SAND2000-0125J

Integer Multiplication with Overflow Detection or Saturation

Michael J. Schulte, Pablo I. Balzola, and Ahmet Akkas
Computer Architecture and Arithmetic Laboratory
Electrical Engineering and Computer Science Department
Lehigh University
Bethlehem, PA 18015

Robert W. Brocato
Digital Microelectronics
Sandia National Laboratories
Albuquerque, NM 87185

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, 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]. The 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 is 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 the results are too large or too small to represent, they saturate to the most positive or most negative representable number [13], [14], [15].

Previous research on overflow detection and saturation has focussed on fractional operands [14], [16], or operations other than multiplication [17], [18]. 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 detection and saturation are much easier, since overflow 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

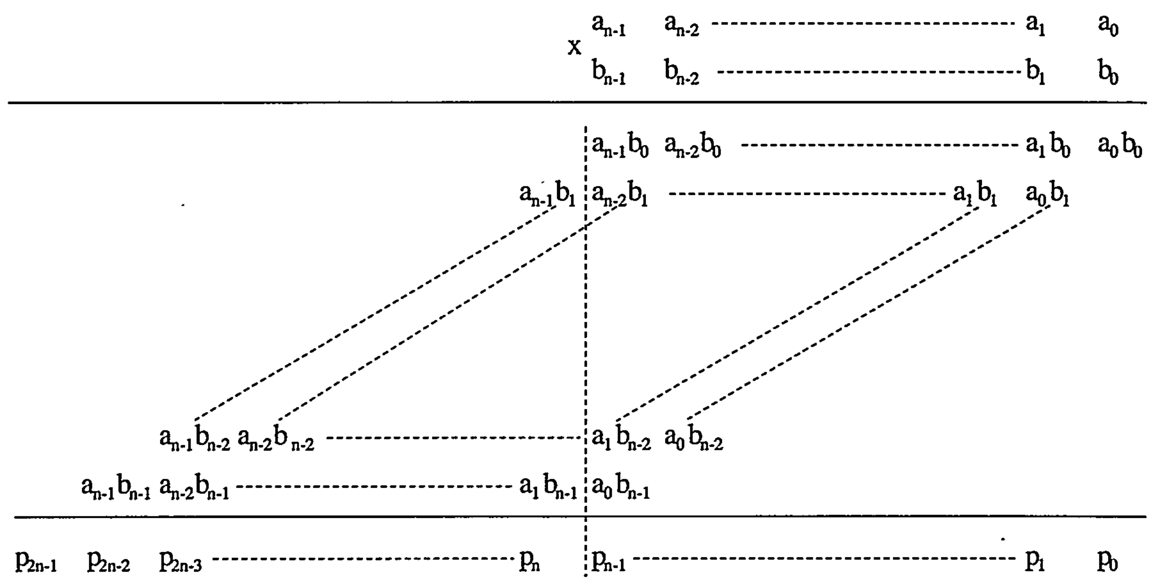

x
a_{n-1} a_{n-2} a_1 a_0
b_{n-1} b_{n-2} b_1 b_0
$a_{n-1}b_0$ $a_{n-2}b_0$ a_1b_0 a_0b_0
$a_{n-1}b_1$ $a_{n-2}b_1$ a_1b_1 a_0b_1
$a_{n-1}b_{n-2}$ $a_{n-2}b_{n-2}$ a_1b_{n-2} a_0b_{n-2}
$a_{n-1}b_{n-1}$ $a_{n-2}b_{n-1}$ a_1b_{n-1} a_0b_{n-1}
p_{2n-1} p_{2n-2} p_{2n-3} p_n p_{n-1} p_1 p_0

Figure 1: Unsigned Multiplication Matrix for $P = A \cdot B$

with unsigned and two's complement multiplication, respectively. Section 4 gives area and delay estimates for tree 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 Overflow/Saturation for Unsigned Multipliers

Figure 1 shows the partial product 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

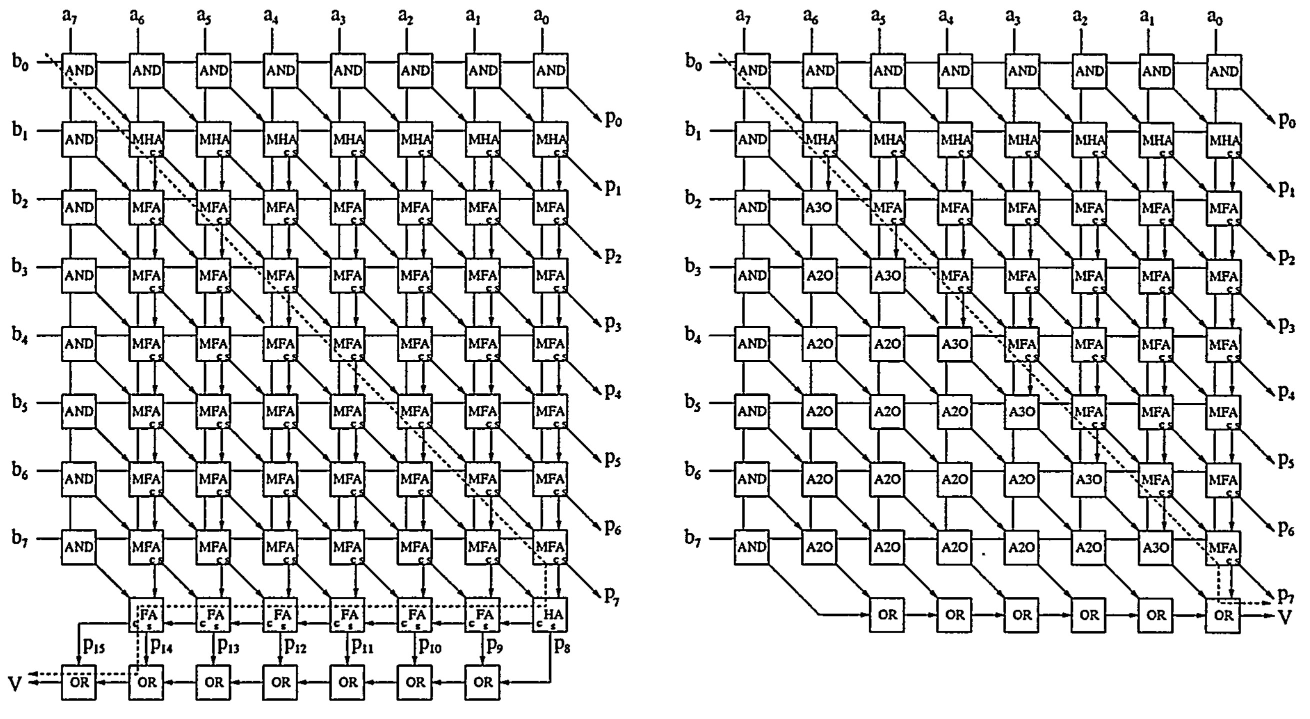
$$A = \sum_{i=0}^{n-1} a_i \cdot 2^i \qquad B = \sum_{i=0}^{n-1} b_i \cdot 2^i \qquad P = \sum_{i=0}^{2n-1} p_i \cdot 2^i$$

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

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

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 machines 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 can be



a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
AND
MHA
MFA
FA
HA
OR
A2O
A3O
p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7
p_8 p_9 p_{10} p_{11} p_{12} p_{13} p_{14} p_{15}
V

(a) Conventional Overflow Detection

(b) Proposed Overflow Detection

Figure 2: Unsigned 8-bit Array Multipliers

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

$$< p_i >= p_i + V \qquad (0 \leq i \leq n-1)$$

2.1 Unsigned array multipliers

A block diagram of an unsigned 8-bit array multiplier that performs conventional overflow detection is shown in Figure 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 a 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 a 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 Figure 2a indicates a typically worst case delay path through the multiplier. The worst case delay is approximately equal to the delay through one AND gate[1], two OR gates,

[1]All partial products bits are available after one AND gate delay.

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 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 Figure 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 Figure 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. This approach is illustrated in Figure 2b for an 8-bit array multiplier. In this figure, the MFAs below the diagonal dashed line have been replaced by AND-OR cells, which are referred to as A3O cells and A2O cells. Each A3O cell consists of an AND gates that computes a partial product bit, and a 3-input OR gate that ORs the partial product bit and two signals from neighboring cells. Each A20 cell consist of an AND gates that computes a partial product bit, and a 2-input OR gate that ORs the partial product bit and one signal from a neighboring cell. An n-bit array multiplier that uses our technique requires n^2 AND gates, $(n^2-3n+2)/2$ 2-input OR gates, $(n-2)$ 3-input OR gates, $(n-1)$ HAs, and $(n^2-3n+2)/2$ FAs. Depending on the technology used for the multiplier, the actual implementation of the cells may vary. For example, with static CMOS, the AND-OR cells may be implemented using NAND/NOR gates or complex gates to improve area and performance. Our method for overflow detection has $(n^2-n)/2$ fewer adders and $(n^2-3n)/2$ more OR gates than the conventional method.

Since the AND-OR cells have less area and delay than the modified adder cells, a significant reduction in area and delay is achieved. The dashed line in Figure 2b indicates a typical worst case delay path through this multiplier. This 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.

To perform saturating multiplication with either our method or the conventional method, the V is ORed with the n least significant bits of the partial product. This requires n additional OR gates, and the worst case delay path increase 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 two rows, which are summed using a fast carry-propagate adder (CPA). Figure 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], [19], [20],they are useful to examine because the number and type of cells required to implement them can easily be determine based on n. Results similar to the ones presented here are expected for other types of tree multipliers.

In Figure 3a, a partial product bit is represented by a dot, the outputs of a full adder are

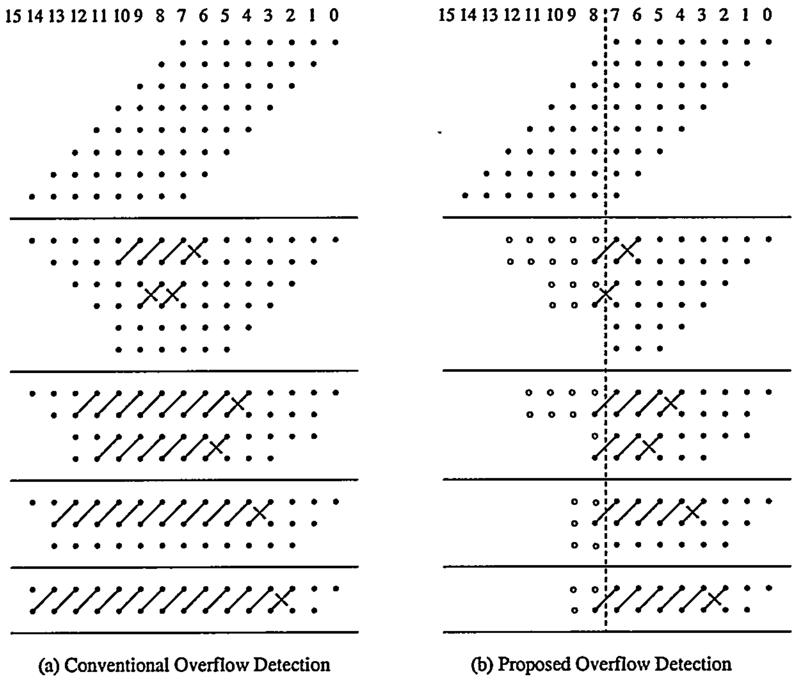


Figure 3: Unsigned 8-bit Dadda Tree Multipliers

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. A 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]. Computing overflow using the conventional method, requires $(n-1)$ OR gates.

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 full adder 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 performed with a tree of $(n-1)$ 2-input OR gates, which has a delay equal to $\lceil \log_2(n) \rceil$ 2-input OR gates.

When detecting overflow or performing saturation, the n most significant bits of the product are not needed, adders are not required in columns n to $(2n-2)$. With our method, the adders in these columns are replaced by a tree of OR gates, which OR the partial products bits in columns n to $(2n-2)$ and the carries going into column n. Figure 3b illustrates the hardware saving that is 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 $n(n-1)/2$ 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

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

Table 1: Number of Stages s for n-Bit Dadda Tree Multipliers.

overflow detection with this technique is

$$n(n-1)/2 + (n-1) - 1 = (n^2 + n - 4)/2$$

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 a $(n-1)$-bit carry-propagate adder. This is $(n^2-3n+2)/2$ fewer adders and $(n^2+n-4)/2$ more OR gates than the conventional method. The size of the CPA is also reduced by $(n-1)$ bits.

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 of partial product reduction (s FA delays), plus the delay of a $(n-1)$-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 Overflow/Saturation for Two's Complement Multipliers

With two's complement integer multiplication, the multiplicand $A = a_{n-1}a_{n-2}\ldots a_1a_0$ is multiplied by the multiplier $B = b_{n-1}b_{n-2}\ldots b_1b_0$ to produce $P = p_{2n-1}p_{2n-2}\ldots p_1p_0$. The values of A, B, and P are

$$A = -a_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i \qquad B = -b_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i \qquad P = -p_{2n-1}\cdot 2^{2n-1} + \sum_{i=0}^{2n-2} p_i \cdot 2^i$$

The most significant bit of a two's complement number, called the sign bit, has a negative weight [9]. If the sign bit is one, the number is negative; otherwise, the number is zero or positive.

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

$$P \leq -2^{n-1} - 1 \qquad \text{or} \qquad P \geq 2^{n-1}$$

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

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

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 $-2^n = 10\ldots0$ when overflow occurs and $t = a_{n-1} \oplus b_{n-1} = 1$; and it saturates to $2^{n-1} - 1 = 01\ldots1$ when overflow occurs and $t = a_{n-1} \oplus b_{n-1} = 0$. If overflow does not occur, then the n least significant bits of the product are returned. The bits of the saturated product $<P>=<p_{n-1}><p_{n-2}>\ldots<p_1><p_0>$ are computed as

$$\begin{aligned} <p_{n-1}> &= Vt + \overline{V}p_{n-1} \\ <p_i> &= V\overline{t} + \overline{V}p_i \qquad (0 \leq i \leq n-2) \end{aligned}$$

This is implemented using an XOR gate to compute t and an n-bit 2-to-1 multiplexor to select the correct values for the saturated partial product bits.

Several techniques have been proposed to implement two's complement multipliers including Pezaris arrays [3], variations of the Baugh-Wooley algorithm [21], [22], [23], and Booth's algorithm [24] and its extensions [25]. 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. With Pezaris arrays, adder and subtractor cells are used to combine negative and positive partial products. With the Baugh-Wooley algorithm and its extensions, the partial product matrix is modified so that it contains only positive partial product bits. With Booth's original algorithm, pairs of multiplier bits are encoded to determine if -1, 0, or 1 times the multiplicand is used as a partial product. Extensions to Booth's algorithm encode more than two multiplier bits at a time to reduce the number of partial products that need to be accumulated. After the partial products are generated using one of these techniques, they are then reduced using array or tree multiplier methods.

Unfortunately, previous techniques for two's complement multiplication do not work well for overflow detection or saturation without computing the most significant product bits. To illustrate this, Figure 4 shows partial product matrices using the Baugh-Wooley [21] and the Complemented Partial Product Word Correction [7], [23] algorithms for n-bit multiplies. In these types of multipliers, 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 significant bits of the product.

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

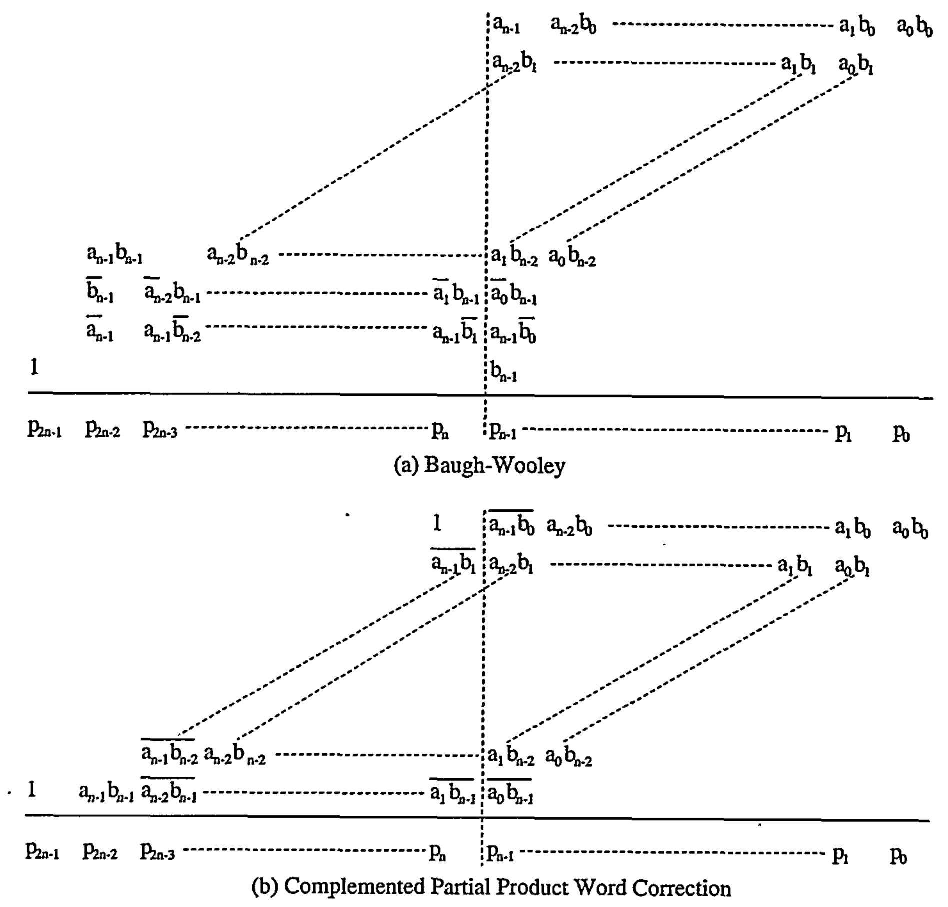

a_{n-1} $a_{n-2}b_0$ a_1b_0 a_0b_0
$a_{n-2}b_1$ a_1b_1 a_0b_1
$a_{n-1}b_{n-1}$ $a_{n-2}b_{n-2}$ a_1b_{n-2} a_0b_{n-2}
$\overline{b}_{n-1}$ $\overline{a}_{n-2}b_{n-1}$ $\overline{a}_1b_{n-1}$ $\overline{a}_0b_{n-1}$
$\overline{a}_{n-1}$ $a_{n-1}\overline{b}_{n-2}$ $a_{n-1}\overline{b}_1$ $a_{n-1}\overline{b}_0$
1 b_{n-1}
p_{2n-1} p_{2n-2} p_{2n-3} p_n p_{n-1} p_1 p_0
(a) Baugh-Wooley
1 $\overline{a_{n-1}b_0}$ $a_{n-2}b_0$ a_1b_0 a_0b_0
$\overline{a_{n-1}b_1}$ $a_{n-2}b_1$ a_1b_1 a_0b_1
$\overline{a_{n-1}b_{n-2}}$ $a_{n-2}b_{n-2}$ a_1b_{n-2} a_0b_{n-2}
1 $a_{n-1}b_{n-1}$ $\overline{a_{n-2}b_{n-1}}$ $\overline{a_1b_{n-1}}$ $\overline{a_0b_{n-1}}$
p_{2n-1} p_{2n-2} p_{2n-3} p_n p_{n-1} p_1 p_0
(b) Complemented Partial Product Word Correction

Figure 4: Two's Complement Multiplication Matrices for $P = A \cdot B$

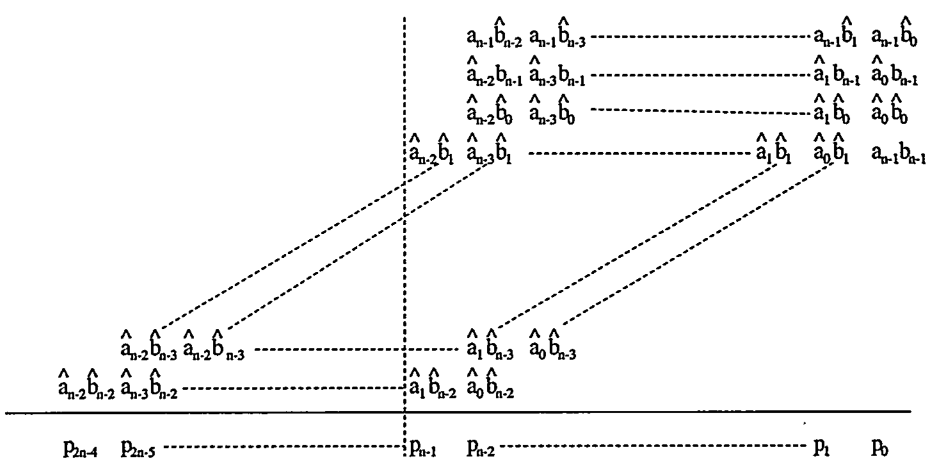

$a_{n-1}\hat{b}_{n-2}$ $a_{n-1}\hat{b}_{n-3}$ $a_{n-1}\hat{b}_1$ $a_{n-1}\hat{b}_0$
$\hat{a}_{n-2}b_{n-1}$ $\hat{a}_{n-3}b_{n-1}$ $\hat{a}_1 b_{n-1}$ $\hat{a}_0 b_{n-1}$
$\hat{a}_{n-2}\hat{b}_0$ $\hat{a}_{n-3}\hat{b}_0$ $\hat{a}_1\hat{b}_0$ $\hat{a}_0\hat{b}_0$
$\hat{a}_{n-2}\hat{b}_1$ $\hat{a}_{n-3}\hat{b}_1$ $\hat{a}_1\hat{b}_1$ $\hat{a}_0\hat{b}_1$ $a_{n-1}b_{n-1}$
$\hat{a}_{n-2}\hat{b}_{n-3}$ $\hat{a}_{n-2}\hat{b}_{n-3}$ $\hat{a}_1\hat{b}_{n-3}$ $\hat{a}_0\hat{b}_{n-3}$
$\hat{a}_{n-2}\hat{b}_{n-2}$ $\hat{a}_{n-3}\hat{b}_{n-2}$ $\hat{a}_1\hat{b}_{n-2}$ $\hat{a}_0\hat{b}_{n-2}$
p_{2n-4} p_{2n-5} p_{n-1} p_{n-2} p_1 p_0

Figure 5: Multiplication Matrix for $| P |=| A | \cdot | B |$

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

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

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 $| A | \cdot | B |$ is shown in Figure 5. This multiplication matrix has $(n+1)$ rows, $(2n-3)$ columns, and n^2 partial products.

Since the product of magnitudes is always non-negative, 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 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

Figure 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 Figure 2a, except that $(n-1)$ of the AND gates on the left side of the array are replaced by NAND gates, $(n-1)$ MFAs toward the bottom of the array are replaced by negating modified full adders (NMFAs), one of the half adders 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 Figure 4b. The SHA takes sum and carry bits from the previous row and adds them with '1' to produce new sum and carry bits. As noted in [7], the specialized half adder has approximately the same amount of area and delay as a regular half adder. The SHA and the inverter that complements p_{2n-1} add the ones shown in columns n and $2n-1$ of Figure 4b [7]. The n XOR gates perform $p_i \oplus p_{n-1}$, for $n \leq i \leq 2n-1$.

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.

Figure 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 Figure 7b, except 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.

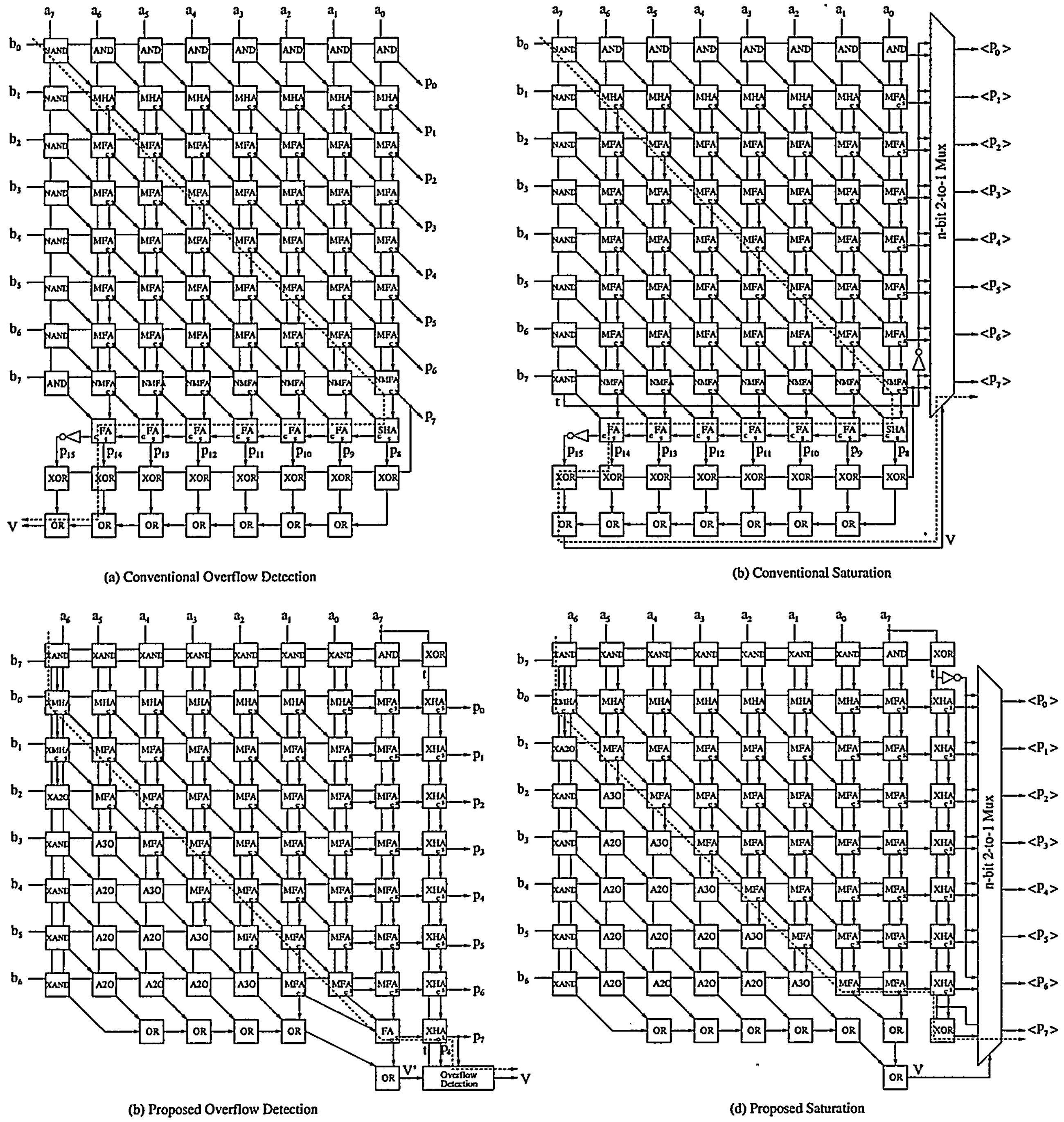
Figure 6c shows an 8-bit two's complement array multiplier that uses our method for overflow detection. The core of the array multiplier computes $|P| = |A| \cdot |B|$, as shown in Figure 5. Each XAND cell in the top row consist of an XOR gate, which produces $\hat{a}_i = a_{n-1} \oplus a_i$, and an AND gate, which produces $b_{n-1}\hat{a}_i$. Similarly, the XAND, XMHA, and XA2O cells on the left side of the array each consist of an XOR gate that computes $\hat{b}_i = b_{n-1} \oplus b_i$, and an AND gate, MHA cell, or A2O cell. On the right side of the array, n XHA cells produce the two's complement of $|P|$, when $t = a_{n-1} \oplus b_{n-1} = 1$. Each XHA cell consists of an XOR gate and a HA. When $t = 1$, the XOR gates invert the bits of $|P|$ and the half adders add one to the inverted bits of $|P|$.

The 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 is one greater than the most positive 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 $p_n p_{n-1} \cdots p_0$ 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 $p_n = 1$ (i.e., $P \geq 2^{n-1}$), or when $t = 1$ and $p_n = 0$ and $p_{n-1} = 0$ (i.e., $P \leq -2^{-n-1} - 1$). Thus, the overflow detection logic computes

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

where V' is the OR of the partial product bits in columns n to $2n-4$ and the carries into column n, as shown in Figure 6c. An n-bit array multiplier that implements overflow detection using our method has three inverters, (n^2+3) AND gates, $(n^2-7n+20)/2$ 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 product is saturated whenever $|P| \geq 2^{n-1}$, 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 $|P|$ need to be calculated. This allows the adders that sum the



(a) Conventional Overflow Detection

(b) Conventional Saturation

(b) Proposed Overflow Detection

(d) Proposed Saturation

Figure 6: Two's Complement 8-Bit Array Multipliers

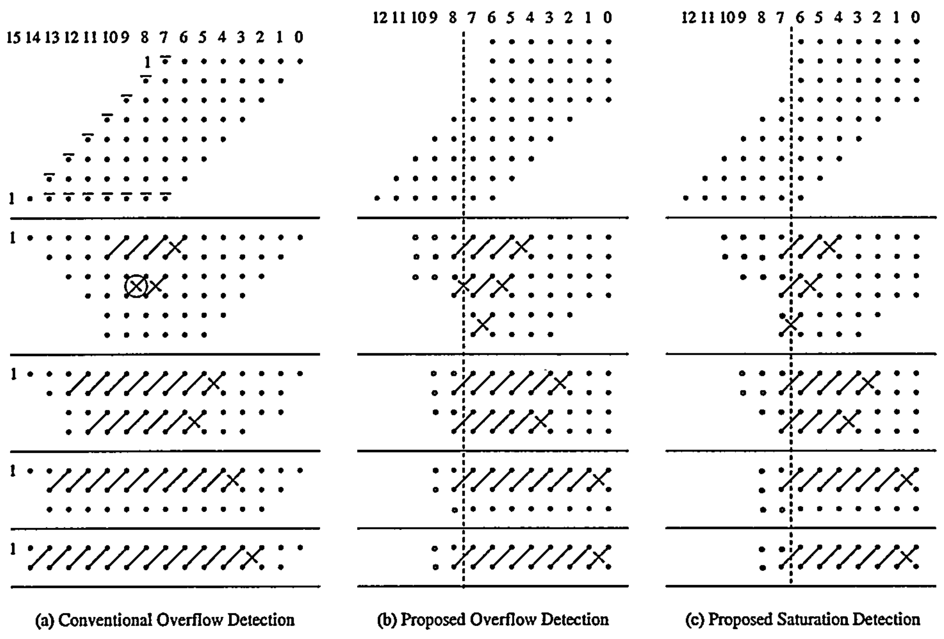

(a) Conventional Overflow Detection
(b) Proposed Overflow Detection
(c) Proposed Saturation Detection

Figure 7: Two's Complement 8-Bit Dadda Tree Multipliers

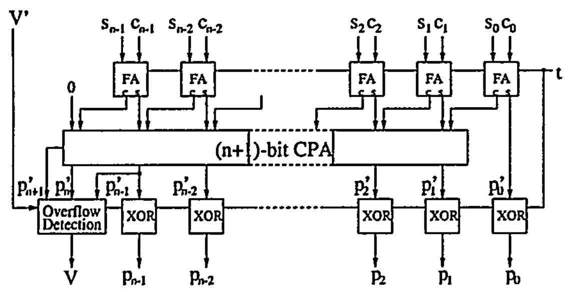
partial product bits in column n to be replaced by OR gates and simplifies the overflow detection logic.

Figure 7d shows an 8-bit two's complement array multiplier that uses our method for saturation. Since the product saturates whenever, $| P | \geq 2^{n-1}$, $| P |$ 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 $| P |$ when $t = 1$. This multiplier requires (n^2+3) AND gates, $(n^2-7n+20)/2$ 2-input OR gates, $(n-4)$ 3-input OR gates, $(3n-1)$ XOR gates, $2n$ HAs, $(n^2+n-6)/2$ FAs, and a n-bit 2-to-1 multiplexor. The worst case delay through this multiplier is equal to the delay through one AND gates, three XOR gates, two HAs, $(n-1)$ FAs, and an n-bit 2-to-1 multiplexor.

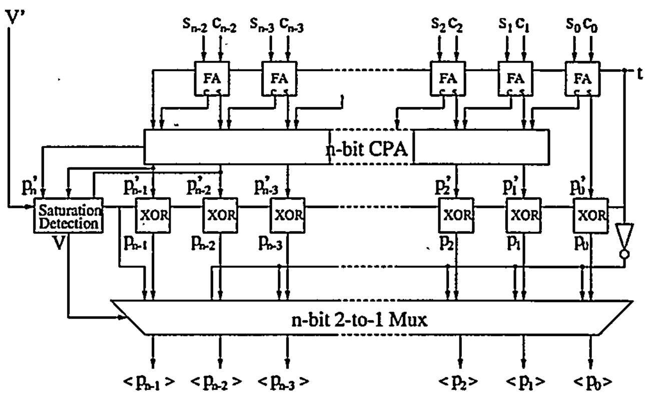
3.2 Two's complement tree multipliers

Figure 7a shows the dot diagram of a 8-bit two's complement tree multiplier that uses the Complemented Partial Product Word Correction algorithm [23] to generate the partial products (see Figure 4b) 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 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 [6], [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 an extra XOR delay from XORing p_{n-1} with p_n to p_{2n-1}.

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



(a) Overflow Detection/Correction



(b) Saturation Detection/Correction

Figure 8: Two's Complement Overflow and Saturation Detection/Correction

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 $(n+1)$-bit CPA to compute the product, and then uses n XOR gates to conditionally invert the product bits. This approach is shown in Figure 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 more significant pre-inverted product bits p'_{n+1}, p'_n, and p'_{n-1}, and the 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 $\mid P \mid + 0 \geq 2^{n-1}$. Simplified logic equations for determining overflow are

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

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 when reducing the partial products to S and C. The logic for computing V is implemented using three 2-input AND gates, four 2-input OR gates, and an inverter. From the time the pre-inverted partial product bits are ready, computing V has a delay equivalent to one AND gate plus three 2-input OR gates.

Figure 8b illustrates the hardware savings achieved by our method for overflow detection (see Figure 5) 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 the overflow detection logic, as shown in Figure 8a. The $2n$-bits in the bottom two rows of Figure 8b to the right of the dashed line correspond to S and C in Figure 6a. 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 $(n + 1)$-bit CPA. The worst case delay through the multiplier and overflow detection logic is equivalent to the delay of one inverter, two AND gates, three OR gates, two XOR gates, $(s + 1)$ FAs, and an $(n + 1)$-bit CPA.

Similar to our method for saturating array multipliers, our method for saturating tree multiplier 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 $S + C = | P |$ and $t \ldots t$, using a $(n-1)$-bit carry-save adder and a n-bit CPA. This is shown in Figure 8b, The saturation logic implements

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

which requires three 2-input AND gates, three 2-input OR gates, and an inverter. An n-bit 2-to-1 multiplexor saturates the product when $V = 1$.

Figure 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 [26], 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 technique differs from the technique presented in [26], in that our technique does not compute the most significant bits of the product and our method for correcting the final product does not require carry propagate addition. In [27], 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 will always have zeros in their most significant bits [27]. Similar reductions in power dissipation are expected using our technique.

4 Area and Delay Estimates

Java programs were used to generate gate-level VHDL code 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 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 2 gives area and delay estimates for unsigned array multipliers using the conventional method and our proposed method for overflow detection.[2] Compared to the conventional method, our method has between 42% and 44% less area and between 39% and 42% less delay. These reductions in area and delay occur because the n most significant partial product bits are not computed and because the adders that compute these bits are replaced by OR gates that detect overflow.

Table 3 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

[2]We plan to include area and delay estimates for two's complement array multipliers with overflow detection in the final version of the paper. We anticipate that two's complement array multipliers that implement our method will greater area and delay than unsigned array multipliers of the same size, due to the overhead of conditionally complementing the input and output operands.

n	Conventional Method		Proposed Method		Percent Reduction	
	Area (gates)	Delay (ns)	Area (gates)	Delay (ns)	Area	Delay
8	662	10.70	372	6.34	44%	41%
16	2862	23.02	1644	13.93	43%	39%
24	6598	35.34	3793	21.30	43%	40%
32	11870	47.66	6845	28.99	42%	42%

Table 2: Estimates for Unsigned Array Multipliers with Overflow Detection.

method, our method has about 47% less area and has between 23% and 28% less delay. As described in Section 2.2, the reduction in area is due to replacing $(n^2 - 3n + 2)/2$ adders with $(n^2 + n - 4)/2$ OR gates, and reducing the size 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..

n	Conventional Method		Proposed Method		Percent Reduction	
	Area (gates)	Delay (ns)	Area (gates)	Delay (ns)	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 3: Estimates for Unsigned Dadda Tree Multipliers with Overflow Detection.

Table 4 gives area and delay estimates for two's complement Dadda tree multipliers using the conventional method and our proposed method for overflow detection. Compared to the conventional method, our method requires between 14% and 37% less area and has between 1% and 7% less delay. The percent reductions in area and delay are less than the percent reductions for the unsigned tree multipliers, due to the overhead of conditionally complementing the input and output operands. As the multiplier size increases, the percent reduction in area due to using 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 OR gates increases quadratically. 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.

n	Conventional Method		Proposed Method		Percent Reduction	
	Area (gates)	Delay (ns)	Area (gates)	Delay (ns)	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 4: Estimates for Two's Complement Dadda Tree Multipliers with Overflow Detection.

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 require overflow detection or saturation.

Acknowledgments

This material is based upon work supported by Sandia National Laboratories under Grant No. BF-1029 and the National Science Foundation under Grant No. 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.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

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 Transactions on Computers*, vol. 20, pp. 442–447, April 1971.

[4] K. Z. Pekmestzi, "Multiplexer-Based Array Multipliers," *IEEE Transactions on Computers*, vol. C-48, pp. 15–23, January 1999.

[5] C. S. Wallace, "Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-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," *Journal of 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 Journal of Solid-State Circuits*, vol. 26, pp. 1184–1198, September 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, pp. 133–135, January 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, J. Glossner, and M. J. Schulte, "Parallel Saturating Fractional Arithmetic Units," in *Proceedings of the Ninth Great Lakes Symposium on VLSI*, pp. 214–217, March 1999.

[15] F. Mintzer and A. Peled, "A Microprocessor for Signal Processing, the RSP," *IBM Journal of Research & Development*, vol. 26, pp. 413–423, July 1982.

[16] A. Landauro and J. Lienard, "On Overflow Detection and Correction in Digital Filters," *IEEE Transactions on Computers*, vol. C-24, pp. 1226–1228, December 1975.

[17] P. D. Pai and A. Tran, "Overflow Detection in Multioperand Addition," *International Journal of Electronics*, vol. 73, pp. 461–469, September 1992.

[18] D. G. East and J. W. Moore, "Overflow Indication in Two's Complement Arithmetic," *IBM Technical Disclosure Bulletin*, vol. 19, pp. 3135–3136, January 1977.

[19] Z. Wang, G. A. Jullien, and W. C. Miller, "A New Design Technique for Column Compression Multipliers," *IEEE Transactions on Computers*, vol. C-44, pp. 962–970, August 1995.

[20] V. J. Oklobdzija and D. Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 3, pp. 292–301, June 1995.

[21] C. R. Baugh and B. A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. C-22, pp. 1045–1047, December 1973.

[22] P. E. Blankenship, "Comments on A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. C-23, p. 1327, 1974.

[23] J. A. Gibson and R. W. Gibbard, "Synthesis and Comparison of Two's Complement Parallel Multipliers," *IEEE Transactions on Computers*, vol. C-24, pp. 1020–1027, October 1975.

[24] A. D. Booth, "A Signed Binary Multiplication Technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, pp. 236–240, 1951.

[25] H. Sam and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and Its Proof with Application in Multiplier Implementations," *IEEE Transactions on Computers*, vol. 39, no. 8, pp. 1006–1015, 1990.

[26] R. De Mori and A. Serra, "A Parallel Structure for Signed-Number Multiplication and Addition," *IEEE Transactions on Computers*, vol. C-21, pp. 1453–1454, December 1972.

[27] M. Zheng and A. Albicki, "Low Power and High Speed Multiplication Design through Mixed Number Representations," in *Proceedings of the International Conference on Computer Design*, pp. 566–570, 1995.