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

M. R. Patel and K. H. Bennett

In a recent paper (1976), the analysis of a binary multiplier using a variable number of shifts per multiplication cycle was described. The method was based on a discrete time finite state system model. In this paper we continue the analysis for the more complex case of binary division. The results enable the optimum hardware configuration to be determined for a given cost.

(Received January 1977)

1. Introduction

In Patel and Bennett (1976) the analysis of a multiplier using a variable number of shifts per multiplication cycle was described. Here we continue the analysis for a variable number of shifts per cycle divider. In an analogous technique, two or more bits of the intermediate dividend are inspected simultaneously to speed up the division beyond one quotient bit per cycle. The method of non restoring division is used and after each cycle the remainder is shifted in an attempt to normalise it, thereby forming the intermediate dividend for the next cycle. The maximum shift possible is determined by the shift paths available in the hardware and the actual shift value used in a cycle is the same as the number of quotient bits generated in that cycle.

The variable shift facility on its own results in a significant improvement in divider speed. The speed may be increased further by providing multiples of the divisor which are negative integral powers of two, e.g. $\pm 0{\cdot}5 \times$ divisor, $\pm 0{\cdot}25 \times$ divisor, etc. The analysis enables us to choose the best hardware configuration, in terms of divisor multiples and shift paths, for a given cost.

In Section 2 the division algorithm is described in detail and Section 3 contains a description of the divider structure and operation. Sections 4 and 5 present the analysis. The approach used in the analysis of the multiplier cannot be extended directly to the divider and a modified technique is required. In Section 6 the results of the analysis are summarised.

The divider design for the ICL 2980 computer is based on the principles described in this paper.

2. The binary division method

Non restoring binary division has been described in a number of publications (e.g. Phister, 1958; Richards, 1955; Walker, 1967). In **Fig. 1** we demonstrate such a division, using two's complement representation for negative numbers. The dividend R is assumed to be less in magnitude than the divisor D at the start of the division (to ensure a fractional quotient) and both have been scaled into fractions. If D is positive and R_j negative then we are omitting the actual restoration by making use of the fact that (since $R_{j-1} - D = R_j$):

$$R_j + D - \frac{D}{2} = R_j + \frac{D}{2}$$

where R_j is the intermediate dividend (henceforth called the dividend) at the start of the jth cycle. Hence we *add* the divisor when R_j is negative. The quotient is initially zero and in each cycle the next bit of the quotient is added in at the rightmost end. The actions for the various combinations of sign of R_j and D are summarised in **Fig. 2.**

Following Freiman (1961), we will say that a binary number b, scaled into fractional form, is in *normal* range if $0{\cdot}5 \leqslant b < 1{\cdot}0$. For positive numbers the first two bits will be 0·1 and for negative numbers 1·0.

Rather than shift the divisor to the right on each iteration, it is more economical in hardware cost (in the divider to be discussed) to shift the dividend to the left by a like amount, at the same time the quotient is shifted also by the same value so that the new quotient bit is always inserted in the same register position.

So far the division process has yielded only one quotient bit per cycle. We now describe three extensions to the basic method with the objective of improving the speed.

Let S^*_j denote the unnormalised result of the $R_j \pm D$ operation (before any shifting has taken place). In the method above a one bit shift only is performed on S^*_j to form R_{j+1}, so that $R_{j+1} = 2S^*_j$ which may not be normalised. As the first extension to the basic division technique we introduce additional shift paths so that the amount of shifting from S^*_j to R_{j+1} may be 1, 2 . . . n bits ($n \geqslant 1$) where n is the maximum value of shift allowed. We thus attempt to form a normalised dividend for the start of each new iteration and the increased shifting capability means that up to n quotient bits can be generated in one cycle.

The technique of normalising remainders to increase the division speed beyond one quotient bit per cycle has been described by Tocher (1958) and is illustrated in **Fig. 3.**

Despite the extra finite shifting capability, it is still possible that the dividend R_{j+1} cannot be normalised. In such cases, subsequent addition or subtraction leads to a result which is normalised, or nearly so. In the second extension to the division process we permit two alternative courses of action in a division iteration:

(*a*) ADD . SHIFT mode. If the dividend is normalised, we form $S^*_j = R_j \pm D$ and then shift to form R_{j+1} (as above).

(*b*) $\overline{\text{ADD}}$. SHIFT mode. If the dividend is unnormalised, we do not perform the addition, but simply shift $S^*_j = R_j$ to form R_{j+1}.

The second mode results in a larger shift than would otherwise have occurred. In **Fig. 4** an example using the technique is illustrated. Four cycles are required whereas five cycles were necessary with only the variable shift facility and six cycles would have been required for the basic method.

A further improvement in performance is achieved by introducing multiples of the divisor which are negative integral powers of two. The correct multiple is added/subtracted (according to the rules in Fig. 2) to the dividend when it is unnormalised. Let a shift of y bits be required to normalise the dividend. Then the 2^{-y} multiple of the divisor is used in the current cycle. If that multiple is not available we enter the $\overline{\text{ADD}}$. SHIFT mode and proceed directly with the shifting.

The authors are respectively with International Computers Limited and the Computer Science Department, University of Keele, Staffs, ST5 5BG.

R_0	1·01110001	$Q_0 = 0$
$+D$	0·10110000	
R_1	0·00100001	$Q_1 = 1$
$-D/2$	0·01011000	
R_2	1·11001001	$Q_2 = 1{\cdot}1$
$+D/4$	0·00101100	
R_3	1·11110101	$Q_3 = 1{\cdot}01$
$+D/8$	0·00010110	
R_4	0·00001011	$Q_4 = 1{\cdot}001$
$-D/16$	0·00001011	
R_5	0	$Q_5 = 1{\cdot}0011$

Fig. 1 Example of non restoring division

Divisor sign	*Dividend sign*	*Operation*	*Quotient*
+	+	SUB	+1
+	−	ADD	−1
−	+	ADD	−1
−	−	SUB	+1

Fig. 2 Non restoring division: summary of action for two's complement representation

R_0	0·1000001111		$Q_0 = 0$
$-D$	0·1111100000		
	1·1000101111	SUB . SHIFT 1	
R_1	1·0001011110		$Q_1 = 1{\cdot}0$
$+D$	0·1111100000		
	0·0000111110	ADD . SHIFT 2	
R_2	0·0011111000		$Q_2 = 0{\cdot}100$
$-D$	0·1111100000		
R_3	1·0100011000	SUB . SHIFT 0	$Q_3 = 0{\cdot}101$
$+D$	0·1111100000		
	0·0011111000	ADD . SHIFT 2	
R_4	0·1111100000		$Q_4 = 0{\cdot}10000$
$-D$	0·1111100000		
R_5	0		$Q_5 = 0{\cdot}10001$

Fig. 3 Example of non restoring division with variable sized shifting only and with maximum permitted shift of two bits

R_0	0·1000001111		$Q_0 = 0$
$-D$	0·1111100000		
	1·1000101111	SUB . SHIFT 1	
R_1	1·0001011110		$Q_1 = 1{\cdot}0$
$+D$	0·1111100000		
	0·0000111110	ADD . SHIFT 2	
R_2	0·0011111000	$\overline{\text{ADD}}$. SHIFT 2	$Q_2 = 0{\cdot}100$
R_3	0·1111100000		$Q_3 = 0{\cdot}10000$
$-D$	0·1111100000		
R_4	0		$Q_4 = 0{\cdot}10001$

Fig. 4 Example of non restoring division with variable sized shifting and $\overline{\text{ADD}}$.SHIFT mode and with maximum permitted shift of two bits

This is the third modification to the basic division process.

Fig. 5 shows an example of the technique, using the same divisor and dividend as Figs. 3 and 4. Only three cycles are now required. Note the extra complexity to generate the quotient.

To summarise, a division cycle consists of one of two actions—either an addition followed by a shift, or a shift only. If the ADD . SHIFT mode is entered, the correct multiple of the divisor is chosen by determining the rightmost significant bit of the dividend. These decisions are made by inspecting the $n + 1$ most significant bits (including the sign) of the dividend, where n is the maximum shift value permitted as before. The $n + 1$ bits are collectively called the dividend prefix. Note that, if a positive dividend prefix is represented by x, then the action is exactly the same as for the negative prefix ($-x - 2^{-n}$) using two's complement representation.

3. The binary divider structure

Fig. 6 is a block diagram of the divider to be discussed in this paper. The normalised divisor and the dividend (also normalised as long as $|R_0| < |D|$) are placed in single length registers as shown. The quotient is generated in the right hand half of the double length register T. Gated paths exist between the divisor and adder to generate the divisor submultiples at the adder input; a similar technique is used to perform the shifting between the register T and the buffer register. The behaviour in a division cycle is now summarised:

(*a*) the dividend prefix is inspected to decide whether to enter the $\overline{\text{ADD}}$. SHIFT mode or the ADD . SHIFT mode; if it is the latter, the appropriate multiple of the divisor is gated into the adder

(*b*) if we are in the $\overline{\text{ADD}}$. SHIFT mode, the dividend alone is passed through the adder and placed in the top half of T. Otherwise, the adder output $R_j \pm D/2^y$ is put in the top half

(*c*) the quotient bit(s) are inserted at the bottom of T

(*d*) the contents of T are moved into the buffer with the appropriate amount of shifting by opening the correct shift path gates

(*e*) finally, the top half of the buffer is moved to the dividend register, whilst the bottom half overwrites the bottom half of the T register.

The increase in performance is achieved at the cost of extra hardware to produce the divisor multiples and to shift the remainder and quotient. The register T is twice the length of the divisor register, so that a measure of the hardware cost is given by $2n + m$ where m is the number of divisor multiples available and n is the number of shift paths. Positive and negative versions of the same divisor multiple are regarded as separate for the time being.

As with the multiplier the analysis is designed to show the best combination of m and n for a given cost. The expression for the cost ignores the extra control logic that is required to organise the extended divider; this cost is relatively small as it is required once only and not once per bit position.

4. The analysis

In the analysis of the variable shift multiplier, each shift value was regarded as a state in a finite state system. The probability of a transition from shift state i to shift state j was determined for all i and j. By regarding the system as a Markov process, the limiting state probabilities were found.

A necessary characteristic of a Markovian process is that the transition probability from i to state j is independent of the state occupied before reaching state i. In the multiplier the result of the operation (multiplicand ± multiplier) is inde-

R_0	0·1000001111		$Q_0 = 0$
$-D$	0·1111100000		
	1·1000101111	SUB . SHIFT 1	
R_1	1·0001011110		$Q_1 = 1{\cdot}0$
$+D$	0·1111100000		
	0·0000111110	ADD . SHIFT 2	
R_2	0·0011111000		$Q_2 = 0{\cdot}100$
$-D/4$	0·0011111000	use 0·25 × D	
R_3	0		$Q_3 = 0{\cdot}10001$

Fig. 5 Example of non restoring division with ±0·5 × divisor and ±0·25 × divisor available and with maximum shift capability of two bits

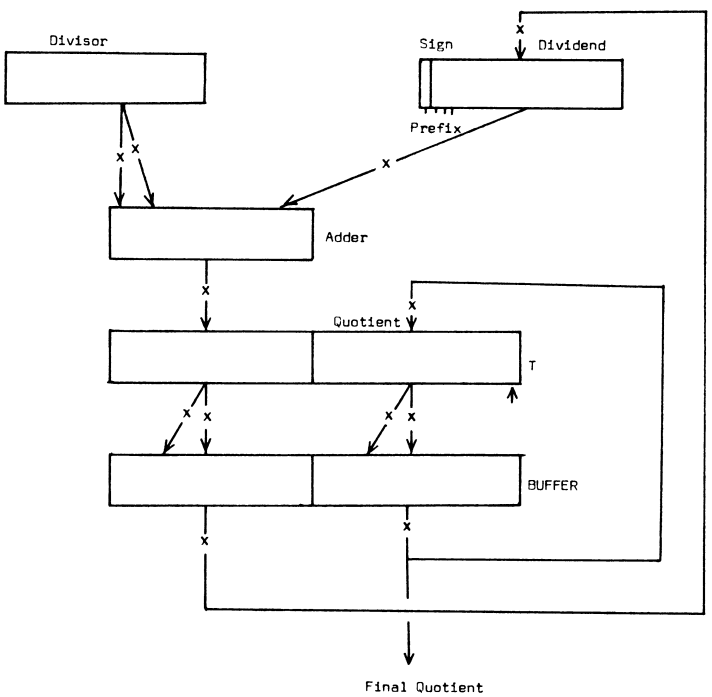

Fig. 6 Block diagram of the divider

Dividend prefix	*Action in current cycle*
000	$\overline{\text{ADD}}$. SHIFT 2
001	ADD . SHIFT, $-0{\cdot}5D$
010	ADD . SHIFT, $-D$
011	ADD . SHIFT, $-D$
111	$\overline{\text{ADD}}$. SHIFT 2
110	ADD . SHIFT, $+0{\cdot}5D$
101	ADD . SHIFT, $+D$
100	ADD . SHIFT, $+D$

Fig. 7 Action undertaken in current division cycle according to dividend prefix

		Prefix after shift							
		000	001	010	011	100	101	110	111
	000	¼	¼	¼	¼	0	0	0	0
Prefix	001	0	0	½	½	0	0	0	0
before	110	0	0	0	0	½	½	0	0
shift	111	0	0	0	0	¼	¼	¼	¼

Fig. 8 Transition probability matrix A

pendent of the next bits of the multiplier to be inspected and the Markovian condition is met.

In the divider, the result of (dividend ± divisor followed by a shift) determines the dividend prefix in the next iteration; this in turn determines the mode of action and in particular the amount of shifting, in that iteration. The probability of a transition from shift state i to shift state j is dependent upon the states occupied before state i and the Markovian condition fails.

Instead we regard each possible *value* of the dividend prefix as a state (there are thus 2^{n+1} states in the system, taking account of signs). For a given dividend prefix and divisor the action in a division cycle is fixed and determinable unambiguously, so that the possible dividend prefixes in the next cycle and their probabilities are also known unambiguously. The Markovian requirement is fulfilled.

The determination of the transition probabilities is straightforward. If the $\overline{\text{ADD}}$. SHIFT mode is entered, we know the probabilities directly; for example, if the prefix is 0·001 then a two bit shift is performed leading to one of the prefixes in the next iteration of 0·100, 0·101, 0·110, 0·111, each with probability of 0·25. If the ADD . SHIFT mode is entered, we calculate the probabilities in the same way, but using the prefix of $(R_j \pm D \cdot 2^{-y})$ instead of R_j. The value of the divisor must thus be considered and it is interesting to note that different divisor values lead to different speeds (Section 6) for some configurations.

We now describe the analysis in more detail; in parallel we work through a simple example. The configuration chosen for illustration is:

Maximum shift = 2 bits

Divisor multiples = $\pm D, \pm 0{\cdot}5D$.

1. The action in the current cycle is determined for each dividend prefix (**Fig. 7**) as described in Section 2.
2. Let a prefix before normalisation have the value s. We shift the prefix y bits to the left to normalise it, thereby producing one of 2^y possible new prefixes, each with probability $1/2^y$. We now define a matrix A in which an element a_{st} represents the probability of an unnormalised prefix s producing a normalised prefix t after shifting. All zero rows are omitted for clarity in **Fig. 8.**
3. Let i, j respectively be the values of the dividend prefix in successive division cycles. Then an element $q_{ij}^{(d)}$ of the main transition probability matrix $Q^{(d)}$ represents the probability of the dividend prefix being j in the $(k+1)$th cycle, given that it was i in the kth cycle. The transition is achieved in the divider *either* through the ADD . SHIFT mode *or* the $\overline{\text{ADD}}$. SHIFT mode, according to the dividend prefix. There is a set of transition probability matrices over the range of divisor prefixes d.

   The transition probability in $\overline{\text{ADD}}$. SHIFT mode is obtained directly from element a_{ij} of matrix A.
4. We now consider the result of the addition. Only the $n+1$ most significant bits of the divisor (the divisor prefix) are considered and the probability of a carry from the previous bit position is assumed to be 0·5. Let $z_i^{(d)}$ be the result of the addition with *no* carry and $z_i^{(d)'}$ the result with a carry, in both cases the dividend prefix being i and the divisor prefix being d. Then:

   prob [transition $i \rightarrow j$, ADD . SHIFT mode] =

   $\frac{1}{2}$ (prob [transition $z_i^{(d)} \rightarrow j$] +

   + prob [transition $z_i^{(d)'} \rightarrow j$]) (1)

   Again these two probabilities are obtained directly from matrix A, once $z_i^{(d)}$ and $z_i^{(d)'}$ are known.

   A table of $z_i^{(d)}$ and $z_i^{(d)'}$ for the permissible values of dividend and divisor prefixes is readily constructed (**Fig. 9**).
5. Since $z_i^{(d)}$ and $z_i^{(d)'}$ depend on the divisor, we must evaluate

divisor prefix	dividend prefix 011	010	101	100
011	111 / 000	000 / 001	001 / 000	000 / 111
010	110 / 111	111 / 000	000 / 111	111 / 110
use 0·5D: 001	111 / 000	111 / 000	000 / 111	000 / 111
use 0·5D: 110	000 / 111	000 / 111	111 / 000	111 / 000
101	001 / 000	000 / 111	111 / 000	000 / 001
100	000 / 111	111 / 110	110 / 111	111 / 000

$z_i^{(d)'}$ / $z_i^{(d)}$

Fig. 9 Table of $z_i^{(d)}$ and $z_i^{(d)'}$

dividend prefix on $(k+1)$th cycle	100	101	110	111	000	001	010	011
100	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	0	0	0	0
dividend 101	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
prefix on 110	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
kth 111	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0
cycle 000	0	0	0	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
001	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{4}$
010	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
011	0	0	0	0	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{3}{8}$

Fig. 10 Probability transition matrix $Q^{(d)}$ for divisor = 010

Dividend prefix	*Limiting state probability*
100	0·1429
101	0·1429
110	0·1071
111	0·1071
000	0·1071
001	0·1071
010	0·1429
011	0·1429

Fig. 11 Limiting state probability vector $\pi^{(d)}$ for divisor prefix = 010

Divisor prefix d	*Mean speed $\bar{s}^{(d)}$ in bits per cycle*
010, 101	1·8571
011, 100	1·8571

Fig. 12 Table of mean speed and divisor prefix

the mean shift for each divisor prefix. In **Fig. 10** we show one transition matrix, for the divisor prefix 010. The set of limiting state probability vectors $\pi^{(d)}$ can now be determined (**Fig. 11**) by straightforward Markov analysis (see for example Howard, 1971).

6. The *mean* shift $\bar{s}^{(d)}$ for divisor prefix d is given by

$$\bar{s}^{(d)} = \sum_i \pi_i \times (\text{shift value for dividend } i) \qquad (2)$$

$$= \sum_i \pi_i \times \textbf{if } \overline{\text{ADD}}.\text{ SHIFT mode entered for dividend prefix } i \textbf{ then} \text{ shift value for } i \textbf{ else } \tfrac{1}{2}(\text{shift value for } z_i^{(d)} + \text{shift value for } z_i^{(d)'}) \textbf{ fi} \qquad (3)$$

In (2) and (3), i ranges over the values of the dividend prefix, the divisor prefix being d. A table of $\bar{s}^{(d)}$ and d is shown in **Fig. 12.**

In this configuration, the mean shift does not vary with the divisor prefix, but in a later example, such behaviour is illustrated.

7. The overall mean shift $\bar{S}$ is calculated as follows. Let the length of the operand be w bits. As before, $\bar{s}^{(d)}$ represents the mean shift speed for divisor d. Hence the mean *time* for a division with divisor d is given by $w/\bar{s}^{(d)}$ (in units of cycle-times). We assume that the probability distribution for the 2^n values of d is uniform. Hence, given that the divisor is

Configuration: Maximum shift = 3 bits
Divisor multiples = ± 1 $\pm\frac{1}{2}$
Divisor considered = 0·110

Dividend prefix (positive values only show)	*Ordinary decode* $z_i^{(d)}$	$z_i^{(d)'}$	*shift*	*Improved decode* $z_i^{(d)}$	$z_i^{(d)'}$	*shift*
0010	−2	−1	2·5	−2	−1	2·5
0011	−1	0	3	−1	0	3
0100	−3	−2	1·5	0	1	2·5*
0101	−2	−1	2·5	−2	−1	2·5
0110	−1	0	3	−1	0	3
0111	0	1	2·5	0	1	2·5

*0·5D used rather than D.

Fig. 13 Example of improved decode

normalised:

$$\text{Overall mean division time} = \frac{1}{2^n}\sum_d \frac{w}{\bar{S}^{(d)}} \text{ cycle times} \quad (4)$$

This may be expressed as a speed:

$$\frac{1}{\bar{S}} = \frac{1}{2^n}\sum_d \frac{1}{\bar{S}^{(d)}} \text{ bits/cycle} \quad (5)$$

For our example, $\bar{S} = 1{\cdot}8571$ quotient bits/cycle.

The size of the matrices in the analysis soon becomes very large (of the order of $2^{2(n+1)}$, and hence an ALGOL 60 program was written to calculate the division speeds; data for the program consisted simply of the details of the divider configuration. Maximum advantage was taken of any symmetries in the matrices to reduce the data space required by the program. Even so, a maximum shift value of six was the largest that could be analysed in the small core memory (32K words) of a CDC 7600.

5. Further performance improvement

In Section 2 it was stated that the appropriate multiple of the divisor is chosen by determining the position of the leftmost significant bit in the dividend. By slightly modifying this condition, a significant performance improvement is gained. The multiple of the divisor is chosen such that $R_j \pm D \cdot 2^{-y}$ is equal or nearest to 0 (positive result) or -1 (negative result). This ensures that the maximum shift possible is attained.

In practice the new condition means that for a given pair of dividend and divisor prefixes, one of three situations now arises:

1. The next smaller multiple (in magnitude) of the divisor is needed. For example, if D would have been used previously, it is in fact better to use $0{\cdot}5D$. Similarly $0{\cdot}25D$ may be used instead of $0{\cdot}5D$ in other cases.
2. The next larger multiple of the divisor is needed, e.g. $0{\cdot}5D$ instead of $0{\cdot}25D$.
3. The same multiple of the divisor is needed.

An example of case 1 (and case 3) is given in **Fig. 13.**

The improvement relies almost completely on the use of the next smaller multiple (case 1) when the dividend is normalised. Results of the analysis showed an improvement at best of about 0·3% in mean speed when the next larger multiple (case 2) was used. The reason is that the next larger multiple is used only in conjunction with unnormalised dividends and these have a relatively low probability of occurrence (**Fig. 14**). Hence in the derivation of the results presented in the next section, only the use of the next smaller multiple is considered. Let us name this technique the 'improved decode' and the original technique the 'ordinary decode'.

6. Results of analysis

The availability of both positive and negative versions of the same divisor multiple is essential in the divider we have described. If as is likely the adder unit is used in the execution of other functions, e.g. multiplication, it is cheaper to provide a complementer as an integral part of the adder and simply have one path into the adder for each divisor multiple. For this type of configuration the measure of hardware cost is estimated as $2n + m/2$ (cf. Section 3).

The mean shift $\bar{S}$ as determined by our analysis is listed in **Table 1** in the appendix, for various divider configurations. In **Fig. 15** some selected results have been plotted against cost.

Freiman (1961) has described the analysis of a variable shift divider with multiples of the divisor available. In his scheme, the amount of shift in an iteration was unlimited so his results provide a useful but approximate comparison with the results described here.

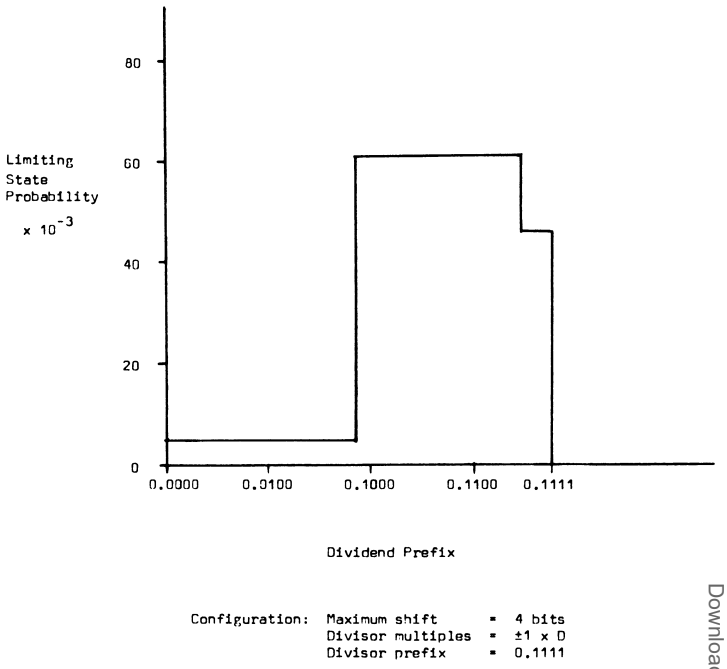

Fig. 14 Limiting State Probability Distribution with Dividend Prefix

Table 1 Results of analysis

Configuration		*Mean Shift* $\bar{S}$ *bits per iteration*	
Max shift n (bits)	*Divisor multiples*	*ordinary decode*	*improved decode*
2	± 1	1·6667	1·6667
2	$\pm 1\ \pm\frac{1}{2}$	1·8574	1·8574
3	± 1	2·0768	2·0768
3	$\pm 1\ \pm\frac{1}{2}$	2·2650	2·4173
3	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}$	2·3504	2·5053
4	± 1	2·3042	2·3042
4	$\pm 1\ \pm\frac{1}{2}$	2·4220	2·6766
4	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}$	2·4847	2·7579
4	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}\ \pm\frac{1}{8}$	2·5121	2·7880
5	± 1	2·4307	2·4307
5	$\pm 1\ \pm\frac{1}{2}$	2·4954	2·8027
5	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}$	2·5297	2·8493
5	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}\ \pm\frac{1}{8}$	2·5471	2·8725
5	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}\ \pm\frac{1}{8}\ \pm\frac{1}{16}$	2·5544	2·8809
6	± 1	2·4972	2·4972
6	$\pm 1\ \pm\frac{1}{2}$	2·5312	2·8552
6	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}$	2·5487	2·8790
6	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}\ \pm\frac{1}{8}$	2·5577	2·8912
6	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}\ \pm\frac{1}{8}\ \pm\frac{1}{16}$	2·5622	2·8971
6	$\pm 1\ \pm\frac{1}{2}\ \pm\frac{1}{4}\ \pm\frac{1}{8}\ \pm\frac{1}{16}\ \pm\frac{1}{32}$	2·5640	2·8992

Using the ordinary decode, as progressively more shift paths are provided then the probability of an unnormalised dividend becomes increasingly smaller; in the limit there will always be a shift path to enable S^*_j to be normalised. The multiples of the divisor $\pm 0{\cdot}5D$, $\pm 0{\cdot}25D$ etc. are only used in conjunction with *un*normalised dividends, so we expect the results to converge to the $\pm 1D$ case, as confirmed by Fig. 15. Freiman's limiting result for $\pm 1D$ is 2·667 quotient bits per cycle.

A similar argument holds for the improved decode technique. In this case the $0{\cdot}5D$ multiples (and these two multiples only) are used in conjunction with *normalised* dividends, so in the limit they still have an effect on performance. Freiman's

limiting result for the $\pm 1D$, $\pm 0{\cdot}5D$ case is 2·875 quotient bits per iteration.

For the $\pm 1D$ case, Freiman's limit seems too high in comparison with our results (Fig. 14) whilst for the $\pm 1D$, $\pm 0{\cdot}5D$ case it is almost certainly too low. The reason is that for these cases Freiman has assumed a *uniform* probability distribution in the range [0·5, 1·0] for the dividend, whereas the results from our analysis (Fig. 14, normalised dividends) shows a clear non uniform distribution for the dividend, once the division process has started.

The non uniformity of the distribution was recognised by Freiman who in the same paper developed the analysis to take account of this behaviour; however, the analysis assumed an unlimited shift capability and the choice of different configurations means that Freiman's numerical results are not directly comparable with those described here.

In the $\pm 1D$ case, the probability distribution of the dividend favours the smaller shifts whilst in the $\pm 1D$ $\pm 0{\cdot}5D$ case it favours the larger shifts. This can be seen more clearly from the joint probability density diagrams in Freiman's paper.

Table 2 shows an example of how the divider speed varies considerably with the divisor value, for $n > 2$.

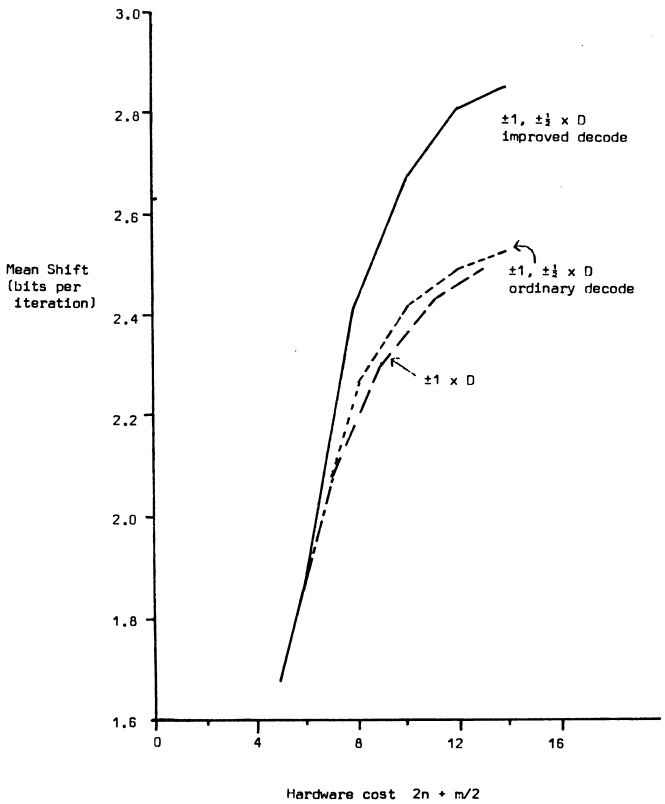

Fig. 15 Selected cost performance results

7. Conclusions

A simulator has been written to obtain performance data for a divider with the following configuration:

Table 2 Variations in performance with divisor prefix

Configuration: Maximum shift = 4 bits
Divisor multiples = ± 1, $\pm \frac{1}{2}$, $\pm \frac{1}{4}$

Divisor prefix	*Mean speed (bits/iteration)*	*Divisor prefix*	*Mean speed (bits/iteration)*
01000	2·3134	01100	2·7201
01001	2·6947	01101	2·3985
01010	2·8695	01110	2·2531
01011	2·8106	01111	2·0677

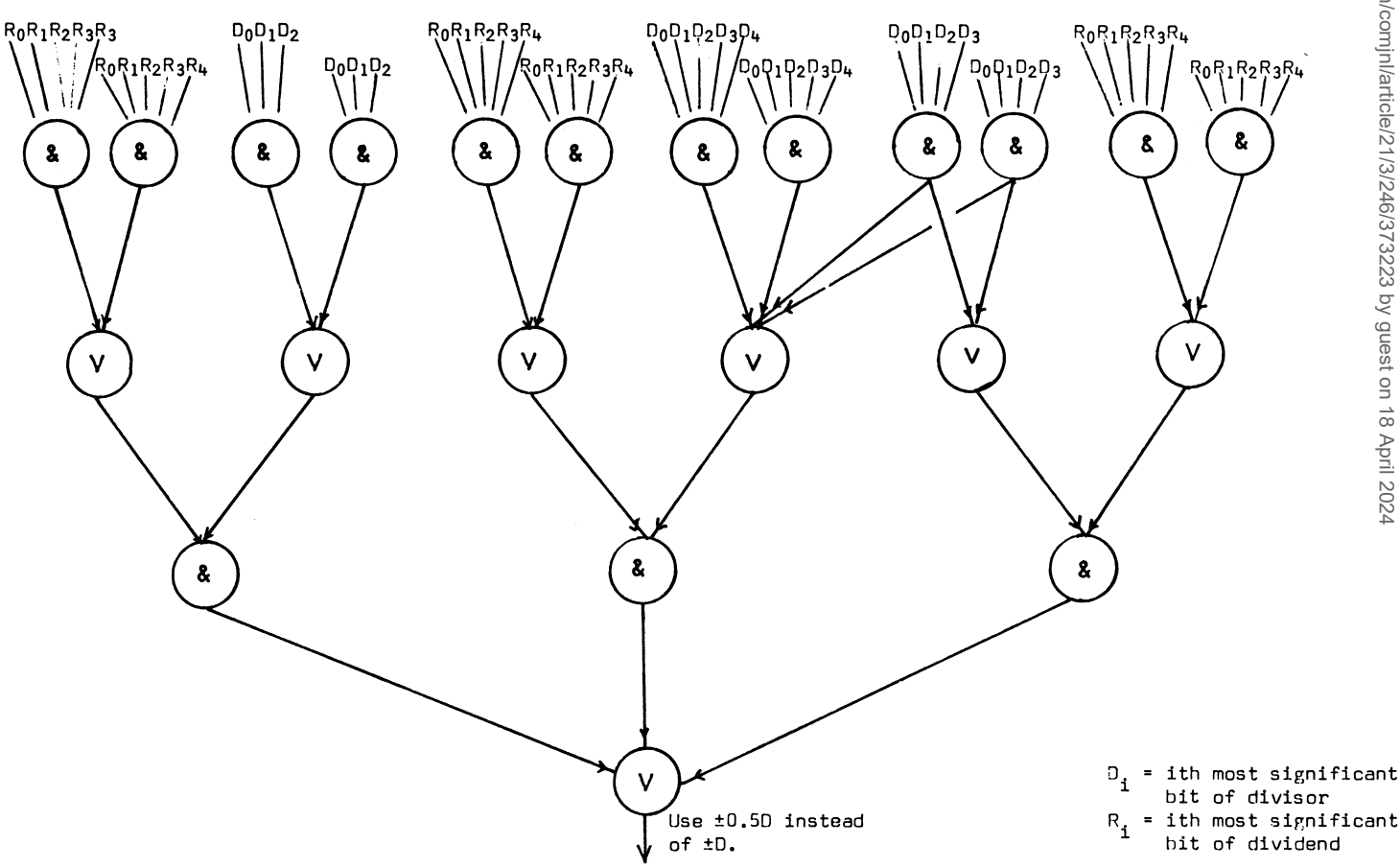

Fig. 16 Logic required to determine the use of $\pm 0{\cdot}5 \times$ divisor instead of $\pm 1 \times$ divisor for the configuration $n = 4$, with $\pm D$, $\pm 0{\cdot}5D$ available

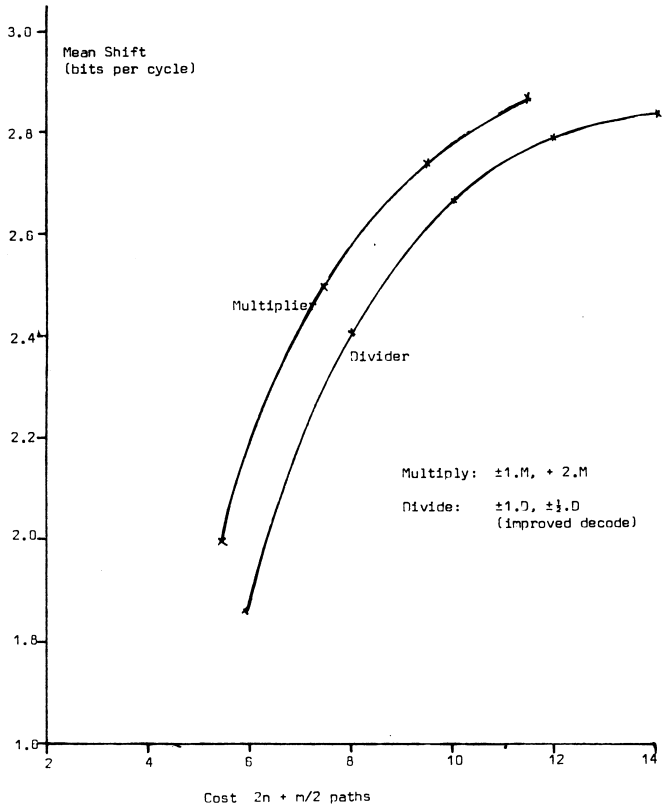

Fig. 17 Comparison of variable shift division and multiplication

Maximum permitted shift = 4 bits

Divisor multiples available = $\pm 1 \times$ divisor

The simulator was exercised over a fairly large sample of divisions and the mean division speed by this method was 2·3 bits/cycle. This is close to the predicted speed (Table 1) of 2·304 bits/cycle. The divider on the ICL 2980 has the above configuration and its design is based on the principles described in this paper.

The optimum configuration for a given hardware cost can be seen from Fig. 15. The improved decode technique is well worthwhile, yet it is cheap to implement. **Fig. 16** shows the logic required to determine when to use the next smaller multiple, for one configuration.

There is little to choose between providing more shift paths or more divisor multiples with the improved decode and in practice the choice would probably be based on other criteria. Other CPU functions stand to gain by providing extra shift paths (e.g. 'scale' and 'rotate' instructions) so it is more effective to implement shift paths than extra divisor multiples.

It is slightly more cost effective with the ordinary decode to provide extra divisor multiples, but the gains are outweighed by the improved decode.

In **Fig. 17**, the mean shift of the divider (with improved decode) is compared with the mean shift of the multiplier described in Patel and Bennett (1976), using a cost index of $2n + m/2$ paths (i.e. under the assumption that a complementer is available in the adder). A satisfactory division to multiplication performance ratio is obtained.

Freiman in his paper discussed the performance of a divider with divisor multiples which are not negative integral powers of two. These cannot of course be generated directly. In this paper our interest has centred on computers where the expense of pre-calculating and storing divisor multiples cannot be justified. In large machines and when heavy use of division is made, algorithms such as those described by Ahmad (1972) would be used.

It is clear that end effects are an important practical consideration of the actual performance of the divider described here. Typically, of the order of three beats are required; the dividend and divisor must be moved in and out of the divider, initial normalisation may be required and condition code and other registers may need clearing or setting. The reduction of the time taken by end effects is often only achieved at very great expense.

In the analysis, shift values approaching the maximum may be inapplicable towards the end of a division. We have also assumed that the dividend and divisor have values which are randomly distributed to start with. This is usually a reasonable assumption in **real** arithmetic (but see Hamming, 1970); however, in many programs **integer** constants are mostly small and positive. Their relative occurrence in division operations is not known. In addition, the division process can proceed at maximum speed as soon as a dividend of zero is obtained; again we do not know how often this occurs.

It was one of the objectives of the divider design to utilise existing hardware as much as possible to minimise the cost of the implementation of an arithmetic function whose use is largely confined to certain computing application areas. It is worth noting that as the trend towards large scale integration continues, it may well turn out in the future to be cheaper to provide a separate hardware divider for those installations requiring this facility.

Acknowledgement

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

References

AHMAD, M. (1972). Iterative schemes for high speed division, *The Computer Journal*, Vol. 15, No. 4.

FREIMAN, C. V. (1961). Statistical analysis of certain binary division algorithms, *Proc. I.R.E.*, Vol. 49, No. 1.

HAMMING, R. W. (1970). On the distribution of numbers, *Bell Systems Tech. Journal*, Vol. 49, No. 8, pp. 1609-1625.

HOWARD, R. A. (1971). *Dynamic probabilistic systems*, Vol. 1 Markov Models, Wiley, N.Y.

PATEL, M. R. and BENNETT, K. H. (1976). Analysis of speed of a binary multiplier using a variable number of shifts per cycle, *The Computer Journal*, Vol. 19, No. 3, pp. 254-257.

PHISTER, M. (1958). *Logical design of digital computers*, Wiley, New York.

RICHARDS, R. K. (1955). *Arithmetic operations in digital computers*, Van Nostrand, New York.

TOCHER, T. D. (1958). Techniques of multiplication and division for automatic binary computers, *Quart. J. Mech. Appl. Maths.*, Vol. 11, Pt. 3, pp. 364-384.

WALKER, B. S. (1967). *Introduction to computer engineering*, University of London Press.