Performance/Power Tradeoffs in ASIC Multipliers

B.Laurent G.Saucier
Institut National Polytechnique de Grenoble/CSI
46, Avenue Felix Viallet 38031 Grenoble cedex, FRANCE

Abstract

This paper presents a wide array of efficient multipliers on an ASIC submicronic technology. Connection delays are taken into account through statistical wire lengthes. For power dissipation point of view, logic switching activity as well as glitching activity are both estimated. Firstly, the goal is to provide actual and accurate characterizations of classical multipliers. Secondly, most efficient architectures are compared such that optimized performance/power tradeoffs are derived.

1 Introduction

Due to recent advances in microelectronic technology, power dissipation of integrated circuits has become, after former interest in area and performance, a major concern in VLSI circuit design. Two major reasons feed this interest: (1) power dissipation rises with increased gate density and clock frequency. Devices are overheated and this temperature ultimately degradates the circuit performance. (2) Portable systems have shifted from low performance applications to efficient communication products such as cellular phones or laptop computers. Computer Aided Design tools are needed to manage the power of a circuit design: automatic synthesis has to produce power-saving systems.
On the other hand, multiplication is involved in about 10% of all instructions in typical scientific programs [1]. Due to its quite long latency, it usually takes several cycles to perform. Sequential multipliers may even generate and accumulate summands sequentially. As a consequence, designing fast multiplications is crucial for processor efficiency.
Those two major requirements drive us to study performance/power tradeoffs in multipliers during the logic synthesis phase. The implementation technology is a submicronic ASIC ($0.8\mu m$), from Thomson TMS.

Multiplication consists in accumulating partial products [2]. As a consequence, two major steps need to be optimized: the generation of partial products and their accumultation. Accumulating two levels of partial products is an addition such that two operand addition is the third step in the whole algorithm. Only one type of multiplier is studied here, which is the most efficient, that is parallel multiplier: the partial products are all generated in parallel and then a multi-operand addition is performed for their accumulation. To speed up multiplication, fast algorithms use three techniques: few partial product are generated, their accumultation is performed as fast as possible, and finally, the final

addition is designed to accelerate carry propagation. Fortunately, the fastest strategies are not always the ones that dissipate the more power such that performance/area tradeoffs are actually good solutions.

The remainder of this paper is organized as follows. Section 2 presents the estimation of critical path and power dissipation we used. Sections 3 to 5 characterize the three steps of the multiplication. The suitable strategies are combined to form complete multipliers in section 6, which are compared to reach optimal performance/power tradeoffs. Finally, section 7 concludes the paper.

2 Performance and Power Estimation

2.1 Critical Path Estimation

For combinational circuits whose inputs are synchronized to a system clock, which is common for pipelining techniques, each input has only one arrival time. The synthesis tool used for experiment supports intrinsic and extrinsic delays to propagate these times according to the following equation:

$$\Delta(gate, input) = \alpha(input) + \beta(input) * Capacitance \quad (1)$$

Moreover, we included in our timing estimate the distributed RC effects. For very long wires or for deep submicronic technologies, the resistance-capacitance delays tend to dominate. The solution for the propagation of a voltage step along a wire is proportional to $R_{wire}C_{wire}$ [9] (R_{wire} and C_{wire} represent the total resistance and capacitance of a wire). In addition, the capacitance C_{wire} has to be charged by the driver such that the term $\beta(input) * C_{wire}$ is added to the gate delay. The values for wire characteristics are provided in the ASIC library and are proportional to the total area of the circuit. Let us note that the output netlists are not bufferized. The primary inputs may have increasing fanout whose delay could be drastically reduced by insertion of fast buffers. However, as all the multipliers schemes have such fanouts, the comparisons are still relevant but the given delay could be reduced. Primary inputs and outputs intervene as drivers and input capacitances.

2.2 Power Modelling

2.2.1 Power dissipation model

Sources of power dissipation in CMOS devices are splitted into three components. With V_{dd} the supply voltage, N the switching activity, f the frequency, and C the output capacitance, the major term represents the power required to charge and discharge the output node capacitance, and it is given by:

$$P = \frac{1}{2} C.f.V_{dd}^2.N \quad (2)$$

To reach pattern-independence, probabilities where introduced [3] to perform the an averaging of switching activities before running the analysis [4]. Based on the Boolean network functionality and connectivity, these input probabilities are propagated through

the network, using binary decision diagrams (BDD) [5] so that reconvergent fanouts and feedback are taken into account. Assume the probability of internal node N being 1 is P_1 and an uncorrelation between consecutive input vectors, the probability for node N to undergo a change is equal to $P_T = 2P_1(1 - P_1)$.

To apply these probabilistic operations on BDDs, one must take care of the number of input variables because as the size of a BDD grows exponentially with its inputs, the computation may run into memory overflow problems. As a consequence, the circuit is partitionned so that the number of correlated nodes within each partition is maximized [6]. A example of [6] illustrates the BDD computation and partitionning (Fig. 1).

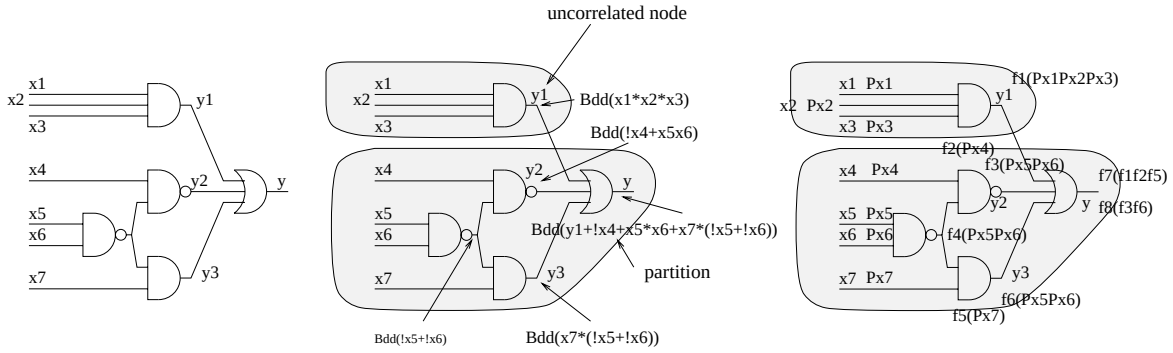

Figure 1: BDD computation/partitionning and glitch modelling

2.2.2 Glitch modelling

The probabilistic model presented in section 2.2.1 allows the computation of the useful power consumption. However, in a real delay model, the output of a gate can change more than one time before settling to its final value due to the successive values of its inputs. These spurious switchings lead to a useless power consumption called glitching dissipation. The glitching of a gate depends on the following two factors [7]: 1) the difference of arrival times at the inputs of a gate, and 2) the sensitivity of the gate to these signals. The probability of a switch due to a glitch cannot be estimated the same way as a useful switch for the simple reason that the probability of a node to undergo a switch does not depend only on the Boolean netwok functionnality but also on its structure (path lengths for instance). The formula $P_T = 2P_1(1 - P_1)$ in no longer valid for all the possible switches. To solve this problem, one has to use the real delay model on which the probability calculation is based: the switching probabilities are supposed to be known at the primary inputs of the circuits. They are propagated to the fanout gates up to the roots of the circuit using the gate delays. Each gate modifies the switching probability according to its ability to propagate the switching from its inputs to the output, what we called sensitivity. Finally, gates have a set of switching probabilities which are added. This power estimator, presented in [8], is quite accurate for combinational circuits, compared with exhaustive simulation. In the third drawing of Fig. 1, the 2 input NAND gate ($x5\bar{x}6$) produces two different switches at the output of gate $y2$, $y3$ and y due to different arrival times (for instance, $y2$ has two switches: the first depends only on $Px7$ and the second is a fonction $f6$ of $Px5$ and $Px6$). These three gates have a switching probability

higher than the one of the BDD estimate.

For the following section comparisons, we give a logic power dissipation, derived from the exact BDD estimate. The total dissipation is obtained with the second approach and the difference is the glitching activity. All the figures correspond to the switched capacitance $\sum pc$ for all the gates of the cicruit. The output capacitance is composed of the fanout gate input capacitances as well as, for the final comparisons of section 6 , the wire capacitances.

3 Encoding Step

3.1 Encoding Algorithms

A small number of partial products reduces the complexity and the time needed to accumulate them. Several methods of encoding the multiplicands are possible. We classified them into three types: add ans shift algorithm, Booth encoding and multiplication decomposition.

In the *Add/Shift* algorithm, the multiplicand is conditionally added and shifted whether the multiplier bit is one or zero. The selection logic, a AND gate, has the advantage to be the fastest approach but produces the greatest number of summands, in fact n.

Booth encoding aims at examining at least three bits (say p) of the multiplier at the same time in order to produce at most $\lceil \frac{n+1}{p-1} \rceil$ summands. The *Modified Booth Algorithm* [10] requires the use of $\pm 2M$ and $\pm M$ multiples, which are easy to produce with complementation and shifting. When more bits are examined, four for the Booth 3 algorithm, the required multiples are $\pm 4M$, $\pm 3M$, $\pm 2M$ and $\pm M$. In fact, $\pm 3M$ is a hard multiple which has to be produced by a carry propagation addition (of M and $2M$). The advantage is the fewest number of partial products among all the examined schemes ($\lceil \frac{n+1}{3} \rceil$) but has the most complex selection logic. To strike a balance between an efficient dot diagram and a fast encoding, Bewick and Flynn proposed in [11] to produce the hard multiple with small carry ripple adders. The *partially redundant* multiple requires some extra summands, filled with the output carries of the adders which overlap the following adder outputs, and a constant to compensate the complementation. In fact, we propose a slight modification of this algorithm: instead of complementing the hard multiple $\pm(2M + M)$, we add the yet complemented basic multiples: $\pm(2M) + \pm(M)$ such that the compensation constant is suppressed. The internal adders must avoid lengthes for which the carries of consecutives summands may overlap such that a size of 4 is choosen as the smallest adequat length. As an intermediate between radix 2 and radix 3 algorithms, we also build a hybdrid encoding producing half of the summands with a hard multiple and the others not. We refer it as the Booth2/3 algorithm.

Finally, we also use *smaller multipliers* to implement larger ones: the multipler and multiplicand are splitted by groups of p bits and the multiplications of all the internal groups produce a reduced number of summands. To get a performant encoding compared with Booth 2 algorithm, we use $p = 4$ because greater groups are too slow. A traditionnally area-oriented scheme is used (Braun multipler), but gives good performance for such

a small size. The number of summands is $\lceil \frac{n}{4} \rceil - 1$ which is sligthly less than Booth 2.

3.2 Encoding Estimate

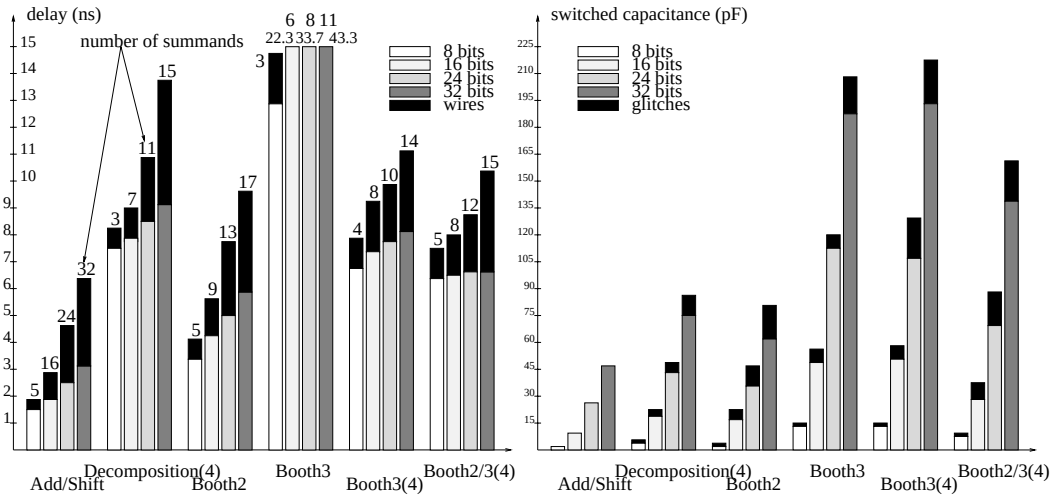

Figure 2: Encoding Estimates

The smallest set up time is achieved with the Add/Shift algorithm, due to the very simple selection logic. Booth 2 is fast because it does not require the generation of the hard multiple with a simple selection logic. To produce less summands, higher Booth radixes or multiplication decomposition are required. Using small array multipliers is slower due to the final ripple adder and Booth 3 is definitively ruled out because of a linear increase of set up time. Partially redundant Booth 3 is a succes because it is better than multiplication decomposition on both number of summands and performance points of view. The hybrid solution Booth2/3 does not bring any improvements. Let us remark than the delays of both gates and wires are increasing with the operands size: this is because the growing input fanout load the input drivers.

For the power dissipation point of view, the first three solutions dissipate much less than the others; this is due to the small area of the selection logic. Using ripple carries for each summands lead to double or triple the dissipation. Although the partially redundant Booth 3 is retained, it should not reach a good performance/power tradeoff.

4 Reduction Step

4.1 Tree Topologies

To reduce the number of summands, a carry-save addition is used such that the carries only propagate to the next bits. Approximatively the same number of (3, 2) counters are used for all the reduction scheme but there are many ways to interconnect them to produce efficient solutions.

The first family is a combinaison of *Linear Arrays* in which the partial products are added sequentially to the yet accumulated ones. The *Double Linear Arrays* [12] sums the odd and even partial products in parallel such that the delay is two times less. Zuras and McAllister [13] proposed to combine progressively longer linear arrays: an array is connected to the rest of the *ZM tree* when their delays are the same. Even more efficient, The *Overturned Stairs Tree* of Mou and Joutand [14] is recursively constructed with smaller *OS trees* and linear arrays. It may be often optimal.

On the other hand, one may try to sum all the partial products in parallel without any penalty due to the slow linear arrays: Dadda proposed in [15] to construct optimal Wallace trees using (2,2) and (3,2) counters. The number of counters used is the number of elements in the serie: $U_0 = 2, U_{n+1} = \lfloor \frac{3U_n}{2} \rfloor$ to reach the multipler summands. When two full adders are combined to form a (4,2) counter, a *4-2 counter tree* is constructed with one level of element for each power of two summands. It may be less efficient than the previous scheme for other sizes. As all the previous tree topologies have the same fanout, the delay of such structures is determined by the number of counter levels.

4.2 Reduction Estimate

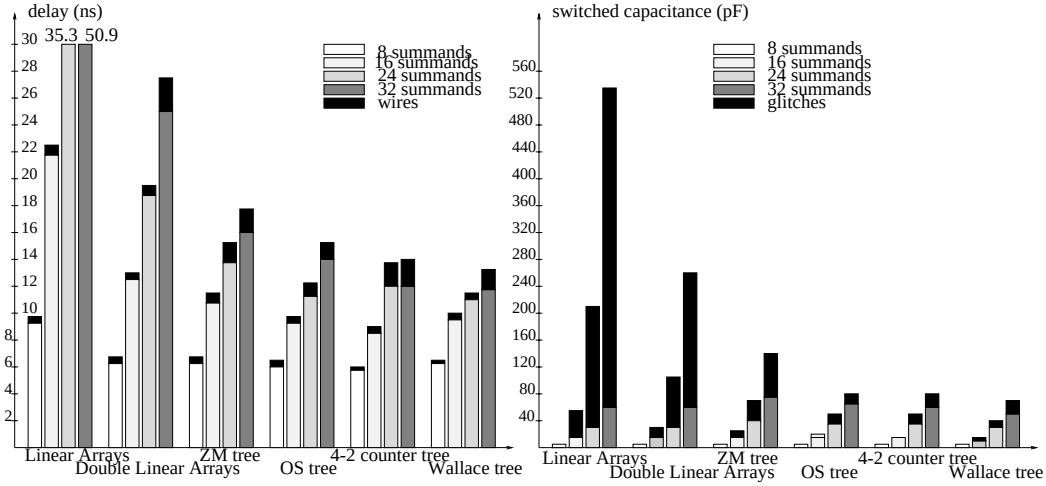

Figure 3: Reduction Estimates

The difference between serial and parallel topologies is blattant on both performance and power dissipation points of view. Fortunately, the slow structures are also the ones that dissipate the more. The explanation is in fact quite simple: the xor operator propagates switches very well since a switch at an input pin always drive to a switch at the output: $a \oplus b \neq \bar{a} \oplus b$. As linear arrays are consecutive levels of full adders, whose output is a three input Xor and whose inputs are fed with different arrival times signals, an adder may have as many different and highly probable switches as there levels of adders below it. As a consequence, the more a tree is composed of linear arrays (*Linear Array*, *Double Linear Array*, *ZM tree*, and *OS tree*), the more it dissipates power (up to 90% of glitches for 32 summands !). Other structures have between 20% and 25% of glitches, the fewest glitches are obtained for the most regular structure: 4-2 counter tree. These data were obtained with input probabilities of the Add/Shift algorithm. Let us note that when

the reduction stage follows other encoding logics, several switches may feed the topology trees such that the amount of glicthes may be bigger. Final comparisons (section 6) will take into account this remark. The Wallace tree is always the fastest as well as the 4-2 counter tree. However, for some number of summands, the 4-2 tree may be slightly less performant (see the 24 summands). From these figures, we decide to rule out all the linear array structures.

5 Final Addition

5.1 Two Operand Adder

The two last summands are summed with a two operand adder. This time, the fastest approaches dissipate the more such that a good tradeoff must be found. The main problem in designing adders is the carry propagation. They distinguish from each others with the degree of serialization and parallelization [16]. Parallelization accelerates the carries but are more complex and so dissipate more power. The staighforward inplementation is obtained by abutment of full adders such that the carry ripples of all the cells. When two sets of outputs and carries are generated assuming that the incoming carry will be either zero or one, progressively bigger ripple carry adders may be parallelized such that the delay is reduced from $O(n)$ to $O(\sqrt{2n})$. With, the progagate/generate formalism, logarithmic delays ($O(log_2(n))$) are achieved [17]: the Sklansky adder [18] has the fastest structural scheme but has an increasing fanout. When the same binary tree of cells is copied for each bit, the Kogge & Stone adder [19] is obtained with optimal delay and fanout but with an increased complexity. Finally, Han & Carlson adder [20] reduces the previous complexity with a slight delay increase.

5.2 Addition Estimate

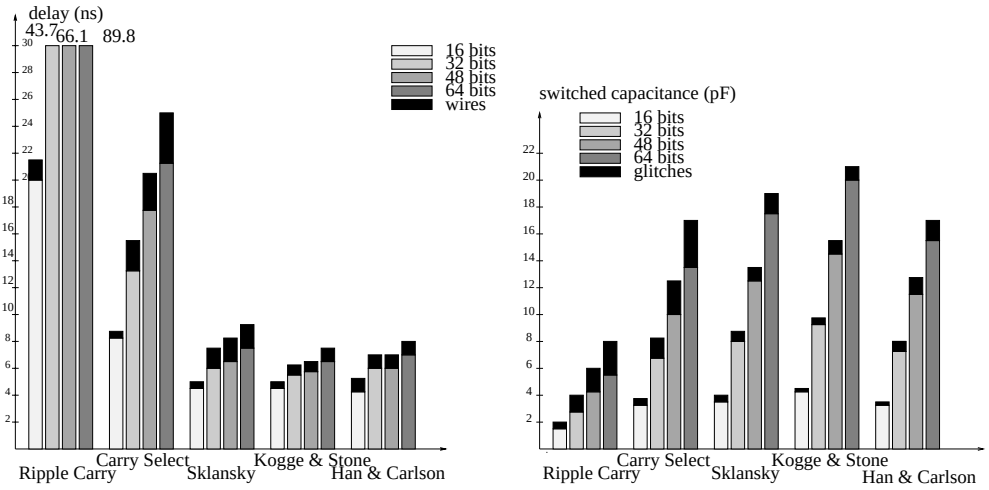

Figure 4: Final Adder Estimates

The difference between linear, square root and logarithmic delays is blattant. The three last approaches only are viable. The ripple adder has the smallest power dissi-

pation with the biggest number of glitches (about 30%). Unlike the previous reduction stage where the switches propagated through Xor operators, the switches now propagate through majority operators which reduce drastically the switching probabilities. The Han & Carlson adder is a good solution because it is not much dissipative one with a near optimal performance. Whereas the glitching activity is very small due to the quite regular structure, the Kogge & Stone adder is more dissipative because the complexity, which rises from $O(\frac{n}{2}(\lceil log_2(n) \rceil + 1))$ to $O(n(\lceil log_2(n) \rceil - 1) + 1)$, tends to double for big operand sizes. On the other hand, the Sklansky adder is slower due to increasing fanouts such that both extrinsic and wire delays are worrying for bigger multipliers. Those three solutions are retained but the Han & Carlson scheme should be the best tradeoff.

6 Experimental results

Previous efficient algorithms were combined to form complete multipliers. Four encodings, two reductions and three adders gives 24 different architectures. There were all estimated up to 64 bits such that 192 multipliers were synthetized. Experimental results are presented below:

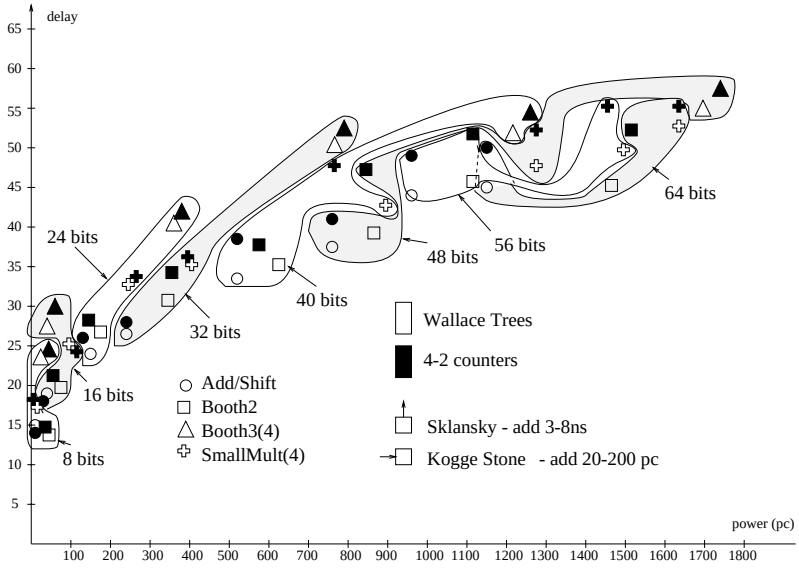

Figure 5: Complete multipliers

Wires delays as weel as glitches are much more important than the sum of the three consecutive phases for two reasons: firstly, the wire lenghtes grow with the size of the circuit. As a consequence, the three steps together have more wiring delays as the have when they are simulated alone. Secondly, the previous glitching estimate expresses the ability of a design to generate and propagate its own spurious transitions. When the three basic circuits are connected, the glitches of a part are propagated to the following ones such that the amount of glitches grows at each step.

The amount of wire load in the critical path is approximatively the same for all the architectures. It accounts for 25% of the delays for the 0.8μ ASIC technology. Glitching

activity is quite interesting for multipliers since arithmetic are known to contain a lot of spurious transitions such that classical estimates fail to provide actual dissipation. Low power effective architectures may consume only 33% of spurious transitions but the majority of all the previously studied schemes dissipate about 50% of useless power. Let us note that we focused on full swing transitions such that the power dissipation may be bigger if intermediate swing are counted. Performant architectures consume less glitches that the less efficient ones. This is because they process partial products in parallel. As a consequence, the pathes have often the same length such that logic activity dominates.

Booth 3 and implementing small multipliers give poor results. Using 4 bit array multiplers is about 20% slower than Add/Shift algorithm, due to the final ripple carry addition. Partially redundant Booth 3 is 50% slower because the ripple carry adders bring additionnal delays equivalent to four stages of a balanced Wallace tree and the encoding has huge fanouts since each partial product needs six times as many fanouts as required in Booth 2. The comparaison between Add/Shift and Booth 2 algorithms is surprising: in fact, Add/Shift appears to be more performant for the majority of the multipliers (for big sizes, they are equivalent). The reason is that the wiring load is more important is the Booth 2 encoding because the number of cells is greater such that the statistical wire lengthes are also greater. This logic synthesis evidence is offset during place and route: when reduction and encoding stages are put into parralelogram shapes, the wires needed to route the Booth muxes to the counter inputs are smaller than those to route the Add/Shift selection to the same counters because the number of summands is reduced by half. This fact drives the Booth 2 to be better whereas the statistical wiring lenghtes seem to prove the contrary. The Add/Shift is the most power-effective architecture with only 33% of glitches because the encoding does not produce any glitches.

The 4-2 counter tree is sometimes more performant for small operand sizes but when the multiplier size grows, Wallace trees are the better with up to 10% gain. The other schemes may consumme up to 90% glitches and even higher for big multipliers. For instance, a Braun multiplier, which is a good area solution, proves to be disastrous for a power dissipation point of view. As a consequence, area and power criteria for arithmetic are actually distinct.

The increasing fanout of the Skansky adder penalizes the global performance for big operand sizes. The critical path is increasing of about 10% only for a few fanouts. The advantage of the Kogge & Stone adder is not significant (about 2%). However, the power dissipation increases of about 10%. This drives us to think that the Han & carlson adder is the best solution.

From those figures, the Add/Shift algorithm followed by a Wallace tree and Han & carlson adder appears to be the best tradeoff because the power dissipation is quite low with a near optimal performance.

7 Conclusion

In this paper, we characterized classical multiplier architectures in order to rule out the less performant and most dissipative ones. The remaining schemes were estimated such that tradeoffs can be determined. Work remains to be done to validate the experimantal

results after place and route. Wire and path lenghtes can vary from logic synthesis estimate and should provide slightly different conclusions.

8 References

[1] S.Oberman, M.Flynn. *Design Issues in Floating Point Division*, Technical Report: CSL-TR-94-647 Stanford University.

[2] I.Koren. *Computer Arithmetic Algorithms*, Prentice Hall, 1993, ISBN 0-13-151952-2.

[3] R.Burch. *Pattern-Independent Current Estimation for Reliability Analysis of CMOS Circuits*, In Proc. of the Design Automation Conference, pp. 294-299, 1988.

[4] F.Najm. *Feedback, Correlation and Delay Concerns in the Power Estimation of VLSI Circuits*, In Proc. of the Design Automation Conference, pp. 612-616, 1995.

[5] F.Najm. *Transition Density, a Stochastic measure of Activity in Digital Circuits*, In Proc. of the Design Automation Conference, pp. 644-649, june 1991.

[6] B.Kapoor. *Improving the Accuracy of Circuit Activity Measurement*, In Proc. of the 31^{st} Design Automation Conference, pp. 734-739, june 1994.

[7] H.Mehta et al. *Accurate Estimation of Combinational Circuit Activity*, In Proc. of the Design Automation Conference, pp. 618-622, 1995.

[8] B.Laurent, G.Saucier. *A Low Power Mapping Optimizing Global Power.*, In Proc. of Synthesis and System Integration of Mixed Technologies (SASIMI), pp. 40-46, nov. 1996.

[9] N.Weste, K.Eshraghian. *Principles of CMOS VLSI Design, A Systems Perspective.*, Addison-Wesley Publishing Company, 1985.

[10] O.L.McSorley. *High Speed Arithmetic in Binary Computers.*, In Proc. of the IRE, 49(1), pp. 67-91, jan 1961.

[11] G.Bewick, M.Flynn. *Binary Multiplication Using Partially Redundant Multiples*, Technical Report: CSL-TR-92-528 Stanford University.

[12] J.L.Hennessy, D.A.Patterson. *Computer Architecture, A Quantative Approach*, Morgan-Kaufmann, pp. A44-A48, 1990.

[13] D.Zuras, W.McAllister. *Balanced Delay trees and Combinational Division in VLSI*, IEEE Journal of Solid-State Circuits, vol SC-21, No-5, pp. 814-819, oct. 1986.

[14] Z.Mou, F.Jutand. *A Class of Close to Optimum Adder Trees Allowing Regular and Compact Layout*, IEEE Trans. on Computers, pp. 251-254, 1990.

[15] L.Dadda. *On Parallel Digital Multipliers*, Alta Frequenza 45 (1976), 574-580.

[16] G.Bosco, M.Belrhiti, B.Laurent. *Adder Solution Space Exploration for Speed/Area Trade-offs*, In Proc. of Int'l Workshop on Logic and Architecture Synthesis, pp. 293-300.

[17] A. Guyot, M. Belrhiti, G. Bosco, *Adders synthesis*, IFIP Workshop on logic and architecture synthesis, pp. 107-117, December 1994.

[18] J.Sklansky, *Conditional-Sum Addition Logic*, IRE Transactions on Electronic Computers, Vol. EC-9, No.2, pp. 226-231, June 1960.

[19] P.M.Kogge and H.S.Stone, *A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations*, IEEE Transactions on Computers, Vol. C-22, No.8, pp. 786-792, August 1973.

[20] T.Han and D.A.Carlson, *fast Area-Efficient VLSI Adders*, 8^{th} Symposium on Computer Arithmetic, pp. 49-56, May 1987.