Integer Multipliers with Overflow Detection

Mustafa Gok,
Michael J. Schulte, *Senior Member*, *IEEE*, and
Mark G. Arnold, *Member*, *IEEE*

Abstract—This paper presents a general approach for designing array and tree integer multipliers with overflow detection. The overflow detection techniques are based on an analysis of the magnitudes of the input operands. The overflow detection circuits operate in parallel with a simplified multiplier to reduce the overall area and delay.

Index Terms—Computer arithmetic, high-speed arithmetic algorithms, combinational logic, overflow detection, multiplication.

1 Introduction

Overflow occurs when an arithmetic operation produces a result outside of the range of representable numbers. Often, an error flag is generated to indicate that overflow has occurred. Although multiplication of two n-bit integers produces a $2n$-bit product, many architectures only return the n least significant bits of the product and an overflow flag, which is set when the product cannot be represented correctly with only n bits [1], [2], [3]. For example, IBM's PowerPC microprocessor family supports a 32-bit by 32-bit two's complement multiply instruction which returns the least significant 32 bits of the 64-bit product and an overflow flag [4]. The Java Virtual Machine supports two integer multiplication instructions; a 32-bit by 32-bit *imul* instruction, which returns the 32 least significant bits of the product, and a 64-bit by 64-bit *lmul* instruction, which returns the 64 least significant bits of the product [5].

Recently, several techniques for overflow detection have been proposed to eliminate the need to compute all $2n$ bits of the product [6], [7], [8], [9], [10], [11]. Instead, they compute approximately n least significant product bits and have overflow detection logic that executes in parallel with the multiplication. In [6], overflow detection circuits for tree multipliers are presented. These circuits require the multiplier to generate $2n$-bit sum and carry vectors and then use the most significant bits of these vectors to detect overflow in parallel with the final carry propagate addition. Although this approach has less delay than conventional methods for overflow detection, it does not provide a significant reduction in area.

In [10], overflow detection is performed on two's complement operands by counting the leading sign bits. The overflow detection method in [10] is only developed for two's complement multiplication and cannot be used for unsigned multiplication and combined unsigned and two's complement multiplication. Furthermore, the method presented in [10] has $O(n^2)$ gates, while the methods presented in this paper have $O(n)$ gates.

In [11], methods are presented for detecting overflow in unsigned, two's complement, and combined multipliers. Compared to the methods presented in this paper, the methods in [11] require more complex conditions to be tested to determine if overflow has occurred. Furthermore, [11] does not describe how their overflow detections methods are implemented and does not provide area and delay estimates for their overflow detection circuits.

In [7], overflow detection circuits and saturation logic are presented for unsigned and two's complement array and tree multipliers. Although these designs have less area and delay than the designs presented in [6], [10], they have more area and delay than the designs presented in this paper and cannot be readily extended to combined multipliers that work on both unsigned and two's complement operands.

In [8], methods are presented that allow a single multiplier to perform both unsigned and two's complement multiplication with overflow detection and saturation. The overflow detection methods presented in [8] work well for array multipliers, but can increase the worst-case delay path of tree multipliers since they have linear delay. In [9], overflow detection methods that have logarithmic delay and work well for combined unsigned and two's complement tree multipliers are presented.

This paper presents a general approach for designing integer multipliers with overflow detection. This general approach is applied to unsigned, two's complement, and combined integer multipliers. A block diagram of the general design approach is shown in Fig. 1. In this diagram, the simplified integer multiplier, multiplies an n-bit multiplicand, A, by an n-bit multiplier, B, and produces an $(n+1)$-bit product, $P = p_n p_{n-1} \cdots p_1 p_0$. In parallel with the simplified multiplication, preliminary overflow detection logic generates a preliminary overflow flag, V', using only operand bits from A and B as inputs. The final overflow detection logic generates an overflow flag, V, using V', p_n, and p_{n-1} as inputs.

Our approach works well in systems that require only the n least significant product bits, such as hardware implementations of the Java Virtual Machine. Compared to previous techniques for overflow detection, our approach requires less area and delay. It also works well for both array and tree multipliers and does not depend on the internal architecture of the simplified multiplier. This paper is an extension of our research presented in [8] and [9].

2 Unsigned Multipliers

With our proposed overflow detection technique for unsigned integer multiplication, only the $n+1$ least significant product bits, p_0 to p_n, are computed. Overflow, which occurs when $P \geq 2^n$, is detected by testing p_n and the total number of significant bits in A and B. For unsigned numbers, the significant bits include the leftmost 1 and all the bits to its right. For example, $A = 00010001$ has five significant bits and $B = 00000110$ has three significant bits. The number of significant bits determines the bounds for A, B, and P. If A has S_A significant bits and B has S_B significant bits, A, B, and P are bounded by

$$2^{S_A-1} \leq A \leq 2^{S_A} - 1, \tag{1}$$

$$2^{S_B-1} \leq B \leq 2^{S_B} - 1, \tag{2}$$

$$2^{S_A+S_B-2} \leq P \leq 2^{S_A+S_B} - 2^{S_A} - 2^{S_B} + 1. \tag{3}$$

The left side of (3) provides the lower bound on the product. Overflow occurs for unsigned multiplication if $2^n \leq 2^{S_A+S_B-2}$ or, equivalently, if

$$n + 2 \leq S_A + S_B. \tag{4}$$

Thus, overflow always occurs if A and B together have at least $n+2$ significant bits since this implies that $P \geq 2^n$.

- *M. Gok is with Cukorova University, Adana, 01330 Turkey. E-mail: musgok@cu.edu.tr.*
- *M.J. Schulte is with the University of Wisconsin, Madison, WI 53706. E-mail: schulte@engr.wisc.edu.*
- *M.G. Arnold is with Lehigh University, Bethlehem, PA 18015. E-mail: maab@lehigh.edu.*

Manuscript received 4 Oct. 2004; revised 25 July 2005; accepted 2 Dec. 2005; published online 21 June 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0314-1004.

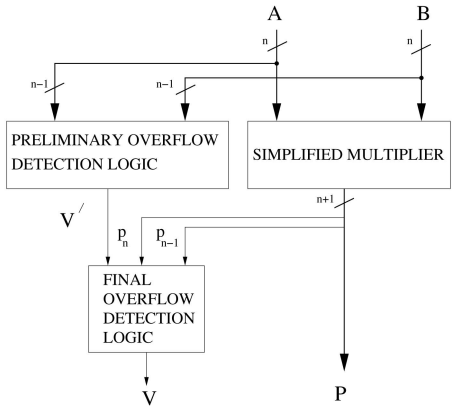

Fig. 1. Proposed integer multiplication with overflow detection.

The right side of (3) gives the upper bound on the product. The product is guaranteed not to overflow when

$$2^{S_A+S_B} - 2^{S_A} - 2^{S_B} + 1 \leq 2^n - 1. \tag{5}$$

Since

$$-2^{S_A} - 2^{S_B} + 1 \leq -1, \tag{6}$$

(5) simplifies to $2^{S_A+S_B} \leq 2^n$ or, equivalently,

$$S_A + S_B \leq n. \tag{7}$$

Thus, overflow never occurs when A and B together have n or fewer significant bits since this implies that $P < 2^n$.

When $S_A + S_B = n + 1$, (3) simplifies to

$$2^{n-1} \leq P \leq 2^{n+1} - 2^{S_A} - 2^{S_B} + 1 \leq 2^{n+1} - 1. \tag{8}$$

Expression (8) implies that, when $S_A + S_B = n + 1$, the product may overflow, but is less than 2^{n+1}. Thus, only product bits p_0 to p_n are needed to represent the product. Overflow occurs in this case if and only if $p_n = 1$.

As shown in [12], the condition given in (4) is detected as

$$V'_u = \sum_{i=1}^{n-1} \sum_{j=1}^{i} a_{n-j} \cdot b_i, \tag{9}$$

where bit summations and bit multiplications correspond to logical ORs and ANDs, respectively. This equation can also be rewritten in long form as

$$\begin{aligned} V'_u = {} & a_{n-1} \cdot b_1 + (a_{n-1} + a_{n-2}) \cdot b_2 + \ldots + \\ & (a_{n-1} + \ldots + a_1) \cdot b_{n-1}. \end{aligned} \tag{10}$$

Equations (9) and (10) test if the total number of significant bits in the input operands is at least $n + 2$, which indicates that overflow must occur. For example, if $a_{n-1} \cdot b_1$ is one, then the total number of significant bits is at least $n + 2$ since A has n significant bits and B has at least two significant bits. Similarly, if $(a_{n-1} + a_{n-2}) \cdot b_2$ is one, then the total number of significant bits is at least $n + 2$ since A has at least $n - 1$ significant bits and B has a least three significant bits. Testing continues until all the bit combinations that are guaranteed to cause overflow are evaluated.

The preliminary overflow flag, V'_u, and the product bit, p_n are ORed to generate the overflow flag V_u, where

$$V_u = V'_u + p_n \tag{11}$$

since overflow occurs if and only if the total number of significant bits in both operands is at least $n + 2$ or p_n is one. The next two sections show how these results are used to design unsigned array and tree multipliers with overflow detection.

2.1 Unsigned Array Multipliers

When implementing the proposed overflow detection method for unsigned array multipliers, directly computing (10) requires $n \cdot (n-1)/2$ 2-input AND gates and $n \cdot (n-1)/2 - 1$ 2-input OR gates. Instead, a significant hardware reduction is achieved by using the following iterative equations [8]:

$$o_{i+1} = o_i + a_{n-i} \quad \text{and} \quad v_{i+1} = v_i + o_{i+1} \cdot b_i \tag{12}$$

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 (12), $V'_u = v_n$ is generated and then V_u is computed as shown in (11).

Fig. 2 shows a block diagram of the proposed unsigned 8-bit array multiplier with overflow detection. In this diagram, a modified half adder (MHA) cell consists of an AND gate and a half adder (HA). The AND gate generates the partial product bit and the HA adds the generated partial product bit with a partial product bit from the previous row to produce sum and carry bits. Similarly, a modified full adder (MFA) consists of an AND gate and a full adder (FA). The AND gate generates a partial product bit and the FA adds the generated bit with sum and carry bits from the previous row.

The simplified unsigned array multiplier computes the least significant product bits, p_0 to p_n. The rest of the multiplier hardware is replaced by overflow detection logic, which operates in parallel with the simplified multiplier. Since the carries generated by additions performed in the nth column of the partial-product matrix are no longer needed, the MFAs along the diagonal that produce p_n are replaced by A2X cells and the HA in the bottom right corner is replaced by an exclusive-or (XOR) gate. Each A2X cell contains one AND gate, which generates a partial product bit, and two XOR gates, which compute the sum of the generated partial product bit and sum and carry bits from the previous row. The logic equation for the A2X cell is $s_{i,j} = (a_i \cdot b_j) \oplus s_{i+1,j-1} \oplus c_{i,j-1}$, where $s_{i,j}$ and $c_{i,j}$ are sum and carry bits from the cell in column i and row j of the multiplication array, respectively. Column 0 is the rightmost column of the array and row 0 is the top row of the array.

The unsigned overflow detection circuit consists of one OR gate and $(n-2)$ overflow detection (OVD) cells. Each OVD cell contains one AND gate and two OR gates, which compute o_{i+1} and v_{i+1} based on (12). Since the overflow detection circuit replaces the more significant half of the multiplier and operates in parallel with the simplified multiplier, large reductions in area and delay are achieved.

2.2 Unsigned Tree Multipliers

A dot diagram for our proposed 8-bit unsigned Dadda tree multiplier is shown in Fig. 3. Dadda multipliers are used because the number of gates for Dadda multipliers can easily be formulated based on n, but the techniques presented in this section work well with other types of multipliers that have logarithmic delay. In Fig. 3, an FA is represented by a diagonal line, an HA is represented by a crossed diagonal line, a 3-input XOR gate is represented by a diagonal line with an x at the bottom, and a 2-input XOR gate is represented by a crossed diagonal line with an x at the bottom. In Fig. 3, partial product bits in columns $(n+1)$ to $(2n-1)$ are not generated and all the hardware that sums these bits and detects overflow is replaced by a simple overflow detection circuit.

Since tree multipliers have logarithmic delay, we modify our implementation of the overflow detection circuit to also have logarithmic delay. As shown in [9], the delay of the unsigned overflow detection circuit is reduced by rewriting (10) as

$$V'_u = A_{n-1} \cdot b_1 + A_{n-2} \cdot b_2 + \ldots + A_1 \cdot b_{n-1}, \tag{13}$$

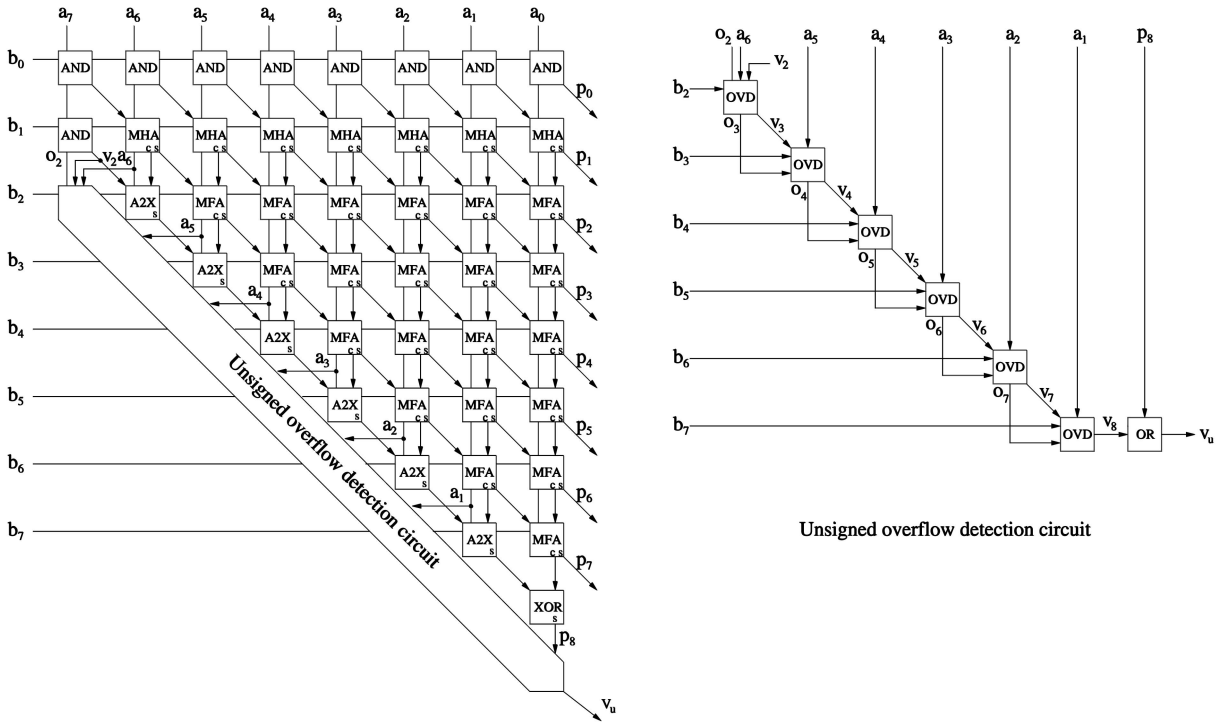

Fig. 2. 8-bit unsigned array multiplier with overflow detection.

where $A_i = \sum_{k=i}^{n-1} a_k$ is computed using a parallel prefix algorithm.

In a size n prefix computation, n inputs, $x_0, x_1, \ldots, x_{n-1}$, and an associative operation, $\circ$, are used to compute n outputs, $y_0, y_1, \ldots, y_{n-1}$ [13]. Each output y_i is dependent on all x_j inputs $(j \leq i)$ as follows:

$$y_i = x_0 \circ x_1 \circ \ldots, x_{i-1} \circ x_i. \tag{14}$$

Equation (14) can be used to sequentially compute y_i for $0 \leq i \leq n-1$ using $(n-1)$ "$\circ$" operations, which is called a serial-prefix computation. Since "$\circ$" is an associative operation, operations in (14) can be performed in any order. For example, $x_0 \circ x_1$ and $x_{i-1} \circ x_i$ can be computed in parallel, which is called a parallel-prefix computation [14].

Various parallel prefix algorithms have been developed for high-speed computations [14], [15], [16], [17]. We use the Brent-Kung algorithm [17] for our proposed overflow detection circuits since it has $O(n)$ gates, while most other parallel prefix algorithms have $O(n \log n)$ gates. It also has a regular layout and maximum fan-in and fan-out of two. Although the delay of the Brent-Kung algorithm is greater than the delay of other parallel prefix algorithms, it is still fast enough not to adversely affect the worst-case delays of our designs.

Fig. 4 shows the block diagram of the proposed overflow detection circuit for an 8-bit unsigned Dadda multiplier. The proposed overflow detection circuit for an n-bit unsigned Dadda multiplier uses an $(n-1)$-bit parallel prefix circuit, followed by $(n-1)$ parallel AND gates, followed by a tree of $(n-2)$ OR gates to compute V_u' based on (13). V_u is then computed based on (11).

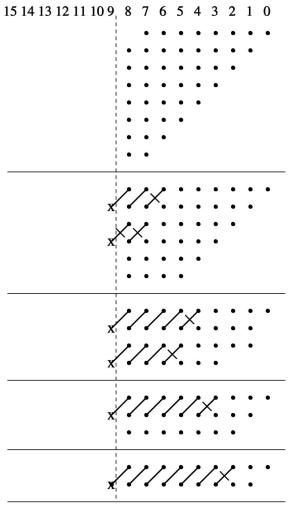

Fig. 3. 8-bit unsigned dadda multiplier with overflow detection.

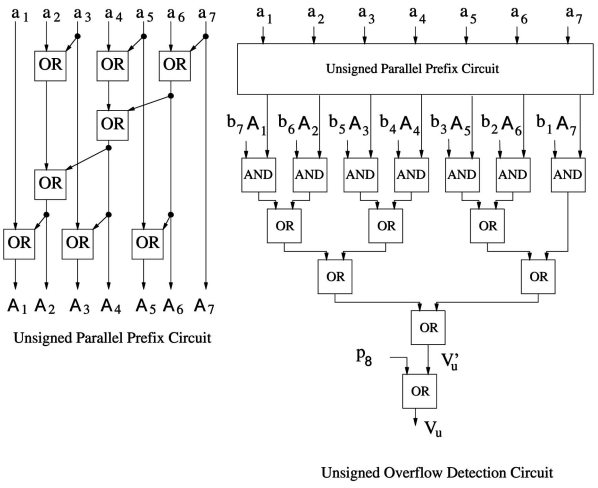

Fig. 4. Overflow detection circuit for an 8-bit unsigned Dadda multiplier.

3 Two's Complement Multipliers

With our proposed overflow detection method for two's complement multiplication, only p_0 to p_n are computed [12]. Overflow, which occurs when $P \geq 2^{n-1}$ or $P < -2^{n-1}$, is detected by comparing p_n and p_{n-1} and testing the total number of significant bits in A and B. For two's complement numbers, the significant bits include the rightmost bit that differs from the sign bit and all the bits to its right. For example, $A = 11100110$ has five significant bits and $B = 00000110$ has three significant bits. The number of significant bits determines the bounds for the magnitudes of A, B, and P. If A has S_A significant bits and B has S_B significant bits, then A, B, and P are bounded by

$$2^{S_A-1} \leq |A| \leq 2^{S_A}, \tag{15}$$

$$2^{S_B-1} \leq |B| \leq 2^{S_B}, \tag{16}$$

$$2^{S_A+S_B-2} \leq |P| \leq 2^{S_A+S_B}. \tag{17}$$

The left side of (17) provides the lower bound on the product. Overflow always occurs for two's complement multiplication if $2^{n-1} \leq 2^{S_A+S_B-2}$ or, equivalently, if

$$n+1 \leq S_A + S_B. \tag{18}$$

Thus, overflow always occurs when A and B together have at least $n+1$ significant bits. The condition given in (18) is detected as

$$V_t' = \widehat{a}_{n-2} \cdot \widehat{b}_1 + \sum_{i=2}^{n-2} \sum_{j=2}^{i+1} \widehat{a}_{n-j} \cdot \widehat{b}_i, \tag{19}$$

where $\widehat{a}_i = a_i \oplus a_{n-1}$, $\widehat{b}_i = b_i \oplus b_{n-1}$, and $V_t' = 1$ when (18) is true. This equation can also be rewritten as

$$\begin{aligned} V_t' = \widehat{a}_{n-2} \cdot \widehat{b}_1 + (\widehat{a}_{n-2} + \widehat{a}_{n-3}) \cdot \widehat{b}_2 + \ldots + \\ (\widehat{a}_{n-2} + \ldots + \widehat{a}_1) \cdot \widehat{b}_{n-2}. \end{aligned} \tag{20}$$

The right side of (17) gives the upper bound on the product. The product is guaranteed not to overflow when $2^{S_A+S_B} < 2^{n-1}$ or, equivalently, when

$$S_A + S_B < n-1. \tag{21}$$

Thus, overflow never occurs if A and B together have less than $n-1$ significant bits.

When $S_A + S_B = n$, (17) simplifies to

$$2^{n-2} \leq |P| \leq 2^n. \tag{22}$$

Expression (22) implies that, when $S_A + S_B = n$, the product may overflow, but its magnitude does not exceed 2^n. Thus, only p_0 to p_n need to be computed to determine if overflow has occurred. When $S_A + S_B = n-1$, (17) simplifies to

$$2^{n-3} \leq |P| \leq 2^{n-1}. \tag{23}$$

Expression (23) implies that, when $S_A + S_B = n-1$, the product may overflow but it does not exceed 2^{n-1}.

When $S_A + S_B = n-1$ or $S_A + S_B = n$, three cases need to be considered based on the signs of the input operands:

1. If both operands are positive, then $P < 2^n$, which means $p_n = 0$ always and $p_{n-1} = 1$ only when overflow occurs.
2. If both operands are negative, then $P \leq 2^n$, which means $p_n = 0$ always and $p_{n-1} = 1$ only when overflow occurs. The one exception is when $P = 2^n$ and then $p_n = 1$ and $p_{n-1} = 0$.
3. If the operands have different signs, then $-2^n < P$, which means $p_n = 1$ always and $p_{n-1} = 0$ only when overflow occurs.

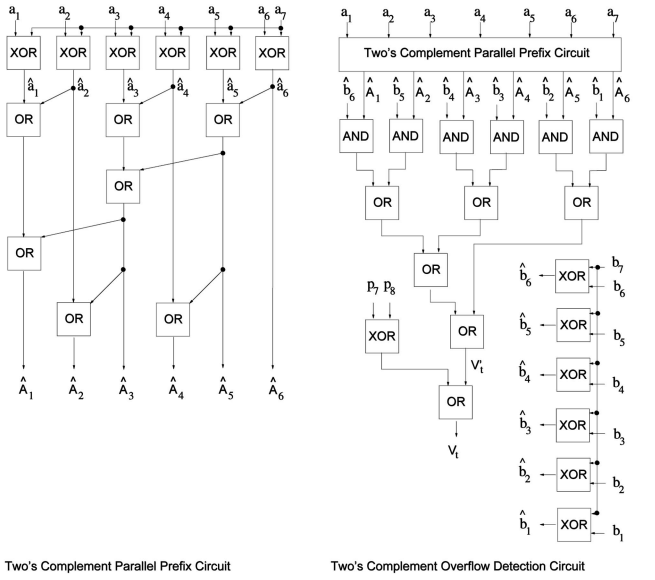

Fig. 5. Overflow detection circuit for an 8-bit two's complement Dadda multiplier.

For all three cases, overflow occurs if and only if $p_n \oplus p_{n-1} = 1$. Thus, the preliminary overflow flag, V_t', and $p_n \oplus p_{n-1}$ are computed in parallel and then ORed to generate V_t as

$$V_t = V_t' + p_n \oplus p_{n-1}. \tag{24}$$

These results can be applied to both two's complement array and tree multipliers [12]. As shown in [9], an overflow detection circuit for tree multipliers can be obtained by rewriting (20) as:

$$V_t' = \widehat{A}_{n-2} \cdot \widehat{b}_1 + \widehat{A}_{n-3} \cdot \widehat{b}_2 + \ldots + \widehat{A}_1 \cdot \widehat{b}_{n-2}, \tag{25}$$

where $\widehat{A}_i = \sum_{k=i}^{n-2} \widehat{a}_k$ is computed using the Brent-Kung parallel-prefix algorithm [17].

Fig. 5 shows the proposed overflow detection circuit for an 8-bit two's complement tree multiplier. This circuit is similar to the unsigned overflow detection logic shown in Fig. 4, except $(2n-4)$ XOR gates compute $\widehat{a}_1$ to $\widehat{a}_{n-2}$ and $\widehat{b}_1$ to $\widehat{b}_{n-2}$, the size of the parallel-prefix circuit is one bit smaller, and one XOR gate is used to compute $p_n \oplus p_{n-1}$, as expressed in (24).

4 Combined Overflow Detection

The proposed unsigned and two's complement multipliers with overflow detection have similar structures. Consequently, a single multiplier can be designed to perform either unsigned or two's complement multiplication with overflow detection. The type of the multiplication performed is based on an input control signal, t, which is one when two's complement multiplication is performed and zero when unsigned multiplication is performed.

Our proposed overflow detection method for combined unsigned and two's complement multiplication unifies the overflow detection equations for unsigned and two's complement multiplication by defining

$$\check{a}_i = a_i \oplus (t \cdot a_{n-1}) \quad \text{and} \quad \check{b}_i = b_i \oplus (t \cdot b_{n-1}) \tag{26}$$

for $1 \leq i \leq n-2$. This gives a_i and b_i when $t = 0$, and $\hat{a}_i$ and $\hat{b}_i$ when $t = 1$. Equations (9) and (19) are merged to generate a combined preliminary overflow flag V_c', which is optimized for

TABLE 1
Comparison of 32-Bit Multipliers with Overflow Detection

	Proposed		Schulte *et. al* [7]		Reduction	
Type	Area	Delay	Area	Delay	Area	Delay
UA	86917	14.21	83577	13.89	-3.9%	-2.3%
UT	79685	5.44	83986	5.69	5.2%	5.4%
SA	90333	14.37	105650	15.48	14.6%	7.2%
ST	89131	6.49	103001	6.68	19.5%	2.9%

either array or tree multiplier implementations. The overflow flag is then computed as

$$V_c = V_c' + p_n \cdot \bar{t} + (p_n \oplus p_{n-1}) \cdot t. \quad (27)$$

The implementation details for combined array and tree multipliers are given in [12].

5 Results and Comparisons

Table 1 gives area and delay estimates for our proposed 32-bit multipliers and the 32-bit multipliers proposed by Schulte et al. in [7], when both types of multipliers are implemented using the TMSC25 0.25 micron CMOS standard cell library and the Leonardo Spectrum Synthesis Tool. For unsigned multipliers, the technique in [7] signals overflow if it detects a logical 1 among any of the partial product bits in the $n-1$ most significant columns of the partial-product matrix or a carry bit equal to 1 into column n. For two's complement multiplication, the overflow detection technique in [7] multiplies the magnitudes of the operands by using XOR gates to modify some of the partial product bits when an operand's sign bit is one and then detects overflow using a technique similar to the one used for unsigned multiplication. After multiplying the magnitudes of the operands, their product is complemented if the signs of the input operands differ. The techniques presented in [7] do not work well for designing combined multipliers due to significant differences in the structures for their unsigned and two's complement multipliers with overflow detection.

As demonstrated in Table 1, the unsigned array multipliers in [7] have slightly less area and delay than our proposed unsigned array multipliers since they only compute the n least significant product bits and use an efficient mechanism to determine if any of the partial product bits in the $n-1$ most significant columns are 1. Our proposed unsigned tree multipliers have less area and delay than the unsigned tree multipliers in [7] since they use a more efficient technique to determine if overflow has occurred. Our two's complement array and tree multipliers have much less area and less delay than those presented in [7] since they avoid the overhead of conditionally complementing the multiplier input and output operands to correctly handle negative numbers.

Our methods for designing multipliers with overflow detection have the following advantages over the methods presented in [7]:

- Our overflow detection circuits do not rely on internal carries generated by the multiplier and do not require the partial-product matrix of the multiplier to be modified. Thus, they can be applied to any type of simplified multiplier that produces $n+1$ product bits.
- Our methods for detecting overflow on unsigned tree multipliers and two's complement array and tree multipliers have less area and delay than those presented in [7].
- Our methods can also be used to design efficient combined multipliers that operate on both unsigned and two's complement operands.

6 Conclusions

A general approach for overflow detection is developed for unsigned and two's complement integer multiplication. Our proposed integer multipliers with overflow detection have less area and delay than previous designs. Similarities between unsigned and two's complement multipliers with overflow detection are utilized to design efficient combined unsigned and two's complement multipliers. Depending on the system requirements, the proposed methods can be further optimized for area, delay, or power.

References

[1] P. Lapsley, *DSP Processor Fundamentals: Architectures and Features.* IEEE Press, 1997.
[2] J.L. Hennessy and D.A. Patterson, *Computer Architecture a Quantitative Approach,* second ed., pp. A-11. Morgan Kaufmann, 1996.
[3] K. Guttag, "Built-In Overflow Detection Speeds 16-Bit Microprocessor Arithmetic," *EDN,* vol. 28, pp. 133-135, Jan. 1983.
[4] IBM, *Power PC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors,* no. G522-0290-01, Feb. 2000.
[5] T. Lindholm and F. Yelin, *The Java Virtual Machine Specification.* Addison-Wesley, 1996.
[6] E.W. Mahurin, "Method and Circuit for Detecting Overflow in Operand Multiplication," Patent No. 6,151,616, Nov. 2000.
[7] M.J. Schulte, P.I. Balzola, A. Akkas, and R.W. Brocato, "Integer Multiplication with Overflow Detection or Saturation," *IEEE Trans. Computers,* vol. 49, no. 7, pp. 681-691, July 2000.
[8] M.J. Schulte, M. Gok, P.I. Balzola, and R.W. Brocato, "Combined Unsigned and Two's Complement Saturating Multipliers," *Proc. SPIE: Advanced Signal Processing Algorithms, Architectures, and Implementations,* pp. 185-196, 2000.
[9] M. Gok, M.J. Schulte, P.I. Balzola, and R.W. Brocato, "Efficient Integer Multiplication Overflow Detection Circuits," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers,* pp. 1661-1665, 2001.
[10] Y. Baydatch and R. Hasharon, "Overflow Detection for Integer Multiply Instruction," Patent No. 5,801,978, Sept. 1998.
[11] Y.H. Cha, G.Y. Cho, H.H. Choi, and H.B. Song, "Bit Result Integer Multiplier with Overflow Detector," *IEE Electronic Letters,* vol. 37, pp. 940-942, July 2001.
[12] M. Gok, "Integer Multiplier and Squarer Architectures with Overflow Detection," PhD thesis, Lehigh Univ., 2003, http://www.cse.lehigh.edu/~caar/publicationpg.html.
[13] R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI and Their Synthesis," PhD thesis, Swiss Federal Inst. of Technology, Zurich, Switzerland, 1997.
[14] R.E. Ladner and M.J. Fischer, "Parallel Prefix Computation," *J. ACM,* vol. 27, pp. 831-838, Oct. 1980.
[15] J. Sklansky, "Conditional Sum Addition Logic," *IRE Trans. Electronic Computers,* vol. 9, pp. 226-231, June 1960.
[16] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. Computers,* vol. 22, pp. 786-792, Aug. 1973.
[17] R.P. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers,* vol. 31, no. 3, pp. 260-264, Mar. 1982.