Power – Performance Optimal 64-Bit Carry-Lookahead Adders

Radu Zlatanovici, Borivoje Nikolic

Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley, CA 94720
{zradu,bora}@eecs.berkeley.edu

Abstract

A circuit sizing tool that minimizes the delay under energy constraints has been developed using optimisation software, tabulated delay models and analytical energy models. The tool is used to generate energy–delay (E-D) tradeoff curves for selected high-performance 64-bit carry-lookahead domino adders. The optimisation indicates that the sparse radix-4 carry-lookahead adder with sparseness factor of 2 has optimal performance in the energy-delay space.

1. Introduction

Carry-lookahead adders are frequently used in high-performance microprocessor datapaths. With the constant increase in clock frequencies, together with reduced logic depths, the timing constraints on basic building blocks are tighter. Power increases as well, and integer execution units typically are among the blocks with the highest power density on a microprocessor chip.

Carry computation is the critical component of wide carry-lookahead (CLA) adders and is usually implemented as a parallel prefix tree. Among all trees, the Kogge-Stone tree [1] is the most commonly used parallel prefix computation topology in high-performance datapaths. Its main features are minimum logic depth, regular structure and uniform fanout. Its main disadvantages are large number of wires and high power dissipation. A recent implementation of the Kogge-Stone tree in a 64-bit adder is reported in [2]. The number of nodes and connections in the tree can be reduced by trading it off for increased logic depth by making the tree sparse; an example of a tree with sparseness of 2 is the Han-Carlson tree [3]. Sparse-tree adder implementations have been recently reported, with sparseness of 2 [4] and sparseness of 4 [5].

An alternate logic optimisation investigates different implementations of carry equations. Ling's equations can lead to the simplification of parallel prefix nodes [6], [7].

Although adder design is a well-researched area, fundamental understanding of adder performance in the energy-delay space is still largely unexplored. There is no definitive answer on which implementation leads to the fastest possible adder, or which one has the smallest delay for given energy consumption. Existing comparisons of delay deal with the impact of wires with fixed sizing [8], impact of carry tree topology on logic depth [9] and optimal transistor sizing using logical effort [10].

This work presents a unified framework for optimal sizing of parallel prefix adders in the energy - delay space. To determine which adder performs best in a specific system, each of the candidate designs has to be optimally sized. In this paper, power-performance tradeoffs are analysed for various 64-bit carry-lookahead (CLA) adders using optimisation software and accurate tabulated delay models. A number of selected parallel prefix adders is evaluated in a typical integer execution unit environment. Transistor sizing for each of these adders is optimised to achieve minimum delay under maximum energy constraints. The sizing accounts for internal capacitive wire parasitics based on a sketch of the floorplan. The present evaluation is restricted to 64-bit adders but the methodology itself is general and can be applied to other structured circuits as well.

2. Optimisation Framework for Gate Sizing

Optimisation framework. The optimisation framework is built around a mathematical optimiser with a static timer in its loop, as shown in Figure 1.

The problem is formulated as a minimization of the circuit block delay under maximum energy constraints by continuously tuning individual gate sizes. Each gate is characterised only by its size. Additional constraints are set to the problem, such as maximum internal slopes and maximum gate pin capacitances.

All circuit-related calculations are performed by a static timer using the classical depth first search algorithm. Its inputs are a gate-level netlist that includes wire loads, a set of input assertions (input arrival times and slopes), a set of output assertions (loads on the output nodes) and the models for the delay and the energy of each gate. The timer computes the delay and energy of the circuit under optimisation as well as some other values such as the pincaps and the signal slopes of the internal nodes of the circuit.

In the main optimisation loop, the optimiser passes a vector of gate sizes (W) to the timer and gets the resulting delay and energy of the circuit. The process is repeated until it converges to a set of optimal gate sizes.

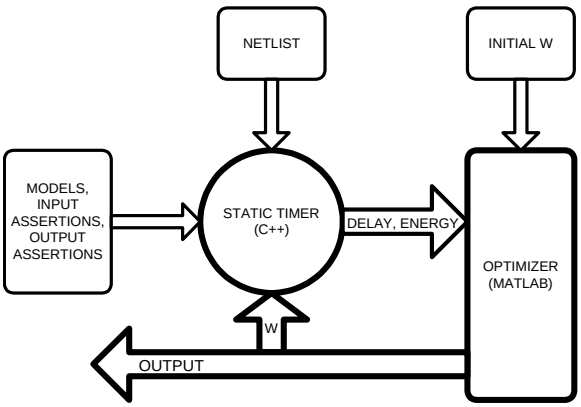

Figure1: Architecture of the optimisation framework.

Since the timer is in the optimisation loop, it is implemented in C++ to accelerate computation. It is coupled with Matlab's non-linear large-scale optimiser. The speed-critical optimisation loop is highlighted in Figure 1.

Delay and Energy Models. The static timer is modular and can use various delay and energy models. Simple analytical models such as Elmore-based RC delay lead to convex optimisation problems that can be solved very efficiently. However, such models have limited accuracy and have difficulties incorporating second order delay dependencies, such as signal slopes.

In this work tabulated delay models are used in order to obtain high accuracy. The gate delay model is represented as a set of tables of simulation data, one table for each input. Simulation data is tabulated over a wide range of fanouts (C_L/C_{in}), and input slopes. Keepers in dynamic gates are additionally modelled by adding a column with the ratio of the size of the keeper transistor to the size of the pull-down network. Linear interpolation between neighbouring points is used to calculate the actual delay and output slope. Apart from the delay tables, gate models also include data about input capacitances, worst-case leakage power for 0 and 1 outputs, and the implemented logic function.

The energy model is analytical and separates the switching component from the leakage component:

$$E = \sum_{all\_nodes} \alpha_i C_i V_{DD}^2 + \sum_{all\_gates} P_{leak,j} / f \qquad (1)$$

Node capacitances are computed by the timer using the input capacitance and wire data. Node activities are computed by logic simulation: a large number of random input vectors is passed through the circuit and the number of zeroes and ones is recorded at each node. Logic simulation was preferred over a probability propagation algorithm because of its simplicity. Leakage power is computed as the weighted average of the worst case 0 and 1 leakage powers from the gate model, using the logic level probabilities as weights.

The energy model has two main limitations: (1) it does not account for the crowbar current; however, by enforcing reasonable slope constraints, this power component is negligible; (2) the worst-case leakage model is pessimistic; since the analysed circuits are high-activity datapath blocks, leakage accounts for only a small percentage of the overall energy.

3. Selected 64-Bit Adders

We selected a set of representative carry-lookahead adder topologies for optimisation and comparison: radix 4 Kogge-Stone tree [2], radix 4 tree with sparseness of 4 [11], [5], radix-2 tree with sparseness of 2 [4] and 4 [12]. Radix-2 Kogge-Stone tree is included as a reference design. Ultimate performance under power constraints usually dictates the use of single rail domino logic.

In order to fairly compare these carry-lookahead schemes, a generic 64-bit adder structure is constructed, shown in Figure 2. It corresponds to a 64-bit integer execution unit, in a high-performance microprocessor, similar to [4] and [11].

As previously analyzed in [6] and [7], Ling's carry equations can lead to a simplification of parallel prefix nodes compared to the conventional CLA equations. A simple optimisation run reveals that adders based on Ling's equations always outperform conventional CLA, except for very tight area constraints. Therefore, all adders in this analysis implement Ling's equations:

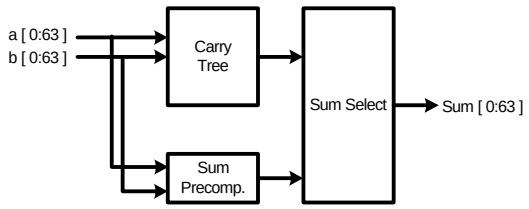

Figure 2: Generic 64-bit adder block diagram.

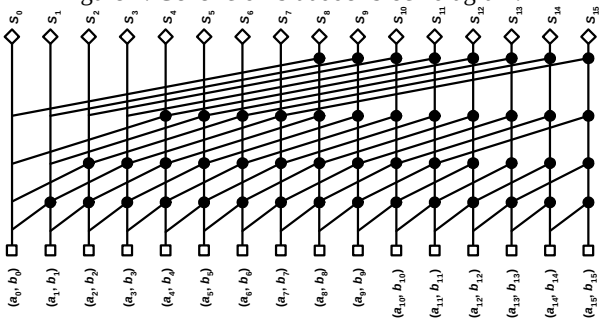

Figure 3: 16-bit radix-2 Kogge-Stone tree.

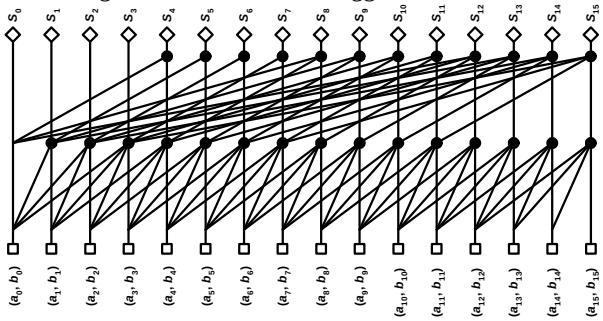

Figure 4: 16-bit radix-4 Kogge-Stone tree.

$$\begin{aligned} &g_i = a_i b_i,\ t_i = a_i + b_i \\ &H_i = g_i + t_{i-1} H_{i-1},\ I_i = t_i t_{i-1} \\ &S_i = t_i \oplus H_i + g_i t_{i-1} H_{i-1} \end{aligned} \qquad (2)$$

Figures 3 and 4 show the carry trees for the reference radix-2 and radix-4 designs (16-bit trees). For these trees, the sum-precompute block implements eq. (3):

$$S^0{}_i = a_i \oplus b_i,\ \ S^1{}_i = a_i \oplus b_i \oplus (a_{i-1} + b_{i-1}) \qquad (3)$$

where $S^0{}_i$ is the pre-computed value of the sum for an incoming carry of 0 and $S^1{}_i$ for an incoming carry of 1 at bit i.

The full trees in Figures 3 and 4 compute the intermediate and final carry for every bit. However, 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 odd carries (H_1, H_3, H_5,…, H_{63}) in the CLA block and use them to set the multiplexers in the sum-select stage. The resulting tree is sparse, with a sparseness of 2, as shown in Figure 5. 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 eq. (3), but odd order sums must be pre-computed using eq. (4):

$$\begin{aligned} S^0{}_i &= a_i \oplus b_i \oplus (a_{i-1} b_{i-1}) \\ S^1{}_i &= a_i \oplus b_i \oplus [a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1})(a_{i-2} + b_{i-2})] \end{aligned} \qquad (4)$$

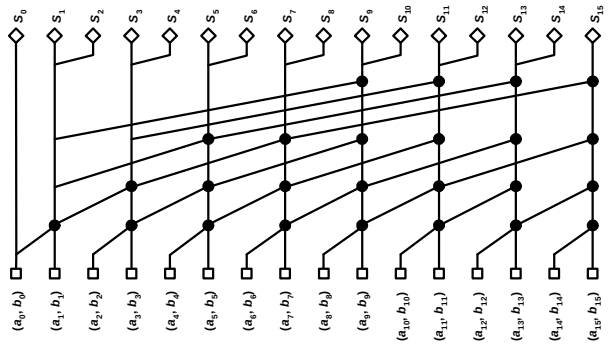

Figure 5: 16-bit radix-2 sparse tree, sparseness 2.

4. Results

Energy–delay (E-D) tradeoff curves are obtained for all analysed adders using the presented framework and a 0.13µm 1.2V technology. All adders are optimised in the same environment, with the same constraint on the input capacitance and maximum internal slopes. The load at the output was chosen 3 times the input capacitance. This is a realistic value for an ALU [11] where additional buffers are inserted to drive large loads. The height of a bit slice is selected to be 18 metal pitches [11] and wire lengths are estimated according to the number of traversed bitslices.

Figure 6 shows the E-D tradeoff curves for the 2 reference designs – the radix 4 and radix 2 full trees. On the x-axis the delay is normalized to the delay of a fanout-of-4 (FO4) inverter. The y-axis shows the total energy per transition, in pJ. On each curve, the point with the lowest delay represents the best performance that can be achieved by the respective structure regardless of the power budget (unconstrained minimum delay). The lowest energy point is determined by minimum size constraints and slope constraints.

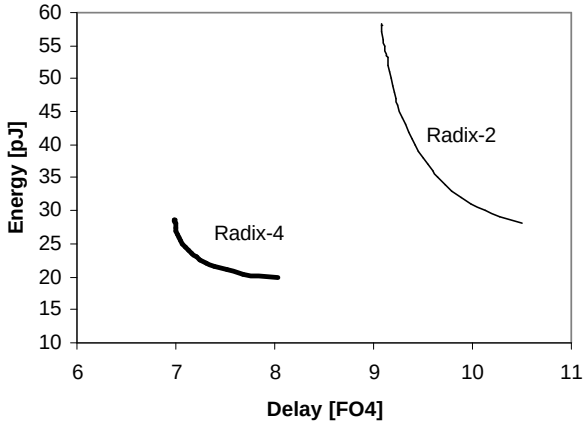

Figure 6: E-D tradeoffs for radix-2 and radix-4 Kogge-Stone trees.

It can be noted from the figure that in these loading conditions, the radix-4 design is both faster and consumes less power than radix-2. The critical path of the radix-2 tree has 7 domino stages, which is larger than what is optimally needed to drive a relatively small overall fanout of the adder. The radix-4 tree has more complex and slower gates, but their number is smaller and the critical path has only 4 domino stages.

Figure 7 further compares the E-D tradeoff curves for all radix-4 designs – full and sparse trees.

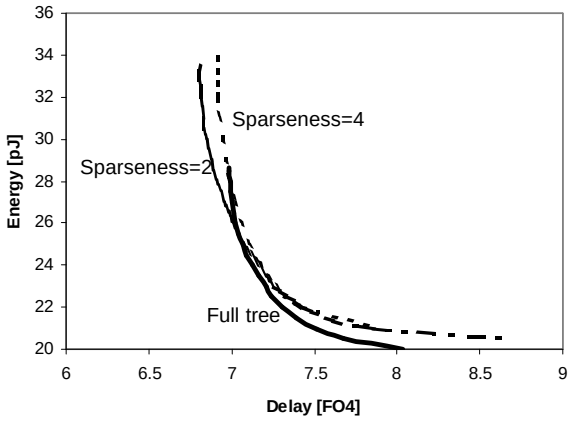

Figure 7: E-D tradeoffs for full and sparse radix 4 trees.

The differences between a sparse and a full tree are reflected in the following ways: (1) Larger output load of the carry tree: one carry output of the CLA block must drive a number of gates that is equal to the sparseness factor of the tree, thus slowing down the circuit. Optimal delay is achieved through upsizing the critical path (2) Reduced input loading for the CLA 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. (3) More complex sum-precompute blocks that slow down the critical path through additional branching.

The overall result is a balance of all the above factors. The tree with a sparseness of 2 that computes even carries achieves the smallest unconstrained delay and therefore is the fastest in this context. Sparseness of the tree does not dramatically affect the energy of the adder. Although sparse designs have fewer gates, their critical paths are upsized proportionally to optimally drive larger loads.

Trading off the delay for energy savings affects the internal signal slopes. In highest performance designs, the slope constraint is usually active on fast non-critical paths, right before they merge into the critical path. The sum-precompute block is normally much faster than the CLA block and if no slope constraint is in effect, the optimiser will downsize all the gates in it, in order to save power. This results in a slow slope at the output of the sum-precompute block, because the sum-select multiplexer is on the critical path and is upsized in order to obtain the required performance. If slope constraints become tighter, gates in non-critical paths have to be either upsized or buffered in order to satisfy them.

The detailed structure of the fastest adder is shown in Figure 8. The carry tree computes even-order carries and is implemented in domino logic. The hard clock edge is imposed on the final sum-select gate. All dynamic gates are footless, except those connected to the primary inputs or with hard clock edges, in order to improve performance and save power.

Best performance is 6.8 FO4 inverter delays at an energy cost of 33.5 pJ per cycle. Stage-by-stage delays for the optimal design are shown in Figure 8. Increasing the delay by 3%, to 7 FO4s, saves almost 20% in energy (26 pJ per cycle).

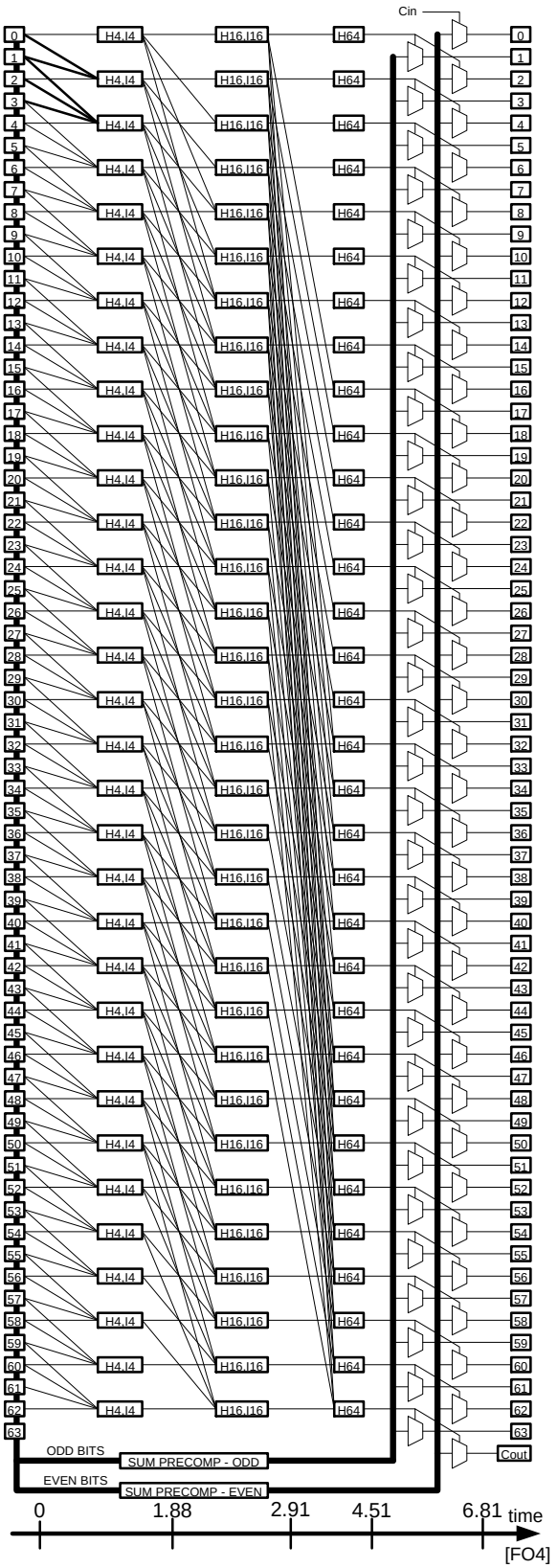

Figure 8: Fastest adder.

5. Conclusions

A circuit sizing tool is used to optimise the performance of carry-lookahead adders in the energy-delay space. Radix-4 carry-lookahead tree implemented in domino logic achieves the minimum of 6.8 fanout-of-4 delays. Full and sparse trees do not differ significantly in delay for the same energy constraints.
This tool can be used for power-performance optimisation of any regular logic block.

6. References

[1] P.M. Kogge, H.S. Stone: A Parallel Algorithm for Efficient Solution of a General Class of Recursive Equations, IEEE Transactions on Computers pp. 786-793 August 1973

[2] J. Park, H.C. Ngo, J.A. Silberman, S.H. Dhong: 470ps 64bit Parallel Binary Adder, 2000 Symposium on VLSI Circuits p.192-193, 2000

[3] T. Han, D.A. Carlson: Fast Area Efficient VLSI Adders, 8th Symposium on Computer Arithmetic, p.49-56, 1987

[4] S. Mathew, R. Krishnamurthy, M. Anders, R.Rios, K. Mistry, K. Soumyanath: Sub-500ps 64b ALUs in 0.18µm SOI/Bulk CMOS: Design & Scaling Trends, International Solid-State Circuits Conference, p.318-319, 2001

[5] S. Naffziger: A sub-nanosecond 0.5µm 64b adder design, International Solid-State Circuits Conference, p.210-211, 1996

[6] H. Ling: High Speed Binary Adder, IBM J. Res. Develop, vol.25, no.3, p. 156-166, May 1981

[7] R.W. Doran: Variants of an improved carry look-ahead adder, IEEE Transactions on Computers, vol. 37 9/1988 p.1110-1113

[8] Z. Huang, M.D. Ercegovac: Effect of Wire Delay on the Design of Prefix Adders in Deep-Submicron Technology, 34th Asilomar Conference on Signals, Systems and Computers, vol. 2 p. 1713-1717, 2000

[9] A. Beaumont-Smith, C.C. Lim: Parallel Prefix Adder Design, 15th Symposium on Computer Arithmetic, p.218-225, 2001

[10] H.Q. Dao, V. Oklobdzija: Application of Logical Effort Techniques for Speed Optimisation and Analysis of Representative Adders, 35th Asilomar Conference on Signals, Systems and Computers, vol. 2 p. 1666-1669, 2001

[11] Y. Shimazaki, R. Zlatanovici, B. Nikolic: A Shared-Well Dual-Supply-Voltage 64-bit ALU, International Solid-State Circuits Conference, p. 104-105 2003

[12] S. Mathew, M. Anders, R. Krishnamurty, S. Borkar: A 4GHz 130nm Address Generation Unit with 32-Bit Sparse-Tree Adder Core, Symposium on VLSI Circuits, p.126-127, 2002