Exploring Multiplier Architecture and Layout for Low Power

Pascal C. H. Meier, Rob A. Rutenbar and L. Richard Carley

Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213
email: pascal@ece.cmu.edu rutenbar@ece.cmu.edu lrc@ece.cmu.edu

Abstract*

Multiplication represents a fundamental building block in all DSP tasks. Due to the large latency inherent in multiplication, schemes have been devised to minimize the delay. Two methods are common in current implementations: regular arrays and Wallace trees. Previous gate-level analyses have suggested that not only are Wallace trees faster than array schemes, they also consume much less power. However these analyses did not take wiring into account, resulting in optimistic timing and power estimates. We develop a simplified comparative layout methodology to analyze the effect of physical layout on these designs. Results for short bit-width (8, 16, 24 bit) DSP multipliers show that while wiring has a major impact on signal delay and power, Wallace trees still show roughly a 10% power advantage over array-based designs.

Introduction

There has been renewed interest in basic digital arithmetic in the last few years, driven originally by the migration of fast floating point hardware into standard microprocessors, and more recently by the demand for multimedia functionality in both desktop and portable systems. In the latter case, the demands of DSP-style systems for both high throughput (e.g., for voice, video) and low energy consumption has spawned new work in low-power circuit styles [1], DSP synthesis [2], and examination of basic tradeoffs in different arithmetic styles [4]. Our interest is in the basic building blocks of arithmetic circuits, in particular, short word width (8 - 24 bit) multipliers of the type that dominate in DSP applications.

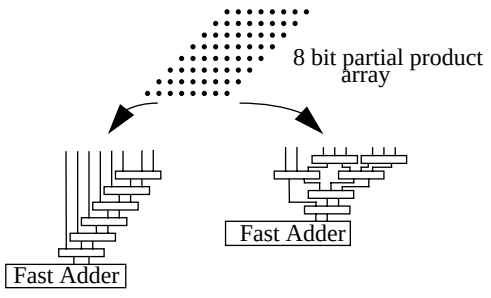
For individual arithmetic blocks, different gate-level architectures have a substantial impact on hardware size, layout complexity, speed and power. For basic adder types (ripple, lookahead, skip, bypass, *etc*.), the power tradeoffs were clearly mapped by Callaway and Swartzlander in [4]. For example, ripple adders are slowest but use the least energy, whereas speculative styles such as bypass adders are fast but consume much more power, since they perform computations they later discard.

The tradeoffs for basic multipliers, however, are much less well characterized. Alternative designs focus on the manner in which the trapezoidal partial product array of individual bit-wise products are reduced (summed) to produce the final product (see Figure 1). *Array* styles [10] use a regular 2-D grid of adders for this reduction. Compact and easy to lay out, the arrays perform this reduction in gate depth that is linear in the bit width. At the other end of the spectrum, *Wallace tree* styles [3] use a log-depth tree network for this reduction. Faster, but irregular, they trade ease of layout for speed. Although the speed-size tradeoffs for these two styles are fairly well characterized, the power tradeoffs are not well understood.

For example, Bellaouar and Elmasry [5] suggest that Wallace tree styles are best avoided for low power applications, since the excess wiring is likely to consume extra power. On the hand, Callaway and Swartzlander [4] demonstrate quantitatively that switching activity within just the partial product reduction hardware is substantially better for the tree over the array-–if one ignores the wires completely. Montoye [7] alludes to the difficulties in dealing with the tree wiring for one high-performance commercial processor (IBM RS/6000), but concludes that the speed gain is worth the trouble; unfortunately power was not a concern in this design.

In this paper, we attempt to shed some light on the question: *are Wallace tree style multipliers competitive with array styles for speed and power*? Intuition suggests that the log-versus-linear depth of the reduction network for the tree might well lead to shorter propagation paths and less power-consuming glitching. First order analysis of the average transition counts across sets of random vectors applied to both array and Wallace tree designs supports this point of view (see Figure 2.), as does the work of

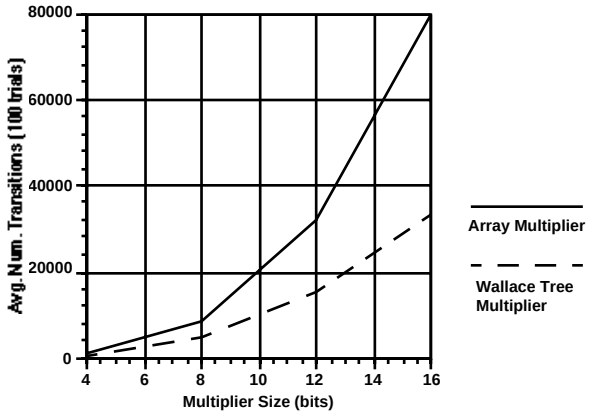
Figure 1. Multiplier Architectures



(a) Array architecture (b) Wallace Tree architecture

* This work was supported in part by the Semiconductor Research Corp, the National Science Foundation and ARPA.(3)

Figure 2. Transition count comparison for multipliers.



[4]. However, these analyses are incomplete due to the lack of any consideration of wiring effects, and are therefore unable to support or refute the claim of [5].

To remedy this, we suggest a more detailed evaluation model that considers not only the gate-level differences, but also the wiring effects due to layout. As we shall see, experiments suggest a posture of cautious optimism for the Wallace trees: they are neither as bad as [5] suggests, nor as good as [4] estimates, but they do appear roughly comparable within the limits of our model, with the tree style somewhat better on both energy consumption and on speed. The rest of the paper describes our modeling methodology, our layout strategy for comparing arrays and trees, and our experimental results.

Analysis Methodology

To fairly compare the array and Wallace tree multiplier styles, we need the following:

- **Cell-level generators for the multipliers' partial product reduction networks:** where "cell" here means the component (3,2) carry save adders and single-product AND gates needed in the design.
- **Cell-level generators for multipliers' final 2-level reducing adders:** each style requires a final addition, and each can use different adder styles to achieve different tradeoffs.
- **Cell characterization:** to understand power consumption in each of these primitive component cells, as implemented by conventional static CMOS circuits.
- **Layout model:** to allow exploration and first-order estimation of wiring, delay and area, we need a simple layout style that is easily automatable, yet not so detailed as to demand manual, full-custom layout.
- **Power and delay estimation:** to measure power and delay for the completed multiplier layouts–including the delay and capacitance of all the intercell wiring.

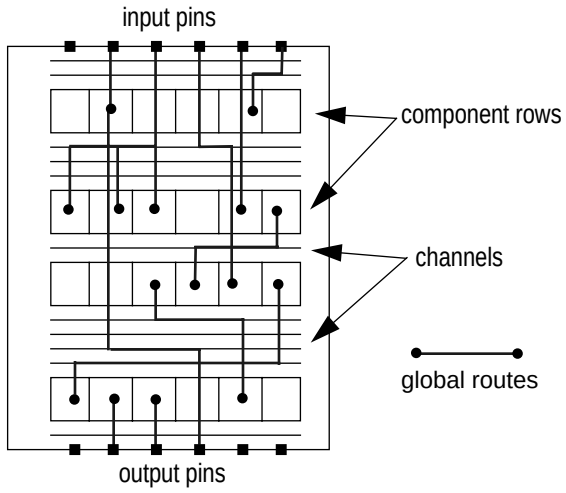
We summarize our strategy for each of these in this section.

For simplicity, we restrict ourselves to unsigned operands, and focus exclusively on the short word widths common in DSP applications: 8, 16 and 24 bits. Cell-level generation of the netlist for each multiplier is straightforward. The only degrees of freedom here are the detailed arrangement of the levels of carry save adders in the Wallace tree, for which we use the basic arrangement suggested by Wallace[3], and the final reducing adder at the bottom of each architecture. Since this adder is a large part of the overall delay, we evaluate with respect to three different adder styles for each: carry ripple, carry skip, and carry select addition. Carry select addition precomputes carries during the addition operation. Carry skip addition uses parallel evaluation of the criterion for carry propagation to allow the carry to traverse the adder faster. Ripple addition is simply a chain of full adders, and represents the most basic addition technique. See [11] for more details.

Given these netlists of basic cells, we next characterize the power and delay of each cell prior to analysis of the overall multiplier. We designed simple static CMOS circuits for each component in our multipliers, using parameters from the Hewlett-Packard 0.8μm CMOS process (CMOS26G); the designs assume a 3 V supply and are based on circuits presented in [10]. There are two components to power dissipation in these cells: a) the power dissipated in the circuit switching, and b) power delivered and removed from the capacitive load on each output. Both effects were measured using HSPICE simulations for all output transitions, at various capacitive loads. Cell delay was also calculated, in this case as the difference between the input and output signals at their 50% voltage points.

To evaluate complete multipliers, we require a common layout model for both the regular array and the Wallace tree. It is important here to avoid penalizing one style artificially at the expense of the other. We use a simple unit grid model, in which each component cell of the multiplier netlist occupies one slot in a set of standard cell-style rows (see Figure 3). We assume over-the-row routing of vertical wires connecting cells in different rows, but we conservatively estimate the impact on total height and wire length of the wires that must make horizontal jogs in each wiring channel. A post-process global router embeds wires into the placed model, and the maximum density of each channel is used to derive channel height and final wirelength estimates.

Figure 3. Multiplier layout model.



For the array, we procedurally tile the regular partial product reduction cells into the grid, with the final reduction adder carefully located at the bottom of the array. Since nearly all the connections in the array are nearest neighbor between cells in the partial product reduction network, the array fits well into this model. On the other hand, the Wallace tree requires constructive placement and routing of its partial product generation (AND gates), reduction network ((3,2) carry save adder tree) and final reduction adder. We use a simple annealing-based placement strategy [7] that strives to minimize the overall wirelength while densely packing the grid.

Finally, given a complete netlist and a real, although highly simplified placement, we can extract back per-net capacitance and determine estimates of both power and delay for each complete multiplier. Initially, we developed Verilog files for the various multiplier architectures, where power was computed by counting the number of transitions which occurred at each cell output. To accelerate this, we developed a simplified, custom logic simulator for these designs. The simulator is an event-driven evaluation engine, where drivers send signals to receivers. If the value of a receiver is unresolved when a new driving signal arrives, this corresponds to a potential glitch, and the previous driving signal may be preempted. Since the filtering of small glitches from the event queue can cause power estimation to be inaccurate (*i.e.*, too low), power consumed during glitches is also estimated. Comparisons between our custom logic simulator, operating on characterized cell-level netlists with backannotated wire delays, against transient device-level HSPICE simulation on small (4 bit) multipliers has shown this method to be acceptably accurate, typically within 5% of HSPICE estimates.

Determining the complete power dissipated in a multiplier requires sensitizing all possible combinations of inputs. This means for a circuit of n inputs, 2^{2n} input combinations have to be simulated. However, it is possible to estimate *average* power dissipation by running sets of test vectors, and performing mean estimation using statistical techniques. Such techniques have been shown to be effective in determining the actual power dissipation within a known tolerance [6]. In our approach, we run batches of vectors to insure 95% confidence intervals, *i.e.*, to estimate that the computed average power dissipation is within 2% of the actual mean, with a confidence of 95%.

To estimate delay, we use three strategies. For the smaller multipliers (up to 8 bit operands) we exhaustively simulate all pairs of inputs and use our custom logic simulator to measure the worst case delay. For larger designs, we use simple static timing analysis (topological longest path) to derive a *pessimistic* worst case for delay. We also simulate batches of random vectors to estimate an *optimistic* worst case delay. Together these two measurements bound–albeit loosely–the worst case delay. In the following section, we present results of all these analyses.

Experimental Results

We explored 18 different multiplier implementations in all: two different architectures (array versus Wallace tree), three different final reduction adders for each multiplier (carry select, carry skip, carry ripple), and three different word widths (8, 16 and 24 bits).

Table 1. and Table 2. show the area estimates for each multiplier Recall that wiring impacts the overall area because of the density estimates for each wiring channel. Unsurprisingly, the array multipliers are always smaller due to their much more local wiring. Also, the ripple-adder versions are always the smallest, again due to the smaller hardware and more local wiring in these adders. As a point of comparison here, [12] describes a 6 bit array multiplier as part of an 8-tap FIR filter that occupies roughly 0.4 mm^2 in a 0.8µm CMOS process; this suggests that our area estimates are reasonable.

Table 1. Array multipliers - estimated area (mm^2)

Adder Type	8 bit	16 bit	24 bit
Carry Sel.	0.668	2.195	4.579
Carry Skip	0.627	2.099	4.430
Ripple	0.532	1.913	4.156

Table 2. Wallace tree multipliers - estimated area (mm^2)

Adder type	8 bit	16 bit	24 bit
Carry Sel.	0.759	2.626	5.625
Carry Skip	0.736	2.537	5.576
Ripple	0.725	2.488	5.576

Figure 4. shows estimated average energy per multiply operation for each of the multipliers. Results suggest a fairly consistent 10% energy advantage for the Wallace trees, across the three bit widths examined. Despite the larger amount of irregular wiring, the shallower partial product reduction in the Wallace tree now appears to be advantageous for power. However, given the coarseness of our layout model, a more cautious conclusion is simply that the trees' energy use appears roughly competitive with that of arrays.

Figure 5. shows estimated delay for each multiplier. In gen-

Figure 4. Estimated average energy per multiply op.

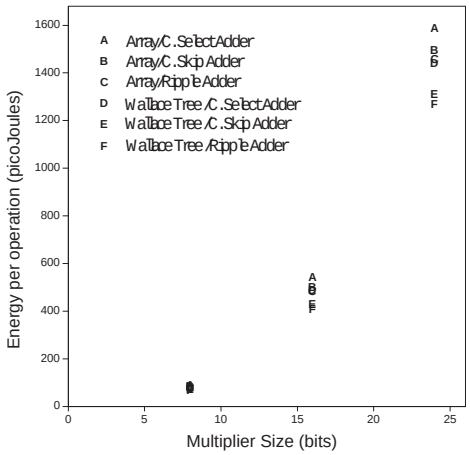


Table 3. 8 bit multiplier - estimated delay (ns)

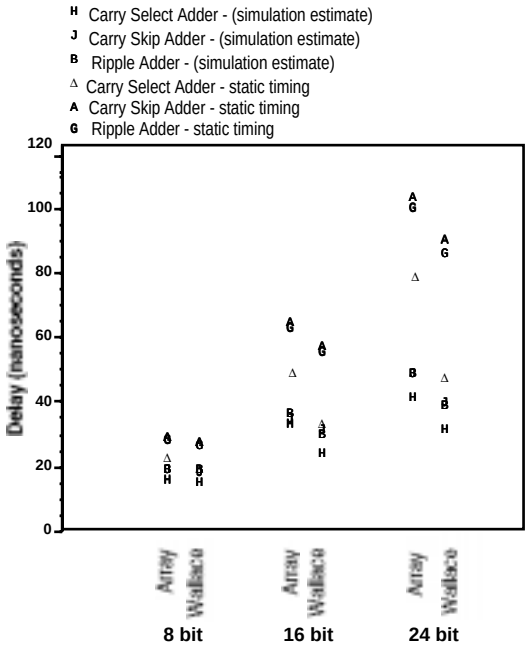
	array		Wallace tree	
	exhaustive simulation	static timing	exhaustive simulation	static timing
Carry sel.	16.12	22.73	14.86	18.90
Carry skip	18.93	28.93	18.42	27.25
Ripple	18.79	27.94	18.81	26.47

eral, the delays are fairly long since we use a 0.8µm CMOS process, and because all devices are of minimum size. For the small 8 bit design, results of exhaustive simulation over all pairs of inputs appears in Table 3., along with more conservative static timing estimates for comparison. The static timing estimates are pessimistic by 30-50%. The greatest discrepancy occurs with the carry-skip adder, which has many false paths, due to carry propagate prediction circuitry. Note that the carry skip adder does not show a speed advantage at low bit width, due to the lookahead circuitry. Without this circuitry, the carry skip adder behaves like a ripple adder. Also note that the use of a ripple adder completely negates the advantage of using a Wallace tree, as expected.

Note that Figure 5. offers both a pessimistic (static timing) and an optimistic (worst case encountered during simulation of random patterns) timing estimate for each multiplier. The wide overlap of the array and Wallace tree timing intervals certainly suggests that the Wallace trees are at least competitive in delay. Indeed, the intervals for each Wallace tree cover smaller delays than the corresponding array interval, which also suggests that the trees are faster, again as expected.

Taken together, Figure 4. and Figure 5. suggest that the Wallace trees may indeed have an energy•delay advantage over the regular array multipliers. Wallace trees are not so bad as suggested by [5], nor as significantly superior as estimated by [4].

Figure 5. Estimated worst-case delay (ns).



Conclusions

By introducing a simple unit grid layout model, we have been able to compare regular array and Wallace tree style unsigned multipliers over bit widths 8 to 24 bits, including first-order delay and area effects due to physical wiring. The model is clearly coarse, but capable of making basic predictions for area, for average power, and delay. Interestingly, the Wallace trees fare rather well, despite their irregularity and excess wiring. The smaller depth of their partial product reduction hardware seems to offset the power lost in the wiring, offering improved energy and delay. We believe this preliminary result justifies closer investigation with more refined models of problem, *e.g.*, use of more aggressive exact critical path analysis[11], and better layout optimization, to determine where the Wallace tree style may be useful.

Acknowledgments

We thank Dennis Ciplickas of Carnegie Mellon for his insights on circuit timing and power simulation issues.

References

[1] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low Power Digital Design," *IEEE JSSC*, vol. 27, pp. 473-484, April 1992.

[2] A.P. Chandrakasan, M.Potkonjak, R. Mehra, J. Rabaey, R.W. Brodersen, "Optimizing Power Using Transformations," *IEEE Transactions on CAD*, Vol. 14, No.1 pp.12-31, Jan. 1995.

[3] C.S. Wallace, "Suggestions for a Fast Multiplier," *IEE Trans. Electron. Computers*, EC-13, pp. 14-17, 1964.

[4] T.K. Callaway, and E.E. Swartzlander Jr., "Low Power Arithmetic Components." in *Low Power Design Methodologies*, Rabey, J. and Pedram, M., eds., pp. 161-198, Norwell, Mass: Kluwer Academic Publishers, 1996.

[5] A. Bellaouar, and M.I. Elmasry, *Low-Power Digital VLSI Design, Circuits and Systems*, pp. 442-450, Norwell, Mass: Kluwer Academic Publishers, 1995.

[6] R. Burch, F. Najm, P. Yang, and T. Trick, "McPOWER: A Monte Carlo Approach to Power Estimation," *IEEE/ACM ICCAD*, pp.90-97, Santa Clara, CA, Nov. 8-12, 1992.

[7] R. Rutenbar, "Simulated Annealing Algorithms: An Overview," *IEEE Circuits and Devices Magazine*, pp. 19-26, Jan. 1989.

[8] R.K. Montoye E. Hokenek, S.L. Runyon, "Design of the IBM RISC System/6000 Floating-Point Execution Unit," *IBM Journal of Research and Development*, pp. 59-77, Vol. 34, No. 1, January 1990.

[9] N. Weste and K.Eshraghian, *Principles of CMOS VLSI Design*, p. 312, Addison-Wesley 1988.

[10] D. Goldberg, "Computer Arithmetic," in *Computer Architecture A Quantitative Approach*, J.L. Hennessy and D.A. Patterson, pp. A1-A66, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.

[11] M. Sivaraman and A.J. Strojwas, "Towards Incorporating Device Parameter Variations in Timing Analysis", *Proceedings of the European Design Conference*, 1994, pp. 338-342.

[12] L.E. Thon, P. Sutardja, F. Lai and G. Coleman, "A 240MHz 8-Tap Programmable FIR Filter for Disk-Drive Read Channels," *IEEE ISSCC*, 1995, pp. 82-83.