A Novel Technique for Fast Multiplication

Sadiq M. Sait Aamir. A. Farooqui Gerhard. F. Beckhoff

King Fahd University of Petroleum and Minerals
Dhahran-31261, Saudi Arabia
e-mail: facy009@saupm00.bitnet

Abstract

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 algorithm 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-look-ahead adder. In 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 algorithms is modeled in a hardware description language and its VLSI chip implemented. The performance of the new design is compared with 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 [7, 5, 8, 4, 3], and most of them are based on the Booth algorithm [2] 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 (CSA) tree.
3. A single addition of the carry and sum vectors using a carry-lookahead adder (CLA).
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 signed 2's complement multiplier is presented for 8-bits. It is also extendable to 10-bits (using non-overlapped 4-bit fields) with no additional time delay. The methodology can be very easily extended to words of larger sizes. The paper is divided into 5 sections. In the following section (Section 2) we present the basic design for unsigned numbers. In Section 3 the design to accommodate 2's complement numbers is discussed. A mathematical proof to validate the correction circuit output is also presented. Implementation details are presented in Section 4 and conclusions in Section 5.

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 six is expressed as the addition of multiplication by 4 and multiplication by 2 (multiplication by powers of 2 is accomplished by shifts only). The multiplication by 7 (111) requires three addition. Therefore, this operation is replaced by subtracting multiplicand from multiplicand times 8. In order to avoid the delay produced to obtain the 2's complement of multiplicand, the LSB of the three times shifted multiplicand is pre-set to one. Then,

A	$A \times B$
multiplier-field	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 0 through 7 expressed in terms of addition/subtraction of powers of 2.

0-7803-2492-7/95 $4.00 © 1995 IEEE

only the inverse of the multiplicand is added to it. That is,

$$A \times 111 = A \times 1000 + \overline{A} + 1 = A000 + \overline{A} + 1 = A001 + \overline{A}$$

where A000 represents three zeros catenated to A, or A multiplied by $2^3 = 8$.

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 $\overline{1}$, the last representing complementation. Therefore, as shown in Figure 1 the two columns consists of hard-wired shifters which are enabled by a simple encoder logic. Details of the encoder logic are given in Figure 2. 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.

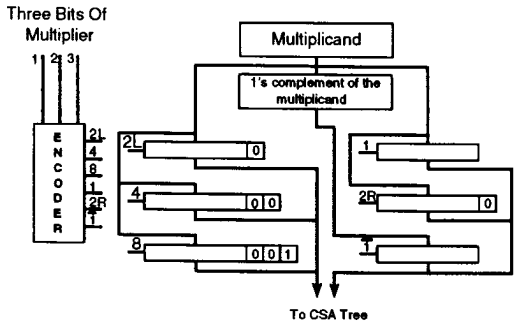

Figure 1: Block diagram of a *pp-cell*.

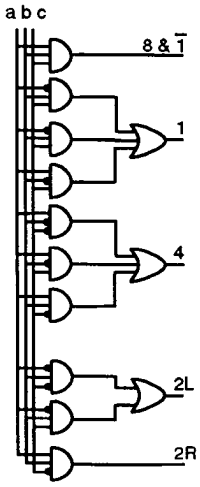

Figure 2: Logic diagram of the encoder cell shown in Figure 1.

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.

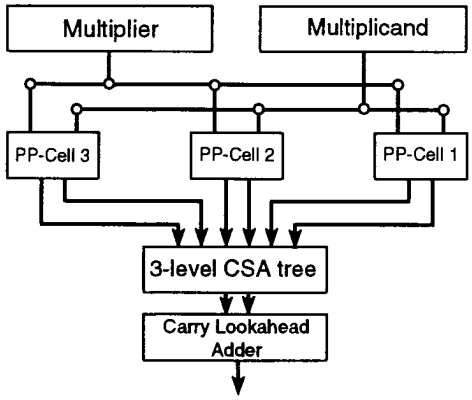

Figure 3: Block diagram of the 9-bit unsigned multiplier.

The outputs produced by these *pp-cells* are added using a three-level tree of CSAs to produce the sum and carry vectors. Note that by using the above *pp-cells*, the number of additions is considerably 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 CSAs are added using a carry-lookahead-adder 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 circuitry remains the same for larger bits of the multiplicand, (except for the small increase in delay of the CLA). In 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.

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 the unsigned multiplication and is depicted below.

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

Correction Circuit

Generally, the 2's complement of an integer is obtained by complementing individual bits and then adding 1 to the number. One feature of the correction circuitry is that it avoids the delay due to rippling of carry produced by the addition of 1 in 2's complementation both before and after multiplication.

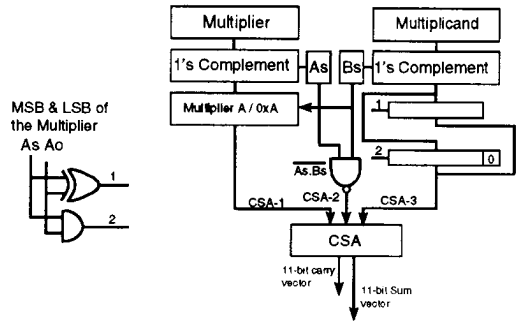

Figure 4: Correction circuitry for 2's complement multiplication.

Two's complement numbers are accommodated by just inverting the negative operands (1's complement), and postcomplementation (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, $A \times (\overline{B} + 1) = A \times (\overline{B}) + A$; 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 of R is the same as the 1's complement of $(R-1)$.

Based on the above two observations, an algorithm has been developed for generating a correction factor. The various values produced by the correction circuitry ($Corr$ or $Corr'$ if post complementation is required) are given in Figure 5, they have to be added to the output of the unsigned multiplier.

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

Table 2: Table illustrating the third input of the CSA of correction circuit.

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 1, then the complemented A_0 will produce a zero, and the correction factor due to adding a 1 (for 2's complement) will be the same as adding 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 the addition of 1 (for 2's complement) will be the same as the addition of $2B$ later. In addition, since one of the operand 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 $A_0 = 0$) (see Figure 5).

This correction factor ($Corr$) is added to the output produced by the other partial product cells using the carry-save adder tree. If 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 3. The mapping of this algorithm into hardware is shown in Figure 4 and yeilds 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.

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 labeled 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 summarized in Tables 2 and 3.

Implementation

The above explained techniques are implemented by the combinational hardware shown in Figure 6. Note that when any 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 -1 (11111111) is added before multiplication by the 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 are 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.

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

$P = (A-1)B + Corr = AB - B + Corr$

P = AB

$AB - B + Corr = AB$

Corr = B

Case 1: $A_s = 0, B_s = 1$

i) $A_0 = 0 \Rightarrow A$ is even

$P = A(B-1) + Corr$

$P = -AB = \overline{(AB-1)}$

$AB - A + Corr' = AB - 1$

$Corr' = A - 1$ (Postcomplementation is required)

ii) $A_0 = 1 \Rightarrow A$ is odd

$P = (A-1)(B-1) + Corr = AB - B - A + 1 + Corr$

$P = -(AB) = \overline{AB-1}$

$AB - B - A + 1 + Corr' = AB - 1$

$Corr' = \overline{B + A - 2} = (B-1) + A - 1$

$Corr' = \overline{\bar{B} + 1} + A - 1$ (Postcomplementation is required)

Case 2: $A_s = 1, B_s = \overline{0}$

i) $A_0 = \overline{0} \Rightarrow \overline{\bar{A} + 1} = A - 1$ is odd

$P = (\overline{\bar{A} + 1} - 1)B + Corr = (A-2)B + Corr = AB - 2B + Corr$

$P = -(AB) = \overline{AB-1}$

$AB - 2B + Corr' = AB - 1$

$Corr' = 2B - 1$ (Postcomplementation is required)

ii) $A_0 = \overline{1} \Rightarrow \overline{(\bar{A} + 1)} = A - 1$ is even

$P = (\bar{A} + 1)B + Corr = (A-1)B + Corr = AB - B + Corr$

$P = -(AB) = \overline{AB-1}$

$AB - B + Corr' = AB - 1$

$Corr' = B - 1$ (Postcomplementation is required)

Case 3: $A_s = B_s = \overline{1}$

i) $A_0 = \overline{0} \Rightarrow \overline{(\bar{A} + 1)} = A - 1$ is odd

$P = (\overline{\bar{A} + 1} - 1)\overline{(\bar{B} + 1)} + Corr = (A-2)(B-1) + Corr = AB - 2B - A + 2 + Corr$

P = AB

$AB - 2B - A + 2 + Corr = AB$

$Corr = 2\overline{(B-1)} + A = 2(B-1) + (A-1) + 1 = 2\overline{(\bar{B} + 1)} + \overline{(\bar{A} + 1)} + 1$

ii) $A_0 = \overline{1} \Rightarrow \overline{(\bar{A} + 1)} = A - 1$ is even

$P = (\bar{A} + 1)\overline{(\bar{B} + 1)} + Corr = (A-1)(B-1) + Corr = AB - A - B + 1 + Corr$

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.

As	Bs	$\overline{As \cdot Bs}$	A	B	$CSA\text{-}1$	$CSA\text{-}2$	$C_{in}CLA$	*Invert output.*
0	0	1	No change	No change	0	-1	1	0
0	1	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. $CSA\text{-}i$ represents the i_{th} input. For input CSA-3 see Table 2.

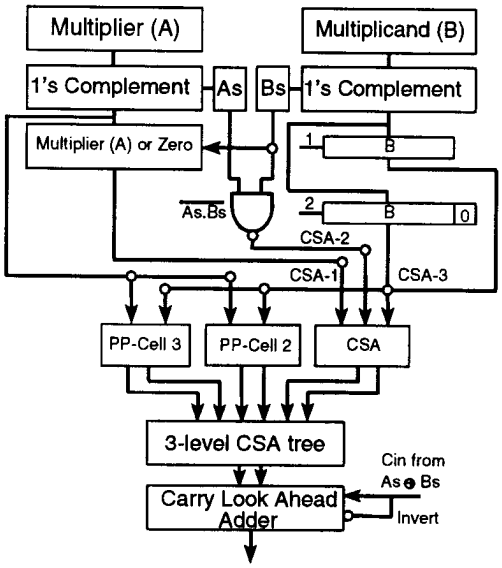

Figure 6: Block diagram of the 8-bit 2's complement multiplier.

Example

Let A = 4=0000 0100 and B = −7=1111 1001.
Since B_s=1 complement it. Then,
A = 0 000 010 0; $\overline{B}$ = 0 000 011 0

Now multiplying these numbers using the *pp-cell* circuit used for unsigned multiplication of numbers will produce 0 001 100 0; (that is, multiplying $\overline{B}$ by A=00000100 (4) is the same as shifting $\overline{B}$ 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 (see Figure 5). Adding this to the *pp-cell* output yields 00011000 + 00000011 = 00011011.
Inverting this output will give 11100100 = −28, the required result.

4 VLSI Implementation and Performance

The hardware described above was modeled using Logic3, a hardware description language, and simulated to verify functional correctness. The Logic3 description was then input to the OASIS silicon compiler to produce a standard-cell layout in SCMOS technology. [6]. To verify the correctness of the layout, 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 tristate buffers, (2) using separate And/Or logic gates for multiplexing and finally using And-Or-Invert cells available in the library of OASIS. The last implementation produced best results with a core area of $1.6 \times 1.6 mm^2$ in 2 micron technology and with a propagation delay of 9ns. 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 [1].

Note that the designs presented here, with only minor modifications, can be easily made cascadable to multiply numbers with larger number of bits. The increase in propagation delay is marginal since the number of additions is reduced, and in addition the sum and carry vectors are produced using a CSA tree.

5 Conclusions

A new technique has been developed and implemented for the multiplication of two signed numbers. Advantage of this technique over the others available in literature include smaller propagation delay [1] and reduced area [8]. The circuitry required to produce a correction output and its mathematical proof are also presented. The technique was implemented in VLSI using the OASIS silicon compiler by modeling the hardware at the netlist level using LOGIC3 see Figure 7. The layout has been submitted to Orbit Semiconductors, California (foundry for MOSIS) for fabrication. The estimated speed of the multiplier is 9ns, and further speed can be possible by using full custom approach and special adders [1]. 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 technical and financial support.

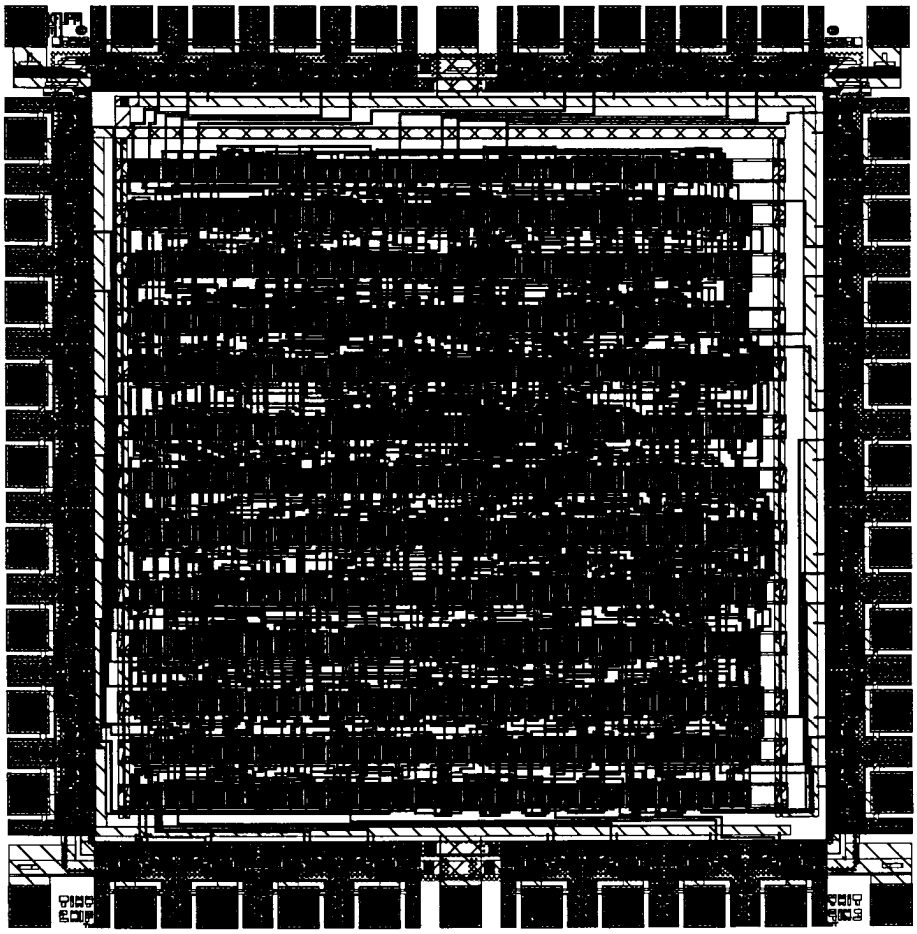

Figure 7: VLSI layout of the 2's complement multiplier.

References

[1] H. T. Hung A. Y. Kwentus and A. N. Wilson Jr. An Architecture for High Performance/Small Area Multiplier for use in Digital Filtering Applications. *IEEE Journal Of Solid State Circuits*, 29(2), February 1994.

[2] A. D. Booth. A Signed Binary Multiplication Algorithm. *Quarterly Journal of Mechanics and Applied Mathematics*, 4:236–240, 1951.

[3] A. R. Cooper. Parallel Architecture Modified Booth Multiplier. *Proceedings of the Institution of Electrical Engineers*, 135(3):125–128, June 1988.

[4] J. Fadavi-Ardekani. $M \times N$ Booth Encoded Multiplier Generator Using Optimized Wallace Trees. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):120–125, June 1993.

[5] Xiaoping Hung, Wen-Jung Liu, and Belle W.Y.Wei. A High Performance CMOS Redundant Binary Multiplication and Accumulation (MAC) Unit. *IEEE Transactions On Circuits and Systems*, 41(1):33–39, January 1994.

[6] K. Kozminski. OASIS: Open Artchitecture Silicon Implementation System Users Guide. *MCNC, Research Triangle Park, North Carolina*, October 1992.

[7] P. E. Madrid, B. Millar, and E. E. Swartzlander Jr. Modified Booth Algorithm for High Radix Fixed-Point Multiplication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):164–167, June 1993.

[8] S. Sunder. A Fast Multiplier Based On Modified Booth Algorithm. *International Journal of Electronics*, 75(22):199–208, 1993.