i = a_i + b_i$) operations. Black rhomboids (♦) refer to the merged bitwise operations ($H_{i:i-1} = a_ib_i + a_{i-1}b_{i-1}$) and ($T_{i-1:i-2} = (a_{i-1}+b_{i-1})(a_{i-2}+b_{i-2})$. The squares (□) refer to the sum select operation ($s_i = s_i^1h_{i-1} + s_i^0h_{i-1}$'). The

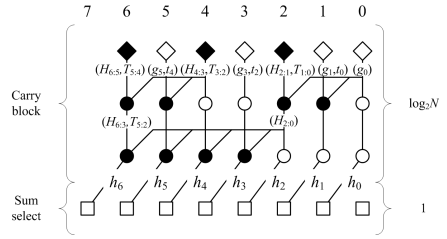


Figure 2. Prefix graph and operators.

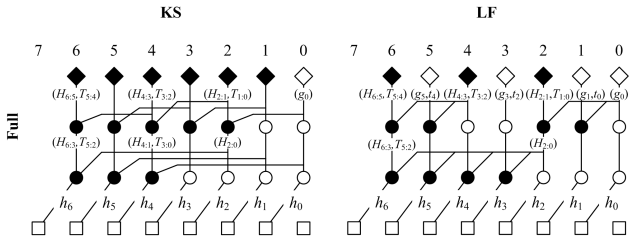


Figure 3. Prefix graph of 8-bit (a) Kogge-Stone and (b) Ladner-Fischer adders (full carry tree).

actual carry signals (c) can be calculated from the pseudo carry signals (h) as $c_i = t_{i-1}h_{i-1}$where $h_{i-1} = g_{i-1}+c_{i-1}$.

Prefix algorithms are classified by considering structural features as logic depth, prefix, fan-out, and wiring complexity. Logic depth is the maximum number of stages from input to output. Prefix is the maximum number of input signals (or maximum fan-in) of a prefix operator in the prefix tree. For example, prefix 2 means two signals are processed in a black node. Prefix is closely related to the logic depth of a prefix algorithm. It determines the lower bound for the number of stages in a carry tree which is $\log_P N$ for an N-bit adder with prefix P operators. For a 64 bit adder, the depth of the carry tree will be 6 for prefix 2 and 3 for prefix 4. Fan-out is the maximum number of logical branching in the prefix tree. Wiring complexity is the maximum number of wires used to connect signals between consecutive stages.

Minimum depth prefix structures yield better performance than structures with higher number of stages [16]. Kogge-Stone (KS) and Ladner-Fischer (LF) adders are the well-known minimum depth high performance prefix computation structures. KS adder keeps fan-out minimum in price of increased wiring complexity. On the other hand, in a LF adder fan-out increases exponentially while wiring complexity is kept at minimum. In [20], various minimum depth adder topologies are introduced that trade wiring complexity for fan-out and vice versa. In short, KS has the highest wiring complexity and minimum fan-out, and LF has the highest fan-out and minimum wiring complexity. The prefix graphs of KS and LF adders using Ling signals are given in Fig. 3.

III. Sparse Carry Tree

The carry tree generates carry signals in parallel for each bit to speed up addition. To overcome the computational burden of generating all carry signals, in a sparse carry block

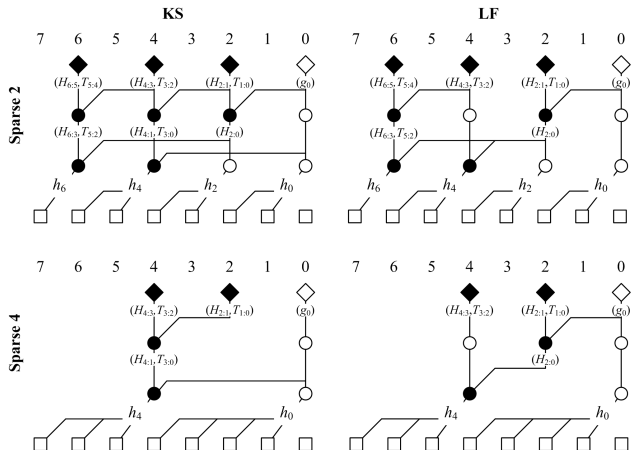


Figure 4. Prefix schemes of the carry tree for 8-bit Kogge-Stone (KS) and Ladner-Fischer (LF) adders (sparse-2 and sparse-4 tree).

only multiple radix carries are generated. The motivation of a sparse carry tree is to reduce the hardware overhead incurred by generating all carry signals. In order to achieve a sparse carry tree, addition operation is divided into sub-blocks where for each sub-block conditional sum signals are generated forcing the corresponding carry signal to be '0' and '1'. True carry signals for each sub-block are simultaneously generated in the carry block. True sum signals are selected from the conditionally generated sum signals depending on the value of true carry signals.

Adders with sparse carry trees are proposed in the literature with various degrees of sparseness [4-5],[9-12],[14],[22-23]. A sparse carry tree can be generated for any prefix adder. Here, we will focus on the fastest prefix adder structures, namely the Kogge-Stone and Ladner-Fischer which are minimum depth structures. Both adders have their advantages and disadvantages. We will show that by using a sparse carry tree the disadvantages can be eliminated to a certain extent and energy-delay performance can be improved.

Both LF and KS adders have their shortcomings. With the demand for higher length (bit-width) adders, the high fan-out of LF adder considerably hurts the adder performance. For example in a 64 bit adder fan-out reaches a number of 32 gates. On the other hand, with technology scaling down, wiring load has become close to fan-out load that adversely affects the KS adder. In a 64 bit adder, wires span through 32 bit slices (bitpitch). The high wiring density in KS adder and high fan-out in LF is solved by using the sparse computation technique without degrading their performance. The long wires are used to connect the intermediate generate and propagate signals in the carry block. In the sparse technique, only some of the carry signals are generated as shown in Fig. 4. The sum signals are generated using only those generated carry signals. The wiring cost is reduced to the amount of generated carry signals. The overhead is: increased loading on the generated carry signals and a more complex sum path. The complexity tradeoff between carry and sum blocks states an energy optimal degree for sparseness which will be discussed in the next section.

IV. Energy Optimal Degree of Sparseness

A. Theoratical Results

Here we will derive a coarse upper limit for the energy optimal degree of sparseness in parallel adders. It is based on critical path length balancing where the critical path length is measured as number of logic stages. As mentioned before, minimum depth parallel adders provide better performance [16] and the sum block logic depth must be equal or lower than the carry generation block to satisfy the same performance target when different degree of sparseness is applied. It is a rough equality to prevent any performance penalty that may occur because of increased sparseness.

Consider an N-bit adder with a sparse M carry tree. We limit the derivation for prefix 2 computations. The analysis can be extended to higher prefix computations as well. The critical path of a parallel adder is determined by the logical depth of carry block for which the minimum possible value for Ling recurrence is

$$1 + \log_2 N$$

The logical depth of sum path is determined by the depth of M-bit adders used in the sum block. The minimum possible depth for an M-bit prefix 2 adder is

$$2+\log_2 M$$

where 1 stage is the bitwise generate-propagate, $\log_2 M$ comes from the carry tree, and 1 stage is the final sum selection stage. By increasing sparseness we are actually increasing the depth of sum block which should not exceed the depth of carry block. So

$$2+\log_2 M \le 1 + \log_2 N \quad (1)$$

which reduces to

$$M \le N/2. \quad (2)$$

This is the theoretical upper bound for the energy optimal sparseness of an N bit prefix 2 Ling adder. In case of a merged carry tree the upper bound for sparseness will become $M \le N/4$.

B. Sum Path in Ling Adder

Ling's transformation reduces complexity in the carry tree at the expense of increased complexity in the sum generation block. Therefore, implementation of sum block differs in Ling adders. In general, sum in an adder is computed by

$$s_i = a_i \oplus b_i \oplus c_i$$

When using Ling recurrence the carry signal (c) needs to be generated from the pseudo-carry (h) signal as $c_i = t_{i-1}h_{i-1}$ where $t_i = a_i + b_i$. So

$$s_i = a_i \oplus b_i \oplus (t_{i-1}h_{i-1})$$

The sum is computed conditionally depending on the value of the pseudo carry (h_{i-1}) using a multiplexer:

$$s_i = \begin{cases} a_i \oplus b_i & \text{when } h_{i-1} = 0 \\ a_i \oplus b_i \oplus t_{i-1} & \text{when } h_{i-1} = 1 \end{cases}$$

For an M-bit block of sum signals, the computation of the sum conditionally depends on the value of the pseudo-carry (h_{i-1}):

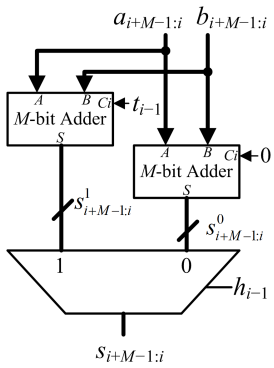


Figure 5. Conditional sum computation in a sparse M Ling adder using M-bit adders.

$$S_{i+M-1:i} = \begin{cases} S^{Ci=0}_{i+M-1:i} & \text{when } h_{i-1} = 0 \\ S^{Ci=t_{i-1}}_{i+M-1:i} & \text{when } h_{i-1} = 1 \end{cases}$$

For a sparse M adder, for each M-bit block two M-bit adders are used for the pre-computation of the sum signals $S^{Ci=0}_{i+M-1:i}$ and $S^{Ci=t_{i-1}}_{i+M-1:i}$ as shown in Fig. 5.

C. Implementation of Sum Block

Conditional computation of sum signals introduces hardware overhead due to the requirement of two adders to compute partial (or conditional) sum signals. Therefore, the simplest adder structure, the ripple carry adder, is preferred. After simplification of the initial stage for c_{in}= '0' and c_{in}= 't_{i-1}', an M-bit ripple carry adder has a logical depth of 1+M stages. An equivalent minimum depth prefix adder has 2+$\log_2 M$ stages. For M = 2, both ripple and prefix adders have the same depth, 3. Linear proportionality of the critical path length in a ripple carry adder limits it to be used for up to sparse 4 for a 64-bit prefix 2 adder. In order to reach higher sparseness degrees while preserving critical path length equality, a parallel prefix adder with shorter critical path needs to be used in the sum block. Table I shows the change in number of logic stages in critical path when ripple carry and minimum depth prefix adders are used in the sum block.

TABLE I. CRITICAL PATH LENGTH COMPARISON WITH INCREASING SPARSENESS FOR SUM BLOCKS IMPLEMENTED USING RIPPLE CARRY AND PARALLEL PREFIX ADDERS

Degree of Sparseness (M)	Ripple carry (1+M)	Parallel prefix (2+$\log_2 M$)
2	3	3
4	5	4
8	9	5
16	17	6

Parallel adders require more gates and wires as compared to serial adders and that increases their implementation cost considerably. In order to reduce the hardware overhead incurred by using prefix adders in the sum block, common signals generated within the two adders can be shared. Any gate that has no signal depending on the input carry signal in its fanin cone is shared among the two sum path adders. For a ripple carry adder only the gates that produce the bitwise generate/propagate signals and their corresponding buffers are shared. On the other hand, when sum path adders are implemented using parallel prefix adders, it is possible to share more gates because of the independent paths from inputs to outputs. Shared gates and signals are shown in shaded areas in Fig. 6 for an 8-bit sum path adder.

By using parallel adders in the sum path, it is possible to increase sparseness without losing performance and it provides a larger exploration area for the energy optimality in design space. The gates and wires are the major sources of switched capacitance, hence dynamic power, in the adder. With sharing common signals in the implementation of prefix adders we prevent excessive hardware overhead and increased power consumption so that reduced gate count in the carry tree is not offset by increased gate count in sum path. While sharing of common gates prevents excessive hardware overhead and increased power consumption to some extent, wiring complexity is increased in the sum block. Hence, to better understand the effect of increased sparseness on energy, we need to look at the change in both total gate complexity and total wire complexity of a sparse adder.

Although KS and LF adders have the same gate count,

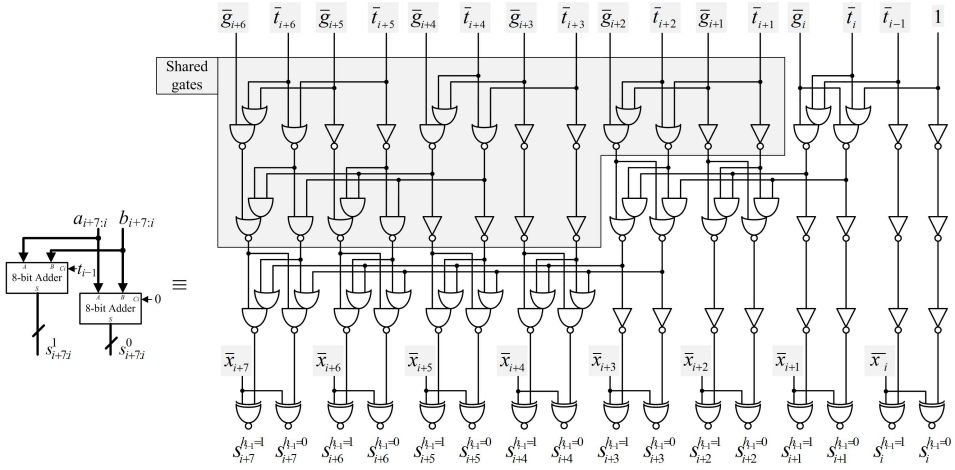
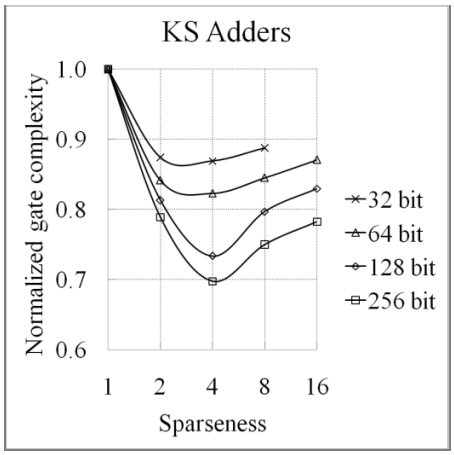
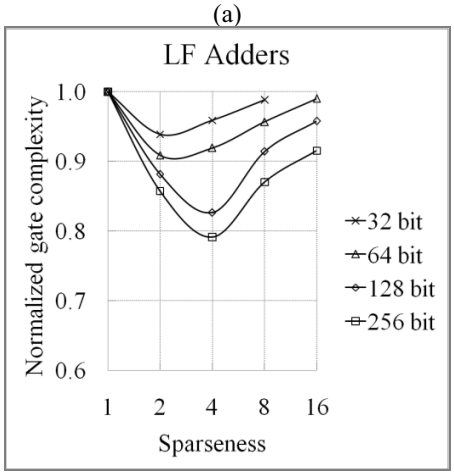


Figure 6. 8-bit conditional sum computation using parallel prefix adder.

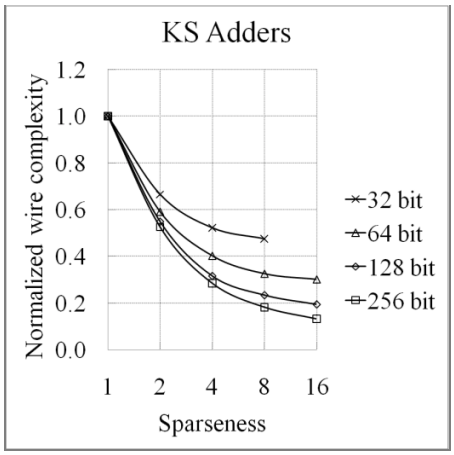


(a)

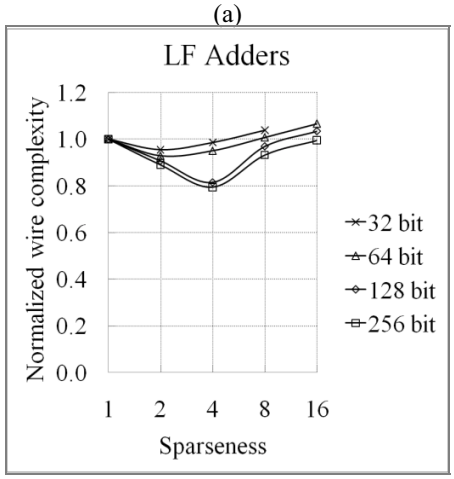


(b)

Figure 7. The effect of increased sparseness on gate complexity for Ling adders (a) KS and (b) LF.



(a)



(b)

Figure 8. The effect of increased sparseness on wire complexity for Ling adders (a) KS and (b) LF.

LF adders have more buffers than complex gates. Therefore, using total gate count for total gate complexity will be inadequate to measure the effect of changing hardware complexity with sparseness. As shown in [15], the dynamic energy consumption of a logic gate depends on the logic complexity, fan-out load and the switching activity rate. In parallel adders and most of digital circuits, output of a gate drives another logic gate or the output load though a wire. The total circuit capacitance (area) excluding the output loads and the input signal drives is calculated by summing all gate and wire capacitances (gate sizes). Also, static energy consumption is also directly related to total capacitance as well. For most of the static logic gates, the gate capacitance increases as the number of inputs are increased. Therefore, the number of inputs is used as a better measure for the complexity of a gate. Hence, the complexity of an inverter will be 1, a two input NAND (NOR) gate will be 2, a *K*-input gate will be *K*. The total gate complexity is calculated as the sum of individual gate complexities. This complexity measure neglects the effect of gate size (or strength) on complexity. Thus, the accuracy of the results are limited to small gate sizes, i.e. either low speed target designs where all gates tend to be unit size or circuits having a dominant critical path and all gates in side paths are minimum sized (like in LF adder). Nevertheless, it can still be used as a first degree analysis method.

Wire complexity is calculated as the sum of horizontal wires spanning from one bit position to another. This analysis is based on the assumption of a rectangular layout structure where the width (correlated to number of adder bits, i.e. 64) is much larger than the height (correlated to the number of levels, i.e. 7), hence the effect of vertical wiring is neglected. A gate at bit position 31 driving a gate at bit position 63 will have a connecting wire spanning through 32 bit positions, hence the wire length will be 32 units. The actual wire length will be determined by the width of a bit position in the layout (i.e. bitpitch). To account for wiring within a bitpitch a wire length of 0.5 units is assumed.

Fig. 7 and 8 show the change in total gate complexity (TGC) and total wire length (TWL) with increasing sparseness for 32-, 64-, 128-, and 256-bit KS and LF adders. Complexities are normalized with respect to full carry tree adders, i.e. M=1. The reference gate complexities for 64-bit KS and LF adders are 3447 and 3029, respectively. The

reference wire complexities for 64-bit KS and LF adders are 4567 and 1234, respectively. For 64-bit KS adders, gate complexity reaches a minimum for sparse 4 designs and increases at a slower rate through sparse 16, whereas wire complexity continues reducing till sparse 16. For 64-bit LF adders, sparse 2 yields both minimum gate complexity and total wire length though there is little difference with sparse 4. It must be noted that the reduction in gate complexity in LF adder is due to removal of buffers as opposed to the more complex AND-OR gates in KS adder. Hence, the improvement in gate complexity for LF adder is smaller compared to the improvement in KS adder. The increase in gate complexity beyond sparse 8 in KS adder will circumvent energy savings achieved through reduced wiring complexity. Furthermore, energy optimum sparseness degree will be determined by the gate capacitance to the wire capacitance ratio. For low performance design region, gate sizes are small hence wire capacitances will dominate and KS sparse 8 is expected to outperform KS sparse 4 in terms of energy at same performance. For LF adder on the other hand, it is not worth going beyond sparse 4 due to increased complexity in both measures. For 128- and 256-bit adders sparse 4 yields the most savings for both KS and LF structures.

V. RESULTS

To prove the theoretical foundations stated in the previous section we have analyzed sparseness degrees of 1, 2, 4, and 8 on 64-bit Ling adders using Kogge-Stone (KS) and Ladner-Fischer (LF) prefix schemes. A standard cell library is prepared in Cadence Virtuoso using TSMC 45nm CMOS technology . The smallest gate has a width of 0.8µm and the largest gate has a width of 3.6µm. The gates in the library are given in Table II.

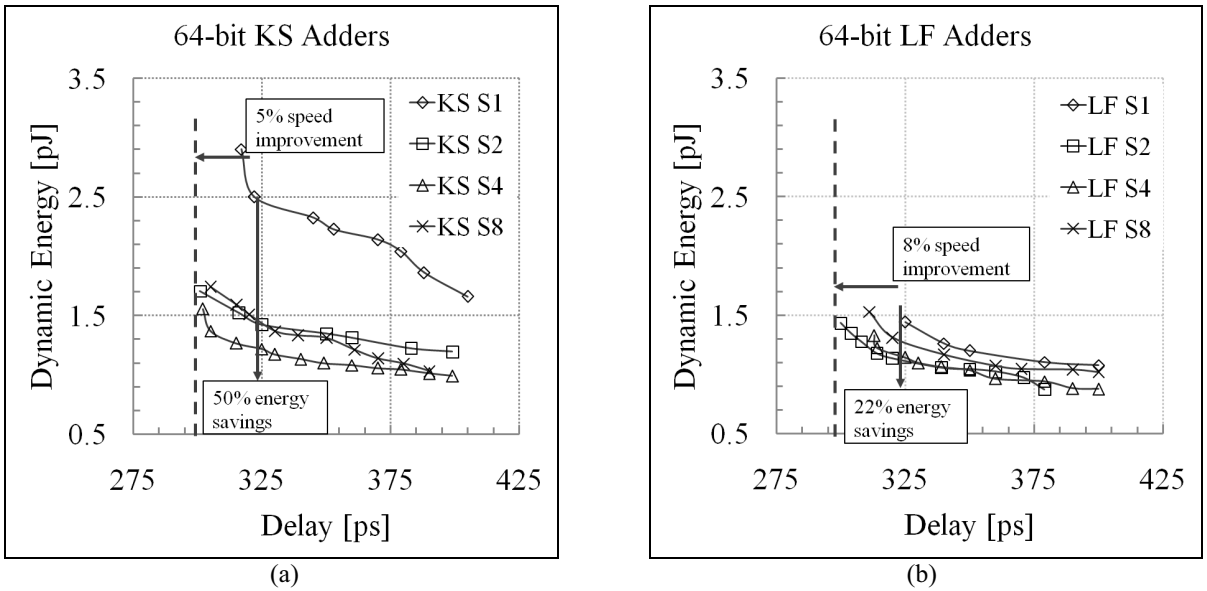
All gates in Table II are designed for 3 threshold levels (low, standard, and high) allowing leakage power optimization. Total number of gates in the library is 114. Each gate is characterized for energy, leakage power, and delay (worst case single input switching conditions) using typical process corner parameters. The nominal supply voltage is 1.1V. The nominal operating temperature is 25°C.

TABLE II. STANDARD CELL LIBRARY

Gate	Available Strength
AOI21	1x,2x,4x,6x,8x
AOI22	1x,2x,4x,6x,8x
INVERTER	1x,2x,4x,6x,8x,12x,16x,32x
NAND2	1x,2x,4x,6x,8x
NOR2	1x,2x,4x,6x,8x
OAI21	1x,2x,4x,6x,8x
OAI22	1x,2x,4x,6x,8x

Synopsys Design Compiler is used to do the gate sizing optimization and Cadence Encounter to do automatic placement and routing. Post-layout energy and delay simulations (using extracted parasitics) are carried out using Synopsys Primetime. 25% switching activity is assumed at the inputs. Gate sizes are optimized for energy under a range of operating conditions (delay and input/output loading) for each adder. Primary inputs are assumed to be driven by 16x inverters. The output load is set to 5fF which is equivalent to the load introduced by terminating the outputs with a 16x inverter. Due to the limited number of available gate strengths buffering is allowed in the gate sizing optimization. Insertion of buffers is critical in the design of LF adder which suffers from high fanout loading.

The effect of increased sparseness on energy and delay for 64-bit Kogge-Stone and Ladner-Fischer Ling adders is shown in Fig. 9. For both KS and LF adders sparse 1, 2, 4, and 8 are synthesized for delay target ranging from 300ps to



(a) (b)

Figure 9. The effect of increased sparseness in the energy-delay space for 64-bit Ling adders (a) KS and (b) LF.

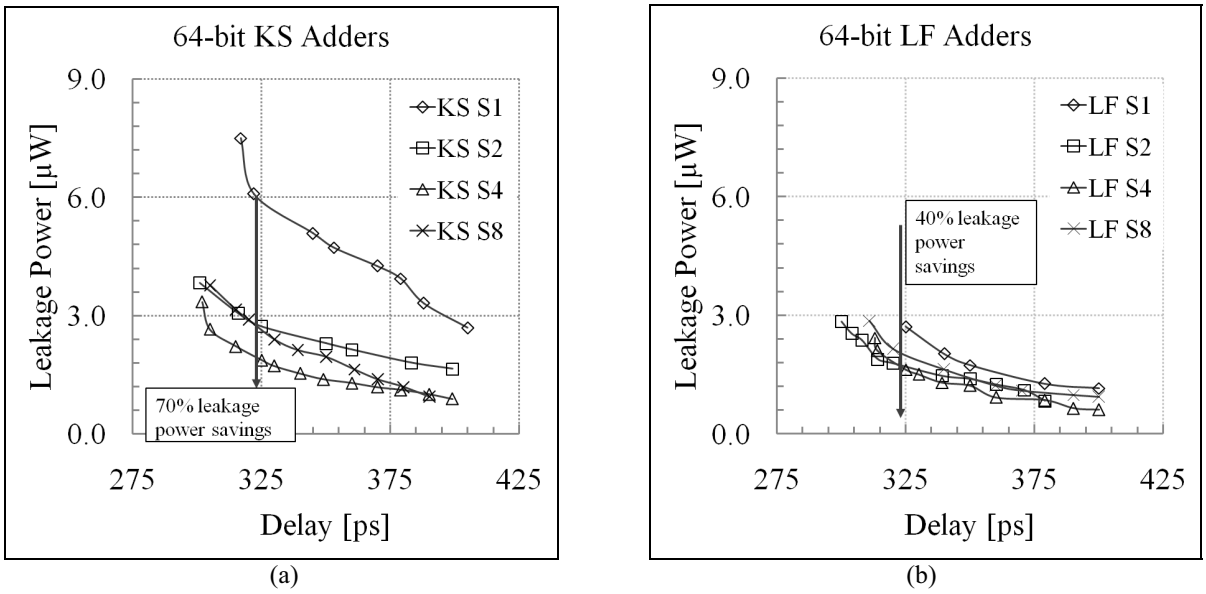


Figure 10. The effect of increased sparseness on leakage power for 64-bit Ling adders (a) KS and (b) LF.

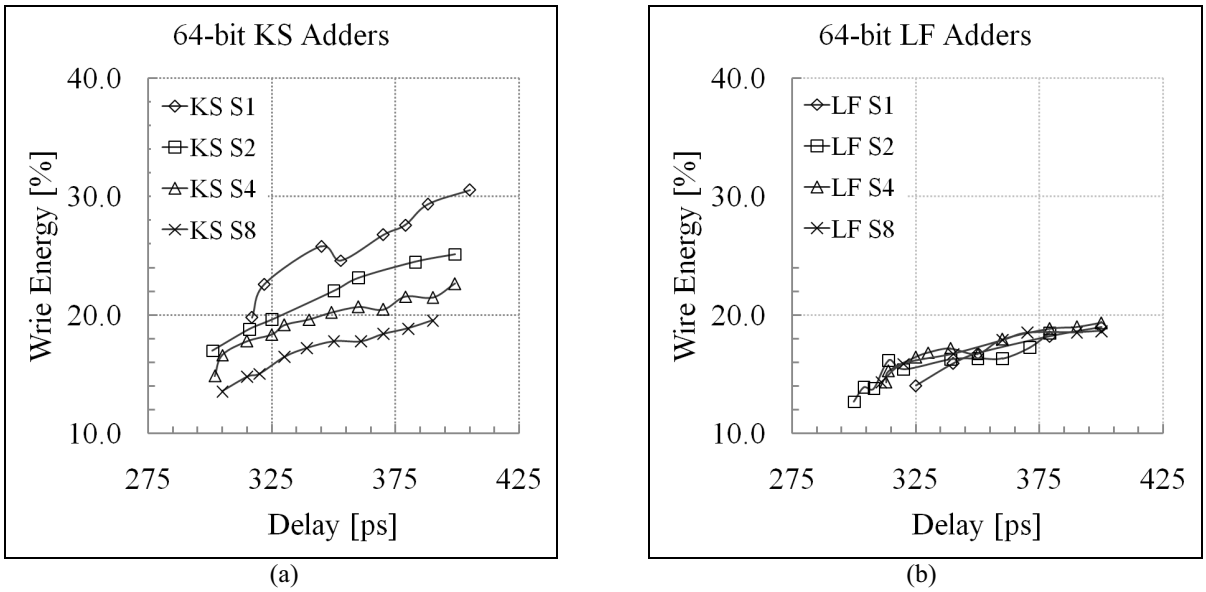


Figure 11. The effect of increased sparseness on wire energy contribution for 64-bit Ling adders (a) KS and (b) LF.

400ps. The resulting energy-delay space is shown in Fig. 9a and Fig. 9b for KS and LF adders, respectively.

For KS adder increased sparseness significantly improves energy without degrading performance (50%). Sparse 4 gives the best result. Going beyond sparse 4 has reversed returns. For low speed targets energy level of KS sparse 8 catches up with sparse 4. Since gate sizes are small, the results closely match the circuit complexity analysis given in previous section. For the LF adder, sparse 2 yields 22% energy savings over the non-sparse LF adder. LF sparse 4 yields comparable results for lower performance delay targets. Similar results are obtained for leakage power (Fig. 10). Leakage power savings reach up to 70% and 40% respectively for KS and LF adders with increased sparseness. Comparing KS and LF adders, sparse 2 LF adder is more energy efficient than the best KS adder, namely KS sparse 4. However, for high performance (<315ps) KS reaches comparable energy levels of LF adder (only 7% higher).

The contribution of wire energy to total energy is shown in Fig. 11. As was shown in Fig. 8, increasing sparseness is expected to reduce wiring complexity, hence its contribution to total energy. Fig. 11a shows the energy contribution of wires to the total energy for KS adders. KS Sparse 8 has the lowest wire energy contribution. The contribution of wire energy is reduced in price of increased circuit complexity. Hence, it is not translated into overall energy reduction. Note

that the wire energy contribution increases as performance decreases. This is because for low performance targets, gate sizes are small and wiring does not scale down proportionally. Hence, wire capacitance start to dominate over gate capacitance. On the other hand, for the LF adder, wire energy does not change significantly with increased sparseness. Although results in Fig. 8 show a small decrease in wire energy Fig. 11b does not show significant change. The discrepancy can be attributed to the increased vertical wiring neglected in the analysis.

VI. Conclusion

In this work, the energy optimal sparseness degree in the carry tree of static CMOS parallel adders is investigated. Efficient sum path adders are proposed to achieve higher sparseness levels that could not be achieved by using ripple carry adders without degrading performance. Sparseness degrees of 1, 2, 4, and 8 are analyzed on 64-bit Ling adders using Kogge-Stone (KS) and Ladner-Fischer (LF) carry prefix schemes at a 45nm technology node using a standard cell library and automatic synthesis and layout optimization flow. Energy savings of 50% and 22%, and leakage power savings of 70% and 40% are achieved with increased sparseness degree of carry trees for KS and LF adders, respectively. For 64-bit KS adder, dynamic energy and leakage power optimal sparseness is 4. For 64-bit LF adder, dynamic energy optimal sparseness is 2 whereas leakage power optimal is 4. Both optimal KS and LF adders reach the same minimum delay target of 300ps with LF sparse 2 consuming 1.43pJ dynamic energy which is 7% less than KS sparse 4.

References

[1] A. Weinberger and J.L. Smith, “A Logic for High-Speed Addition,” Nat. Bur. Stand. Circ., 591:3-12, 1958.

[2] H. Ling, “High-Speed Binary Adder,” IBM Journal of Research and Development, vol. 25, no.3, pp. 156-166, May 1981.

[3] R. W. Doran, “Variants of an Improved Carry Look-Ahead Adder,” IEEE Transactions on Computers, Vol. 37, No.9, Sept. 1988.

[4] J.Sklansky, "Conditional-Sum Addition Logic," Electronic Computers, IRE Transactions on , vol.EC-9, no.2, pp.226-231, June 1960.

[5] O. J. Bedrij, “Carry-Select Adder,” IRE Trans. on Electronic Computers, Vol. EC-11, pp. 340-346, 1962.

[6] P.M. Kogge and H.S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations”, IEEE Trans. Computers Vol. C-22, No. 8, Aug. 1973, pp.786-793.

[7] R. P. Brent and H. T. Kung, “A Regular Layout for Parallel Adders,” IEEE Transactions on Computers, C-31(3), March 1982.

[8] T. Han and D.A. Carlson, “Fast Area-Efficient VLSI Adders,” 8th IEEE Symposium on Computer Arithmetic, Como, Italy, pp.49-56, May 1987.

[9] T. Lynch and E. Swartzlander, "The redundant cell adder," Computer Arithmetic, 1991. Proceedings., 10th IEEE Symposium on , vol., no., pp.165-170, 26-28 Jun 1991.

[10] A. A. Farooqui, V. G. Oklobdzija, and F. Chechrazi, "Multiplexer based adder for media signal processing," VLSI Technology, Systems, and Applications, 1999. International Symposium on , vol., no., pp.100-103, 1999.

[11] W. Ramchan, L. Se-Joong, and Y. Hoi-Jun, "A 670 ps, 64 bit dynamic low-power adder design," Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on , vol.1, no., pp.28-31 vol.1, 2000.

[12] S. Mathew, M. Anders,R. K. Krishnamurthy,and S. Borkar, "A 4-GHz 130-nm address generation unit with 32-bit sparse-tree adder core," Solid-State Circuits, IEEE Journal of , vol.38, no.5, pp. 689-695, May 2003.

[13] B. R. Zeydel, T. Kluter, and V. G. Oklobdzija, “Efficient Mapping of Addition Recurrence Algorithms in CMOS,” 17th IEEE Symposium on Computer Arithmetic, Cape Cod, Mass., USA, June 2005.

[14] R. Zlatanovici, S. Kao,B. Nikolic, "Energy–Delay Optimization of 64-Bit Carry-Lookahead Adders With a 240 ps 90 nm CMOS Design Example," Solid-State Circuits, IEEE Journal of , vol.44, no.2, pp.569,583, Feb. 2009.

[15] 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 Transaction on VLSI Systems, vol. 13, no. 6, pp. 754-758, June 2005.

[16] B. R. Zeydel, D. Baran, and V. G. Oklobdzija, "Energy-Efficient Design Methodologies: High-Performance VLSI Adders," Solid-State Circuits, IEEE Journal of , vol.45, no.6, pp.1220-1233, June 2010.

[17] R. Zimmermann, ‘Binary Adder Architectures for Cell-Based VLSI and their Synthesis’, PhD Thesis, Swiss Federal Institute of Technology (ETH), 1998.

[18] D. Patil, O. Azizi, M. Horowitz, R. Ho, and R. Ananthraman, "Robust Energy-Efficient Adder Topologies," Computer Arithmetic, 2007. ARITH '07. 18th IEEE Symposium on, vol., no., pp.16-28, 25-27 June 2007.

[19] D. Markovic, V. Stojanovic, B. Nikolic, M. Horowitz, and R. Brodersen, “Methods for True Energy-Performance Optimization,” IEEE J. Solid- State Circuits, vol. 39, no. 8, pp. 1282–1293, Aug. 2004.

[20] S. Knowles, “A family of adders,” in Proc. 14th IEEE Symp. ComputerArithmetic, Apr. 1999, pp. 277–281.

[21] R. E. Ladnerand M.J. Fischer; ‘Parallel PrefixComputation’ JACM, 27(4):831-838, Oct. 1980.

[22] A. Neve, H. Schettler, T. Ludwig, and D. Flandre, “Power-Delay Product Minimization in High-Performance 64-bit Carry Select Adders,” Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.12, no.3, pp.235-244, March 2004.

[23] J. Grad and J.E. Stine, "A Multi-Mode Low-Energy Binary Adder," 40th Asilomar Conference on Signals, Systems and Computers (ACSSC'06.), Oct. 2006, pp.2065-2068.

[24] D. Harris, "A taxonomy of parallel prefix networks," 37th Asilomar Conference on Signals, Systems and Computers, Nov. 2004, vol. 2, pp. 2213,2217.