A Fast Parallel Squarer Based on Divide-and-Conquer

Jae-tack Yoo, Kent F. Smith, Ganesh Gopalakrishnan

UUCS-95-011

Department of Computer Science
MEB 3190, University of Utah
Salt Lake City, UT. 84112

August 4, 1995

Abstract

Fast and small squarers are needed in many applications such as image compression. A new family of high performance parallel squarers based on the divide-and-conquer method is reported. Our main result was realizing the basis cases of the divide-and-conquer recursion by using optimized n-bit primitive squarers, where n *is in the range of 2 to 6. This method reduced the gate count and provided shorter critical paths. A chip implementing an 8-bit squarer was designed, fabricated and successfully tested, resulting in 24 MOPS using a 2-μ CMOS fabrication technology. This squarer had two additional features: increased number of squaring operations per unit circuit area, and the potential for reduced power consumption per squaring operation.*

1 Introduction

The need to square numbers arises in a large number of image processing algorithms. For example, in many subband vector quantization systems (*e.g.* [1]), the L_2-norm calculations in the vector quantizer can involve the order of 288 million squaring operations per second (MOPS) to process HDTV image data. Usually these L_2-norm calculations need to be done only on short words: *e.g.* 8-, 12-, or 16-bits wide. Such short word sizes may allow parallel implementations of squarers with $O(n^2)$ area requirements, where n represents the number of bits of an input data.

Dadda designed bit-serial parallel squarers [2] based on his conceptually parallel scheme shown in Figure 1. Figure 1 (b) illustrates that the required number of bit-products are only half when reduced from a CSA array shown in Figure 1 (a). Dadda implemented bit-serial version of the scheme for general purpose squarers since area requirement grows fast with $O(n^2)$. In contrast to Dadda's design, the recent trend in the design of this class of circuits closely related to squarers - namely multipliers - is to move away from bit-serial operations, and instead use parallel schemes such as mutibit recording to improve computation speed. Sam *et al.* [3] showed that recoding over a larger number of bits resulted in better computation speed, and suggested that 5-bit recoding is desirable since it strikes a good compromise between computation speed and circuit complexity. In 5-bit recoding, five rows of partial products are added at a time, thus reducing the number of clock cycles to complete an operation to one-fifth in comparison with its bit-serial version. Meanwhile, what we consider to be short words, *i.e.* 8-bits, are only marginally above the 5-bit limit suggested by Sam *et. al.* Thus, implementing direct parallel squarers with short input words is not unreasonable. In addition, the fact that the partial product array for a CSA squarer can be folded also helps reduce the area.

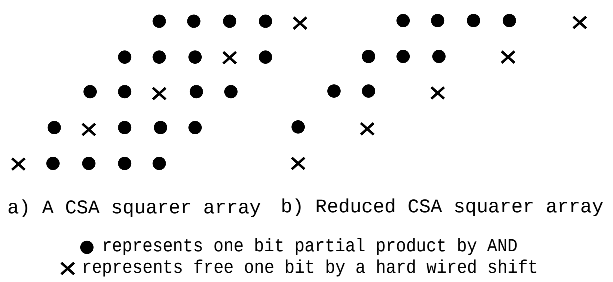

Figure 1: CSA squarers

The main idea presented in this paper is that when the parallel scheme is directly implemented, its CSA array can be reduced further by employing a *divide-and-conquer technique.* A new set of parallel CSA-based squarers based on divide-and-conquer have resulted in less number of bit products than in Dadda's conceptually parallel scheme. This squarer needs fewer gates and has shorter critical paths than in previously reported squarers.

2 Divide-and-Conquer Squarer

A parallel squarer is simpler than a parallel multiplier owing to the fact that in squaring $A = a_{n-1} \ldots a_0$, terms such as $a_i \times a_j + a_j \times a_i$ arise, which can be reduced to $2 \times (a_i \times a_j)$ which is $(a_i \times a_j)$ shifted left, as opposed to terms such as $a_i \times b_j + a_j \times b_i$ that arise in multiplying A by $B = b_{n-1} \ldots b_0$ which cannot be reduced. This saving is illustrated in Figure 1. We now show that a careful implementation of the circuits, designed by a divide-and-conquer scheme, can result in additional circuit area-reductions and delay-reductions.

In the following, we denote subvectors of A by A_{m-n} for various m and n, and the individual bits of A by a_i, for various i. All of our discussions consider the problem of squaring an 8-bit vector A. The first step of the divide-and-conquer recursion equation is:

$$A_{7-0}^2 = (A_{7-4} + A_{3-0})^2 = A_{7-4}^2 + A_{3-0}^2 + 2(A_{7-4} \times A_{3-0}) \quad \ldots\ldots\ldots (1)$$

In equation (1), we assume that A_{7-4} is a vector of bits 7 through 4 of A shifted four positions to the left. (As will be clear from the following, such scalings by powers of two do not change our algorithm—their only effect is to decide where A_{7-4} gets added in.) Equation (1) can be further refined into the following:

$$A_{7-0}^2 = A_{7-6}^2 + A_{5-4}^2 + A_{3-2}^2 + A_{1-0}^2 + 2(A_{7-6} \times A_{5-4} + A_{3-2} \times A_{1-0} + A_{7-4} \times A_{3-0}) \quad \ldots (2)$$

$$\begin{aligned} A_{7-0}^2 = {} & a_7^2 + a_6^2 + a_5^2 + a_4^2 + a_3^2 + a_2^2 + a_1^2 + a_0^2 + 2(a_7 \times a_6 + a_5 \times a_4 + a_3 \times a_2 + a_1 \times a_0) \\ & + 2(A_{7-6} \times A_{5-4} + A_{3-2} \times A_{1-0} + A_{7-4} \times A_{3-0}) \quad \ldots\ldots\ldots (3) \end{aligned}$$

All the equations have different base cases: 1-bit squaring to 4-bit. Figure 2 shows all three possible designs. The question now is which of these equations represents a better hardware implementation. A more general question is what grain size for the implementation to pick for these equations.

The following summarizes our observations:

- Taking equation (2) as the basis for implementation, we need four two-bit squarer primitives, two two-bit multipliers, and one four-bit multiplier, as illustrated in Figure 2 (b). A two-bit squarer converts 0, 1, 2, or 3 to 0, 1, 4, or 9, respectively. Figure 3 (a) shows how to directly achieve this effect, while Figure 3 (b) shows what would result if the two-bit squaring were to be carried further down, to involve one-bit squarings and additions. Clearly, stopping the divide-and-conquer recursion at the two-bit size seems to pay off. This circuit shown in Figure 2 (b) involves 32 and-gates, 19 full-adders, and 8 half-adders, and involves a critical path of 7 full-adder and 5 half-adder delays.

- Taking equation (1) as the basis for implementation results in the circuit of Figure 2 (a). This involves 30 and-gates, 4 exor-gates, 14 full-adders, and 8 half-adders, and involves a critical path of 6 full-adder and 4 half-adder delays. This shows an improvement over the case of using equation (2).

- Taking equation (3) results a direct realization of an 8-bit squarer: Figure 2 (c) uses the same scheme as in Figure 1 (b). This would result in 28 and-gates, 21 full-adders, and 11 half-adders, and have 11 full-adder delays.

The above analysis shows that the optimum primitive squarer size is four bits for 8-bit squaring. Another possible primitive squarer is a 3-bit squarer, which provides base case computation for 12-bit squaring (which can be divided into two 6-bit squaring, one of which can be divided into two 3-bit squaring.) A 3-bit primitive squarer is as shown in Figure 3 (c) which has fewer gates than the parallel version of Dadda's scheme as shown in Figure 3 (d). An optimized 4-bit primitive squarer was designed similarly by directly optimizing its logic. Theoretical limits of this approach depends on the basic divide-and-conquer equation as

$$(number)^2 = (MSB\_number)^2 + (LSB\_number)^2 + 2(MSB\_number \times LSB\_number).$$

The last term of this equation shows that the complexity of a squarer is bounded from below by the size of a half-sized multiplier. Following section discusses implementation issues.

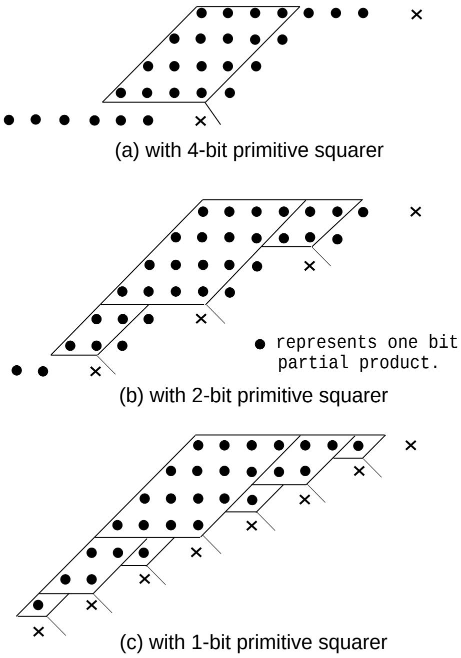

Figure 2: 8-bit squarers

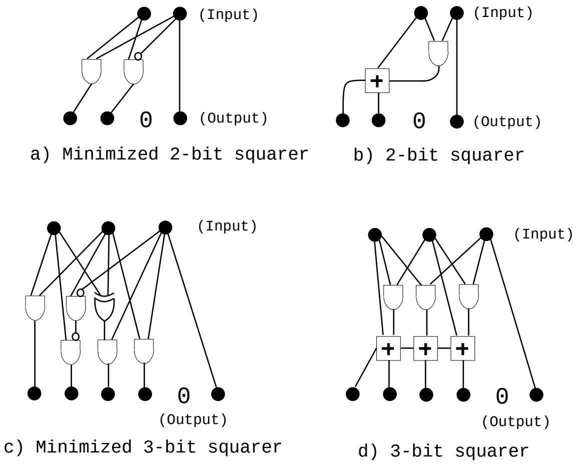

Figure 3: Primitive squarers

3 Experiments and Additional Features

Figure 4 shows the parallel design of an 8-bit divide-and-conquer squarer based on equation (2) of previous section. This design employs a carry save addition for the first stage and a ripple-carry addition for the second stage (similar to that done in [4] which favored the addition with short input words for their fast multipliers). This achieves one squaring operation per clock cycle. The design was implemented in 2-μ CMOS using a design system known as ACME [5]. Figure 4 also shows the regularity (higher replication factor) in primitive squarer placement as shown in the top row of bit-products. This leads us to believe that all members of the divide-and-conquer family of squarers will have regular layouts. Its implementation including input/output pads is shown in Figure 5. This squarer chip was tested and operates at the maximum speed of 24 MHz (MOPS) and occupies area of $1.4mm \times 0.8mm$ (excluding extra latches for testability) which is 37% of a MOSIS "tiny" chip. Using this squarer, we have also built and tested a "difference-square-and-accumulation" unit, the function of which arises during L_2 norm calculation. This unit operated at 23 MOPS in 2-μ CMOS technology.

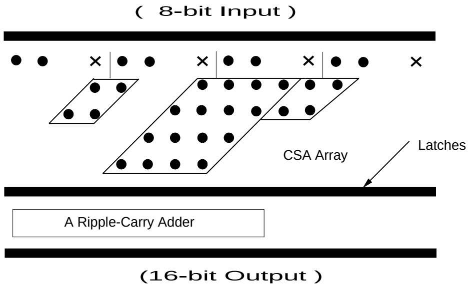

Figure 4: Design of an 8-bit squarer

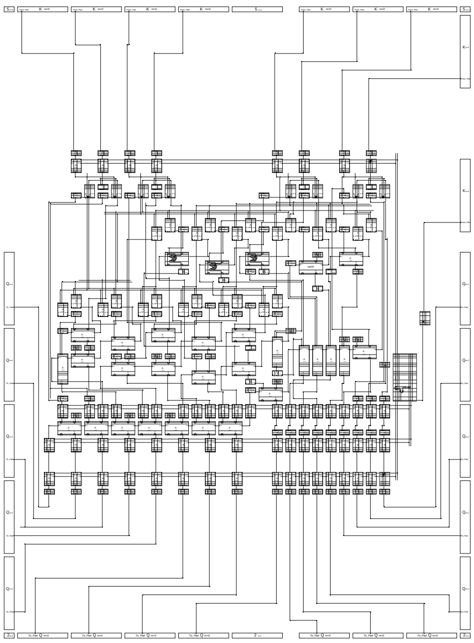

Figure 5: Implementation of the 8-bit squarer in a tiny chip frame

All squarers will, theoretically, have the same order of performance (*Normalized-OPS to the circuit area*): $O(n^2)$ area using $O(1)$ time for ours versus $O(n)$ area using $O(n)$ time for Dadda's bit-serial design. But there is a significant difference in the constant factor depending on its implementation. Figure 6 shows a design of Dadda's [2] (a bit-serial version) with 8-bit positive integer inputs (the same input-size used in our parallel squarer design in Figure 4). Figure 6 shows (1) that the area for latches and registers is relatively large in comparison to its area for partial product generation and its addition and (2) that eight iterations are needed to complete an operation. If, to make a comparison with the parallel version which does an operation per clock cycle, we unroll the iteration structure of the bit-serial version to achieve an operation per clock cycle, the resulting structure will still contain many latches and registers that take up a considerable percentage of the overall area. In contrast to this, the parallel version includes only three latch rows in addition to the CSA adder area, which is more compact than the above unrolled structure. Meanwhile, the cycle times of both versions will be about the same since the critical path in the bit-serial one is in the adders and that of the parallel one is in the ripple-carry adder, both of which are about the same size and result in about the same speed. These area and cycle time arguments conclude that the normalized-OPS to the circuit area of a parallel version with short input words is higher than that of its serial version. In addition to this, the use of a smaller number of clock cycles per operation has become important with the move into subhalf-micrometer fabrication [6] and higher clock speed exposing clock skew problems.

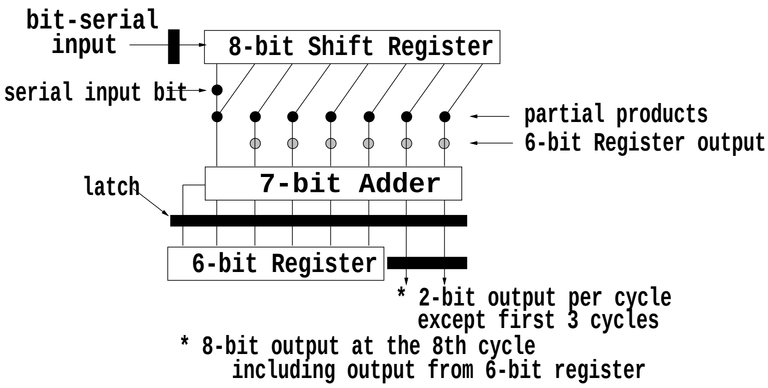

Figure 6: Serial design of Dadda's scheme

Power consumption per squaring operation can be approximated by *the number of transitions per operation* based on the transition density [7]. Assuming that the number of transitions due to arithmetic calculations are the same for both versions, the trade-offs between the two versions are additional transitions caused by increased latching (for the bit-serial version) versus extra transitions due to glitches occurring in the CSA array (for the parallel version). The bit-serial one will induce many transitions due to the eight latch and register activations. But the parallel one will generate glitches during the simultaneous firing of CSA adder activities at the first stage. Fortunately, the parallel one has five rows of partial products, resulting in three serial adder rows. When we use a tree topology, *e.g.* Wallace tree [8], the logic depth of adders will be as low as two, which will result in extremely low number of glitches per squaring operation. This transition density argument illustrates the potential for lower power consumption per squaring operation of our parallel version compared to a bit-serial version.

4 Conclusions

To build fast and effective squarers especially for short input words, a new divide-and-conquer technique is applied to Dadda's conceptually parallel scheme. An 8-bit parallel squarer implementing a new method based on divide-and-conquer was developed and demonstrated. The squarer chip fabricated in 2-μ CMOS operates at a maximum rate of 24 million OPS. The regularity in placing primitive squarers made VLSI layout easy.

This paper concludes that the new method is an effective approach for high performance squaring applications such as L_2 norm calculations in very high rate of image processing, etc. The merits of this method are: (1) reduction of bit-products from Dadda's parallel scheme, (2) reduced gate count and shorter critical path than its parallel version, and (3) potentially less power per squaring operation in comparison to its serial version.

We estimate that, although the area grows as $O(n^2)$, parallel implementations of squarers using divide and conquer may be cost-effective up to 16-bits with base cases of 3-bit, 4-bit or 6-bit primitive squarers.

References

[1] R. Jain, A. Madisetti, and R. L. Baker, "An integrated circuit design for pruned tree-search vector quantization encoding with an off-chip controller," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 2, No. 2, June 1992.

[2] L. Dadda, "Squares for binary numbers in serial from," *IEEE 11th Sym. on Computer Arithmetic*, 1985.

[3] H. Sam and A. Gupta, "A generalized multibit recoding of two's complement binary numbers and its proof with application in multiplier implementations," *IEEE Trans. on Computer*, Vol. 39, No. 8, August 1990.

[4] A. Habibi and P. A. Wintz, "Fast multipliers," *IEEE Trans. on Computers*, Vol. C-19, February 1970.

[5] T. M. Carter, K. F. Smith, S. R. Jacobs, and R. M. Neff, "Cell matrix methodologies for integrated circuit design," *Integration, The VLSI Journal*, July, 1989.

[6] G. Tiwary, "Below the half-micrometer mark," *IEEE Spectrum*, November, 1994

[7] F. Najm, "Transition density, a stochastic measure of activities in digital circuits," DAC, 1991

[8] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, Feb. 1964.