Combined Multiplication and Sum-of-Squares Units

Michael J. Schulte†, Louis Marquette‡, Shankar Krithivasan†,
E. George Walters III‡, and John Glossner↑

†Dept. of ECE, University of Wisconsin-Madison, Madison, WI
‡CSE Dept., Lehigh University, Bethlehem, PA
↑ Sandbridge Technologies, White Plains, NY

Abstract

Multiplication and squaring are important operations in digital signal processing and multimedia applications. This paper presents designs for units that implement either multiplication, $A \times B$, or sum-of-squares computations, $A^2 + B^2$, based on an input control signal. Compared to conventional parallel multipliers, these units have a modest increase in area and delay, but allow either multiplication or sum-of-squares computations to be performed. Combined multiplication and sum-of-squares units for unsigned and two's complement operands are presented, along with integrated designs that can operate on either unsigned or two's complement operands. The designs can also be extended to work with a third accumulator operand to compute either $Z + A \times B$ or $Z + A^2 + B^2$. Synthesis results indicate that a combined multiplication and sum-of-squares unit for 32-bit two's complement operands can be implemented with roughly 15% more area and nearly the same worst case delay as a conventional 32-bit two's complement multiplier.

1: Introduction

Sum-of-squares computations are found in many digital signal processing (DSP) and multimedia applications including adaptive filtering [1], image compression [2], Euclidean branch calculation [3], pattern recognition [4], equalization [5], vector normalization, and Viterbi coding [6]. Sum-of-squares computations typically take either the two operand form,

$$SS = A^2 + B^2 \ , \tag{1}$$

or the vector form,

$$SSV = \sum_{i=0}^{n-1} A_i^2 \ , \tag{2}$$

which is commonly used in mean-square-error calculations.

Due to the abundance of sum-of-squares calculations in DSP and multimedia applications, many processors including Texas Instruments' TMS320C2x and TMS320C55x, DSP Group's PineDSPCore and OakDSPCore, and Analog Device's ADSP-218x, provide square and/or square-and-accumulate instructions [7, 8, 9]. These processors use an existing multiplier or multiply-and-accumulate (MAC) unit to perform a single square operation, A^2,

or square-and-accumulate operation, $Z + A^2$. On these processors, which provide indirect memory addressing, square and square-and-accumulate instructions are useful because they require less memory bandwidth than multiply and multiply-and-accumulate instructions.

Previous research on squaring has mainly focussed on designs for dedicated squarers, which compute the square of an operand [1] - [6], [10] - [21]. This previous research demonstrates that dedicated parallel squarers can be implemented with roughly half as much hardware and less delay than parallel multipliers. Dedicated squarers work well for processor designs in which the relative frequency of squaring operations is known in advance. They are, however, less suitable for programmable digital signal processors and multimedia processors, in which the frequency of squaring operations is application dependent. Consequently, for applications that do not frequently perform squaring operations, the extra hardware needed for dedicated squarers goes unused.

This paper introduces designs for combined multiplication and sum-of-squares units (CMSSUs). The CMSSUs perform either a multiplication, $A \times B$, or a sum-of-squares computation, $A^2 + B^2$, based on an input control signal, s. The CMSSUs use conventional multiplier hardware, plus a modest amount of additional hardware. Thus, they require much less area than a dedicated squarer plus a multiplier. The CMSSUs have an additional advantage over dedicated squarers in that they allow two squares and their summation to be performed as a single operation, which has the potential to improve performance and power consumption.

The remainder of this paper is organized as follows. Sections 2 and 3 present designs for CMSSUs for unsigned and two's complement operands, respectively. Section 4 describes designs for integrated CMSSUs that operate on either unsigned or two's complement operands, based on a second input control signal, t. Section 5 provides area and delay estimates for the CMSSUs and compares them to estimates for conventional tree multipliers for various operand sizes. Section 6 gives conclusions. The designs presented in this paper assume the input operands are n-bit integer operands, but they can easily be extended to other types of fixed-point operands.

2: Unsigned CMSSUs

Two n-bit unsigned binary integers, $A = a_{n-1}a_{n-2}\ldots a_1a_0$, and $B = b_{n-1}b_{n-2}\ldots b_1b_0$, have the values

$$A = \sum_{i=0}^{n-1} a_i 2^i \quad \text{and} \quad B = \sum_{j=0}^{n-1} b_j 2^j \ . \tag{3}$$

Their product, $A \cdot B$, has the value

$$A \cdot B = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j 2^{i+j} \ . \tag{4}$$

In developing the designs for the CMSSUs, it is useful to rewrite (4) as

$$A \cdot B = \sum_{i=0}^{n-1} a_i b_i 2^{2i} + \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} (a_i b_j + a_j b_i) 2^{i+j} \ . \tag{5}$$

Figure 1 shows the partial product matrix for $A \cdot B$ with $n = 8$. The first summation in (5) corresponds to partial product bits on the anti-diagonal, which has been marked

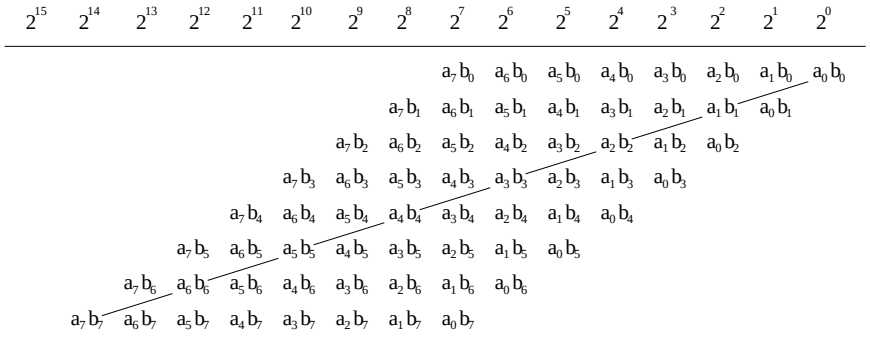


Figure 1. 8-bit Unsigned Multiplication Matrix.

in Figure 1. The paired summations correspond to the remaining bits in Figure 1. For each partial product bit a_ib_j above the anti-diagonal, there is a partial product bit a_jb_i below the anti-diagonal. This observation is useful in designing the CMSSUs. After the partial product bits are generated, they are reduced to sum and carry vectors using a tree of parallel counters [22]. The sum and carry vectors are then added using a high-speed carry-propagate adder to give the product P. The primary difference between CMSSUs and conventional multipliers is in how the partial products are generated.

When computing A^2, A replaces B in (5). Using the identities $a_ia_i = a_i$ and $a_ia_j + a_ja_i = 2a_ia_j$ gives

$$A^2 = \sum_{i=0}^{n-1} a_i 2^{2i} + 2\left(\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} a_i a_j 2^{i+j}\right) \quad . \tag{6}$$

Equation (6) is well known and has been presented previously in [3, 10, 17]. The first summation in (6) corresponds to the anti-diagonal of the squaring matrix, and is shown as the top row of the matrix in Figure 2a. The second summation corresponds to the partial product bits above the anti-diagonal of the squaring matrix shifted left one position, and is shown as the remaining rows in Figure 2a. Similarly,

$$B^2 = \sum_{i=0}^{n-1} b_i 2^{2i} + 2\left(\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} b_i b_j 2^{i+j}\right) \quad , \tag{7}$$

and is shown in Figure 2b.

Adding (6) and (7), and then rearranging terms gives

$$A^2 + B^2 = \sum_{i=0}^{n-1} (a_i + b_i) 2^{2i} + 2\left(\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} (a_i a_j + b_j b_i) 2^{i+j}\right) \quad . \tag{8}$$

The first summation in (8) is replaced by using the identity

$$\sum_{i=0}^{n-1} (a_i + b_i) 2^{2i} = \sum_{i=0}^{n-1} (a_i \oplus b_i) 2^{2i} + 2\left(\sum_{i=0}^{n-1} a_i b_i 2^{2i}\right) \quad , \tag{9}$$

which effectively replaces $a_i + b_i$ by a sum bit, $a_i \oplus b_i$, in the same column, and a carry-bit, a_ib_i, in the next column to the left. Substituting (9) into (8) and rearranging terms gives

$$A^2 + B^2 = 2\left(\sum_{i=0}^{n-1} a_i b_i 2^{2i} + \sum_{i=1}^{n-1}\sum_{j=0}^{i-1} (a_i a_j + b_j b_i) 2^{i+j} + \sum_{i=0}^{n-1} a_i \oplus b_i 2^{2i-1}\right) \quad . \tag{10}$$

The matrix for $A^2 + B^2$ is shown in Figure 2c, where the bottom row in the matrix corresponds to the final summation in (10).

Comparing Equations (5) and (10), and Figures 1 and 2c reveals minor differences between the multiplication and sum-of-squares matrices. Specifically,

- The sum-of-squares matrix has an additional row with n partial product bits $a_i \oplus b_i$, for $0 \leq i \leq n-1$.
- The sum-of-squares matrix is shifted left by one bit position relative to the multiplication matrix.
- Each partial product bit $a_i b_j$ above the anti-diagonal in the multiplication matrix is replaced by $a_i a_j$ in the sum-of-squares matrix.
- Each partial product bit $a_j b_i$ below the anti-diagonal in the multiplication matrix is replaced by $b_j b_i$ in the sum-of-squares matrix.

To unify the partial product matrices for multiplication and sum-of-squares computations, we define an input control signal s, which is set to '1' for sum-of-squares computations and '0' for multiplication. We then use s to define the following variables:

$$c_j = a_j s \vee b_j \overline{s} \quad (\text{for } 0 \leq j \leq n-1) \tag{11}$$

$$d_j = a_j \overline{s} \vee b_j s \quad (\text{for } 0 \leq j \leq n-1) \tag{12}$$

$$e_i = s(a_i \oplus b_i) \quad (\text{for } 0 \leq i \leq n-1)\ , \tag{13}$$

where c_j is used above the anti-diagonal, d_i is used below the anti-diagonal, and e_i is used in the final row of the partial product matrix.

Since the squaring matrix is shifted left by one position relative to the multiplication matrix, the partial product matrix in the CMSSU actually computes $(A^2 + B^2)/2$ when $s = 1$. Using c_j, d_i, and e_i to unify (5) and (10) results in the following combined equation for unsigned multiplication or sum-of-squares computations:

$$P = \sum_{i=0}^{n-1} a_i b_i 2^{2i} + \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} (a_i c_j + d_j b_i) 2^{i+j} + \sum_{i=0}^{n-1} e_i 2^{2i-1}\ . \tag{14}$$

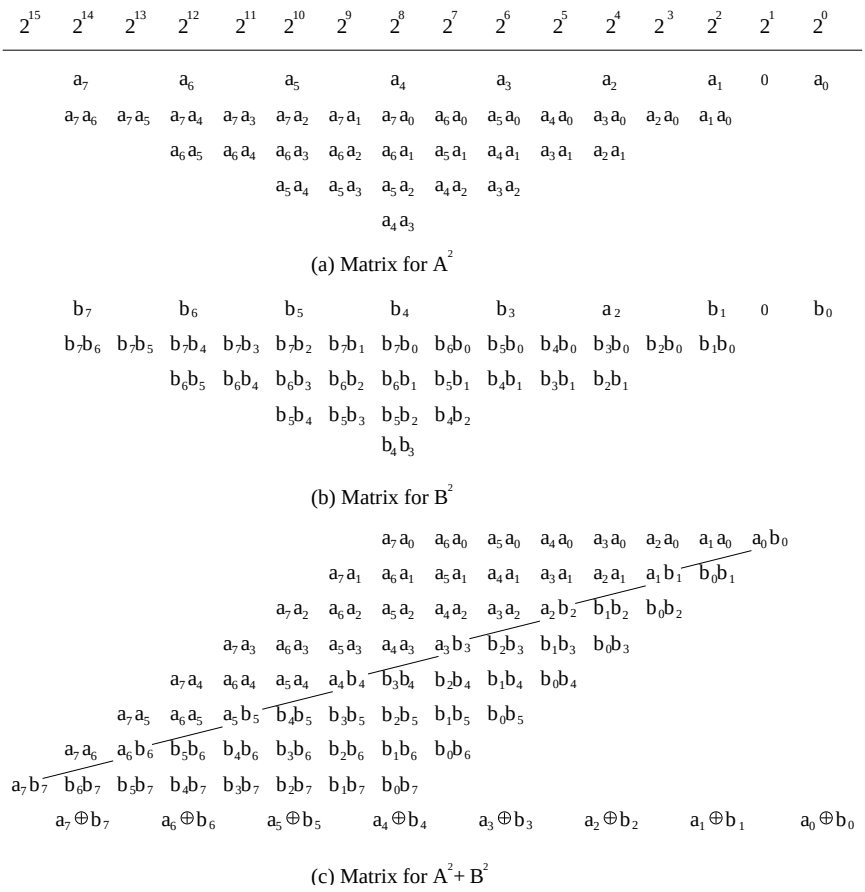
The unsigned CMSSU matrix is shown in Figure 3 for $n = 8$.

Since for unsigned operands, $0 \leq A^2 + B^2 \leq 2(2^{2n} - 2^{n+1} + 1)$, P requires $2n + 1$ bits, $p_{2n-1} p_{2n-2} \ldots p_0 p_{-1}$. When $s = 0$, $A \cdot B$ is available from $p_{2n-1} p_{2n-2} \ldots p_1 p_0$. Similarly, when $s = 1$, $A^2 + B^2$ is available from $p_{2n-2} \ldots p_0 p_{-1}$, and $p_{2n-1} = 1$ if $A^2 + B^2$ overflows the $2n$-bit result.

Compared to an n-bit unsigned tree multiplier, an n-bit unsigned CMSSU requires an additional:

- $2n$ 1-bit multiplexors (to compute $c_j = a_j s \vee b_j \overline{s}$ and $d_j = a_j \overline{s} \vee b_j s$)
- n 2-input XOR gates (to compute $a_i \oplus b_i$)
- n 2-input AND gates (to compute $e_i = s(a_i \oplus b_i)$
- $\approx n$ full adders (to add in e_i)
- one $2n$-bit multiplexor (to select $p_{2n-1} \ldots p_0$ or $p_{2n-2} \ldots p_{-1}$ based on s)

For practical designs, additional buffers are also needed to handle the large fan-out of s. Assuming that the delay of an XOR gate and a multiplexor are approximately equal, the



(a) Matrix for A^2

(b) Matrix for B^2

(c) Matrix for $A^2 + B^2$

Figure 2. 8-bit Unsigned Squaring Matrices.

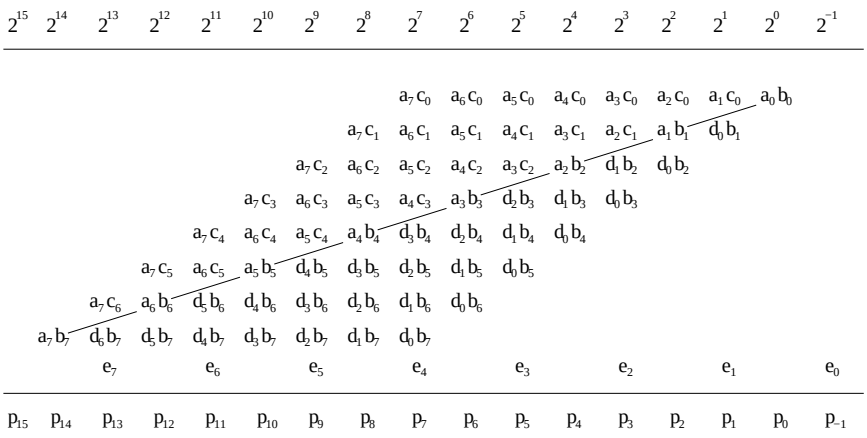


Figure 3. 8-bit Unsigned CMSSU Matrix.

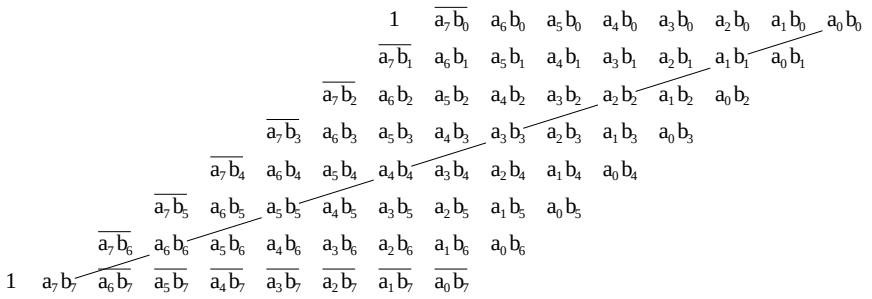


Figure 4. 8-bit Two's Complement Multiplication Matrix.

worst case delay path increases by approximately two multiplexor delays plus the buffering delay. Since e_i is generated in parallel with the computation of c_j and d_j and the partial product generation, generating and adding e_i does not increase the worst case delay of the multiplier, for most operand sizes. For certain operand sizes (e.g, 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, ...) the addition of the e_i term may cause an extra delay, which is equal to the delay of one full adder.

3: Two's Complement CMSSU

The methodology presented in the previous section for unsigned CMSSUs can be extended to designs for two's complement CMSSUs. Two n-bit two's complement integers, $A = a_{n-1}a_{n-2}\ldots a_1a_0$ and $B = b_{n-1}b_{n-2}\ldots b_1b_0$, have the values

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \quad \text{and} \quad B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \ . \tag{15}$$

Using the technique for two's complement multiplication presented in [23], their product has the value

$$A \cdot B = 2^n + 2^{2n-1} + a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}(\overline{a_i b_{n-1}} + \overline{a_{n-1}b_i})2^{n+i-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} \ , \tag{16}$$

which can be rewritten as

$$A \cdot B = 2^n + 2^{2n-1} + \sum_{i=0}^{n-2}(\overline{a_i b_{n-1}} + \overline{a_{n-1}b_i})2^{n+i-1} + \sum_{i=0}^{n-1} a_i b_i 2^{2i} + \sum_{i=1}^{n-2}\sum_{j=0}^{i-1}(a_i b_j + a_j b_i)2^{i+j} \ . \tag{17}$$

The two's complement matrix $A \cdot B$ obtained using the above formula is shown in Figure 4 for $n = 8$. This matrix is similar to the matrix for unsigned operands, shown in Figure 1, except that $2n-2$ partial product bits are inverted and '1's are added in columns n and $2n-1$. The '1' in column n can be added using a specialized half adder and the '1' in column $2n-1$ can be added by simply inverting the most significant product bit [23].

When computing A^2, A replaces B in (17). Using the identities $a_i a_i = a_i$ and $a_i a_j + a_j a_i = 2a_i a_j$ gives

$$A^2 = 2^n + 2^{2n-1} + 2\left(\sum_{i=0}^{n-2} \overline{a_{n-1}a_i} 2^{n+i-1}\right) + \sum_{i=0}^{n-1} a_i 2^{2i} + 2\left(\sum_{i=1}^{n-2}\sum_{j=0}^{i-1} a_i a_j 2^{i+j}\right) \ . \tag{18}$$

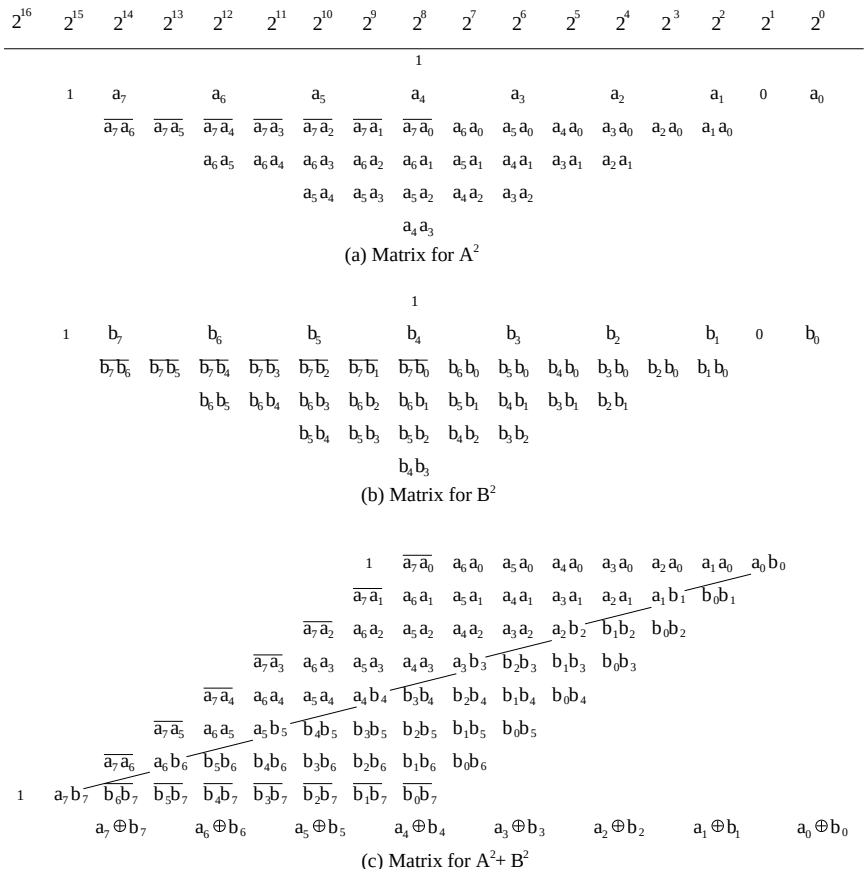


Figure 5. 8-bit Two's Complement Squaring Matrices.

The matrix derived from (18) for A^2 is shown in Figure 5a for $n = 8$. Figure 5b shows a similar matrix for B^2.

Adding the expressions for A^2 and B^2, and using the identity given in (9) yields

$$\begin{aligned} A^2 + B^2 &= 2(2^n + 2^{2n-1} + \sum_{i=0}^{n-2}(\overline{a_i a_{n-1}} + \overline{b_{n-1} b_i})2^{n+i-1} \\ &+ \sum_{i=0}^{n-1} a_i b_i 2^{2i} + \sum_{i=1}^{n-2}\sum_{j=0}^{i-1}(a_j a_i + b_j b_i)2^{i+j} + \sum_{i=0}^{n-1} a_i \oplus b_i 2^{2i-1}) \ . \end{aligned} \tag{19}$$

The two's complement matrix for $A^2 + B^2$ is shown in Figure 5c for $n = 8$. As was done for the unsigned case, the variables c_j, d_j, and e_i are used to unify the two partial product matrices and their associated equations. For two's complement operands, the combined equation is

$$\begin{aligned} P &= 2^n + 2^{2n-1} + \sum_{j=0}^{n-2}(\overline{a_{n-1} c_j} + \overline{d_j b_{n-1}})2^{n+j-1} \\ &+ \sum_{i=0}^{n-1} a_i b_i 2^{2i} + \sum_{i=1}^{n-2}\sum_{j=0}^{i-1}(a_i c_j + d_j b_i)2^{i+j} + \sum_{i=0}^{n-1} e_i 2^{2i-1} \ . \end{aligned} \tag{20}$$

The partial product matrix for the two's complement CMSSU is shown in Figure 6 for $n = 8$. When $s = 0$, $A \cdot B$ is available from $p_{2n-1}p_{2n-2}\ldots p_1 p_0$. Similarly, when $s = 1$, $A^2 + B^2$

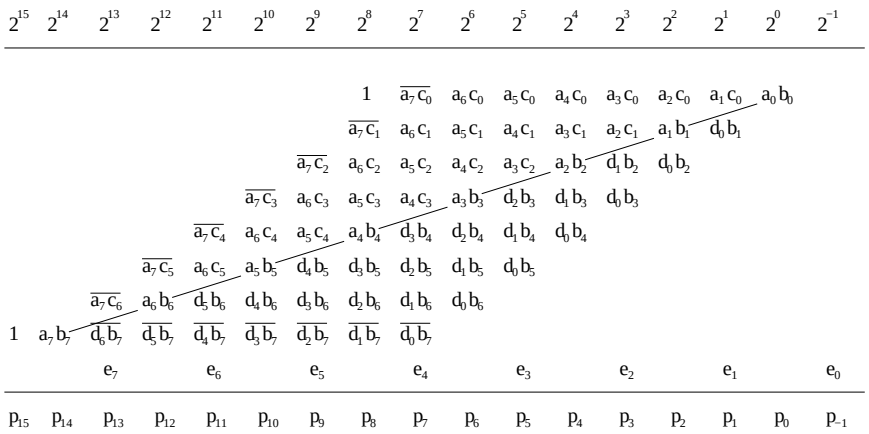


Figure 6. 8-bit Two's Complement CMSSU Matrix.

is available from $p_{2n-2} \ldots p_0 p_{-1}$. For two's complement operands, $0 \leq A^2 + B^2 \leq 2^{2n-2}$. Overflow can only occur in the $2n$-bit two's complement result if both A and B are -2^{n-1}. Since the result must be positive and $A^2 + B^2 < 2^{2n-2}$ for all other cases, $p_{2n-2} = 1$ only when this situation occurs and can be used to indicate overflow. The additional hardware and delay needed to implement the two's complement CMSSU, instead of a conventional two's complement tree multiplier, is the same as for the unsigned case.

4: Integrated Unsigned and Two's Complement CMSSUs

Similarities in the partial product matrices for the unsigned and two's complement CMSSUs allow designs for integrated CMSSUs (ICMSSUs) to be developed that handle either unsigned or two's complement operands. The operand type is specified by a second input control signal, t, where $t = 0$ for unsigned operands and $t = 1$ for two's complement operands. This technique is similar to the technique presented in [17] to develop designs for combined unsigned and two's complement squarers.

Since $2n - 2$ of the partial product bits are inverted for two's complement operands and not inverted for unsigned operands, we define the variable $\widehat{p} = p \oplus t$, and use it to represent partial product bits that are inverted when $t = 1$ and not inverted when $t = 0$. Using this notation and taking advantage of similarities in (14) and (20) gives the following equation for the ICMSSU:

$$\begin{aligned} P &= t \cdot 2^n + t \cdot 2^{2n-1} + \sum_{j=0}^{n-2} (a_{n-1}\widehat{c_j} + d_j\widehat{b_{n-1}}) 2^{n+j-1} \\ &+ \sum_{i=0}^{n-1} a_i b_i 2^{2i} + \sum_{i=1}^{n-2} \sum_{j=0}^{i-1} (a_i c_j + d_j b_i) 2^{i+j} + \sum_{i=0}^{n-1} e_i 2^{2i-1} \quad . \end{aligned} \tag{21}$$

Figure 7 shows the ICMSSU partial product matrix for $n = 8$. In addition to conditionally inverting $2n - 2$ partial product bits and the most significant product bit, p_{2n-1}, when $t = 1$ (to add the '1' in column $2n - 1$), t is also added in column n. Compared to the unsigned CMSSU in Figure 3, the ICMSSU requires $2n - 1$ additional XOR gates and an additional full adder. The delay increase is roughly equivalent to two XOR gate delays. For practical designs, the t signal should be buffered to avoid large fan-out.

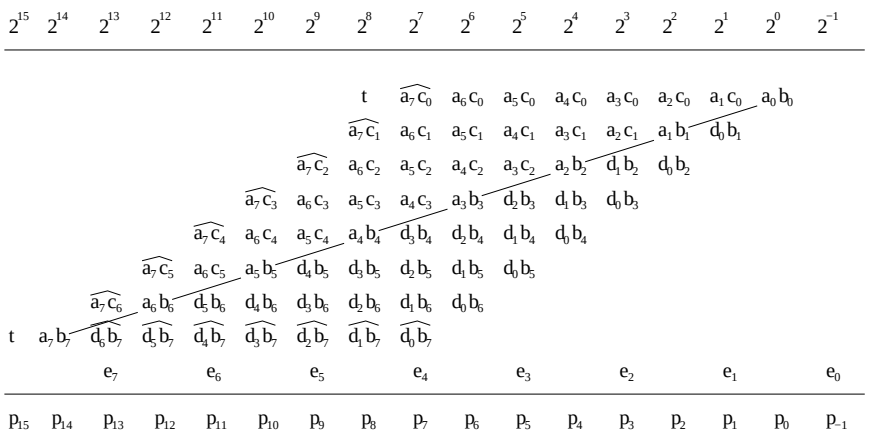


Figure 7. 8-bit Integrated Unsigned and Two's Complement CMSSU Matrix.

5: Area and Delay Estimates

To obtain area and delay estimates, a specialized Java program was written that generates structural VHDL models for CMSSUs and conventional tree multipliers, given an operand size, n, and operand type; unsigned, two's complement, or integrated. The CMSSUs and multipliers use the Reduced Area Multiplier technique to reduce the partial products [23] and a high-speed carry-lookahead adder for the final addition. The Java program was used to generate VHDL models for CMSSUs and multipliers with operand sizes of 8, 16, 24, and 32 bits. After simulating the designs extensively, they were synthesized using Mentor Graphics' LeonardoSpectrum synthesis tool and the TSMC 0.25 micron CMOS standard cell library. The designs were optimized for area with a maximum fan-out of four. The simulation results use an operating voltage of 2.5 Volts and a temperature of 25 degrees centigrade. Optimizing for area, instead of delay, reduces the changes made by the synthesis tool to the structure of the original VHDL code and is useful when comparing the relative benefits of different designs.

Table 1 shows area estimates for unsigned, two's complement, and integrated multipliers and CMSSUs for operand sizes of 16, 24, and 32 bits. To facilitate comparisons, the percent increase in area between the multipliers and CMSSUs for a given operand size are also shown. Based on these estimates, the CMSSUs have between 14.5% and 23.5% more area than the multipliers. As the operand size, n, increases, the relative difference in area between the multipliers and CMSSUs decreases. This occurs because the area of both types of units increases as $O(n^2)$, but the difference in area between the units increases as $O(n)$. Compared to the unsigned CMSSUs, the integrated CMSSUs have between 1.7% and 3.5% more area. This small increase in area of the integrated CMSSU is due to its additional $2n-1$ XOR gates, one full adder, and buffers for the t signal, as explained in Section 4.

Table 2 shows delay estimates for each of the multipliers and CMSSUs. Compared to the multipliers, the CMSSUs have between -0.1% and 8.5% more delay. The increase in delay for the CMSSUs is mainly due to their two extra multiplexor delays and their extra delay from buffering s. The theoretical difference in delay between the CMSSUs and multipliers is constant with respect to operand size, and the overall delays of the units vary logarithmically with operand size. Thus, the relative difference in delay between the CMSSUs and the multipliers decreases as the operand size increases. For $n = 32$ the relative difference in delay between the CMSSUs and the multipliers is negligible. The

32-bit unsigned CMSSU has slightly less delay than the 32-bit unsigned multiplier due to the ability of the synthesis tool to make tradeoffs between area and delay. Based on a theoretical analysis of the worst case delay paths, we expect the worst case delay of the 32-bit CMSSUs to be about 2% more than the worst case delay of the 32-bit multipliers. The integrated CMSSUs have between 1.0% and 2.0% more delay than the unsigned CMSSUs due to buffering the t signal and conditionally inverting some of the partial product bits.

Size (n)	Unit type	Area (equivalent gates)		
		Multiplier	CMSSU	% increase
16	unsigned	35634	43991	23.5%
16	signed	35893	44267	23.3%
16	integrated	37167	45518	22.5%
24	unsigned	83252	97588	17.2%
24	signed	83389	97899	17.4%
24	integrated	85401	99836	16.9%
32	unsigned	149958	172056	14.7%
32	signed	150204	172286	14.7%
32	integrated	152816	174960	14.5%

Table 1. Area Estimates for Multipliers and CMSSUs.

Size (n)	Unit type	Delay (nanoseconds)		
		Multiplier	CMSSU	% increase
16	unsigned	10.34	11.05	6.9%
16	signed	10.39	11.27	8.5%
16	integrated	10.40	11.27	8.4%
24	unsigned	14.10	14.33	1.6%
24	signed	14.18	14.48	2.1%
24	integrated	14.18	14.48	2.1%
32	unsigned	17.86	17.84	-0.1%
32	signed	17.93	18.01	0.4%
32	integrated	17.93	18.01	0.4%

Table 2. Delay Estimates for Multipliers and CMSSUs.

6: Conclusions

CMSSUs are useful in several digital signal processing and multimedia applications, since they allow two squares and their summation to be performed as a single operation, which has roughly the same delay as a multiplication. As the number of bits in the input operands increases, the percentage increase in delay and area between the standard tree multipliers and CMSSUs decreases. Hence, for large input sizes, the areas and delays of the CMSSUs are close to those of the conventional tree multipliers. Integrated CMSSU support both unsigned and two's complement operands with minor increases in area and delay compared to unsigned CMSSUs. By adding an extra row of adders to the partial product reduction tree, the CMSSUs can be extended to also perform $Z + A \times B$ or $Z + A^2 + B^2$.

References

[1] R. H. Strandberg, L. G. Bustamante, V. G. Oklobdzija, M. Soderstrand, and J. C. Le Duc, "Efficient Realizations of Squaring Circuit and Reciprocal used in Adaptive Sample Rate Notch Filters," *Journal of VLSI Signal Processing*, vol. 14, no. 3, pp. 303–309, 1996.

[2] J. T. Yoo, K. F. Smith, and G. Gopalakrishnan, "A Fast Parallel Squarer Based on Divide-and-Conquer," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 6, pp. 909–912, 1997.

[3] R. K. Kolagotla, W. R. Griesbach, and H. R. Srinivas, "VLSI Implementation of 350 MHz 0.35 Micron 8 Bit Merged Squarer," *Electronic Letters*, vol. 34, no. 1, pp. 47–48, 1998.

[4] J. Pihl and E. J. Aas, "A Multiplier and Squarer Generator for High Performance DSP Applications," in *Proceedings of the 39th Midwest Symposium on Circuits and Systems*, vol. 1, pp. 109–112, 1996.

[5] G. Kempa and P. Jung, "FPGA Based Logic Synthesis of Squarers Using VHDL," in *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping. Second International Workshop on Field Programmable Logic and Applications*, pp. 112–123, 1993.

[6] A. Eshraghi, T. S. Fiez, K. D. Winters, and T. R. Fischer, "Design of a New Squaring Function for the Viterbi Algorithm," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 4, pp. 1102–1107, 1994.

[7] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features*. IEEE Press, 1997.

[8] Anaolog Devices, *DSP-218x DSP Instruction Set Reference*, February 2001. Available from www.analog.com.

[9] Texas Instruments, *TMS320C55x DSP Mnemonic Instruction Set Reference Guide*, October 2002. Available from www.ti.com.

[10] T. C. Chen, "A Binary Multiplication Scheme Based on Squaring," *IEEE Transactions on Computers*, vol. C-20, pp. 678–680, 1971.

[11] T. JayaShree and D. Basu, "On Binary Multiplication Scheme Using the Quarter Square Algorithm," *IEEE Transactions on Computers*, vol. C-25, no. 9, pp. 957–960, 1976.

[12] E. L. Johnson, "A Digital Quarter Square Multiplier," *IEEE Transactions on Computers*, vol. C-29, pp. 258–261, March 1980.

[13] C.-L. Wey, "On Design of Efficient Square Generator," in *Proceedings of the International Conference on Computer Design*, pp. 506–511, 1996.

[14] C.-L. Wey and M.-D. Shieh, "Design of a High-Speed Square Generator," *IEEE Transactions on Computers*, vol. C-47, no. 9, pp. 1021–1026, 1998.

[15] B. Potu and P. A. Ramamoorthy, "A Pipelined Bit-Serial Squarer," in *International Conference on Systems Engineering*, pp. 579–582, 1987.

[16] M. Putrino, S. Vassiliadis, and E. Schwarz, "Array Two's Complement Multiplier and Square Function," *Electronic Letters*, vol. 23, no. 22, p. 1185, 1987.

[17] K. E. Wires, M. J. Schulte, L. P. Marquette, and P. I. Balzola, "Combined Unsigned and Two's Complement Squarers," in *Conference Record of the Thirty Third Asilomar Conference on Signals, Systems, and Computers*, pp. 1215–1219, 1999.

[18] E. G. Walters, J. Schlessman, and M. J. Schulte, "Combined Unsigned and Two's Complement Hybrid Squarers," in *Conference Record of the Thirty Fifth Asilomar Conference on Signals, Systems, and Computers*, pp. 861–866, 2001.

[19] D. De Caro and A. G. M. Strollo, "Parallel Squarers Using Booth Folding Technique," *Electronic Letters*, vol. 37, no. 6, pp. 346–347, 2001.

[20] Y. Fengqi and A. N. Wilson, "Multirate Digital Squarer Architectures," in *8th IEEE International Conference on Electronics, Circuits, and Systems*, pp. 177–180, 2001.

[21] A. G. M. Strollo, E. Napoli, and D. De Caro, "New Design of Squarer Circuits Using Booth Enccoding and Folding Techniques," in *8th IEEE International Conference on Electronics, Circuits, and Systems*, pp. 193–196, 2001.

[22] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, 1965.

[23] K. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, Jr., "Parallel Reduced Area Multipliers," *Journal of VLSI Signal Processing*, vol. 9, pp. 181–192, 1995.