INT. J. ELECTRONICS, 1999, VOL. 86, NO. 1, 67–77

A novel technique for fast multiplication

SADIQ M. SAIT†, AAMIR A. FAROOQUI‡ and
GERHARD F. BECKHOFF§

In this paper we present the design of a new high-speed multiplication unit. The design is based on non-overlapped scanning of 3-bit fields of the multiplier. In this technique the partial products of the multiplicand and three bits of the multiplier are pre-calculated using only hardwired shifts. These partial products are then added using a tree of carry-save-adders, and finally the sum and carry vectors are added using a carry-lookahead adder. In the case of 2′s complement multiplication the tree of carry-save-adders also receives a correction output produced in parallel with the partial products. The algorithm is modelled in a hardware description language and its VLSI chip implemented. The performance of the new design is compared with that of other recent ones proposed in literature.

1. Introduction

Multipliers find use in high-speed real-time applications where a large amount of data is to be processed. One such application is digital signal processing (DSP). Several multiplication algorithms for high-speed implementation have been proposed (Cooper 1988, Fadavi-Ardekani 1993, Madrid *et al.* 1993, Sunder 1993, Hung *et al.* 1994b), and most of them are based on the Booth (1951) algorithm and its modifications. In this work we present a new technique for high-speed 2′s complement multiplication which can be easily implemented in hardware. The high speed features of this algorithm are due to:

(1) pre-calculation of partial products of non-overlapped 3-bit fields by hardwired shifting;
(2) addition of partial products using a carry-save-adder tree;
(3) a single addition of the carry and sum vectors using a carry-look-ahead adder;
(4) a new technique to accommodate multiplication of negative operands.

The design of unsigned multiplier is presented for 9-bits and can be extended to 12-bits with no additional time delay at the cost of increased hardware. The design of a signed 2′s complement multiplier is presented for 8-bits and is also extendable to 10-bits (using non-overlapped 4-bit fields) with no additional time delay. The methodology can be easily extended to words of larger sizes.

The paper is divided into six sections. In the following section (§2) we present the basic design for unsigned numbers. In §3 the design to accommodate 2′s complement numbers is discussed. A mathematical proof to validate the correction circuit

Received 8 October 1997. Accepted 19 May 1998.
† King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
‡ Department of Electrical and Computer Engineering, University of California, Davis, CA-95616, USA. e-mail: aamirf@ece.ucdavis.edu
§ Queens University, Ontario, Canada.

output is also presented. Implementation details are presented in §4. Performance details are given in §5 and conclusions in §6.

2. Basic design

The basic idea behind the algorithm consists of first finding *partial products* which are the products of the multiplicand (B) with 3-bit fields of the multiplier (A). The product of the multiplicand and a three-bit number (0 to 7) is obtained by means of *shift* and *add* operations. As shown in table 1, multiplication by a three-bit number can be expressed as a single addition of two multiplicands shifted by a fixed amount. For example, multiplication by 6 is expressed as the addition of multiplication by 4 and multiplication by 2 (multiplication by powers of 2 is accomplished by shifts only). Multiplication by 7 (111) requires three additions. This operation is replaced by subtracting a 1 from a number multiplied by 8. In order to avoid the delay produced by obtaining the 2′s complement of 1, the LSB of the three times shifted multiplicand is pre-set to one. Then, only the inverse of the multiplicand is added to it. That is,

$$B \times 111 = B \times 1000 + \bar{B} + 1 = B000 + \bar{B} + 1 = B001 + \bar{B}$$

where B000 represents three zeros catenated to B, or B shifted left three times.

The two partial products to be added in table 1 can be considered as two operands of an adder, designated as the left-operand and the right-operand. Note that the left-operand is multiplied by either 2, 4 or 8, whereas the right-operand needs to be multiplied by 1, 2 or $\bar{1}$, the last representing complementation. Therefore, as shown in figure 1, the two columns consist of hard-wired shifters which are enabled by a simple encoder logic. The logic design of the encoder block that enables one of the three shifters in each block of the *pp-cell* of figure 1 is given in figure 2.

For a 9-bit wide multiplier-operand, three such pp-cells are required. Each pp-cell takes the entire multiplicand operand, and 3 bits of the multiplier operand. This is illustrated in figure 3. In general, the number of pp-cells is equal to $p = \lceil \frac{n}{3} \rceil$, where n is equal to the number of bits of the multiplier operand A.

The output produced by these pp-cells is the shifted operand, which is added using a three-level tree of carry-save-adders to produce the sum and carry vectors. Note that, by using the above pp-cells, the number of additions is considerably

A Multiplier-field	$A \times B$ Expressed as shift/add
0 0 0	$0 + 0$
0 0 1	$0 + 2^0(B)$
0 1 0	$0 + 2^1(B)$
1 0 0	$2^2(B) + 0$
0 1 1	$2^1(B) + 2^0(B)$
1 0 1	$2^2(B) + 2^0(B)$
1 1 0	$2^2(B) + 2^1(B)$
1 1 1	$2^3(B) - 2^0)B)$

Table 1. Multiplication by numbers 0 through 7 expressed in terms of addition/subtraction of powers of 2.

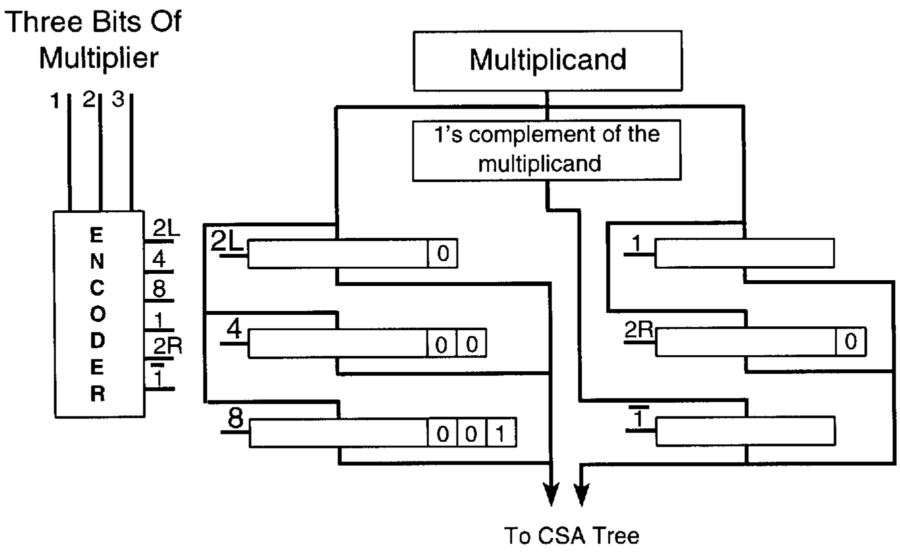

Figure 1. Block diagram of a pp-cell.

reduced. In this case the reduction is from 9 additions to 6. In general, the decrease in the number of additions is from n to $\lceil\frac{2n}{3}\rceil$.

In the final stage, the outputs of these carry-save-adders are added using a carry-lookahead-adder (CLA) circuit. The output of this adder is the product of the 9-bit multiplier with the m-bit multiplicand. Note that the number of levels of carry save adder (CSA) circuitry remains the same for larger bits of the multiplicand. In the case of increase in the number of bits of the multiplier, the only delay-causing circuitry is due to increase in the number of levels in the CSA tree, and the increase in delay of the CLA.

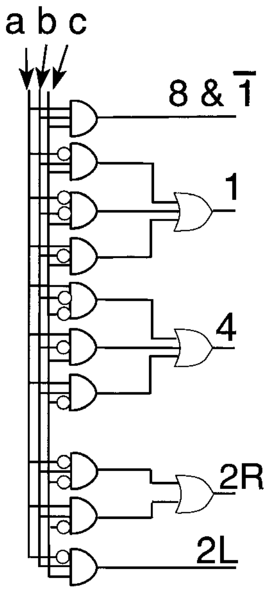

Figure 2. Logic diagram of the encoder cell shown in figure 1.

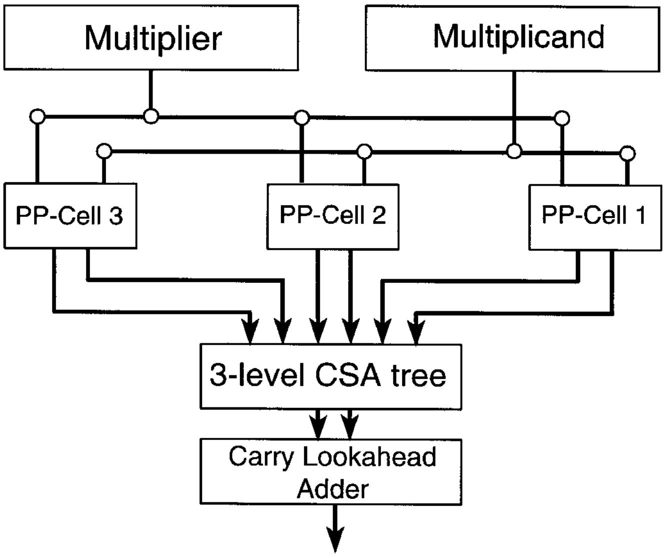

Figure 3. Block diagram of the 9-bit unsigned multiplier.

3. Design of 2′s complement multiplier

For multiplication of signed 2′s complement numbers, the design is similar, except that additional circuitry is included to accommodate correction. The multiplier (A) is again divided into fields of at most 3-bits. The division of the multiplier into fields is slightly different from the case of unsigned multiplication and is depicted below.

$$A_{n-2}A_{n-3}A_{n-4} \quad \cdots \quad A_6A_5A_4 \quad A_3A_2A_1 \quad A_0A_s$$

Observe that the sign bit $(A_s = A_{n-1})$ is grouped with the LSB of the multiplier. Bits A_1 to A_3 form one field, similarly bits A_4 to A_6 form the other, and so-on. The pp-cell required is identical to that in the previous multiplier for unsigned operands.

3.1. *Correction circuit*

Generally, the 2′s complement of a number is obtained by complementing individual bits and then adding 1 to the number. One motivation for the design of correction circuitry is to avoid the delay due to rippling of carry produced by the addition of 1 in 2′s complementation both before and after multiplication.

Two’s complement numbers are accommodated just by inverting the negative operands (1′s complement) and post-complementation (1′s complement) of the negative results. Two’s complementation is avoided since it causes a delay due to addition of one. The error is adjusted using a correction factor as explained below. An additional circuit is used that generates in parallel a correction factor to be added to the result of 1′s complement multiplication.

The technique that avoids the ripple delay due to adding a 1 is based on two observations: (1) the distributive property of multiplication, that is, $A \times (\bar{B} + 1) = A \times (\bar{B}) + A$; therefore the addition of 1 is replaced by the addition of operand A to the result of 1′s complement multiplication; and (2) the fact that the 2′s complement

$$\begin{tikzcd}
(\bar{A},B) \arrow[r, "+(1{,}0)"] \arrow[d, "x"'] & ((\bar{A}+1),B) \arrow[d, "x"] \\
(\bar{A}xB) \arrow[r, "+(0{,}B)"] & ((\bar{A}+1)xB)
\end{tikzcd}$$

(a

$$\begin{tikzcd}
A \arrow[r, "C"] \arrow[d, "-1"'] & \bar{A} \arrow[d, "+1"] & \text{OR} & \bar{A}+1 \arrow[r, "C"] \arrow[d, "-1"'] & A-1 \arrow[d, "+1"] \\
A-1 \arrow[r, "C"] & \bar{A}+1 & & \bar{A} \arrow[r, "C"] & A
\end{tikzcd}$$

(b

Figure 4. Commutative diagram showing the two operations. $\times$ =multiplication and C =complementation.

of R is the same as the $1'$s complement of $(R-1)$. The two observations can be put into a commutative diagram, as shown in figure 4.

Based on the above two observations, an algorithm has been developed for correction factor generation; the various values produced by the correction circuitry (*Corr,* or *Corr*$'$ if post complementation is required) to be added to the output of the unsigned multiplier are given in table 2.

Below we explain one segment of the code when the multiplier (A) is negative $(A_s = 1)$ and the multiplicand (B) is positive $(B_s = 0)$.

When $A_sB_s = 10$, the negative operand (A) is complemented and applied at the input of the encoder. The two operands are then multiplied using the method explained for unsigned numbers. The LSB (A_0) may be a zero or a 1. If A_0 is a one, then the complemented A_0 will produce a zero, and the correction factor due to adding a one (for $2'$s complement) will be the same as adding a B later.

However, if A_0 was a zero, then the complemented A_0 would have produced a one, and the correction factor due to addition of one (for $2'$s complement) will be the same as the addition of $2B$ later. In addition, since one of the operands is negative, it is required to find the $2'$s complement of the result later. This again is replaced by a simple inversion of the final result ($1'$s complement) but subtracting a 1 in the correction circuit. Therefore, the correction circuitry output which must be added to the result is either $B-1$ (for $A_0 = 1$) or $2B-1$ (for $A_0 = 0$).

Case	A_sB_s	A_0	*Corr*	C_{in}(CLA)	Post-complementation
0	00	0	$0-1$	1	No
		1	$B-1$	1	No
1	01	0	$0-1+A$	0	Yes
		1	$\overline{\bar{B}+1}-1+A$	0	Yes
2	10	0	$2B-1$	0	Yes
		1	$B-1$	0	Yes
3	11	0	$2\overline{\bar{B}+1}-0+\overline{\bar{A}+1}$	1	No
		1	$\overline{\bar{B}+1}-0+\overline{\bar{A}+1}$	1	No

Table 2. Algorithm of correction circuitry for $2'$s complement multiplication.

Case 0: $A_s = B_s = 0$
i) $A_0 = 0 \Rightarrow A$ is even
$P = AB$
$Corr = 0$
ii) $A_0 = 1 \Rightarrow A$ is odd
$P = (A - 1)B + Corr = AB - B + Corr$
$\mathrm{P} = AB$
$AB - B + Corr = AB$
$\mathrm{Corr} = B$

Case 1: $A_s = 0, B_s = 1$
i) $A_0 = 0 \Rightarrow A$ is even
$P = A(B - 1) + Corr$
$P = -AB = (\overline{AB - 1})$
$AB - A + Corr' = AB - 1$
$Corr' = A - 1$ (Post-complementation is required)
ii) $A_0 = 1 \Rightarrow A$ is odd
$P = (A - 1)(B - 1) + Corr = AB - B - A + 1 + Corr$
$P = -(AB) = \overline{AB - 1}$
$AB - B - A + 1 + Corr' = AB - 1$
$Corr' = B + A - 2 = (B - 1) + A - 1$
$Corr' = \overline{\bar{B} + 1} + A - 1$ (Post-complementation is required)

Case 2: $A_s = 1, B_s = 0$
i) $A_0 = 0 \Rightarrow \overline{\bar{A} + 1} = A - 1$ is odd
$P = (\overline{\bar{A} + 1} - 1)B + Corr = (A - 2)B + Corr = AB - 2B + Corr$
$P = -(AB) = \overline{AB - 1}$
$AB - 2B + Corr' = AB - 1$
$Corr' = 2B - 1$ (Post-complementation is required)
ii) $A_0 = 1 \Rightarrow (\overline{\bar{A} + 1}) = A - 1$ is even
$P = (\overline{\bar{A} + 1})B + Corr = (A - 1)B + Corr = AB - B + Corr$
$P = -(AB) = \overline{AB - 1}$
$AB - B + Corr' = AB - 1$
$Corr' = B - 1$ (Post-complementation is required)

Case 3: $A_s = B_s = 1$
i) $A_0 = 0 \Rightarrow (\overline{\bar{A} + 1}) = A - 1$ is odd
$P = (\overline{\bar{A} + 1} - 1)(\overline{\bar{B} + 1}) + Corr = (A - 2)(B - 1) + Corr = AB - 2B - A + 2 + Corr$
$\mathrm{P = AB}$
$AB - 2B - A + 2 + Corr = AB$
$Corr = 2(B - 1) + A = 2(B - 1) + (A - 1) + 1 = 2(\overline{\bar{B} + 1}) + (\overline{\bar{A} + 1}) + 1$
ii) $A_0 = 1 \Rightarrow (\overline{\bar{A} + 1}) = A - 1$ is even
$P = (\overline{\bar{A} + 1})(\overline{\bar{B} + 1}) + Corr = (A - 1)(B - 1) + Corr = AB - A - B + 1 + Corr$
$\mathrm{P = AB}$
$AB - A - B + 1 + Corr = AB$
$Corr = A + B - 1 = (A - 1) + (B - 1) + 1 = (\overline{\bar{A} + 1}) + (\overline{\bar{B} + 1}) + 1$

Figure 5. The four cases of $2'$s complement multiplication. Note: whenever the (multiplier) operand is in a form such that its LSB = 1, a 1 is subtracted; the correction circuit corrects this modification.

This correction factor (*Corr*) is added to the output produced by the other partial product cells using the carry-save adder tree. If only one of the operands is negative, then the final output of the CLA is complemented.

The complete mathematical proof for the output produced by the correction circuitry is given in figure 5 and summarized in table 2. The mapping of this algorithm into hardware results in an extremely simple and fast correction circuitry. The correction circuitry output is produced in parallel with the output of the pp-cells, and is added to the sum of partial products using the carry-save adder tree.

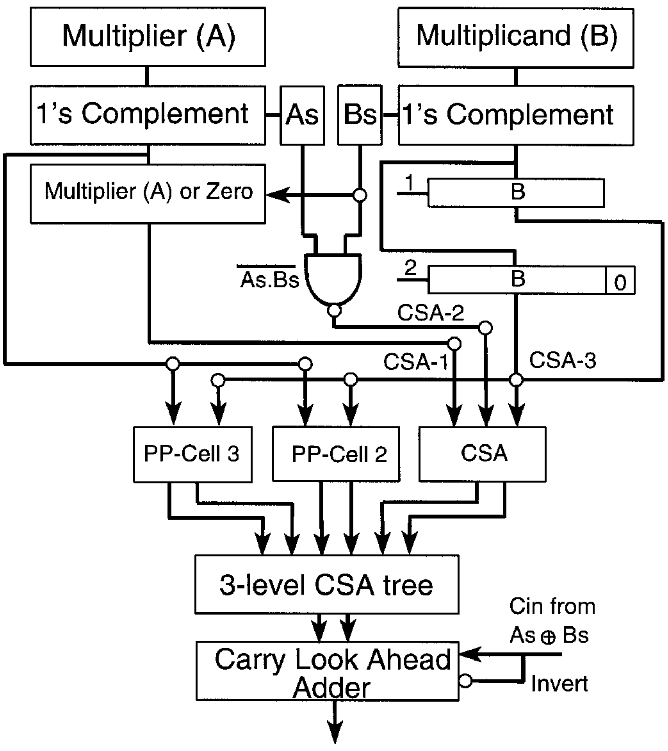

Figure 6. Block diagram of the 8-bit 2′s complement multiplier.

Example:
Let $A = 4 = 00000100$ and $B = -7 = 11111001$.
Since $B_s = 1$, complement it. Then,

$\underline{A} = 00000100$
$\bar{B} = 00000110$

Now, multiplying these numbers using the *pp-cell* circuit used for unsigned multiplication of numbers will produce 00011000 (that is, multiplying $\bar{B}$ by $A = 00000100$ (4) is the same as shifting $\bar{B}$ left twice). To this a correction output is added. Since $A_s = 0$, $B_s = 1$, and $A_0 = 0$, the correction circuit produces an output $A - 1 = 00000011$. Adding this to the pp-cell output yields (see figure 6)

$00011000 + 00000011 = 00011011$. Inverting this output will give

$11100100 = -28$, the required result.

4. Implementation

4.1. *Design of pp-cell*

The design of the pp-cell consists of hardwired shifters which can be implemented using either tri-state buffers or multiplexers. The multiplicand to be shifted enters two columns of shifters. The left column shifts the multiplicand by 2, 4 and 8. Similarly, the right column shifts the multiplicand by 1 or 2 or produces the complement of the multiplicand. Which of the shifted outputs is selected depends on the three-bit fields of the multiplier for each pp-cell. The details of the encoder are given in figure 2, based on table 1.

4.2. *Design of correction circuit*

The correction circuitry consists of a carry-save adder with inputs arriving from three sources. The three inputs of the CSA are labelled *CSA*-1, *CSA*-2 and *CSA*-3. The different values received by these inputs depend on the sign of the multiplier/multiplicand and the value of bit A_0. These values are given in tables 3 and 4 and summarized in table 5.

4.3. *Design of a multiplier for 2′s complement numbers*

The techniques explained above are implemented by the combinational hardware shown in figure 6. In the cases when either of the two operands is negative, in order to 2′s complement the result and to avoid the ripple carry effect produced by the addition of one to the 1′s complement, a −(11111111) is added at the beginning of

A_s	B_s	$\overline{A_s \cdot B_s}$	A	B	*CSA-1*	*CSA-2*	C_{in}(*CLA*)	Invert output
0	0	1	No change	No change	0	−1	1	0
0	0	1	No change	Inverted	A	−1	0	1
1	0	1	Inverted	No change	0	−1	0	1
1	1	0	Inverted	Inverted	A	0	1	0

Table 3. Table illustrating the *i*th input received by the CSA adder of the correction circuitry. *CSA-i* represents the *i*th input. For input *CSA*-3 see table 4.

A_s	A_0	*CSA*-3
0	0	0
0	1	B
1	0	2B
1	1	B

Table 4. Table illustrating the third input of the CSA of this correction circuit.

A_0	A_s	B_s	CSA input	C_{in}
0	0	0	$0+0-1$	1
0	0	1	$0+\mathrm{A}-1$	0
0	1	0	$2\underline{\mathrm{B}}+\underline{0}-1$	0
0	1	1	$2B+A+0$	1
1	0	0	$\underline{\mathrm{B}}+0-1$	1
1	0	1	$B+\mathrm{A}-1$	0
1	1	0	$\underline{\mathrm{B}}+\underline{0}-1$	0
1	1	1	$B+A+0$	1

Table 5. Table summarizing the inputs to CSA adders.

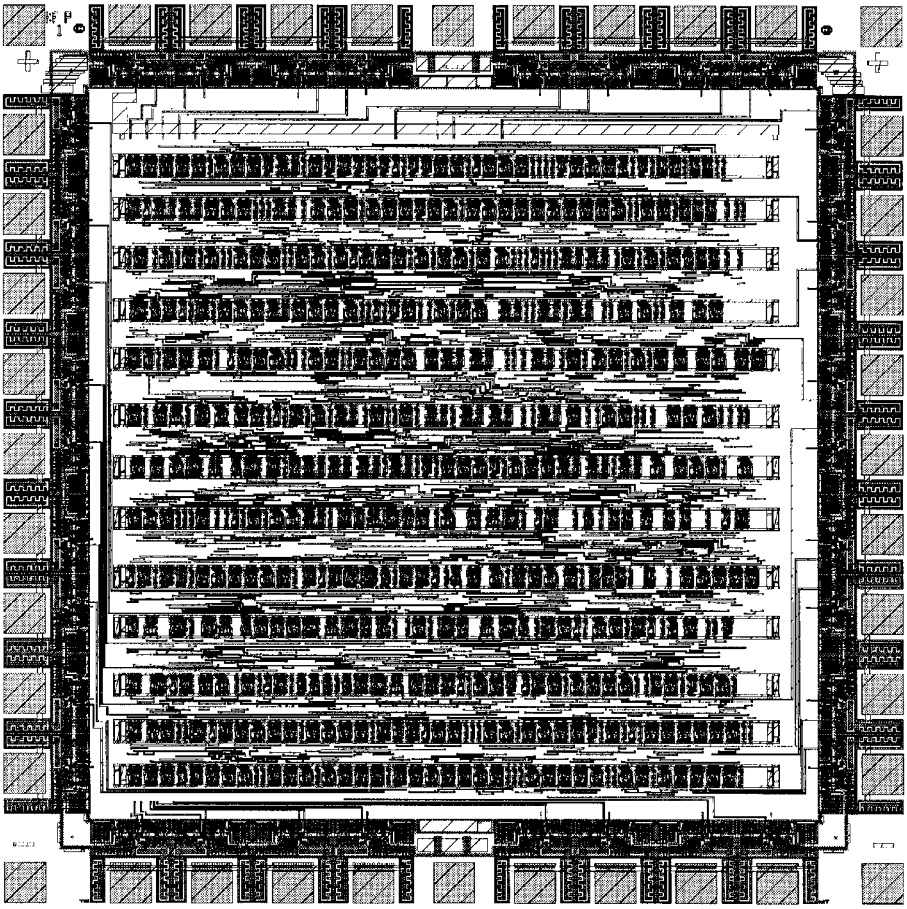

Figure 7. VLSI layout of the 2′s complement multiplier.

multiplication by correction circuitry (see figure 6) and finally the output is inverted to get the 2′s complemented result. Note that an EXOR gate is required to get the -1 when A or B is negative but a NAND gate is used to reduce the gate delay in the first stage. To compensate for the addition of -1 when A and B are positive, a carry is added by the CLA in the final stage.

5. Performance

The structural level hardware of the multiplier is expressed in the LOGIC3 hardware description language and simulated using *ldvsim* (Kozminski 1992). The OASIS silicon compiler is used to produce the layout in Magic (Ousterhout *et al.* 1984, Mayo *et al.* 1990). To verify the correctness of the layout (see figure 7) and determine the operating speed, the circuit is extracted from the layout and simulated using the switch level simulator *irsim.*

In order to get the best results in terms of time and area, three different implementations of the multiplier were attempted: (1) using tri-state buffers, (2) using separate And/Or logic gates for multiplexing, and finally (3) using And-Or-Invert

Characteristic	OMB multiplier	QMB multiplier	Our multiplier
Technology	1.2 μ NTE CMOS4S	1.2 μ NTE CMOS4S	2μ SCMOS
Area of chip	$2026.43 \times 2026.43 \mu m^2$	$1920.91 \times 1920.91 \mu m^2$	$1620.00 \times 1607.00 \mu m^2$
Prop. delay	9.27 ns	13.17 ns	9.00 ns
Area × time	$3.8066 + 07 \mu m^2$ ns	$3.8066 + 07 \mu m^2$ ns	$2.3430 + 07 \mu m^2$ ns
Gate count	860	720	573

Table 6. Comparison with other proposed approaches.

cells available in the library of OASIS. The last implementation produced best results with a core area of $1.6 \times 1.6\,mm^2$ and a speed of 9 ns. The area can further be reduced if, instead of relying on the silicon compiler, special handcrafted cells are designed for the various arithmetic and logical functions (Hung *et al.* 1994a). Further increase in speed can be achieved by using the PP addition technique (three-dimensional optimization, TDM) proposed by Oklobdzija *et al.* (1996).

The results of our multiplier for signed numbers are compared with Modified Booth Multiplier (Sunder 1993) for unsigned numbers and summarized in table 6.

6. Conclusions

A new technique has been developed and implemented for the multiplication of two signed numbers. The advantages of this technique over the others available in literature include smaller propagation delay (Hung *et al.* 1994a) and reduced area (Sunder 1993). The circuitry required to produce a correction output and its mathematical proof are also presented. Hung *et al.* (1994) also proposed a multiplier based on non-overlapped bit scanning. Their design is limited to two bits and is valid for unsigned numbers only. The 3-bit version requires an eight to one multiplexer. The partial products are stored in a RAM which requires a large area and also makes the multiplier slow (the delay of an 11×11 multiplier in 2 μSCMOS is 22 ns). The technique was implemented in VLSI using the OASIS silicon compiler by modelling the hardware at the netlist level using LOGIC3. The chip was fabriated by Orbit Semiconductors, California, and was found to work as per specification. The speed of the multiplier is 9 ns, and speed can be further improved by using a full custom approach and special adders (Hung *et al.* 1994a). The predicted increase in propagation delay with increase in word sizes of operands is small.

Acknowledgments

The authors acknowledge King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for support under Project #COE/ARRAYS/177. We are also thankful to Mr Asjad M. T. Khan for his assistance.

References

Booth, A. D., 1951, A signed binary multiplication algorithm. *Quarterly Journal of Mechanics and Applied Mathematics,* **4**, 236–240.

Cooper, A. R., 1988, Parallel architecture modified Booth multiplier. *Proceedings of the Institution of Electrical Engineers,* **135**(3), 125–128.

Fadavi-Ardekani, J., 1993, $M \times N$ Booth encoded multiplier generator using optimized Wallace trees. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* **1**(2), 120–125.

Hung, H. T., Kwentus, A. Y., and Wilson, A. N., 1994a. An architecture for high performance/small area multiplier for use in digital filtering applications. *IEEE Journal of Solid State Circuits,* **29**(2).

Hung Xiaoping, Liu, Wen-Jung, and Wei, W. Y., 1994b, A high performance CMOS redundant binary multiplication and accumulation (MAC) unit. *IEEE Transactions on Circuits and Systems,* **41**(1), 33–39.

Kozminski, K., 1992, OASIS: Open architecture silicon implementation system user's guide (Research Triangle Park, North Carolina: MCNC).

Madrid, P. E., Millar, B., and Swartzlander, E. E., Jr, 1993, Modified Booth algorithm for high radix fixed-point multiplication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* **1**(2), 164–167.

Mayo, R., *et al.*, 1990, DECWRL/Livermore Magic Release (Digital Western Research Laboratory).

Oklobdzija, V. G., Villeger, D., and Liu, S., 1996, A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach. *IEEE Transactions on Computers,* **45**(3), 294–306.

Ousterhout, J. K., *et al.*, 1984, MAGIC: A VLSI layout system. *Proceedings of 21st Design Automation Conference*, pp. 152–159.

Sunder, S., 1993, A fast multiplier based on modified Booth algorithm. *International Journal of Electronics,* **75**(2), 199–208.