Integer Multiplication with Overflow Detection or Saturation

Michael J. Schulte, *Member*, *IEEE*, Pablo I. Balzola, Ahmet Akkas, and Robert W. Brocato

Abstract—High-speed multiplication is frequently used in general-purpose and application-specific computer systems. These systems often support integer multiplication, where two n-bit integers are multiplied to produce a $2n$-bit product. To prevent growth in word length, processors typically return the n least significant bits of the product and a flag that indicates whether or not overflow has occurred. Alternatively, some processors saturate results that overflow to the most positive or most negative representable number. This paper presents efficient methods for performing unsigned or two's complement integer multiplication with overflow detection or saturation. These methods have significantly less area and delay than conventional methods for integer multiplication with overflow detection or saturation.

Index Terms—Overflow, saturation, two's complement, unsigned, integer, array multipliers, tree multipliers, computer arithmetic.

1 Introduction

Most modern computers directly support multiplication in hardware [1]. In high-performance systems, multiplication is typically implemented using either array multipliers [2], [3], [4] or tree multipliers [5], [6], [7], [8]. Both types of multipliers have area proportional to the square of the operand word length. The delay of array multipliers is proportional to the operand word length, whereas the delay of tree multipliers is proportional to the logarithm of the operand word length [9]. An advantage of array multipliers is that they are more regular than tree multipliers. Consequently, they require less area and are easier to implement in VLSI technology. As operand word lengths and clock speeds continue to increase, it is important to reduce the area, delay, and power dissipation of high-performance multipliers.

When two n-bit numbers are multiplied, a $2n$-bit product is produced. To avoid growth in word length, many computer systems require that the result of each arithmetic operation be the same length as its input operands [1]. Typically, when these systems perform integer multiplication, only the n least significant bits of the product are returned. For example, the Java Virtual Machine supports integer multiplication through the **imul** and **lmul** instructions [10]. The **imul** instruction multiplies 32-bit two's complement integers and returns the 32 least significant bits of their product. The **lmul** instruction is similar, except the input operands are 64 bits and the 64 least significant bits of the product are returned. Since the actual product may not be representable in the format of the result, it is desirable to have a flag that indicates whether or not overflow has occurred [11], [12]. Alternatively, on many digital signal processing systems, if results are too large or too small to represent, they saturate to the most positive or most negative representable number [13], [14], [15]. Overflow detection or saturation is used in several applications, including digital filter implementations, speech encoding, and graphics applications [11], [16].

Previous research on overflow detection and saturation has focused on fractional operands [14], [17] or operations other than multiplication [18], [19]. The main difference between fractional and integer multiplication is that fractional multiplication typically returns the most significant bits of the product. Also, with fractional two's complement multiplication, overflow is much easier to detect since it only occurs when the multiplication -1×-1 is performed [14].

This paper presents efficient techniques for implementing integer multiplication with overflow detection or saturation. Sections 2 and 3 present techniques for overflow detection or saturation with unsigned and two's complement multiplication, respectively. Section 4 gives component counts and area and delay estimates for multipliers that use either our proposed methods or conventional methods for overflow detection. Section 5 presents our conclusions. The technique for two's complement integer multiplication with overflow detection has been used in the design of the Sandia Secure Microprocessor, which implements a subset of the Java Virtual Machine in hardware.

2 Unsigned Multipliers

Fig. 1 shows the multiplication matrix for an n-bit unsigned integer multiplication. In this figure, the n-bit multiplicand $A = a_{n-1}a_{n-2}\ldots a_1a_0$ is multiplied by the n-bit multiplier $B = b_{n-1}b_{n-2}\ldots b_1b_0$ to produce a $2n$-bit product $P = p_{2n-1}p_{2n-2}\ldots p_1p_0$. The values of A, B, and P are

$$A = \sum_{i=0}^{n-1} a_i \cdot 2^i \quad B = \sum_{i=0}^{n-1} b_i \cdot 2^i \quad P = \sum_{i=0}^{2n-1} p_i \cdot 2^i. \qquad (1)$$

- *M.J. Schulte, P.I. Balzola, and A. Akkas are with the Electrical Engineering and Computer Science Department, Lehigh University, Bethlehem, PA 18015. E-mail: schulte@eecs.lehigh.edu.*
- *R.W. Brocato is with the Digital Microelectronics Department, Sandia National Laboratories, Albuquerque, NM 87185.*

Manuscript received 1 Sept. 1999; revised 1 Feb. 2000; accepted 10 Mar. 2000. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111789.

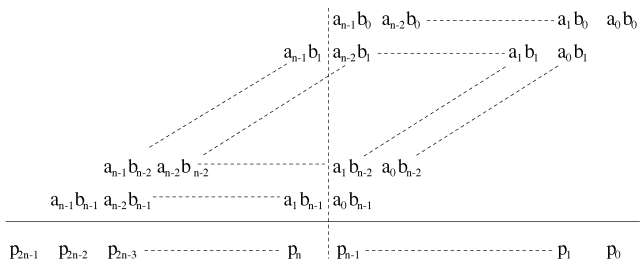


Fig. 1. Unsigned multiplication matrix for $P = A \cdot B$.

If the n least significant bits of the product are used as the result, then overflow occurs if $P \geq 2^n$. The conventional method for detecting this is to compute the entire $2n$-bit product and then compute overflow as

$$V = p_{2n-1} + p_{2n-2} + \ldots + p_{n+1} + p_n, \tag{2}$$

where $+$ denotes logical OR. Computing the entire product is necessary if the computer system supports instructions that require the most significant half of the product. The main disadvantage of this technique is that the hardware used to compute the most significant bits of the product and detect overflow contributes significantly to the area, delay, and power dissipation of the multiplier.

On computers that support unsigned saturating integer multiplication, overflow is usually detected in the manner described above. If overflow occurs, the n least significant bits of the product are all set to ones, which corresponds to $2^n - 1$, the largest representable number. This is accomplished by ORing each of the n least significant product bits with the overflow bit. Thus, the bits of the saturated product $< P >=< p_{n-1} >< p_{n-2} > \ldots < p_1 >< p_0 >$ are computed as

$$< p_i >= p_i + V \qquad (0 \leq i \leq n-1). \tag{3}$$

2.1 Unsigned Array Multipliers

A block diagram of an unsigned 8-bit array multiplier that performs conventional overflow detection is shown in Fig. 2a. The cells along each diagonal in the array multiplier correspond to a column in the multiplication matrix. In this diagram, a modified half adder (MHA) cell consists of an AND gate and a half adder (HA). The AND gate generates a partial product bit and the HA adds the generated partial product bit and a partial product bit from the previous row to produce a sum bit and a carry bit. Similarly, a modified full adder (MFA) consists of an AND gate, which generates a partial product bit, and a full adder (FA) that adds the partial product bit and the sum and carry bits from the previous row. The bottom row of adders produces the n most significant bits of the product. At the bottom of the array, $(n-1)$ OR gates operate on the n most significant product bits to detect overflow. If any of these bits are one, then $V = 1$. An n-bit array multiplier that uses this technique has n^2 AND gates, $(n-1)$ OR gates, n HAs, and $(n^2 - 2n)$ FAs.

The dashed line in Fig. 2a indicates the worst case delay path through the multiplier. The worst case delay is approximately equal to the delay through one AND gate, two OR gates, two HAs, and $(2n-4)$ FAs. To improve performance, the bottom row of adders and the row of OR gates can be replaced by a fast $(n-1)$-bit carry-propagate adder (CPA) and a tree of OR gates. This approach, however, increases the area and reduces the regularity of

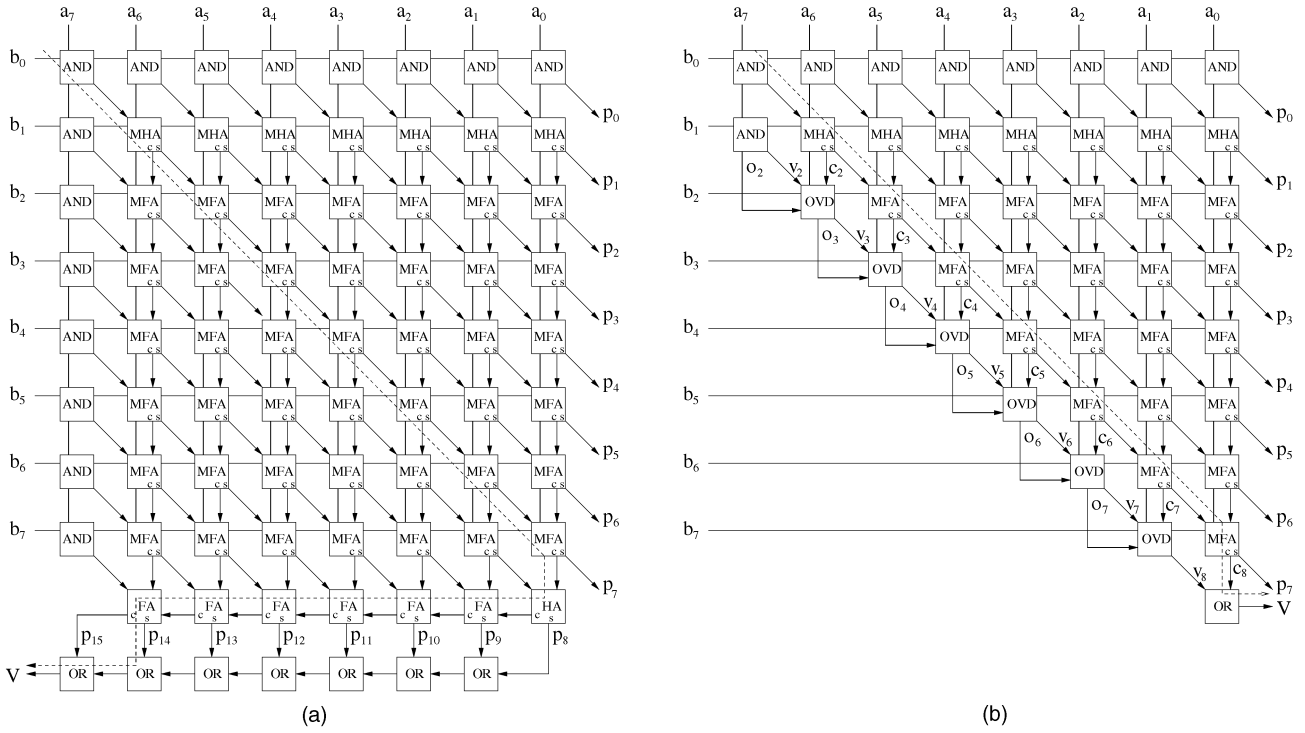


Fig. 2. Unsigned 8-bit array multipliers. (a) Conventional overflow detection. (b) Proposed overflow detection.

the array. In some systems, overflow detection is performed after the entire product is stored in a $2n$-bit register.

In computer systems that require only the n least significant bits of the product and an overflow flag or saturation, a significant reduction in area, delay, and power consumption can be achieved. In Fig. 2a, components below the diagonal dashed line do not contribute to the n least significant bits of the product. This line corresponds to the vertical dashed line in Fig. 1.

Our method for detecting overflow avoids computing the n most significant bits of the product. Instead, it signals overflow if any of the partial product bits in columns n to $(2n-2)$ of the multiplication matrix are one or if any of the carries into column n are one. Mathematically, this is expressed as

$$V = \sum_{i=1}^{n-1} \left(c_{i+1} + \sum_{j=1}^{i} a_{n-j} \cdot b_i \right), \tag{4}$$

where V is the overflow flag, c_i is the ith carry into column n, and the bit summations and bit multiplications correspond to logical ORs and ANDs, respectively. A significant reduction in logic is achieved by taking advantage of common terms in the overflow equation.

With our method, the overflow flag is computed using the following iterative equations:

$$o_{i+1} = o_i + a_{n-i} \tag{5}$$

$$v_{i+1} = v_i + c_i + o_{i+1} \cdot b_i \tag{6}$$

for $2 \leq i \leq n-1$, where o_i is a temporary OR bit and v_i is a temporary overflow bit. Initially, $o_2 = a_{n-1}$ and $v_2 = a_{n-1} \cdot b_1$. After $(n-2)$ iterations of (5) and (6), the overflow flag is computed as $V = v_n + c_n$.

This approach is illustrated in Fig. 2b for an 8-bit array multiplier. All the hardware that previously computed p_n to p_{2n-1} and V is replaced by one AND gate, one OR gate, and $(n-2)$ overflow detection (OVD) cells. Each OVD cell computes o_{i+1} and v_{i+1} based on (5) and (6). Thus, each OVD cell consists of one AND gate, one 2-input OR gate, and one 3-input OR gate. An n-bit array multiplier that uses our method for overflow detection requires $(n^2+3n-2)/2$ AND gates, $(n-1)$ 2-input OR gates, $(n-2)$ 3-input OR gates, $(n-1)$ HAs, and $(n^2-3n+2)/2$ FAs. This is $(n^2-n)/2$ fewer adders, $(n^2-3n+2)/2$ fewer AND gates, and $(n-2)$ more 3-input OR gates than the conventional method. Depending on the technology used for the multiplier, the actual implementation of the cells may vary. For example, with static CMOS, the OVD cells may be implemented using NAND/NOR gates or complex gates to improve area and performance.

The dashed line in Fig. 2b indicates the worst case delay path through the multiplier. Since an MFA has longer delay than an OVD cell, the worst case delay is approximately equal to the delay through one AND gate, one OR gate, one HA, and $(n-2)$ FAs. Compared to array multipliers that detect overflow using the conventional method, array multipliers that use our method have worst case delay paths that are approximately half as long.

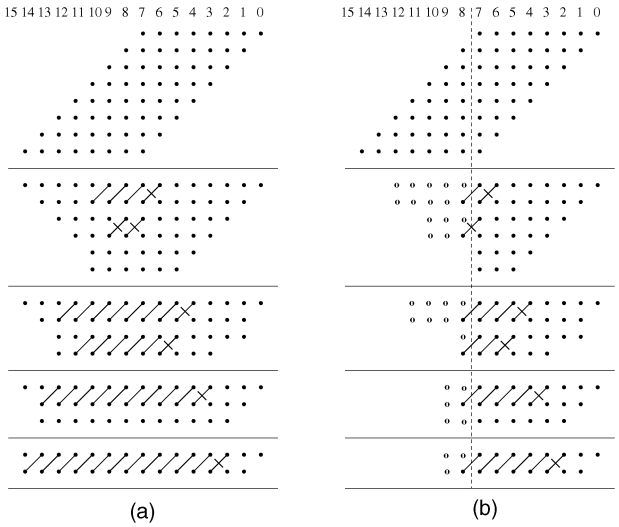


Fig. 3. Unsigned 8-bit Dadda tree multipliers. (a) Conventional overflow detection. (b) Proposed overflow detection.

To perform saturating multiplication with either our method or the conventional method, the V bit is ORed with the n least significant bits of the partial product. This requires n additional OR gates and the worst case delay path increases by the delay of one OR gate.

2.2 Unsigned Tree Multipliers

With tree multipliers, the bits of the multiplicand and multiplier are ANDed to generate an n-word by n-bit partial product matrix. After this, stages of HAs and FAs are used to reduce the partial product matrix to two rows, which are summed using a fast carry-propagate adder (CPA). Fig. 3a shows the dot diagram of an 8-bit tree multiplier that uses Dadda's method of partial product reduction [6]. Although Dadda tree multipliers are not as efficient as the tree multipliers described in [7], [20], [21], they are useful to examine because the number and type of cells required to implement them are easily determined based on n. Results similar to the ones presented here are expected for other types of tree multipliers.

In Fig. 3a, a partial product bit is represented by a dot, the outputs of a full adder are represented by two dots connected by a plain diagonal line, and the outputs of a half adder are represented by two dots connected by a crossed diagonal line [6]. Each stage of adders is divided by a horizontal line. The bottom two rows of the figure correspond to sum and carry vectors S and C. An n-bit unsigned Dadda tree multiplier has n^2 AND gates to generate the partial products, $(n-1)$ HAs and (n^2-4n+3) FAs to reduce the partial products to S and C, and a $(2n-2)$-bit CPA to add S and C to produce P [6], [7].

The worst case delay through the Dadda tree multiplier is equal to the delay for partial product generation (one AND gate delay), plus the delay for partial product reduction through s stages (s FA delays), plus the delay of a $(2n-2)$-bit CPA [7]. For a given multiplier size n, the number of stages s is determined from Table 1. For example, an 8-bit multiplier has four stages and a 32-bit multiplier has eight stages. With tree multipliers, overflow detection is often performed with a tree of $(n-1)$ 2-input

TABLE 1
Number of Stages s for n-Bit Dadda Tree Multipliers

Range of n	s	Range of n	s
$3 \le n \le 3$	1	$14 \le n \le 19$	6
$4 \le n \le 4$	2	$20 \le n \le 28$	7
$5 \le n \le 6$	3	$29 \le n \le 42$	8
$7 \le n \le 9$	4	$43 \le n \le 63$	9
$10 \le n \le 13$	5	$64 \le n \le 94$	10

OR gates, which has a delay equal to $\lceil \log_2(n) \rceil$ 2-input OR gates.

In general, our array multiplier overflow detection technique, which uses a linear array of overflow detection cells, cannot be applied to the design of tree multipliers without increasing the worst case delay path. This is because the delay through the overflow detection cells is linear with respect to n, while the delay through the tree multiplier is logarithmic. To overcome this problem, the adders in columns n to $(2n-2)$ are replaced by a tree of OR gates, which combines the partial products bits in columns n to $(2n-2)$ and the carries going into column n. Although this requires more logic than our approach for the array multiplier, it prevents the delay of the overflow detection logic from increasing the worst case delay path.

Fig. 3b illustrates the hardware savings that are achieved by using our method for overflow detection with an 8-bit Dadda tree multiplier, where the output of a 2-input OR gate is represented by the symbol "o." In each stage, the number of bits to be ORed for overflow detection is reduced by approximately a factor of two. Depending on the implementation technology, OR gates with more inputs or NAND/NOR gates may be used to implement overflow detection.

Since there are $(n^2-n)/2$ partial products bits in columns n to $(2n-2)$ and $(n-1)$ carries going into column n (including one from the CPA), the number of OR gates required to implement overflow detection with this technique is $(n^2+n-4)/2$. The entire multiplier with overflow detection has n^2 AND gates, $(n^2+n-4)/2$ OR gates, $(n-2)$ HAs, $(n^2-5n+6)/2$ FAs, and an $(n-1)$-bit carry-propagate adder.

The tree of OR gates that detects overflow operates in parallel with the partial product reduction and carry propagate-addition. Since the delay for partial product reduction and carry-propagate addition is greater than the delay for overflow detection, the worst case delay through the Dadda tree multiplier is the delay for partial product generation (one AND gate delay), plus the delay for partial product reduction (s FA delays), plus the delay of an $(n-1)$-bit CPA, plus the delay of one OR gate to include the carry out of the CPA. If saturating multiplication is required, the overflow bit is ORed with the n least significant bits of the product.

3 Two's Complement Multipliers

With two's complement integer multiplication, the values of A, B, and P are

$$A = -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i \tag{7}$$

$$B = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i \tag{8}$$

$$P = -p_{2n-1} \cdot 2^{2n-1} + \sum_{i=0}^{2n-2} p_i \cdot 2^i. \tag{9}$$

If the n least significant bits of the product are used as the result, overflow occurs when

$$P \le -2^{n-1} - 1 \quad \text{or} \quad P \ge 2^{n-1}. \tag{10}$$

With the conventional method, overflow is detected by testing if p_{n-1} differs from any product bit to the left of it (i.e., p_n to p_{2n-1}). Thus, overflow is computed as

$$V = \hat{p}_{2n-1} + \hat{p}_{2n-2} + \ldots + \hat{p}_{n+1} + \hat{p}_n, \tag{11}$$

where $\hat{p}_i = p_i \oplus p_{n-1}$ and $\oplus$ denotes logical exclusive-or (XOR). Detecting overflow with this method requires n XOR gates and $(n-1)$ OR gates.

For saturating multiplication, the product saturates to $-2^{n-1} = 10\ldots0$ if $t = a_{n-1} \oplus b_{n-1} = 1$ and $V = 1$ and it saturates to $2^{n-1} - 1 = 01\ldots1$ if $t = a_{n-1} \oplus b_{n-1} = 0$ and $V = 1$. If $V = 0$, then the n least significant bits of the product are returned. Thus, the bits of the saturated product,

$$< P > = < p_{n-1} >< p_{n-2} > \ldots < p_1 >< p_0 >, \tag{12}$$

are computed as

$$< p_{n-1} > = Vt + \overline{V} p_{n-1} \tag{13}$$

$$< p_i > = V\overline{t} + \overline{V} p_i \quad (0 \le i \le n-2). \tag{14}$$

Equations (13) and (14) can be implemented by an n-bit 2-to-1 multiplexor that uses V as the select signal.

Several techniques have been proposed to implement two's complement multipliers, including Pezaris arrays [3], variations of the Baugh-Wooley Algorithm [22], [23], [24], and Booth's Algorithm [25] and its extensions [26]. These techniques modify the way in which the partial products are generated or combined in order to handle partial products with negative and positive weights. After the partial products are generated using one of these techniques, they are then reduced using array or tree structures.

Unfortunately, previous techniques for two's complement multiplication do not work well for overflow detection or saturation without computing the most significant product bits. Fig. 4 shows the n-bit multiplication matrix used in the Complemented Partial Product Word Correction Algorithm [7], [24]. In multipliers that use this algorithm, carry out of column $(2n-1)$ is ignored. When two numbers that do not cause overflow are multiplied (e.g., 1×1), some of the partial product bits in columns n to $(2n-2)$ are one and others are zero. The partial product bits that are one may cause carries out of column $(2n-1)$, which do not contribute to the product. Thus, it becomes difficult to test for overflow without computing the n most

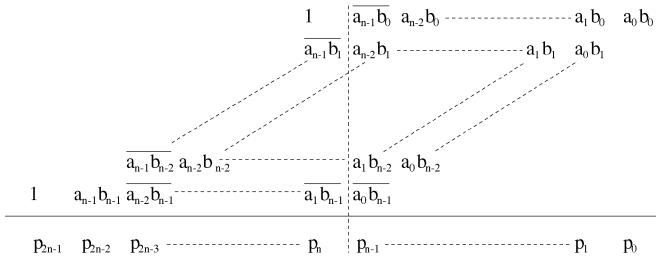


Fig. 4. Two's complement multiplication matrix for $P = A \cdot B$.

significant bits of the product. This is also true for two's complement multipliers that use other techniques to generate the partial products.

To provide overflow detection or saturation for two's complement multiplication without computing the n most significant bits of the product, our technique multiplies the magnitudes of the input operands. This allows overflow detection to be performed using a technique similar to the one presented in Section 2 since the magnitudes can be treated as unsigned numbers. Typically, the magnitude of a two's complement integer is obtained by inverting the bits of the number and adding a one when the sign bit is one (i.e., the number is negative). To avoid the carry-propagation required to add the one, the multiplication matrix is modified to directly incorporate computing the magnitudes.

The product of the magnitudes of the input operands is computed as

$$\begin{aligned} |P| &= |A| \cdot |B| \\ &= \left(a_{n-1} + \sum_{i=0}^{n-2} \hat{a}_i \cdot 2^i\right) \cdot \left(b_{n-1} + \sum_{j=0}^{n-2} \hat{b}_j \cdot 2^j\right) \\ &= a_{n-1}b_{n-1} + \sum_{i=0}^{n-2} \hat{a}_i b_{n-1} \cdot 2^i \\ &\quad + \sum_{j=0}^{n-2} a_{n-1}\hat{b}_j \cdot 2^j + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} \hat{a}_i\hat{b}_j \cdot 2^{i+j}, \end{aligned} \tag{15}$$

where $\hat{a}_i = a_i \oplus a_{n-1}$ and $\hat{b}_i = b_i \oplus b_{n-1}$. An n-bit multiplication matrix for $|P| = |A| \cdot |B|$ is shown in Fig. 5. This multiplication matrix has $(n+1)$ rows, $(2n-3)$ columns, and n^2 partial products.

Since the product of magnitudes is always nonnegative, it is necessary to return the two's complement of the product when the sign bits of the multiplier and multiplicand differ (i.e., when $t = a_{n-1} \oplus b_{n-1} = 1$) since the true product is not positive. For two's complement multiplication, our methods for performing overflow detection or saturation and conditionally producing the two's complement of the product depend on whether array multipliers or tree multipliers are being implemented.

3.1 Two's Complement Array Multipliers

Fig. 6a shows an 8-bit two's complement array multiplier that uses the Complemented Partial Product Word Correction Algorithm to generate the partial products and the conventional method for overflow detection. This multiplier is similar to the one shown in Fig. 2a except that $(n-1)$ AND gates on the left side of the array are replaced by

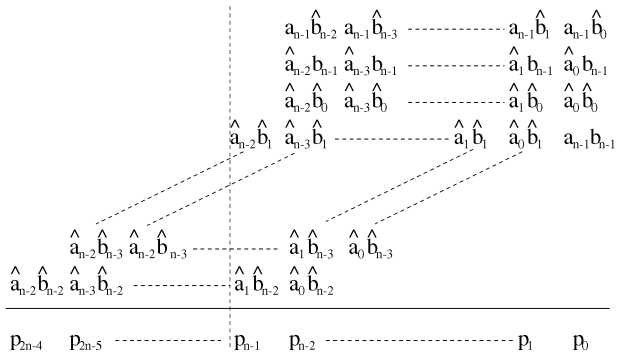


Fig. 5. Multiplication matrix for $|P| = |A| \cdot |B|$.

NAND gates, $(n-1)$ MFAs toward the bottom of the array are replaced by negating modified full adders (NMFAs), one HA is replaced by a specialized half adder (SHA), p_{2n-1} is inverted, and n XOR gates are used at the bottom of the array. The NMFAs and the NAND gates invert $(2n-2)$ partial product bits, as shown in Fig. 4. The SHA takes sum and carry bits from the previous row and adds them with "1" to produce new sum and carry bits. The SHA has approximately the same area and delay as an HA [7]. The SHA and the inverter that complements p_{2n-1} add the ones shown in columns n and $(2n-1)$ of Fig. 4 [7]. The n XOR gates perform $\hat{p}_i = p_i \oplus p_{n-1}$, for $n \le i \le 2n-1$.

An n-bit two's complement array multiplier that uses conventional overflow detection has $(2n-1)$ inverters, n^2 AND gates, $(n-1)$ OR gates, n XOR gates, n HAs, and (n^2-2n) FAs. When implemented in CMOS technology, $(2n-2)$ of the AND gates and inverters can be combined to form NAND gates. The worst case delay is approximately equal to the delay through one inverter, one AND gate, two OR gates, one XOR gate, two HAs, and $(2n-4)$ FAs.

Fig. 6b shows an 8-bit two's complement array multiplier that uses the conventional method for saturation. It is similar to the multiplier shown in Fig. 6a except that it has an n-bit 2-to-1 multiplexor that saturates the product when overflow occurs. Also, the AND gate in the bottom left corner of the array is changed to an XAND cell. The XAND cell produces the partial product bit $a_{n-1}b_{n-1}$ and the signal $t = a_{n-1} \oplus b_{n-1}$, which is used to set the product bits when saturation occurs.

Our test for overflow for two's complement multiplication is more complicated than for unsigned multiplication because of asymmetry in the two's complement number system (i.e., the magnitude of the most negative number), and the need to correctly handle zero times a negative number. Overflow occurs when any of the partial product bits in columns n to $(2n-4)$ are one, any carries into column n are one, or $p_n p_{n-1} \ldots p_0$ cannot be represented as an n-bit two's complement number. This last condition is detected by examining t, p_n, and p_{n-1}. Overflow occurs when $t = 0$ and $p_{n-1} = 1$ (i.e., $P \ge 2^{n-1}$) or when $t = 1$ and $p_n = 0$ and $p_{n-1} = 0$ (i.e., $P \le -2^{-n-1} - 1$). Thus, the final overflow detection logic computes

$$V = \overline{t}\, p_{n-1} + t\, \overline{p_n}\, \overline{p_{n-1}} + V', \tag{16}$$

where the preliminary overflow flag is

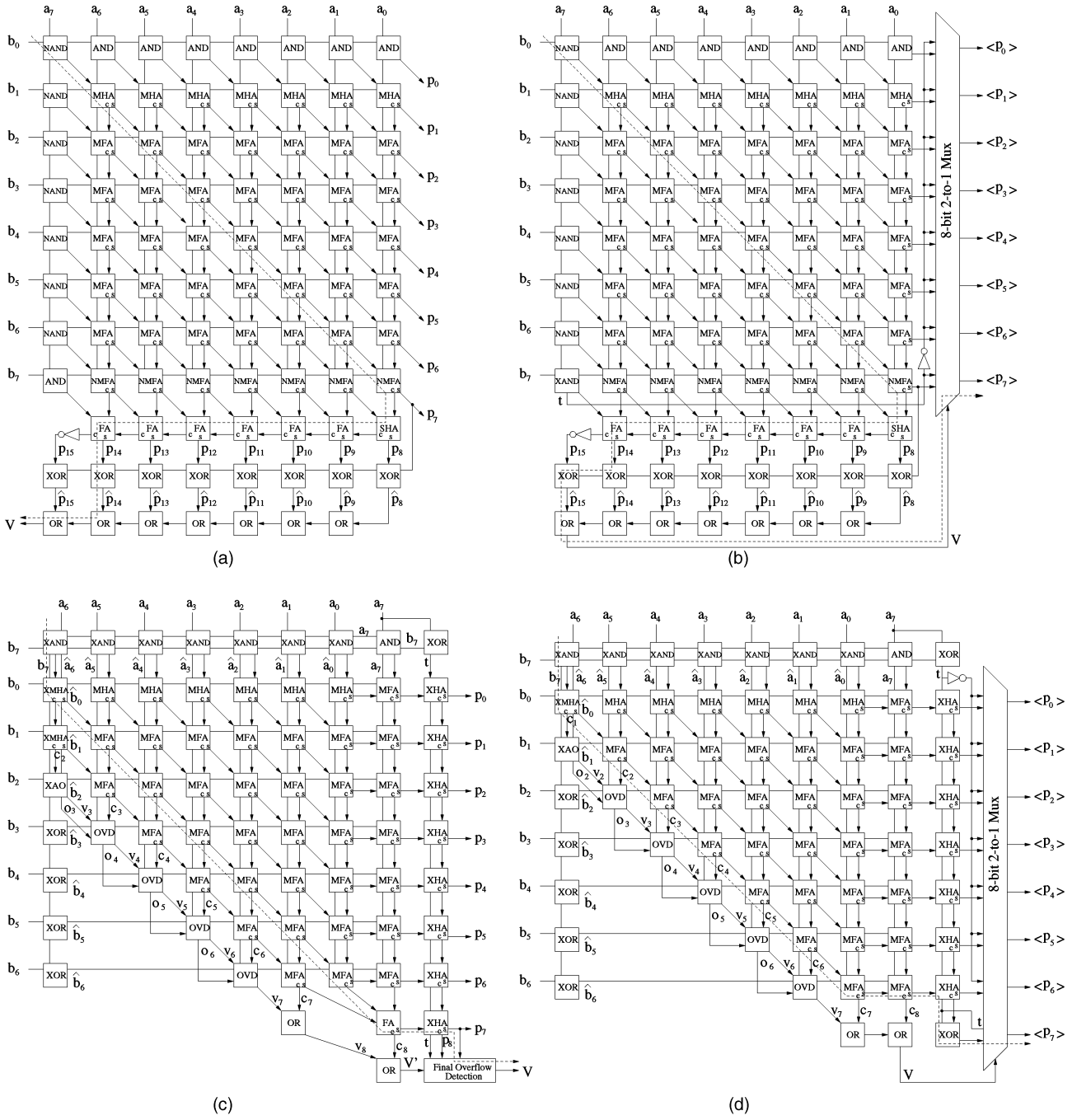


Fig. 6. Two's complement 8-bit array multipliers. (a) Conventional overflow detection. (b) Conventional saturation. (c) Proposed overflow detection. (d) Proposed saturation.

$$V' = \sum_{i=2}^{n-2} \left(c_i + \sum_{j=2}^{i} \hat{a}_{n-j} \cdot \hat{b}_i \right) + c_{n-1} + c_n. \tag{17}$$

Instead of computing V' by ORing the partial product bits in columns n to $(2n-4)$ and the carries into column n, V' is computed using the following iterative equations

$$o_{i+1} = o_i + \hat{a}_{n-i} \tag{18}$$

$$v_{i+1} = v_i + c_i + o_{i+1} \cdot \hat{b}_i \tag{19}$$

for $3 \le i \le n-2$, with the initial conditions $o_3 = \hat{a}_{n-2}$ and $v_3 = \hat{a}_{n-2} \cdot \hat{b}_2 + c_2$. After $(n-4)$ iterations of (18) and (19), the preliminary overflow flag is computed as

$$V' = v_{n-1} + c_{n-1} + c_n. \tag{20}$$

Fig. 6c shows an 8-bit two's complement array multiplier that uses our method for overflow detection. The core of the array multiplier computes the n least significant bits of $|P| = |A| \cdot |B|$, based on columns 0 to $(n-1)$ of the multiplication matrix shown in Fig. 5. Each XAND cell in the top row consists of an XOR gate that produces $\hat{a}_i = a_{n-1} \oplus a_i$ and an AND gate that produces $\hat{a}_i b_{n-1}$. Similarly,

each XMHA on the left side of the array consists of an XOR gate that produces $\hat{b}_i = b_{n-1} \oplus b_i$, an AND gate that produces $a_{n-2}\hat{b}_i$, and an HA that adds $a_{n-2}\hat{b}_i$ and a bit from the cell above it. On the right side of the array, n XHA cells produce the two's complement of $\mid P \mid$, when $t = 1$. Each XHA cell consists of an XOR gate and an HA. When $t = 1$, the XOR gates invert the bits of $\mid P \mid$ and the half adders add one to the inverted bits of $\mid P \mid$. The overflow flag is computed using one XOR-AND-OR (XAO) cell to produce $\hat{b}_2$ and v_3, $(n-4)$ overflow detection (OVD) cells to implement (18) and (19), two OR gates to implement (20), and a final overflow detection circuit to compute V based on (16).

An n-bit array multiplier that implements overflow detection using our method has three inverters, $(n^2 + 5n + 2)/2$ AND gates, n 2-input OR gates, $(n-4)$ 3-input OR gates, $(3n-1)$ XOR gates, $2n$ HAs, and $(n^2 + n - 6)/2$ FAs. The worst case delay is equivalent to the delay through one inverter, three AND gates, two OR gates, two XOR gates, two HAs, and $(n-1)$ FAs.

With our method, saturation detection is simpler than overflow detection for two's complement multiplication. When the actual product is the most negative representable number (i.e., -2^{n-1}), saturating the product to the most negative number produces the correct result. Thus, the final product is saturated whenever $\mid P \mid \geq 2^{n-1}$ or, equivalently, whenever any partial product bit in columns $(n-1)$ to $(2n-4)$ or any carry into column $(n-1)$ is one. Consequently, only the $(n-1)$ least significant bits of $\mid P \mid$ need to be calculated. In this case, V is computed using the following iterative equations

$$o_{i+1} = o_i + \hat{a}_{n-i} \tag{21}$$

$$v_{i+1} = v_i + c_i + o_{i+1} \cdot \hat{b}_i \tag{22}$$

for $2 \leq i \leq n-2$, with the initial conditions $o_2 = \hat{a}_{n-2}$ and $v_2 = \hat{a}_{n-2} \cdot \hat{b}_1 + c_1$. After $(n-3)$ iterations of (21) and (22), the overflow flag is computed as

$$V = v_{n-1} + c_{n-1} + c_n. \tag{23}$$

Fig. 6d shows an 8-bit two's complement array multiplier that uses our method for saturation. Since the product saturates when $\mid P \mid \geq 2^{n-1}$, $\mid P \mid$ only requires $(n-1)$ bits. The $(n-1)$ XHA cells and the XOR gate on the right side of the array produce the two's complement of $\mid P \mid$ when $t = 1$. This multiplier requires $(n^2 + 5n - 6)/2$ AND gates, n 2-input OR gates, $(n-3)$ 3-input OR gates, $(3n-1)$ XOR gates, $(2n-2)$ HAs, $(n^2 - n)/2$ FAs, and an n-bit 2-to-1 multiplexor. The worst case delay through this multiplier is equal to the delay through one AND gate, three XOR gates, two HAs, $(n-1)$ FAs, and an n-bit 2-to-1 multiplexor.

3.2 Two's Complement Tree Multipliers

Fig. 7a shows the dot diagram of an 8-bit two's complement tree multiplier that uses the Complemented Partial Product Word Correction Algorithm [24] to generate the partial products (see Fig. 4) and Dadda's technique [6] to reduce the partial products to sum and carry vectors. In this diagram, a line over a partial product bit indicates that the partial product bit is inverted. The circled half adder in

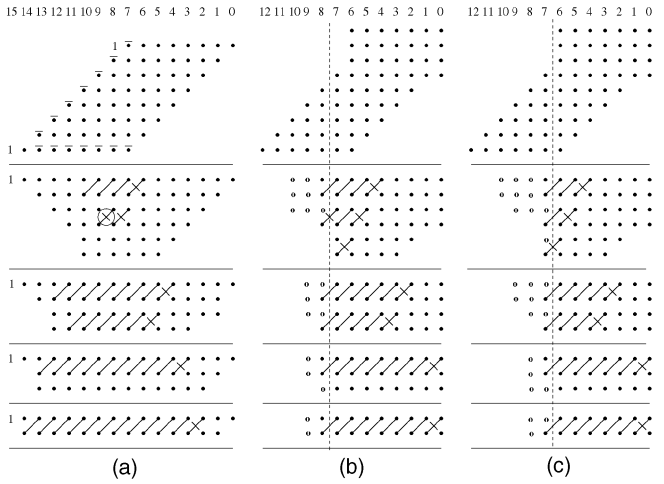


Fig. 7. Two's complement 8-bit Dadda tree multipliers. (a) Conventional overflow detection. (b) Proposed overflow detection. (c) Proposed saturation detection.

column n corresponds to a specialized half adder (SHA). The one in column $(2n-1)$ is added by inverting the most significant bit of the product [7].

An n-bit two's complement Dadda tree multiplier has $(2n-1)$ inverters, n^2 AND gates, $(n-1)$ HAs, $(n^2 - 4n + 3)$ FAs, and a $(2n-2)$-bit CPA [7]. Implementing overflow detection using the conventional method requires n XOR gates and $(n-1)$ OR gates. The delay of the conventional tree multiplier with overflow detection is the same as for the unsigned case except for one extra XOR delay from XORing p_{n-1} with p_n to p_{2n-1} and one extra inverter delay to complement $(2n-2)$ of the partial products. Adding saturation logic to a tree multiplier that already detects overflow requires an XOR gate to compute $t = a_{n-1} \oplus b_{n-1}$, an inverter to compute $\bar{t}$, and an n-bit 2-to-1 multiplexor to select the computed product or the saturation value, based on (13) and (14).

For array multipliers that use our methods for overflow detection or saturation, the two's complement of the product is computed by conditionally inverting the bits of the product and adding one when $t = 1$. For tree multipliers, however, this approach requires two carry propagate additions; one adds S and C to produce $\mid P \mid$ and the second conditionally adds one to $\mid P \mid$ after its bits have been inverted. Instead, our method for tree multipliers uses an n-bit carry-save adder to conditionally add $(2^n - 1)$ (i.e., n consecutive ones) to S and C, uses an n-bit CPA to compute the product, and then uses n XOR gates to conditionally invert the product bits. This approach is shown in Fig. 8a, where $(2^n - 1)$ is added and the product bits are inverted if $t = 1$. The n-bit carry-save adder is implemented using n full adders and has a delay equivalent to one full adder.

In parallel with conditionally inverting the product, the preinverted product bits p'_{n+1}, p'_n, and p'_{n-1}, and the two's complement signal t are examined to test if overflow occurs. Overflow occurs if $t = 1$ and $\mid P \mid + (2^n - 1) \geq 2^n + 2^{n-1}$, or if $t = 0$ and $\mid P \mid + 0 \geq 2^{n-1}$. A simplified logic equation for determining overflow is

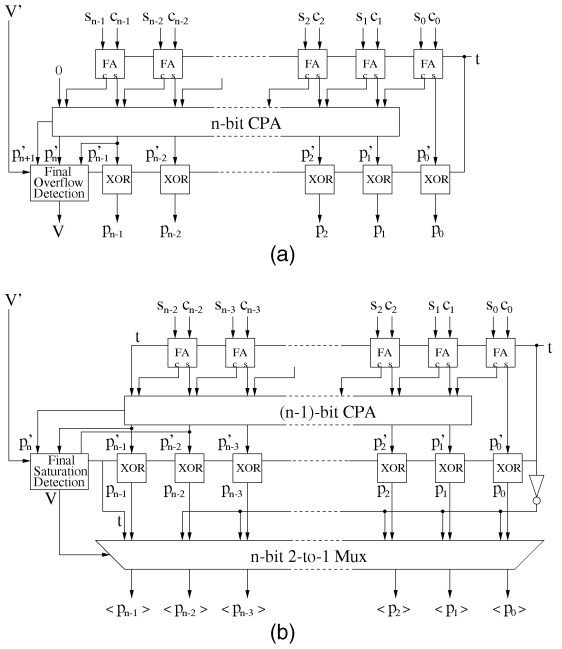


Fig. 8. Overflow/saturation detection and product correction. (a) Proposed overflow detection and product correction. (b) Proposed saturation detection and product correction.

$$V = p'_{n+1} + p'_n p'_{n-1} + \bar{t}p'_n + \bar{t}p'_{n-1} + V', \tag{24}$$

where V' is one if any of the partial product bits in columns n to $(2n-4)$ are one or if carries into column n are one.

Fig. 7b illustrates the hardware savings achieved by our method for overflow detection with an 8-bit two's complement Dadda tree multiplier. In this diagram, partial product bits in columns n to $(2n-4)$ are ORed with carries into column n to compute V', which is used as an input to the final overflow detection logic, shown in Fig. 8a. The $2n$-bits in the bottom two rows of Fig. 7b to the right of the dashed line correspond to S and C in Fig. 8a. Our method for two's complement multiplication with overflow detection uses one inverter, (n^2+3) AND gates, $(n^2-3n+10)/2$ OR gates, $(3n-1)$ XOR gates, n HAs, $(n^2+n-2)/2$ FAs, and an n-bit CPA. The worst case delay through the multiplier and overflow detection logic is equivalent to the delay of two AND gates, three OR gates, two XOR gates, $(s+1)$ FAs, and an n-bit CPA.

Similar to our method for saturating array multipliers, our method for saturating tree multipliers causes saturation whenever $\mid P \mid \geq 2^{n-1}$. Thus, $\mid P \mid$ only requires $(n-1)$ bits. The correct product is obtained by adding $S + C = \mid P \mid$ and $t \ldots t$, using a $(n-1)$-bit carry-save adder and an n-bit CPA. This is shown in Fig. 8b, where the final saturation detection logic implements

$$V = p'_n p'_{n-1} + \bar{t}p'_{n-1} + \bar{t}p'_{n-2} + V'. \tag{25}$$

An n-bit 2-to-1 multiplexor saturates the product when $V = 1$, based on (13) and (14).

Fig. 7c illustrates our method for saturating multiplication with an 8-bit two's complement Dadda tree multiplier. For an n-bit Dadda tree multiplier, our method requires two inverters, (n^2+4) AND gates, $(n^2-n+6)/2$ OR gates, $(3n-1)$ XOR gates, $(n-1)$ HAs, $(n^2-n+6)/2$ FAs, an n-bit 2-to-1 multiplexor, and an $(n-1)$-bit CPA. The worst case delay through the multiplier and saturation logic is equivalent to the delay of one inverter, three AND gates, three OR gates, two XOR gates, $(s+1)$ FAs, an n-bit 2-to-1 multiplexor, and an $(n-1)$-bit CPA.

In [27], two's complement multiplication is also performed by multiplying the magnitudes of the numbers and correcting the product when the sign bits of the multiplier and multiplicand differ. Our techniques differ from the technique presented in [27] because our techniques do not compute the most significant bits of the product and our methods for correcting the final product do not require carry propagate addition. In [28], the multiplicand is converted to a sign-magnitude representation and the multiplier is Booth-encoded. Using the magnitude of the multiplicand reduces switching activity and power dissipation since integers with small magnitudes have zeros in their most significant bits [28]. Similar reductions in power dissipation are expected using our techniques.

4 Component Counts, Delay, and Area

Tables 2 and 3 summarize component counts and worst case delays for multipliers with overflow detection. For Multiplier Type, C and P denote conventional and proposed overflow detection techniques, U and S denote unsigned and signed operands, and A and T denote array and tree multipliers. Table 2 gives the total number of each component and the length of the CPA. This table shows that the proposed methods for overflow detection significantly reduce the number of AND gates and FAs for array multipliers and the number of FAs and the length of the CPA for tree multipliers. Table 3 gives the number of each component and the length of the CPA on the worst case delay path. The proposed methods for overflow detection reduce the number of full adders on the critical path for array multipliers. For tree multipliers, they reduce the number of OR gates and the length of the CPA on the critical path. In general, the proposed array multipliers with overflow detection use fewer components and have more delay than the proposed tree multipliers.

Depending on the operand size and implementation technology, it is often possible to further reduce the amount of logic required to implement overflow detection for tree multipliers without increasing their worst case delay path. This is accomplished by combining common terms in the overflow detection logic until the delay through the overflow detection logic matches the delay through the multiplier tree. For the area and delay estimates given in this section, we initially use a tree of OR gates to implement the overflow detection logic and then allow synthesis tools to optimize this implementation. Thus, the component counts for the proposed tree multipliers in Table 2 should be considered to be worst case values when no optimization has been performed.

Java programs were used to generate gate-level VHDL models for various array and Dadda tree multipliers with overflow detection. The VHDL code was then synthesized and optimized for area using LSI Logic's 0.6 micron LCA300K gate array library and the Leonardo synthesis tool from Exemplar Logic. The estimates given assume a

TABLE 2
Component Counts for n-Bit Multipliers with Overflow Detection

Multiplier Type	Number of Components							
	INV	AND	OR2	OR3	XOR	HA	FA	CPA
CUA	-	n^2	$n-1$	-	-	n	n^2-2n	-
PUA	-	$(n^2+3n-2)/2$	$n-1$	$n-2$	-	$n-1$	$(n^2-3n+2)/2$	-
CUT	-	n^2	$n-1$	-	-	$n-1$	n^2-4n+3	$2n-2$
PUT	-	n^2	$(n^2+n-4)/2$	-	-	$n-2$	$(n^2-5n+6)/2$	$n-1$
CSA	$2n-1$	n^2	$n-1$	-	n	n	n^2-2n	-
PSA	3	$(n^2+5n+2)/2$	n	$n-4$	$3n-1$	$2n$	$(n^2+n-6)/2$	-
CST	$2n-1$	n^2	$n-1$	-	n	$n-1$	n^2-4n+3	$2n-2$
PST	1	n^2+3	$(n^2-3n+10)/2$	-	$3n-1$	n	$(n^2+n-2)/2$	n

TABLE 3
Worst Case Delay for n-Bit Multipliers with Overflow Detection

Multiplier Type	Number of Components on Worst Case Delay Path						
	INV	AND	OR2	XOR	HA	FA	CPA
CUA	-	1	2	-	2	$2n-4$	-
PUA	-	1	1	-	1	$n-2$	-
CUT	-	1	$\lceil log_2(n) \rceil$	-	-	s	$2n-2$
PUT	-	1	1	-	-	s	$n-1$
CSA	1	1	2	1	2	$2n-4$	-
PSA	1	3	2	2	2	$n-1$	-
CST	1	1	$\lceil log_2(n) \rceil$	1	-	s	$2n-2$
PST	-	2	3	2	-	$s+1$	n

nominal operating voltage and temperature of 5.0 Volts and 25° C, respectively. Area estimates are reported in equivalent gates and delay estimates are reported in nanoseconds. For each tree multiplier, the final CPA is implemented using a block carry-lookahead adder (CLA) with a block size of four [9].

Table 4 gives area and delay estimates for unsigned array multipliers using the conventional method and our proposed method for overflow detection. Compared to the conventional method, our method has between 50 and 53 percent less area and between 41 and 42 percent less delay. These reductions in area and delay occur because the adders that compute the n most significant product bits are replaced by simple overflow detection logic.

Table 5 gives area and delay estimates for unsigned Dadda tree multipliers using the conventional method and our proposed method for overflow detection. Compared to the conventional method, our method has about 47 percent less area and between 23 and 28 percent less delay. The reduction in area is due to replacing $(n^2-3n+2)/2$ adders and $(n-1)$ OR gates with $(n^2+n-4)/2$ OR gates and reducing the length of the CLA from $(2n-2)$ bits to $(n-1)$ bits. The reduction in delay is due to using a smaller CLA and performing most of the overflow detection in parallel with the partial product reduction and carry-lookahead addition.

Table 6 gives area and delay estimates for two's complement array multipliers using the conventional method and our proposed method for overflow detection. Compared to the conventional method, our method requires between 12 and 43 percent less area and has between 10 and 33 percent less delay. The percent reductions in area and delay are less than the percent reductions for the unsigned array multipliers due to the overhead of conditionally complementing the input and output operands. As the multiplier size increases, the percent reduction in area achieved by our proposed method also increases. This is because the overhead to conditionally complement the operands increases linearly with multiplier size, while the savings from replacing adders by overflow detection logic increase quadratically.

Table 7 gives area and delay estimates for two's complement Dadda tree multipliers using the conventional method and our proposed method for overflow detection.

TABLE 4
Unsigned Array Multipliers with Overflow Detection

n	Conventional		Proposed		Reduction	
	Area	Delay	Area	Delay	Area	Delay
8	662	10.70	330	6.19	50%	42%
16	2862	23.02	1366	13.54	52%	41%
24	6598	35.34	3106	20.92	53%	41%
32	11870	47.66	5550	28.26	53%	41%

TABLE 5
Unsigned Dadda Tree Multipliers with Overflow Detection

n	Conventional		Proposed		Reduction	
	Area	Delay	Area	Delay	Area	Delay
8	821	8.07	434	6.21	47%	23%
16	2905	10.53	1546	7.85	47%	25%
24	6270	13.57	3344	9.71	47%	28%
32	10918	14.98	5818	11.17	47%	25%

TABLE 6
Signed Array Multipliers with Overflow Detection

	Conventional		Proposed		Reduction	
n	Area	Delay	Area	Delay	Area	Delay
8	673	11.03	590	9.93	12%	10%
16	2881	23.35	1935	17.24	33%	26%
24	6625	35.67	3979	24.59	40%	31%
32	11905	47.99	6727	31.92	43%	33%

Compared to the conventional method, our method requires between 14 and 37 percent less area and has between 1 and 7 percent less delay. Our method for two's complement tree multipliers provides only a small reduction in delay since the reduction in the delay of the CLA and overflow detection is offset by the increase in delay from conditionally complementing the operands.

32-bit unsigned array multipliers and 32-bit two's complement tree multipliers that use our methods and the conventional methods for overflow detection were also synthesized with the Synopsys Design Compiler and a 0.25 micron CMOS standard cell library with four layers of metal. The designs were first optimized for area and then restricted to have a maximum transition time of 1.5 nanoseconds. Silicon Ensemble then performed place-and-route on the optimized designs, with the restriction of 90 percent row utilization. Delay estimates were made by the compiler using a slow process, 110° C, 2.3 Volt technology file. Area measurements were performed after place-and-route.

Area measurements and worst case delay estimates for these multipliers are shown in Table 8, where UA corresponds to the unsigned array multipliers and ST corresponds to the signed tree multipliers. Area is given in square millimeters and delay is given in nanoseconds. The unsigned array multiplier that uses our method for overflow detection has 49 percent less area and 57 percent less delay than the one that uses the conventional method. The two's complement tree multiplier that uses our method has 39 percent less area and 30 percent less delay than the one that uses the conventional method.

Since the multipliers that use our methods have less rectangular structures than multipliers that use conventional methods for overflow detection or saturation, our methods require careful placement of cells to ensure effective area utilization for custom layouts. When these multipliers are implemented with synthesis and place-and-route tools, the tools do a very effective job of placing the cells in a rectangular layout.

TABLE 7
Signed Dadda Tree Multipliers with Overflow Detection

	Conventional		Proposed		Reduction	
n	Area	Delay	Area	Delay	Area	Delay
8	837	8.59	723	8.29	14%	3%
16	2925	11.12	2141	11.02	27%	1%
24	6298	13.81	4163	13.22	34%	4%
32	10948	15.24	6950	14.19	37%	7%

TABLE 8
32-bit Multipliers with Overflow Detection

Mult.	Conventional		Proposed		Reduction	
Type	Area	Delay	Area	Delay	Area	Delay
UA	.1994	56.19	.0994	24.21	49%	57%
ST	.1998	22.50	.1229	15.71	39%	30%

5 Conclusions

The methods for multiplication with overflow detection and saturation presented in this paper are applicable to array multipliers or tree multipliers for unsigned or two's complement numbers. Compared to conventional methods, they have significantly less area and delay. These methods should also reduce power dissipation due to decreased hardware requirements. They can be used in VLSI or reconfigurable computing systems for applications that do not require the most significant product bits, but that do require overflow detection or saturation.

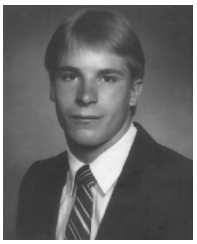
Acknowledgments

This material is based upon work supported by Sandia National Laboratories under Grant Number BF-1029 and the US National Science Foundation under Grant Number MIP-9703421. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Sandia National Laboratories or the National Science Foundation. The authors are grateful to Kwok Kee Ma from Sandia National Laboratories, who supervised the work performed on the Sandia Secure Microprocessor, to K'Andrea Bickerstaff and Mehran Jalaliani of Cirrus Logic for providing area measurements and delay estimates from the Synopsis Design Compiler and Silicon Ensemble, and to the anonymous reviewers for their excellent suggestions regarding revisions to the paper.

References

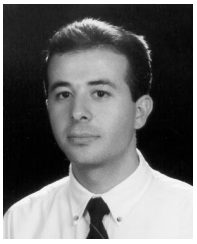
[1] D. Tabak, *RISC Systems and Applications.* John Wiley & Sons, 1996.
[2] J.C. Hoffman and R. Kitai, "Parallel Multiplier Circuit," *Electronic Letters,* vol. 4, p. 178, May 1968.
[3] S.D. Pezaris, "A 40 ns 17-bit Array Multiplier," *IEEE Trans. Computers,* vol. 20, no. 4, pp. 442-447, Apr. 1971.
[4] K.Z. Pekmestzi, "Multiplexer-Based Array Multipliers," *IEEE Trans. Computers,* vol. 48, no. 1, pp. 15-23, Jan. 1999.
[5] C.S. Wallace, "Suggestion for a Fast Multiplier," *IEEE Trans. Electronic Computers,* vol. 13, pp. 14-17, 1964.
[6] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza,* vol. 34, pp. 349-356, 1965.
[7] K. Bickerstaff, M.J. Schulte, and E.E. Swartzlander Jr., "Parallel Reduced Area Multipliers," *J. VLSI Signal Processing,* vol. 9, pp. 181-192, 1995.
[8] P.J. Song and G.D. Micheli, "Circuit and Architecture Trade-Offs for High-Speed Multiplication," *IEEE J. Solid-State Circuits,* vol. 26, no. 9, pp. 1,184-1,198, Sept. 1991.
[9] I. Koren, *Computer Arithmetic and Algorithms.* Brookside Court Publishers, 1998.
[10] T. Lindholm and F. Yelin, *The Java Virtual Machine Specification.* Addison-Wesley, 1996.
[11] K. Guttag, "Built-In Overflow Detection Speeds 16-bit Microprocessor Arithmetic," *EDN,* vol. 28, no. 1, pp. 133-135, Jan. 1983.

[12] J.L. Hennessy and D.A. Patterson, *Computer Architecture a Quantitative Approach,* second edition, pp. A-11, Morgan Kaufmann, 1996.
[13] P. Lapsley, *DSP Processor Fundamentals: Architectures and Features.* IEEE Press, 1997.
[14] N. Yadav, M.J. Schulte, and J. Glossner, "Parallel Saturating Fractional Arithmetic Units," *Proc. Ninth Great Lakes Symp. VLSI,* pp. 214-217, Mar. 1999.
[15] F. Mintzer and A. Peled, "A Microprocessor for Signal Processing, the RSP," *IBM J. Research and Development,* vol. 26, no. 4, pp. 413-423, July 1982.
[16] A. Peleg and U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro,* vol. 6, no. 4, pp. 42-50, Aug. 1996.
[17] A. Landauro and J. Lienard, "On Overflow Detection and Correction in Digital Filters," *IEEE Trans. Computers,* vol. 24, no. 12, pp. 1,226-1,228, Dec. 1975.
[18] P.D. Pai and A. Tran, "Overflow Detection in Multioperand Addition," *Int'l J. Electronics,* vol. 73, no. 3, pp. 461-469, Sept. 1992.
[19] D.G. East and J.W. Moore, "Overflow Indication in Two's Complement Arithmetic," *IBM Technical Disclosure Bulletin,* vol. 19, no. 8, pp. 3,135-3,136, Jan. 1977.
[20] Z. Wang, G.A. Jullien, and W.C. Miller, "A New Design Technique for Column Compression Multipliers," *IEEE Trans. Computers,* vol. 44, no. 8, pp. 962-970, Aug. 1995.
[21] V.J. Oklobdzija and D. Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," *IEEE Trans. VLSI,* vol. 3, no. 2, pp. 292-301, June 1995.
[22] C.R. Baugh and B.A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Trans. Computers,* vol. 22, no. 12, pp. 1,045-1,047, Dec. 1973.
[23] P.E. Blankenship, "Comments on a Two's Complement Parallel Array Multiplication Algorithm," *IEEE Trans. Computers,* vol. 23, p. 1,327 1974.
[24] J.A. Gibson and R.W. Gibbard, "Synthesis and Comparison of Two's Complement Parallel Multipliers," *IEEE Trans. Computers,* vol. 24, no. 10, pp. 1,020-1,027, Oct. 1975.
[25] A.D. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanics and Applied Mathematics,* vol. 4, pp. 236-240, 1951.
[26] 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. Computers,* vol. 39, no. 8, pp. 1,006-1,015, Aug. 1990.
[27] R. De Mori and A. Serra, "A Parallel Structure for Signed-Number Multiplication and Addition," *IEEE Trans. Computers,* vol. 21, no. 12, pp. 1,453-1,454, Dec. 1972.
[28] M. Zheng and A. Albicki, "Low Power and High Speed Multiplication Design through Mixed Number Representations," *Proc. Int'l Conf. Computer Design,* pp. 566-570, 1995.



Michael J. Schulte received BS degrees in electrical engineering and computer science from the University of Wisconsin-Madison in 1991, and MS and PhD degrees in electrical and computer engineering from the University of Texas at Austin in 1993 and 1996, respectively. Since 1996, he has been an assistant professor in the Electrical Engineering and Computer Science Department at Lehigh University, where he directs the Computer Architecture and Arithmetic Research Lab. His research and teaching interests include computer architecture, computer arithmetic, embedded systems, and processor design and implementation. In 1997, he received a US National Science Foundation CAREER Award to research hardware support for accurate and reliable numerical computations. He is a member of the IEEE.

Pablo I. Balzola received his BS degree in electrical engineering from Lehigh University in 1997 and his MS degree in computer science in 1999. He is currently a PhD candidate in the Electrical Engineering and Computer Science Department at Lehigh University. His research is focused on hardware and software for cryptography. He is supported by a grant from Sandia National Laboratories.



Ahmet Akkas received the BS degree in electrical engineering from Gazi University, Turkey, in 1990 and the MS degree in computer science from Lehigh University in 1996. Currently, he is a PhD candidate in the Electrical Engineering and Computer Science Department at Lehigh University. His research interests include computer arithmetic and computer architecture for reliable computing. He is supported by a grant from the US National Science Foundation.

Robert W. Brocato received the BS degree in electrical engineering from the University of California, San Diego, in 1983 and the MS degree in electrical engineering from the University of California, Santa Barbara, in 1984. In 1985, he joined Sandia National Labs, Albuquerque, New Mexico, where he is currently with the Digital ASIC Department. He is responsible for designing microprocessors and mixed signal integrated circuits.