A Generalized Multibit Recoding of Two's Complement Binary Numbers and Its Proof with Application in Multiplier Implementations

Article *in* IEEE Transactions on Computers · August 1990

DOI: 10.1109/12.57039 · Source: DBLP

2 authors, including:

Arup Gupta
Intel
1 PUBLICATION **95** CITATIONS

A Generalized Multibit Recoding of Two's Complement Binary Numbers and Its Proof with Application in Multiplier Implementations

HOMAYOON SAM AND ARUPRATAN GUPTA, MEMBER, IEEE

Abstract—**A multibit recoding algorithm for signed two's complement binary numbers is presented and proved. In general, a $k+1$-bit recoding will result in a signed-digit (SD) representation of the binary number in radix 2^k, using digits -2^{k-1} to $+2^{k-1}$ including 0. It is shown that by scanning $k+1$-tuples ($k \geq 1$) with one bit overlapping between adjacent groups, a correct SD representation of the original number is obtained. Recoding of binary numbers has been used in computer arithmetic with 3-bit recoding being the dominant scheme. With the emergence of very high speed adders, hardware parallel multipliers using multibit recoding with $k > 2$ (i.e., more-than-3-bit recoding) are feasible with potentials of improving both the performance and the hardware requirements. A parallel hardware multiplier based on the specific case of 5-bit recoding is proposed. Such an implementation would use three fast adders to produce the odd multiples of the multiplicand while reducing the size of the carry-save-adder array by 50% compared to the classic case of 3-bit recoding. Extensions beyond 5-bit recoding for multiplier design are studied for their performance and hardware requirements. Other issues relating to multiplier design such as multiplication by a fixed or controlled coefficient are also discussed in the light of multibit recoding.**

Index Terms—**Booth's Algorithm, computer arithmetic, fixed coefficient multiplier, multibit recoding, multiplier recoding, overlapped scanning, parallel multiplier, signed-digit arithmetic.**

I. Background

RECODING of binary numbers was first hinted at by Booth [1] close to four decades ago while he was considering the problem of multiplying two signed binary numbers in digital computers. His algorithm treated positive and negative numbers in two's complement format indiscriminately and therefore proved to be an attractive solution. In essence, the Booth's algorithm looks at 2 bits of the multiplier number X and depending on the particular combination at hand it does one of the three operations.

- If $x_i x_{i+1} = 00$ or $x_i x_{i+1} = 11$ then shift the existing sum of partial products one bit to the right.
- If $x_i x_{i+1} = 01$ then add the multiplicand Y to the existing sum of partial products and then shift the result one bit to the right.

Manuscript received August 22, 1989; revised March 2, 1990.
The authors are with AT&T Bell Laboratories, Allentown, PA 18103.
IEEE Log Number 9036136.

TABLE I
MODIFIED BOOTH'S RECODING

$x_{i+2}x_{i+1}x_i$	Add to Partial Product
000	+0Y
001	+1Y
010	+1Y
011	+2Y
100	−2Y
101	−1Y
110	−1Y
111	−0Y

- If $x_i x_{i+1} = 10$ then subtract the multiplicand Y from existing sum of partial products and then shift the result one bit to the right.

The process starts by appending a 0 to the right of the LSB of the multiplier X (denoted by x_{-1}) and then looking at binary pairs beginning with x_{-1}.[1] Adjacent pairs share one bit so that in every iteration only one bit of the multiplier retires. It is worth noting here that the required multiples of the multiplicand are $+Y$ and $-Y$ which are readily available if adders and subtracters are employed. n iterations are required for a multiplier X with word size n.

A decade after Booth's paper, MacSorley [2] proposed a modification to Booth's algorithm in which a triplet of bits instead of a pair was looked at in every iteration.[2] His scheme is summarized in Table I.

The Modified Booth's Algorithm also starts by appending a 0 to the right of x_0. Here triplets are taken again beginning at position x_{-1} and continuing to the MSB with one bit overlapping between adjacent triplets. If the number of bits in X (excluding x_{-1}) is odd, the sign (MSB) is extended one position to ensure that the last triplet contains 3 bits. At every step, a multiple of multiplicand Y is added to the partial product according to Table I and the result shifted right by 2 bits. If n represents the (even) number of bits in the multiplier X after any sign-extension is performed, then a total of $n/2$ iterations would be required in a multiplication $X \times Y$, two bits of X retiring and two product bits resulting from every step.

[1] This and all the following recoding algorithms can start at either end of X. In this paper, we start the recoding always from x_{-1}.
[2] A proof of the Modified Booth Algorithm was later given by Rubinfield [3].

0018-9340/90/0800-1006$01.00

TABLE II
EXAMPLES OF SD-REPRESENTATION IN RADIX 4

X in Decimal	X in 2C Format	X in SD radix 4
-29	100011	$\bar{2}\,1\,\bar{1}$
-5	111011	$0\,\bar{1}\,\bar{1}$
0	000000	0 0 0
+12	001100	$1\,\bar{1}\,0$
+23	010111	$1\,2\,\bar{1}$

The advantage of the Modified Booth's Algorithm (MBA) over Booth's original algorithm lies in the reduced number of iterations required, the reduction being from n steps to $n/2$ steps for MBA. This improves both speed and hardware requirements (in the case of a hardware parallel multiplier) when doing a multiply operation. This gain is possible at the expense of somewhat more complex operations at every step. It should be noted, however, that the required multiples of Y (0, $\pm Y$, $\pm 2Y$) are available by merely shifting Y to the left, provided subtraction as well as addition can be done at every step.

Although the algorithms and operations specified above seem rather arbitrary at the first sight, they are both based on meaningful number systems. If one focuses on what in effect is being done to X in the two schemes and ignores the multiplicand Y and the multiplication process altogether, then one may arrive at a different representation for the two's complement number X. In the case of MBA, then, we may arrive at the following representation for X:

$$X = \sum_{i=0}^{n/2-1} D_i \cdot 4^i \tag{1.1}$$

where digits D_i are one of $\{-2, -1, 0, +1, +2\}$ found from Table I based on the value of triplets of form (x_{i+2}, x_{i+1}, x_i) by looking up the Y coefficients in that table. Here we have a signed-digit (SD) representation of X in radix 4 (note that the radix 4 representation corresponds to 2-bit shifts of partial product in the MBA multiplication). Some examples may help to illuminate the concept. Applying the MBA scheme mentioned before to the multipliers -29, -5, 0, $+12$, and $+23$ results in the radix 4 numbers shown in Table II. The negative sign of the digit is usually shown as a bar above the digit to aid the clarity of representation. The two's complement (2C) format is assumed to be 6 bits wide in these examples. The first signed digit of -29 was found by appending a 0 to the right of LSB. Then the first triplet (110) was looked up in Table I to correspond to -1. Other digits are found similarly by looking up digits corresponding to triplet values (sharing one bit from the previous triplet) from Table I.

But how are the entries in Table I obtained? Can these results be extended to other radices? And if so, what ensures that algebraic equivalence is preserved? Unfortunately the available literature seldom refers to the number systems underlying recoding and therefore makes such questions difficult to answer. The following section attempts to answer such questions by developing a multibit recoding algorithm which is based on signed-digit number systems.

II. SD Representation and Multibit Recoding

A signed-digit number system [4] in radix r is based on a redundant set of signed digits

$$\{-\alpha, -(\alpha-1), \cdots, -1, 0, 1, \cdots, \alpha-1, \alpha\}$$

where

$$\alpha = \left\lfloor \frac{r}{2} \right\rfloor \tag{2.1.0}$$

is chosen for minimum redundancy. Note that the total number of digits available $(2\alpha + 1)$ does imply a redundant number system in radix r (because $r \leq 2\alpha + 1$). For example, if $r = 8$ then $\alpha = 4$ and a redundant set of 9 signed digits $\{0, \pm 1, \pm 2, \pm 3, \pm 4\}$ is used to represent numbers in this system. Schemes of conversion from decimal numbers to a given SD representation exist (see [4] for example) but are usually not amenable to machine implementation due to their complexity. We will show that if we start from a binary 2C number then the SD representation in radix 2^k can be obtained through what may be called a generalization of Booth's recoding algorithm.

A. Multibit Recoding Algorithm

Let X be an n-bit two's complement format binary integer. The value of X can then be found from:[3]

$$X = -x_{n-1} \cdot 2^{n-1} + \sum_{q=0}^{n-2} x_q \cdot 2^q. \tag{2.1.1}$$

Note that this is a uniform representation for both positive and negative numbers. X is positive if $x_{n-1} = 0$ and is negative if $x_{n-1} = 1$. An SD representation of X in radix 2^k ($k \geq 1$) will have n/k signed digits[4] $D_{n/k-1}, D_{n/k-2}, \cdots, D_1, D_0$. In this new radix ($2^k$) the value of X can be rewritten as

$$X = \sum_{i=0}^{n/k-1} D_i \cdot 2^{ki} \tag{2.1.2}$$

where digits D_i are found from bits x_i of X according to

$$D_i = x_{ki-1} + \left(\sum_{j=1}^{k-1} x_{ki+j-1} \cdot 2^{j-1} \right) - x_{k(i+1)-1} \cdot 2^{k-1} \tag{2.1.3}$$

[3] Binary *integers* are chosen to work with in this paper because their treatment is easier to follow than binary fractions which require negative powers in their representation. Obviously, the results derived are independent of the format used to represent the numbers and are equally valid for any binary 2C number. An n-bit 2C fraction can be conveniently converted to an n-bit integer by multiplying it by 2^{n-1}.

[4] It is easily shown that sign-extension in 2C numbers does not change the value of the number. Therefore, if the number of bits in X is not divisible by k then the sign of X can be extended as many positions as required (at most $k-1$) to make the new number of bits divisible by k. In this paper, n represents the number of bits in X after any sign extensions are performed and hence is always divisible by k.

and

$$x_{-1} = 0 \tag{2.1.4}$$

by definition which represents the appended 0 bit to the right of x_0.

Note that (2.1.3) simply evaluates the inner product

$$[x_{k(i+1)-1} \quad x_{k(i+1)-2} \quad \cdots \quad x_{ki} \quad x_{ki-1}] \cdot \begin{bmatrix} -2^{k-1} \\ 2^{k-2} \\ 2^{k-3} \\ \vdots \\ 2^1 \\ 2^0 \\ 1 \end{bmatrix}. \tag{2.1.5}$$

If we let $\boldsymbol{X}_i$ and $\boldsymbol{K}$ represent the two vectors above then (2.1.3) can be rewritten in short as

$$D_i = \boldsymbol{X}_i \cdot \boldsymbol{K}. \tag{2.1.6}$$

The two vectors in (2.1.5) above each have $k+1$ elements and therefore a $k+1$-bit recoding process corresponds to performing n/k such inner products. Some explanation of various elements in (2.1.3) may help one better see this correspondence. The subscripts of x of the form $k \cdot i$ plus an index refer to the fact that k new bits of X (plus an overlapped bit) are considered at a time for each signed digit D_i. For each digit D_i the subscripts of x range from $k \cdot i - 1$ to $k(i+1) - 1$ covering the same $k+1$ bits of X as specified in the first vector in (2.1.5). This digit calculation needs to be repeated n/k times so that (2.1.2) fully evaluates X. One can also easily see that digits D_i and D_{i+1} share bit $x_{k(i+1)-1}$. This indicates the overlapping bit between adjacent $k+1$-tuples. Note, however, that the coefficient of this bit is -2^{k-1} when calculating D_i while a coefficient of 1 is used for this bit when calculating D_{i+1}. The minimum value of D_i is found by letting X_i be equal to $[1\ 0 \cdots 0]$ which yields -2^{k-1} and the maximum is 2^{k-1} obtained by letting X_i be equal to $[0\ 1 \cdots 1]$. All the integer values in this range can be assumed by digits D_i when X_i assumes all its possible binary values. Finally note that the second vector in (2.1.5) is a constant vector which is fixed for a given k (i.e., for a given radix 2^k). The whole process of calculating the new digits from the binary numbers using the inner product (2.1.6) can therefore be thought of as a discrete convolution between the input data sequence X and the fixed response vector $\boldsymbol{K}$. The only difference is that in a real convolution the input data are shifted only once when a new output is to be calculated whereas here it is shifted k times between two output digits.

The above results on multibit recoding are summarized in Section II-C. The next section proves that the above equations lead to a valid representation of X in radix 2^k. It can be skipped if desired without any loss to the continuity of the material presented.

B. A Proof of the Multibit Recoding Algorithm

We want to show that digits D_i as specified in (2.1.3) above correctly represent X in radix 2^k. Note that based on these signed digits, the value of X is represented as in relation (2.1.2). The proof consists of substituting the value of D_i from (2.1.3) into (2.1.2) and showing that the result is the same as the two's complement binary value of X as given by (2.1.1).

After substituting the value of D_i from (2.1.3) into (2.1.2) and rearranging we obtain

$$X = \sum_{i=0}^{n/k-1} x_{ki-1}2^{ki} - x_{k(i+1)-1}2^{ki+k-1} + \sum_{i=0}^{n/k-1}\sum_{j=1}^{k-1} x_{ki+j-1}2^{ki+j-1}. \tag{2.2.1}$$

The first summation can be expanded and rearranged as follows:

$$\begin{aligned} &x_{-1}2^0 - x_{k-1}2^{k-1} + x_{k-1}2^k - x_{2k-1}2^{2k-1} + x_{2k-1}2^{2k} - \cdots \\ &\quad - x_{n-k-1}2^{n-k-1} + x_{n-k-1}2^{n-k} - x_{n-1}2^{n-1} \\ &\quad = x_{-1}2^0 + x_{k-1}2^{k-1} + x_{2k-1}2^{2k-1} + \cdots \\ &\quad + x_{n-k-1}2^{n-k-1} - x_{n-1}2^{n-1}. \end{aligned}$$

Using the fact that $x_{-1} = 0$ to replace 2^0 with 2^{-1} for the coefficient of x_{-1} and bringing the negative term first, the above value for the first summation of (2.2.1) can be rewritten as

$$\begin{aligned} &-x_{n-1}2^{n-1} + x_{-1}2^{-1} + x_{k-1}2^{k-1} + x_{2k-1}2^{2k-1} + \cdots \\ &\qquad + x_{n-k-1}2^{n-k-1} = -x_{n-1}2^{n-1} + \sum_{i=0}^{n/k-1} x_{ki-1}2^{ki-1}. \end{aligned}$$

Substituting this value for the first summation into (2.2.1) we obtain

$$\begin{aligned} X &= -x_{n-1}2^{n-1} + \sum_{i=0}^{n/k-1} x_{ki-1}2^{ki-1} \\ &\quad + \sum_{i=0}^{n/k-1}\sum_{j=1}^{k-1} x_{ki+j-1}2^{ki+j-1} \\ &= -x_{n-1}2^{n-1} + \sum_{i=0}^{n/k-1}\sum_{j=0}^{k-1} x_{ki+j-1}2^{ki+j-1}. \end{aligned} \tag{2.2.2}$$

Expansion can easily show that, for constant integers k, q, and t, the following relationship always holds true:

$$\sum_{i=0}^{q}\sum_{j=0}^{k-1} x_{ki+j+t} = \sum_{u=t}^{kq+k-1+t} x_u. \tag{2.2.3}$$

Equation (2.2.3) above reflects two different ways of finding the sum of elements in an array with $k(q+1)$ elements. On the left, we put the elements of the array into a two-dimensional

array with $q+1$ rows and k columns and use the double summation to add the sums of individual row elements together. On the other hand, the right side shows the sum of all elements as ordinarily found from a one-dimensional array. In order to use (2.2.3) above to unfold the double sum in (2.2.2) into a single sum, we should replace t with -1 and q with $n/k-1$. Therefore, the new limits of the summation will go from -1 to $k(n/k-1)+k-1-1$ which is equal to $n-2$. Now x_{-1} is equal to 0 by definition and therefore (2.2.2) results in

$$X = -x_{n-1}2^{n-1} + \sum_{u=0}^{n-2} x_u 2^u. \qquad (2.2.4)$$

The proof is therefore complete.[5] The above value of X is identical to the one in (2.1.1) for an n-bit two's complement binary number X and therefore shows that recoding does not alter the algebraic value of the binary number. It also proves that any two's complement number can be represented in any radix 2^k ($k \geq 1$) by signed digits D_i as given by (2.1.3). Finally note that $x_{-1} = 0$ is merely a convention required for relation (2.1.3) to calculate the value of D_0 correctly. If D_0 is treated separately by letting X_0 contain k bits instead of $k+1$ bits, thus dropping x_{-1}, then this convention may be dropped.

C. Summary of Multibit Recoding

The multibit recoding algorithm can be summarized as follows.

An SD representation of a 2C binary number X in a given base 2^k is obtained using the following steps.

1) Extend the sign bit of X by as many positions as necessary (at most $k-1$ positions) so that the total number of bits n is divisible by k.

2) Append a 0 to the right of the LSB of X and denote this bit by x_{-1}.

3) Form vectors of $k+1$ bits starting from x_{-1} (see footnote 1) such that adjacent vectors share one bit. For example, the MSB of vector X_0 is the same as the LSB of vector X_1.

4) The inner product of each vector X_i with the constant vector

$$\boldsymbol{K} = [-2^{k-1} \quad 2^{k-2} \quad 2^{k-3} \quad \cdots \quad 2^1 \quad 2^0 \quad 1]$$

yields a signed digit D_i. Alternatively, relation (2.1.3) can be used to calculate D_i.

5) Digits D_0 through $D_{n/k-1}$ obtained above represent X in base 2^k. Each digit may assume values in the range -2^{k-1} to 2^{k-1} including 0.

To ease the digit computations, one can form tables showing the digits corresponding to all possible $k+1$-tuples for a given k. Given are Tables III–VI for $k=1$, $k=2$, $k=3$, and $k=4$ corresponding to radices 2, 4, 8, and 16, respectively. Note that the Modified Booth Algorithm of Table I corresponds to the multibit recoding with $k=2$ as shown in Table IV.

[5] The authors recently learned of another proof for the general recoding case [5]. In that proof, the authors do not take advantage of the signed-digit number system underlying the recoding process and restrict their treatment to multiplier implementations. The ensuing proof is therefore quite complicated compared to the one given here. See [5] for more information.

TABLE III
2-BIT RECODING ($k=1$)

Pair Value	Signed Digit Value
00	0
01	+1
10	-1
11	0

TABLE IV
3-BIT RECODING ($k=2$)

Triplet Value	Signed Digit Value
000	0
001	+1
010	+1
011	+2
100	−2
101	−1
110	−1
111	0

TABLE V
4-BIT RECODING ($k=3$)

Quadruplet Value	Signed Digit Value
0000	0
0001	+1
0010	+1
0011	+2
0100	+2
0101	+3
0110	+3
0111	+4
1000	−4
1001	−3
1010	−3
1011	−2
1100	−2
1101	−1
1110	−1
1111	0

Also, the original Booth recoding technique (see Section I) corresponds to the case of $k=1$ as shown in Table III.

These tables can be implemented in hardware, as is the case with parallel multipliers and SD adders, or in software/firmware, as is the case with software multiply instructions.

Recoding of binary numbers has been used to implement various arithmetic blocks such as multipliers, dividers, and signed-digit adders (see [4]). In the remainder of this paper, however, we limit our treatment to applications of recoding to hardware parallel multipliers.

III. APPLICATIONS OF RECODING IN MULTIPLIERS

Various classes of multipliers such as serial multipliers, parallel multipliers, and the microprogrammed or software multipliers have all used the Modified Booth Algorithm to reduce the multiplication time. We will focus on the applications of recoding to hardware parallel multipliers and will show how multibit recoding may affect the design and performance of this class of multipliers. Afterwards, some special cases are covered for which multibit recoding may prove particularly advantageous.

TABLE VI
5-BIT RECODING ($k = 4$)

Quintuplet Value	Signed Digit Value
00000	0
00001	+1
00010	+1
00011	+2
00100	+2
00101	+3
00110	+3
00111	+4
01000	+4
01001	+5
01010	+5
01011	+6
01100	+6
01101	+7
01110	+7
01111	+8
10000	−8
10001	−7
10010	−7
10011	−6
10100	−6
10101	−5
10110	−5
10111	−4
11000	−4
11001	−3
11010	−3
11011	−2
11100	−2
11101	−1
11110	−1
11111	0

As mentioned in Section I, hardware multipliers have used recoding of the multiplier number ever since the idea was first proposed. The dominant scheme has been the Modified Booth Algorithm (MBA) used along with the so-called carry-save adder (CSA) array to implement a parallel multiplier. This results in a very regular layout for the multiplier as well as high speed of multiplication. 3-bit recoding of the multiplier number (X in the $X \times Y$ operation) in effect results in multiplication in the higher radix 4 and achieves a more parallel operation. This is achieved by multiplying the multiplicand Y by signed digits 0, ± 1, and ± 2 instead of by binary digits 0 and 1 at each step of operation. It is intuitively obvious that in a higher radix, for example using a 5-bit recoding, fewer operations would be required to finish the multiplication provided multiplication by the higher digits can be easily accommodated. This requires very high speed carry-propagate adders to generate odd-digit multiples of the multiplicand Y. To this end, a high-speed carry-propagate (CP) adder has been designed which makes it possible to generate the odd multiples of Y in a time comparable to the recoding time of X. The adder is of the carry-select type [6] and is designed to optimize the carry-propagate and carry-select paths for higher speed, thus achieving the performance required for higher-radix recoding.

Here we will first review the architecture of a conventional multiplier using MBA. Then a multiplier architecture is proposed which uses 5-bit recoding of X along with the optimized carry-select adders for the generation of odd multiples. Various considerations for multipliers based on multibit recoding technique are addressed in the end.

A. 3-Bit Recoded Multiplier [4], [7]

Fig. 1 shows the architecture of a typical parallel multiplier using MBA.

The multiplier X is n bits wide and the multiplicand Y has m bits. There are $n/2$ 3-bit Booth encoders each taking 3 bits of the X and generating control lines to select one of the multiples of the Y ($0, \pm Y, \pm 2Y$) to be applied to the inputs of one row of the carry-save adder (CSA) array. Each encoder implements the truth table of Table IV which corresponds to radix 4 representation ($k = 2$) for X. Each row of the CSA array consists of $m + 1$ selectors each feeding an adder (no adders for the first row, half adders for the second row, and full adders for the rest). Each row of the selectors therefore selects one of the different multiples of Y and applies it as one input to the adder row. It should be emphasized again that the only multiple of Y requiring special treatment is $2Y$ which can be easily obtained by shifting Y one position to the left (by hardwired displacement of Y one bit to the left in hardware). Negative multiples are obtained by complementing the multiple and then adding a 1 in the final twin adder.

The sums and carries of each row of the array reflect the partial product accumulated up to that point and in turn are applied to the next row to be added to a new multiple of Y. Each row also produces 2 bits of the product in the same way that a long-hand multiplication produces one digit of the product each time two consecutive results are added. After the first n bits of the product are produced on the right through the twin adders (i.e., adders with 2-bit carry look-ahead (CLA) which propagate the carry from one row to the next), m bits of carry propagation is required to produce the final m bits of the product. This can be done through any m-bit carry propagate (CP) adder and twin adders are used here to generate these product bits faster.

The most important attribute of a parallel multiplier is its speed performance. This is usually the driving force for a parallel multiplier. Next is its silicon area and dimensions. For the above multiplier, the multiplication time can be divided into the following components:

- The Booth encoder delay. With the current 0.9 μm CMOS technology a delay of 3 ns is typical. Note that the encoders are used in parallel and therefore their delays do not accumulate.
- The delay of the CSA array or of the $n/2$ twin adders producing lower product bits, whichever is larger. Usually, the twin adders are designed to outperform the adder array for optimum performance. This delay is then equal to $(n/2 - 1) \times$ (one carry-save adder delay). A delay of about 1.80–2.0 ns is obtained with the current technology for a carry-save adder.
- The delay of the final carry propagation[6] which is equal to $m/2 \times$ (one twin adder delay). The twin adder's carry path delay is about 2 ns for the current technology.

The area of a multiplier can be roughly measured by looking at the number of rows and columns that make up the multiplier

[6] A more accurate estimate for the carry propagation should account for the setup time of the first stage of the CLA circuit. This delay increases as the size of CLA is increased.

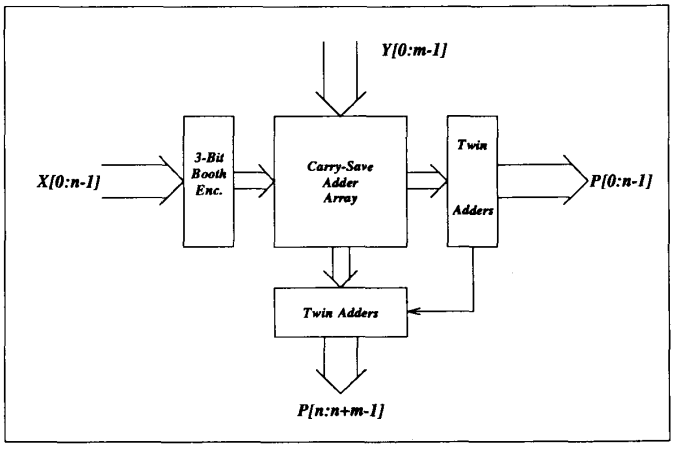

Fig. 1. Parallel multiplier with 3-bit recoding.

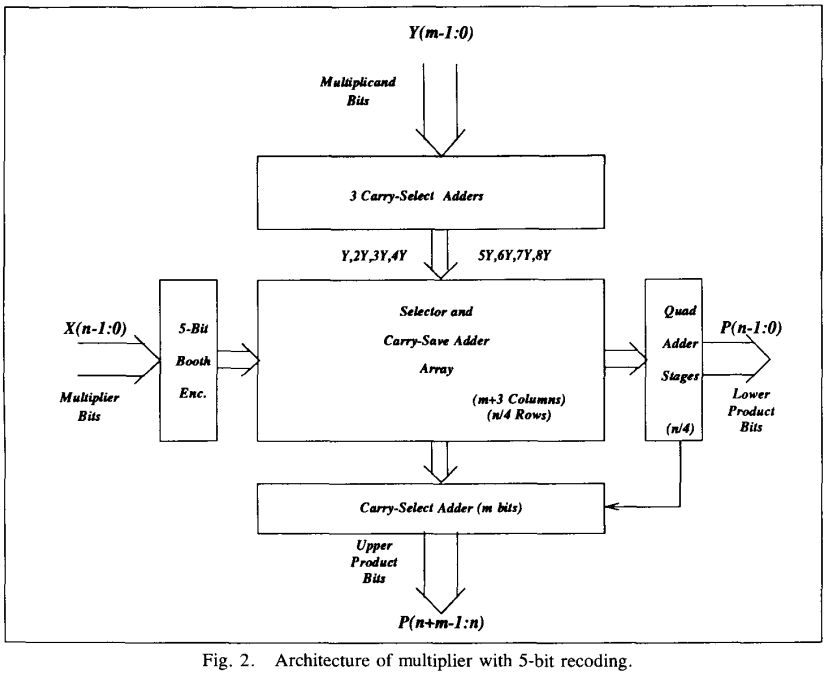

Fig. 2. Architecture of multiplier with 5-bit recoding.

as well as the relative complexity of each cell making up the rows or columns. In this case, we are dealing with $n/2$ rows by $m + 1$ columns. The elements making up each row are one 3-bit Booth encoder, $m + 1$ selectors (1 of 5) and adders, and one twin adder. A recent 16×16 multiplier using 3-bit recoding in 0.9 μm CMOS has a delay of 25 ns and an area of 0.73 mm^2. Other examples of 3-bit recoded multiplier design as well as the design issues involved can be found from the reference list [8], [9].

B. Proposal for a 5-Bit Recoded Multiplier

Maybe the most formidable obstacle in going to higher bit recoding schemes has been the generation of odd multiples of the multiplicand. Such multiples can be formed by addition or subtraction of two (or possibly more) power-of-2 multiples of Y. Such a scheme would have an acceptable performance if the adder delay can be justified in relation to the overall performance gain of the multiplier. Since the generation of the multiples of Y takes place in parallel with the recoding of the multiplier X, it is highly desirable to have an m-bit adder delay comparable to the delay of the recoder. The optimized carry-select adder mentioned before achieves such a performance goal.

Given in Fig. 2 is a block diagram of the 5-bit recoding parallel multiplier. The salient features of this multiplier which set it apart from the previous one are as follows.

1) The multiplier is now recoded in 5-bit groups. This corresponds to radix 16 representation ($k = 4$) for X using signed digits $0, \pm 1, \cdots, \pm 8$. Each recoder cell is larger (five inputs and ten outputs) than before but only $n/4$ of these

TABLE VII
ADDERS NEEDED FOR PROPOSED MULTIPLIER

Multiple	Operation	Adder Size
3Y	Y + 2Y	m+1
5Y	Y + 4Y	m+2
7Y	8Y - Y	m+3

are required as compared to $n/2$ for MBA. The ten outputs of the recoder select one of the 17 different multiples of Y (0, $\pm Y$, $\pm 2Y$, $\pm 3Y$, $\pm 4Y$, $\pm 5Y$, $\pm 6Y$, $\pm 7Y$, $\pm 8Y$). The ten select lines can be mnemonically referred to as Sign, Zero, One, Two, Three, Four, Five, Six, Seven, and Eight. The Boolean equations to implement these signals are easily obtained from Table VI and have some minterms in common.

2) The carry-select adder block is a new block which does not exist in the 3-bit recoded multiplier. It consists of three adders which are used to obtain the odd multiples of Y according to Table VII. $6Y$ is obtained by displacing $3Y$ one position to the left. All the other multiples (Y, $2Y$, $4Y$, and $8Y$) are available by shifting Y (i.e., hardwired displacement) to the left by 0, 1, 2, or 3 bits, respectively.

3) The CSA array now has only $n/4$ rows instead of $n/2$. Each row consists of $m+3$ selectors selecting one of the nine multiples ($0Y$ through $8Y$) with the Sign selecting addition or subtraction. Again the first row has no adders, the second row has half adders, and the rest are full adders. Therefore, the delay of CSA array is approximately equal to $(n/4-1)\times$ (one carry-save adder delay).

4) Each row of the array generates four product bits instead of two. This requires the use of fast carry-propagate adders to generate the 4 product bits and the carry input to the next stage in the same time interval that a carry-save adder row is updating the partial product. The use of quad adders with 4-bit carry look-ahead (CLA) achieves the under 2 ns carry delay required here.

5) The final m bits of the product are generated by an m-bit carry-select adder. This can be done due to the timing nature of the inputs to the final carry-propagate stage which makes all the inputs available by the time the first input (bottom right-hand side) is ready. Another speedup, therefore, is achieved here through the carry-select adder with representative delays of 8 ns for 16 bits, 9.5 ns for 24 bits, and 10.5 ns for 32 bits.

The above features of the new multiplier make it an attractive choice because of its higher speed performance and at the same time good area cost while keeping the regular layout structure. The higher speed is achieved by reducing the array delay by more than half which can be specially significant for very large multiplier sizes (say 48×48). The array delay reduction is gained at the expense of a few nanoseconds of penalty, i.e., the difference between the $m+3$-bit carry-select delay (for generating $7Y$) and the recoder delay. The use of carry-select adders for the final propagation of carry improves the performance even further. As far as the area is concerned, great improvements are obtained by reducing the size of the array by half while paying the price of somewhat bigger selectors and recoders as well as the three additional adder blocks. Whether or not area is increased significantly is to a good degree a function of the layout techniques used to implement the multiplier and of the size of the multiplier itself (i.e., values of m and n). It is believed that superior performance is possible at the expense of at worst a little area penalty using this scheme with both area and performance enhancing further as the word size of multiplier and multiplicand (n and m) are increased. This is in contrast to the Wallace tree [10] implementations which have highly irregular layouts while their area penalty for speed improvement is rather severe. Tables VIII and IX show some of the expected delays for 5-bit scheme compared to the 3-bit (MBA) scheme.

TABLE VIII
REPRESENTATIVE DELAYS FOR 3-BIT RECODED MULTIPLIERS

Multiplier Size	16×16	24×24	36×36	48×48	64×64
Recoder Delay	4ns	4ns	4ns	4ns	4ns
CSA Array Delay	14ns	22ns	34ns	46ns	62ns
Final Carry Propagate Delay	11ns	16ns	24ns	32ns	42ns
Total Delay	29ns	42ns	62ns	82ns	108ns

TABLE IX
REPRESENTATIVE DELAYS FOR 5-BIT RECODED MULTIPLIERS WITH CARRY-SELECT USED FOR THE FINAL STAGE

Multiplier Size	16×16	24×24	36×36	48×48	64×64
m-Bit Carry Select Delay	8ns	9.5ns	11ns	12.5ns	14ns
CSA Array Delay	6ns	10ns	16ns	22ns	30ns
Final Carry Select Delay	8ns	9.5ns	11ns	12.5ns	14ns
Total Delay	22ns	29ns	37ns	47ns	58ns

TABLE X
VARIOUS FEATURES OF MULTIPLIERS USING MULTIBIT RECODING

Recoding Size	No. of SD's	Selector Size	No. of Extra Adders	Normalized CSA Delay	CLA Size
3 Bits	5	1-of-2	0	1.0	2
4 Bits	9	1-of-4	1	0.66	3
5 Bits	17	1-of-8	3	0.50	4
6 Bits	33	1-of-16	7+2	0.40	5
7 Bits	65	1-of-32	15+8	0.33	6

All of the above delays are based on ADVICE [11] simulations of the individual cells under the worst case slow process files at 125°C and 4.2 V supply. Note that in Table IX the recoder delay is replaced by the delay of the carry-select adder generating the odd multiples of Y because this delay is larger than the recoding delay. As is evident from the above delay figures, the future generations of multipliers can particularly benefit from the proposed architecture and the speed gain it offers.

C. Multiplying by a Fixed or Controlled Coefficient

It is worthwhile noting here that in specialized DSP applications, where a multiplier can be dedicated to multiplying input data by a fixed coefficient, considerable savings in area

can be achieved which reduce the area below the equivalent 3-bit implementation. This is because, assuming that the fixed coefficient is applied to the multiplier X, recoders and selectors will no longer be necessary and direct hardwiring of various multiples of Y to various rows of the array will serve the purpose.

Another interesting speed and area enhancement in this case may be possible by approximating the fixed coefficient by another coefficient which uses only quintuplets corresponding to powers of 2 (see Table VI). If such an approximation is possible and acceptable then the three carry-select adders for generating odd multiples of Y can also be removed thereby improving both the area and speed even further.

It seems that one should be able to find analytic transformations to approximate any binary 2C number by another one which can be represented by power-of-2 signed digits in a given radix 2^k (in this case in radix 16). If such a transformation exists and can be applied satisfactorily to the multiplier numbers[7] X or if the algorithms using multiplication can be modified to use only coefficients with power-of-2 signed digits, then a more general multiplier may also do away with the three carry-select adders and their associated delay. In this case, the selectors and the recoding logic are also simplified. Such special purpose multipliers should be studied as potential options for dedicated applications where the higher speed and the smaller area offered are very desirable.

D. Considerations for Higher Bit Recoding

Every time the recoding size is increased by one bit, the number of digits required to represent the binary number is doubled (not counting 0). This means that a multiplier to be implemented with a recoding size of one bit more will require selectors which should select from twice as many digits as before. Also the number of odd multiples of the multiplicand is almost doubled requiring more carry-select adders to implement those multiples. On the other hand, with recoding sizes of more than 5 bits, there are some multiples of Y (such as $11Y$) which can only be obtained by addition of more than two power-of-2 multiples (in this case $8Y$, $2Y$, and Y). Such multiples can be formed by using carry-save adders to reduce the summands (i.e., power of two multiples to be added) to two numbers and then by a carry-select adder to obtain the required multiple. However, the operations required and the number and complexity of circuits needed to implement them very quickly prohibit a practical multiplier design using higher bit recoding schemes. Given in Table X are some of the hardware requirements for various multipliers using $k+1$-bit recoding. For recoding sizes of more than 5 bits, the two different types of adders required are shown as the sum of two numbers under the Extra Adders column. The first number is the number of carry-select adders and the second one of the number of carry-save adders. More than two additions may become necessary for higher bit recodings.

Another design difficulty in going to higher bit recoding comes from the fact that in a $k+1$-bit recoded multiplier, k bits of the product fall off from each row of the CSA array. This requires high-speed k-bit adders to produce k bits of the product and the carry into the next stage in the same time that one row of the CSA array produces the outputs into the next row. This was the reason for using quad adders (i.e., 4-bit CLA adders) in the proposed architecture and currently this should be done in under 2 ns. Obviously the task becomes more formidable as the number of carry-propagations to be done in this time period is increased. The setup time needed for a larger size CLA adder is also increased making the design goal further difficult to achieve. The size of carry-look-ahead circuit needed for various recoding sizes is shown in the last column of Table X.

The main advantage of going to a higher bit encoding lies in its reducing the CSA array delay. Since the delay of the CSA array is only halved by *doubling* the size of k and since the addition of each bit to recoding size requires about *twice* as many digits to generate and choose from, then the point of diminishing returns is reached very quickly considering the exponential growth of hardware elements and the obstacles mentioned above. We believe that 5 bits ($k=4$) is the optimum recoding size given that the delay of CSA array is halved compared to the 3-bit case by requiring only three extra additions to be performed. The 1-of-8 selectors needed also seem to be manageable specially from a layout standpoint where the selectors can feed the adders without compromising the compactness of the layout. The height of the selector cell will increase in the same direction as the reduction of the rows of the CSA array and thus the overall aspect ratio of the multiplier is not considerably changed. Finally the 4-bit CLA circuit needed is manageable from a design and area standpoint as experiments by authors have shown to be the case. As a final indication of the optimality of the 5-bit recoding, a comparison to the 9-bit recoding is given. The use of 9-bit recoding would offer only a modest 25% improvement in the array delay over the 5-bit case while requiring 1-of-128 selectors and the generation of 63 odd multiples of Y!

An estimating formula is presented below which provides the delay figure for a parallel multiplier implemented via a CSA array and using $k+1$-bit recoding of the multiplier X. It is assumed that X and Y are, respectively, n bits and m bits wide. It is further assumed that the k-bit carry-look-ahead delay is less than or equal to the delay of one stage of the CSA array.

$$\begin{aligned} T_{k+1}(n \times m) = \max[&((k+1)\text{-bit recoder delay}), \\ &((m+k-1)\text{-bit CP adder delay} + (\gamma - 1) \\ &\times (\text{one CSA stage delay}))] \\ &+ \left(\frac{n}{k-1} - 1\right) \times (\text{one CSA stage delay}) \\ &+ (m\text{-bit CP adder delay}). \end{aligned} \tag{3.4.1}$$

In (3.4.1) above, the first term represents the maximum of the two delays associated with parallel operations of recoding and odd-digit multiple generation. In generating odd multiples, at most γ additions (of $\gamma+1$ shifted values of Y) should be performed where γ is a heuristic number associated with the

[7] An example would be when the multiplier coefficients are stored in a ROM. In this case, the approximation can be done before storing the values so that no extra operations take place before multiplication.

TABLE XI
WORST CASE NUMBER OF ADDITIONS (γ) TO GENERATE ODD MULTIPLES FOR DIFFERENT RECODING SIZES

Recoding Size (k+1)	2	3	4	5	6	7	8	9
γ	0	0	1	1	2	2	3	3

"worst case" multiples of Y for the given recoding size of $k+1$. The first γ values would be reduced through $\gamma - 1$ CSA stages. The final carry propagation to generate the odd multiple would be performed by a CP adder, such as a carry-select adder, which would be at most $m + k - 1$ bits wide (corresponding to the multiple $[2^{k-1} - 1]Y$). The values of γ for recoding sizes of up to 9 are given in Table XI. As may be seen, a relation of the form $\gamma = \lfloor (k-1)/2 \rfloor$ is strongly suggested by the table.

The second term in the delay equation (3.4.1) corresponds to the CSA array delay which is approximately in inverse proportion to $k - 1$. Finally, the last term in the delay formula corresponds to the last m-bit carry propagation which generates the most significant bits of the product.

It is worth mentioning here that a 4-bit recoding scheme (corresponding to Table V with $k = 3$) is quite feasible and straightforward to implement by requiring only one carry-select adder to produce the term $3Y$ and 3-bit CLA adders to propagate the carry. The selector size of 1-of-4 is also rather convenient to design and work with. The improvement in the delay of the array (33%), however, is not as large as the 5-bit case. Nevertheless, it is a good choice for some moderate improvement in the speed of multiplication. It is interesting to note that MacSorley (see [2]) mentioned the case of 4-bit recoding at the same time as the Modified Booth Algorithm. However, it was not as widely adopted due to the difficulty in generating the term $3Y$.

Again attention should be paid to special purpose multipliers which multiply by fixed or controlled coefficients and the improved area and performance which may result from this (see Section III-C). It should be noted, however, that an increase in the recoding size implies coarser approximations if only power-of-2 signed digits are available. If the resulting approximations are satisfactory, then such multipliers may offer superior area and performance using higher bit recoding schemes.

IV. CONCLUSION

A generalized recoding scheme for signed two's complement binary numbers is presented. It is shown that multibit recoding of two's complement binary numbers corresponds to a signed-digit representation of the number in a power-of-2 radix and leads to a compact representation of the recoded number. The SD representation of the recoded number is used to prove the correctness of the multibit recoding algorithm. The proposed scheme has applications in implementing various arithmetic blocks such as multipliers, dividers, and signed-digit adders where recoding has traditionally been used. The specific case of hardware parallel multipliers is covered in some depth in the light of the multibit recoding algorithm. With the trend towards higher performance processors and arithmetic units, substantial performance improvements can be obtained by applying multibit recoding to the design and implementation of multiplier units. Such implementations are feasible due to emergence of fast carry-propagate adders. Fixed coefficient multipliers can be designed without the need for recoding blocks or selectors thereby reducing the area of the multiplier. Furthermore, it is suggested that by controlling the multiplier numbers to be representable by power-of-2 signed digits, the need for extra adders may be overcome with corresponding improvements in speed and area. Such circuits are no longer true "multipliers" in the conventional sense of the word but approximate the operation of a real multiplier for dedicated purposes which are tolerant of such approximations. Further theoretical work in this area may be necessary to establish approximation algorithms in a streamlined fashion. On the other hand, from a merely theoretical standpoint, it would be interesting to further generalize the proposed multibit recoding to include SD representation in a radix which is *not* a power of 2.

ACKNOWLEDGMENT

The authors would like to thank A. Fisher and J. Henry for the enthusiastic support and encouragement which has made this work possible. H. Scholz and K. Kolwicz provided the opportunity which originated the ideas expressed here and are gratefully acknowledged for their support and interest. H. Moscovitz and B. Krambeck received the ideas of this paper with great interest and are thanked for their encouragement. Discussions with L. Rigge, J. Fadavi-Ardekani, M. Thierbach, and P. Lin also provided much technical support and are gratefully acknowledged.

REFERENCES

[1] A. D. Booth, "A signed binary multiplication technique," *Quarterly J. Mechan. Appl. Math.*, vol. IV, part 2, 1951.
[2] O. L. MacSorley, "High speed arithmetic in binary computers," *Proc. IRE*, Jan. 1961.
[3] L. P. Rubinfield, "A proof of the modified Booth's algorithm for multiplication," *IEEE Trans. Comput.*, Oct. 1975.
[4] K. Hwang, *Computer Arithmetic Principles, Architecture and Design*. New York: Wiley, 1979.
[5] S. Vassiliadis, E. M. Schwarz, and D. J. Hanrahan, "A general proof for overlapped multiple-bit scanning multiplications," *IEEE Trans. Comput.*, Feb. 1989.
[6] J. J. F. Cavanagh, *Digital Computer Arithmetic Design and Implementation*. New York: McGraw-Hill, 1984, pp. 117-122.
[7] A. Gupta, "A 50 ns 16 × 16 bit 2's complement parallel multiplier," *Proc. ISELDECS*, pp. 747-749, 1987.
[8] M. Hatamian and G. L. Cash, "A 70-MHz 8-bit × 8-bit parallel pipelined multiplier in 2.5-μm CMOS," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 4, Aug. 1986.
[9] K.-C. Chu and R. Sharma, "A technology independent MOS multiplier generator," in *Proc. 21st Design Automat. Conf.*, 1984.
[10] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, Feb. 1964.

[11] L. W. Nagel, "ADVICE for circuit simulation," in *Proc. IEEE Int. Symp. Circuits Syst.*, Apr. 28, 1980.

Homayoon Sam received the B.S. degree from the University of California, Berkeley, in 1982 and the M.S. degree from the University of Michigan, Ann Arbor, in 1984, both in electrical engineering and computer science.

From 1984 to 1988 he worked as circuit and system designer with the Telecommunications Department of National Semiconductor. Since 1988 he has been a member of the Technical Staff with the Technology Implementation and the Signal Processing and Integrated Circuit Design Departments of AT&T Bell Laboratories where has been working on multiplier generators and DSP architecture. His research interests include computer algorithms, signal processing, and number theory.

Mr. Sam is a member of Tau Beta Pi and Eta Kappa Nu.

Arupratan Gupta (M'85) received the B.Tech. (Hons.) degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, in 1981 and the M.S. degree in electrical engineering from Washington State University, Pullman, in 1983.

From 1983 to 1985 he worked as a design engineer at the Peripheral Components division of Intel Corporation. Since 1985 he has been a member of the Technical Staff with the Signal Processing and Integrated Circuit Design Department of AT&T Bell Laboratories. His research interests are in the areas of computer architecture, high-performance arithmetic VLSI, and digital signal processing.

Mr. Gupta is a member of Tau Beta Pi and Sigma Xi.