Analysis of speed of a binary multiplier using a variable number of shifts per cycle

M. R. Patel* and K. H. Bennett

Department of Computer Science, University of Keele, Keele, Staffordshire ST5 5BG

Iterative binary multiplication can be speeded up by examining two or more bits of the multiplier in each cycle. Freeman (1967) has described an exact method for calculating the resultant gain in multiplication speed, based on the use of a discrete-time finite state system model. We have used Freeman's model to evaluate the gain in speed for various combinations of two design parameters—the maximum number of shifts per cycle, and the available multiples of the multiplicand. It is shown that a greater performance improvement is obtained by increasing the former parameter rather than the latter.

(Received February 1975)

In a well known technique for speeding up iterative binary multiplication, two or more bits of the multiplier are examined simultaneously in each cycle to determine what multiple of the multiplicand is to be added to or subtracted from the partial product. **Fig. 1** is the block diagram of the multiplier to be discussed.

The multiplicand and partial product reside in single length registers, whilst the multiplier is placed initially in the bottom half of a double length shift register. A multiplication cycle begins by the examination of the n least significant bits of the multiplier; this determines the multiple of the multiplicand that is entered into the adder, and later, the amount to shift that is used. Although the addition (or subtraction) is single length, the adder has an extra bit at the most significant end to hold any overflow (which must not be lost).

The output of the adder is moved into the top half of the shift register. A double length rightward shift is performed so that the product is built up in the lower half of the buffer register. Shifting is implemented by gated paths between the two double length registers. The shift also results in the loss of least significant multiplier bits (which are no longer required), at the same time presenting a new bit pattern in the n least significant bit positions. Flores (1963) has described this type of multipler in greater detail.

By examining n bits of the multiplier at a time, the multiplication is effectively performed to the base 2^n. In theory, this would require all multiples of the multiplicand from 1 to $2^n - 1$ to be available, but in practice the introduction of simple shift paths only generates multiples which are integral powers of two. When other multiples are needed, special action—such as shifting by less than n bits—is taken. This of course is the reason why the performance gain may be less than the ideal factor of n. It is our purpose to show how various combinations of multiples and shift size affect the performance gain.

As soon as we examine more than one bit of the multiplier at a time, extra hardware logic is required for data paths:

(*a*) to move multiples of the multiplicand into the adder

(*b*) to implement the variable number of shifts.

Note that the shift register is of double length, so that approximately twice the hardware is required for (*b*) in comparison to (*a*).

Thus the sum of shift paths and paths into the adder, for a multiplier which can shift up to n bits and has m paths into the adder is $(2n + m)$ paths. This expression gives an approximate measure of the hardware cost involved in providing the variable shift facility. (It is not exact partly because hardware cost is not necessarily linearly proportional to circuit complexity, and partly because it ignores additional overheads such as the extra adder position (see section 1.2) that may not already exist and the decoding of the multiplier bits).

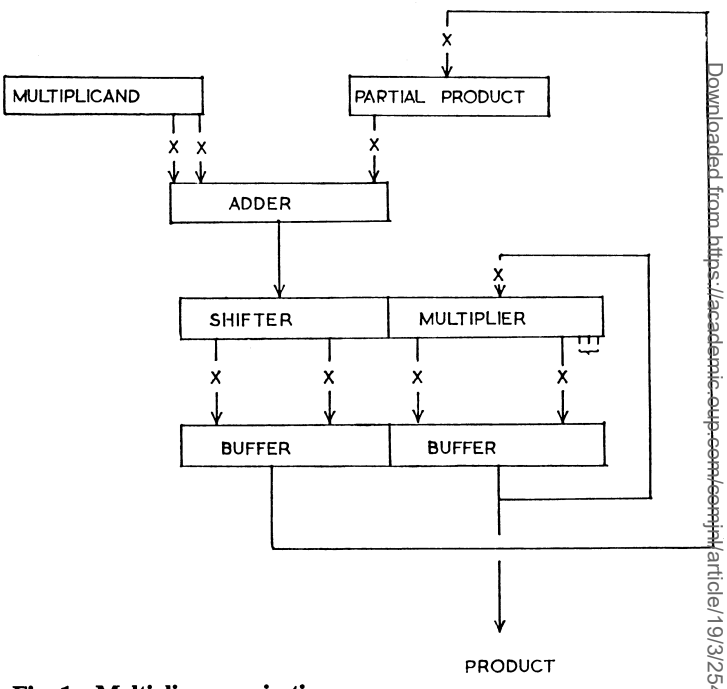

Fig. 1 Multiplier organisation

The optimum combination of n and m is not immediately clear, and the following analysis is designed to answer two questions:

1. Keeping the number of shift paths n constant, which specific m multiples of the multiplicand will give the fastest performance?
2. Keeping the sum of paths $(2n + m)$ constant, which combination of n and m will give the fastest performance?

The analysis is divided into three parts, for two, three and four shift paths. In each case, a selection of the possible combinations of the multiples of the multiplicand is considered and using Freeman's model, the mean shift per multiplication cycle is calculated. The chosen combinations illustrate both the analysis of the model and the functioning of the multiplier. The results for other combinations are presented in **Table 1.**

Similarly, higher numbers of shift paths are not considered explicitly, but the results are included in Table 1.

Simple iterative multiplication is usually found in small and medium sized computers. In large machines with a long word length, e.g. the CDC 6600 (1969) simple iteration is often

*International Computers Limited.

Table 1 Summary of results

Sum of paths	*Max. shift*	*Multiples of multiplicand*	*Limiting mean shift per cycle*
3	1	{+1}	1·0
5	2	{+1}{−1}	1·5
6	2	{+1,+2}*	1·667
6	2	{+1,−1}	1·8
7	2	{+1,−1,+2}{+1,−1,−2}	2·0
8	2	{+1,−1,+2,−2}	2·0
8	3	{+1,+2}*	1·857
8	3	{+1,−1}	2·333
9	3	{+1,+2,+4}	1·917
9	3	{+1,−1,+2}{+1,−1,−2}	2·50
10	3	{+1,−1,+2,−2}	2·571
10	3	{+1,−1,+2,+4}†	2·571
10	4	{+1,−1}	2·647
11	4	{+1,−1,+2}{+1,−1,−2}	2·75
12	4	{+1,+2,+4,+8}*	1·981
12	3	{+1,−1,+2,−1,+4,−4}	2·667
12	4	{+1,−1,+2,−2}	2·8
12	4	{+1,−1,+2,+4}†	2·8
12	5	{+1,−1}	2·818
13	5	{+1,−1,+2}{+1,−1,−2}	2·875
14	6	{+1,−1}	2·910

Notes:
*denotes all further combinations of sign.
†denotes all further combinations of sign of 2 and 4 only.

unacceptably slow. This drawback, together with possible requirements for parallelism in the arithmetic unit usually results in the adoption of other multiplication techniques.

1. Two bits at a time

1.1. ({+1} × *multiplicand*) *available*

This case is the simplest realisation of a variable shift multiplier. Consider the successor to a cycle in which a two bit shift has been made; the bit combination at the least significant end of the multiplier will be one of 00, 01, 10, 11, all with equal probability. In the first two cases, a two bit shift is always made (after addition for 01). In the last two cases, only a one bit shift is made because only the multiplicand itself and not any multiples of it is available to the adder; on the *next* iteration the current first (more significant) bit becomes the second bit of the pair. This bit must necessarily be one, so the only valid combinations are 01, 11, with equal probability. The following table summarises the shift sizes and actions of the adder:

Valid bit combination	*Action of adder*	*No. of shifts performed in this cycle (bits)*
00	NULL	2
01	ADD	2
10	NULL	1
11	ADD	1

When the adder performs a NULL action, the partial product is transferred without change to the shift register.

If a one bit shift is made in the current cycle, then only the 01 and 11 rows of the table are relevant in the next cycle.

Consideration of the above table leads to the following probabilities:

Current shift (bits)	*Following shift (bits)*	*Probability of transition*
2	2	0·5
1	2	0·5
2	1	0·5
1	1	0·5

In Freeman's model, the multiplication is regarded as a Markov process with two states, transition between the states being governed by the above probabilities. The system is considered to be in state j if the shift in the current multiplication cycle was j bits. Let $\pi(k)$ be the state probability row-vector at time k, so that an element $\pi_s(k)$ of $\pi(k)$ represents the probability of being in state s at time k. Also let P be the state transition probability matrix, i.e. an element P_{ij} of P represents the probability of a transition to state j at time $k+1$, given that we are in state i at time k. Then at time $k+1$,

$$\pi(k+1) = P.\pi(k),\ k \geqslant 0 \tag{1}$$

where

$$P = \begin{bmatrix} 0{\cdot}5 & 0{\cdot}5 \\ 0{\cdot}5 & 0{\cdot}5 \end{bmatrix}. \tag{2}$$

Also

$$\pi(0) = [0{\cdot}5 \quad 0{\cdot}5]\ . \tag{3}$$

All bit combinations are equally probable during the first multiplication cycle.

By definition, the sum of the elements of π must be 1, as must the sum of the elements of each row of P.

Equation 1 is solved by the Z (or geometric) transform technique, since this gives the rate of convergence to the limiting state probabilities (when appropriate). In the present case,

$$\pi(k) = [0{\cdot}5 \quad 0{\cdot}5]\ . \tag{4}$$

The *mean shift* is defined as:

$$\sum_{j=1}^{n} j \times \pi_j \tag{5}$$

where π_j is the probability of a j bit shift. The sum is over all possible shifts. For the current case, we find that the mean shift is 1·5 bits per cycle.

Howard (1971) has written a very palatable description of Markov processes and transform analysis.

1.2. ({+1, −1} × *multiplicand*) *available*

The addition of a (−1 × multiplicand) path permits special action when the least significant two bits of the multiplier are 11. The multiplicand is subtracted from the partial product and a two bit shift carried out; the subtraction will always result in the generation of a carry and this is recorded in a flip-flop. The carry is added into the least significant bit of the multiplier in the next cycle. The transition probabilities are not affected, since if the four combinations of the two bits are equally probable just before the addition of the carry, they will still be equally probable afterwards.

The above technique effectively treats (3 × multiplicand) as (−multiplicand + 4 × multiplicand).

The action of the multiplier is summarised in the table below. Apart from the last row, it is identical to the table in Section 1.1.

Valid bit combination	*Action of adder*	*No. of shifts performed in next cycle (bits)*
00	NULL	2
01	ADD	2
10	NULL	1
11	SUBTRACT	2

Only the 01 and 11 rows are relevant following a one bit shift. We can now deduce the transition probabilities and state them as the transition probability matrix (as for equation 1 and matrix 2).

$$P = \begin{bmatrix} 0 & 1{\cdot}00 \\ 0{\cdot}25 & 0{\cdot}75 \end{bmatrix}. \tag{6}$$

Also

$$\pi(0) = [0{\cdot}25 \quad 0{\cdot}75] \ . \qquad (7)$$

From equations 1, 6 and 7, transform analysis leads to the result:

$$\pi(k) = [0{\cdot}2 + 0{\cdot}05\,(-0{\cdot}25)^k \quad 0{\cdot}8 + 0{\cdot}05\,(-0{\cdot}25)^k] \qquad (8)$$

for $k \geqslant 0$.

From equations 5 and 8, the mean shift is:

$$1{\cdot}8 - 0{\cdot}05\,(-0{\cdot}25)^k \text{ bits per cycle} \ . \qquad (9)$$

This expression rapidly converges to its limiting value of 1·8 bits per cycle.

1.3. ($\{+1, +2\} \times$ *multiplicand*) *available*
This is precisely the case that has been analysed by Freeman (1967). The mean shift is 1·667 bits per cycle.

1.4. ($\{+1, -1, +2\} \times$ *multiplicand*) *available*
The three paths supply the three required multiples of the multiplicand (3 × multiplicand is treated as described in Section 1.2). Hence the mean shift attains its maximum value (for $n = 2$) of 2·0 bits per cycle.

2. Three bits at a time

2.1. ($\{+1, -1\} \times$ *multiplicand*) *available*
When three bits of the multiplier are examined at once, there are eight possible valid bit combinations, and three possible shift sizes; the multiplier is regarded as a three state Markov process. The possible actions are summarised below.

Valid bit combination	*Action of adder*	*No. of shifts performed in next cycle (bits)*
000	NULL	3
001	ADD	3
010	NULL	1
011	SUBTRACT	2
100	NULL	2
101	ADD	2
110	NULL	1
111	SUBTRACT	3

Subtraction is performed as described in Section 1.2; the carry is added into the least significant bit of the multiplier on the next iteration. However, in contrast to Section 1.2 the carry does affect the transition probabilities because subtraction can lead to a shift of two bits, rather than three bits. If the current combination is 011, then in the next cycle, before the carry is added, the possible bit combinations are 000, 010, 100, 110. The addition of 001 clearly changes the combinations.

All eight bit combinations are equally probable after a *three* bit shift.

After a *one* bit shift, there are four possible combinations, 001, 101, 011, 111.

After a *two* bit shift, considerations of the above argument lead to the four possible combinations 001, 101, 011, 111 (which are the same for a one bit shift).

The transition probabilities are:

$$P = \begin{bmatrix} 0 & 0{\cdot}5 & 0{\cdot}5 \\ 0 & 0{\cdot}5 & 0{\cdot}5 \\ 0{\cdot}25 & 0{\cdot}375 & 0{\cdot}375 \end{bmatrix} \ . \qquad (10)$$

The initial state probability vector $\pi(0)$ is

$$\pi(0) = [0{\cdot}25 \quad 0{\cdot}375 \quad 0{\cdot}375] \ . \qquad (11)$$

Using Equation 1 and the Z transform technique, the state probability vector at time k is found to be:

$$\pi(k) = [0{\cdot}111 + 0{\cdot}139\,(-0{\cdot}125)^k \quad 0{\cdot}444 - 0{\cdot}69\,(-0{\cdot}125)^k \quad 0{\cdot}444 - 0{\cdot}69\,(-0{\cdot}125)^k] \ . \qquad (12)$$

The mean shift (equation 5) is:

$$2{\cdot}333 - 0{\cdot}208\,(-0{\cdot}125)^k \ . \qquad (13)$$

This converges rapidly to a limiting value of 2·333 bits per cycle.

2.2. ($\{+1, +2, +4\} \times$ *multiplicand*) *available*
Freeman has analysed this configuration in (1967). He found the limiting value of the mean shift to be 1·917 bits per cycle.

The following combinations of three paths into the adder also produce the same mean shift:

$\{-1, +2, +4\}$ $\{-1, +2, -4\}$
$\{+1, -2, +4\}$ $\{+1, -2, -4\}$
$\{+1, +2, -4\}$ $\{-1, -2, -4\}$
$\{-1, -2, +4\}$.

3. Four bits at a time

3.1. ($\{+1, -1\} \times$ *multiplicand*) *available*
The actions following the sixteen possible bit combinations are given in the table below.

Valid bit combination	*Action of adder*	*No. of shifts performed in next cycle (bits)*
0000	NULL	4
0001	ADD	4
0010	NULL	1
0011	SUBTRACT	2
0100	NULL	2
0101	ADD	2
0110	NULL	1
0111	SUBTRACT	3
1000	NULL	3
1001	ADD	3
1010	NULL	1
1011	SUBTRACT	2
1100	NULL	2
1101	ADD	2
1110	NULL	1
1111	SUBTRACT	4

The carry bit from a subtraction must again (see Section 2.1) be considered when determining the transition probabilities. The possible bit combinations in the successor to a cycle with a one, two or three bit shift are:

0001 1001
0011 1011
0101 1101
0111 1111

All valid bit combinations are equally probable.
The transition probability matrix is:

$$P = \begin{bmatrix} 0 & 0{\cdot}5 & 0{\cdot}25 & 0{\cdot}25 \\ 0 & 0{\cdot}5 & 0{\cdot}25 & 0{\cdot}25 \\ 0 & 0{\cdot}5 & 0{\cdot}25 & 0{\cdot}25 \\ 0{\cdot}25 & 0{\cdot}375 & 0{\cdot}1875 & 0{\cdot}1875 \end{bmatrix} \ . \qquad (14)$$

The initial state probability vector is:

$$\pi(0) = [0{\cdot}25 \quad 0{\cdot}375 \quad 0{\cdot}1875 \quad 0{\cdot}1875] \ . \qquad (15)$$

The standard analysis shows the mean shift to be:

$$2{\cdot}647 - 0{\cdot}334\,(-0{\cdot}0625)^k \ . \qquad (16)$$

For large k, the mean shift is 2·647 bits per cycle.

3.2. ($\{+1, +2, +4, +8\} \times$ *multiplicand*) *available*
This is the third case that Freeman analysed. His results show the mean shift to be 1·9087 bits per cycle.

There are fifteen further combinations of paths into the adder that produce this mean shift. All sixteen have the same *magnitudes* of the multiples of the multiplicand (1, 2, 4, 8), but differ in the *signs*, which are the set of all possible combinations.

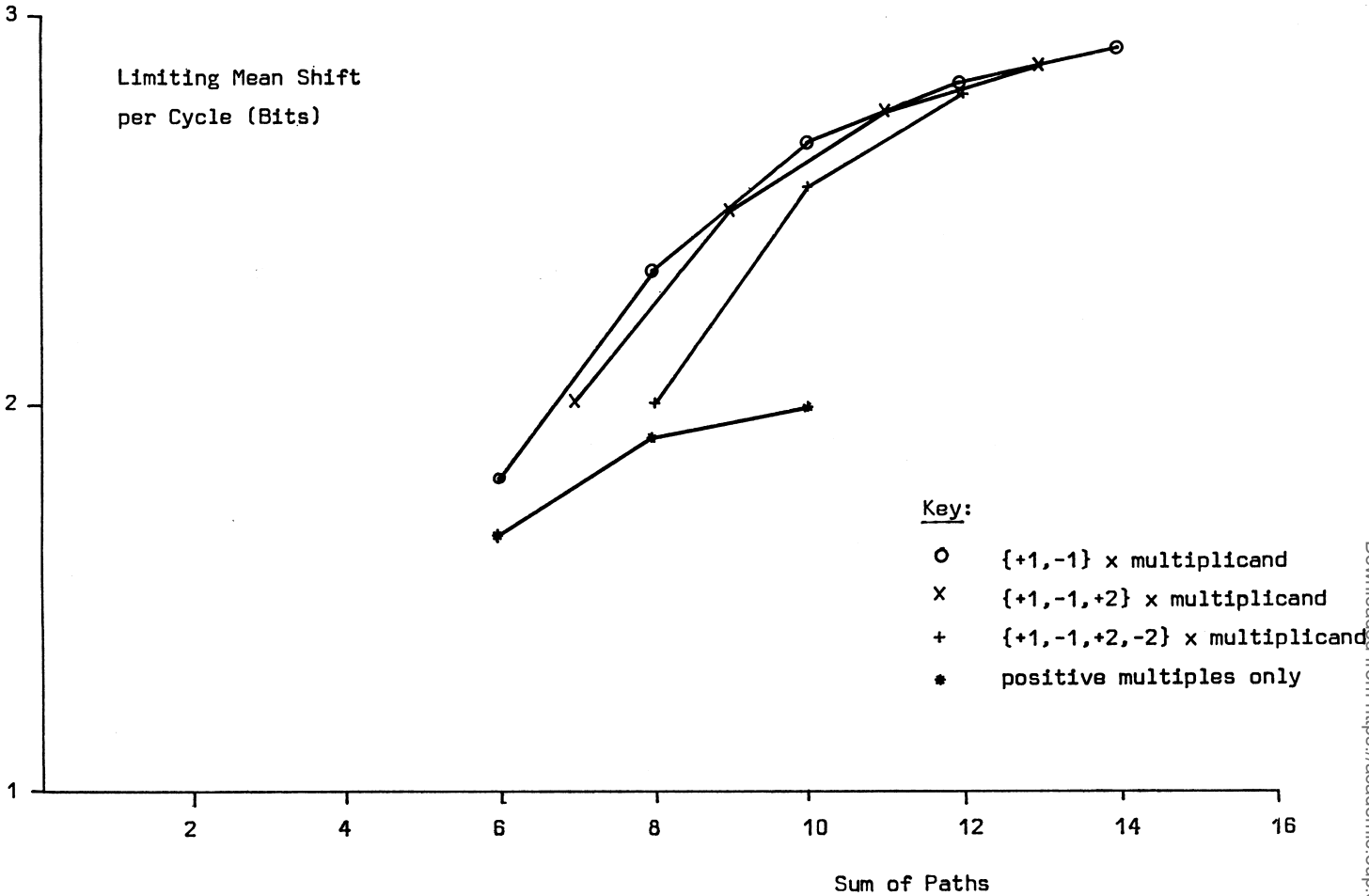

Fig. 2 Multiplier performance

Conclusions

Freeman (1967) has conjectured that the approximate analysis of the speed of variable shift multiplication made by Flores (1963) produces optimistic results when compared with the exact results. This is so in the cases that he cites. For example, when two bits are examined and ($\{+1, -1\}$ × multiplicand) are available, the exact analysis yields 1·667 bits per cycle as the mean shift in comparison to Flores' result of 1·75 bits per cycle. However, our analysis has shown that the conjecture is not necessarily true for all cases.

The optimum combination of multiplicand multiples (m) and shift paths (n) for a given hardware cost ($2n + m$) is clear from Table 1. Both the ($+1$ × multiplicand) and (-1 × multiplicand) paths must be available to the adder; when the sum of the paths is odd, the ($+2$ × multiplicand) path must also be present. All the other paths must be shift paths. In **Fig. 2,** the optimum performance curve is plotted together with some inferior cases.

The values in Table 1 suggest that the mean shift is tending towards a limit of three bits per cycle. This is an interesting result that is not intuitively obvious. It would then represent the absolute maximum performance gain when only multiples of the multiplicand that are integral powers of two are available. For practical purposes there is very little additional gain in speed to be obtained by increasing the number of multiplier bits examined in each cycle beyond four bits.

Although not shown explicitly in the analysis it may be verified that (for the cases analysed) the least probable shift is one bit, and the most probable shift is two bits (for odd values of shift path, a three bit shift is equally most probable).

We have noted that iterative multiplication techniques are more popular in medium and small central processors. In these machines the cost of logic to precalculate and hold odd multiples of the multiplicand (e.g. 3 × multiplicand or 5 × multiplicand) is likely to be prohibitive whereas simple adder and shift paths are more cost-effective.

The same hardware is suitable for the binary division process except for the additional sequence control gates. Variable numbers of shift per cycle can again be used to advantage. The analysis of speed for the division is more complex than that for multiplication and will be presented for publication at a later date.

If the multiplication and division hardware is not a separate module from that performing the other arithmetic instructions then the shift instructions stand to gain from the concentration of paths in the shifter rather than into the adder. If the multiplication and division hardware is a separate module, a carry save scheme can be optionally used instead of a carry propagate scheme with the above general conclusions still valid.

Acknowledgement

One of us (MRP) wishes to acknowledge the permission granted by International Computers Limited to publish this article.

References

FREEMAN, H. (1967). Calculation of Mean Shift for a Binary Multiplier using 2, 3 or 4 bits at a Time, *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 6, December 1967.

FLORES, I. (1963). *The Logic of Computer Arithmetic*, Prentice-Hall, Englewood Cliffs, NJ

Control Data 6400/6500/6600 Computer Systems Reference Manual. Control Data Corporation, St Paul, Minnesota, 1969.

HOWARD, R. A. (1971). *Dynamic Probabilistic Systems*, Vol. 1: Markov Models, John Wiley & Sons, NY.