Ternary logic in parallel multipliers*

Z. G. Vranesic and V. C. Hamacher

Departments of Electrical Engineering and Computer Science, University of Toronto, Toronto, Ontario, Canada

The logic cost and speed of parallel multipliers implemented in both binary and ternary logic is studied. Binary operand lengths of 8 through 32 bits and the corresponding ternary digit range of 6 through 21 are considered. For the particular design technique used, the binary versions are slightly faster where the speed criterion is in terms of the longest logic path from operands to product. Ternary designs show smaller total cost of gates and a major reduction in the number of required inputs, indicating greatly simplified wiring interconnection complexity.

(Received June 1971)

1. Introduction

Recent technical literature shows an increasing incidence of papers describing many-valued switching systems. Workable algebras and minimisation techniques for such systems have been proposed (Allen and Givone, 1968; Vranesic, Lee, and Smith, 1970; Pugh, 1967). However, the fundamental question of applicability of non-binary schemes within the framework of present binary technology has seldom been critically approached. It is apparent that when three-valued storage elements become 'naturally' available, it will be sensible to use them in conjunction with other ternary logic primitives. In fact, a well-known argument is often encountered that since the most efficient radix for implementation of switching systems is the natural base ($e = 2{\cdot}71828\ldots$) it seems likely that the 'best' integral radix is 3 rather than 2. Unfortunately there have been few attempts to show the validity of this hypothesis in the realm of currently available devices.

In this paper we take a close look at the possibility of using ternary arithmetic circuitry to facilitate implementation of large units required for parallel multiplication.

High speed parallel binary multiplication has been studied for a number of years (Ramamoorthy and Economides, 1969; Wallace, 1964). Implementations incorporating some of these design ideas in prototype models (Habibi and Wintz, 1970; Pezaris, 1971) and commercial production versions (Anderson, Earle, Goldschmidt, and Powers, 1967; Control Data Corporation, 1966) also exist.

Our aim is to design a ternary parallel multiplier and compare it with a binary design for equivalent sizes of operands. The comparison is made on the basis of gate and input costs and speed in terms of delay along the longest logic path from operands to product, where each gate is assigned a unit delay τ. In the binary case AND-OR-NOT logic is assumed and in ternary the switching primitives of Vranesic *et al.* (1970), defined in **Table 1,** comprise the basic gates. A gate fan-in limit of 8 is used throughout while fan-out problems are ignored (binary fan-out problems are always worse than in the corresponding ternary system).

The basic concepts used to speed up binary multiplication are adapted and extended in the ternary design. Included are:

1. Digit grouping of the multiplier in pairs and appropriate summand selection.
2. Carry-save reduction of the summands in a 'carry-save adder (CSA) tree'.
3. Fast addition of the final two summands to obtain the required product, using a design incorporating first-level carry lookahead logic, essentially the same as in Flores (1963).

2. High-speed binary multiplier logic design

The numbers to be multiplied are considered as positive integers represented by n bits, the multiplier being denoted as $D = d_n \ldots d_2 d_1$ and the multiplicand denoted as $M = m_n \ldots m_2 m_1$. The multiplier is recoded in pairs using the Wallace (1964) technique, given in **Table 2.** Bit pairs $d_i d_{i-1}$, $i = 2, 4, \ldots$, select the proper summands as a function of the adjacent bit on the right, d_{i-2}. Note that all summands can be obtained by shifting ($\times 2$) and/or complementing M. Negative versions of M are handled in 2's-complement form. It is assumed that the 1's-complement is available at the output of the M register and 1 to be added in the low-order bit position is easily introduced into the CSA summand reduction tree. **Fig. 1** shows a schematic of the multiplier for $n = 24$. There are 13 input summands to

Table 1 Truth table for ternary switching primitives (Vranesic, *et al.*, 1970)

		OR	AND	CYCLING		INVERTER
x	y	$x + y$	xy	$x^{\overset{y}{\rightarrow}}$	$x^{\overset{y}{\leftarrow}}$	x^y
0	0	0	0	0	0	0
0	1	1	0	1	2	1
0	2	2	0	2	1	2
1	0	1	0	1	1	0
1	1	1	1	2	0	0
1	2	2	1	0	2	0
2	0	2	0	2	2	0
2	1	2	1	0	1	0
2	2	2	2	1	0	0

Table 2 Binary multiplier recoding technique (Wallace, 1964)

BIT PAIR		ADJACENT BIT	SUMMAND SELECTED
d_i	d_{i-1}	d_{i-2}	
0	0	0	$0 \times M$
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	−2
1	0	1	−1
1	1	0	−1
1	1	1	0

*This Research was supported in part by the National Research Council of Canada

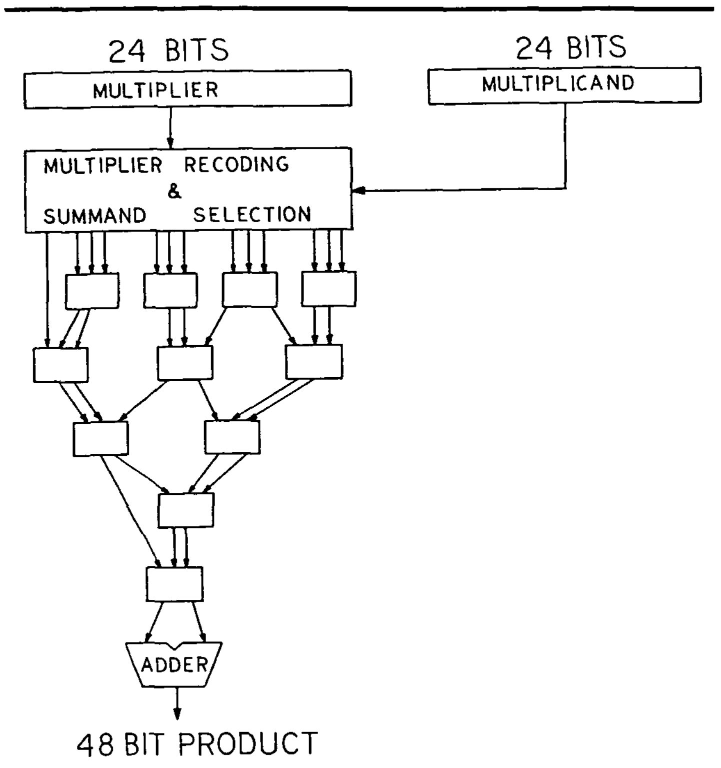

Fig. 1. Binary multiplier for 24-bit operands

Table 3 24-bit binary multiplier cost

SUBUNIT	AND-OR-NOT GATES	INPUTS
Multiplier recoding and summand selection	1,564	3,745
Tree	3,954	9,158
Adder	737	2,216
Totals	6,255	15,119

the tree, one for each bit pair of D plus one last summand which is either M or all zeroes, depending on whether or not $d_{24} = 1$, that is, the thirteenth summand results from recoding an implied 00 digit pair to the left of d_{24}.

Multiplier recoding and summand selection causes 4τ delay. The tree is constructed from binary full adders (the CSA technique) and each level contributes 3τ delay since input complements are not assumed available. Since there are five levels, the total tree delay is 15τ. The final 48-bit adder is of the first-level lookahead type with a group size of 8 and it contributes 8τ of delay. The total delay through the 24-bit multiplier is thus 27τ. The cost in gates and inputs is given in **Table 3.** Similar computations were made for other operand lengths and the results are presented in Section 4.

3. Ternary multiplier design

Ternary multiplication is arranged following the basic structure of Section 2. While we are still using the same operand names M and D, all switching variables in this section are ternary having truth values 0, 1 and 2. We also note that digit by digit multiplication is never needed, since non-trivial summands can be formed initially using the addition process only. The multiplier is recoded in digit pairs, the same grouping as in binary. Whereas in binary all versions of M were easy to obtain, the same is not true in ternary. Besides shifting ($\times 3$) and/or complementing M, it is necessary to generate $\pm 2 \times M$ and $\pm 4 \times M$. An extension of the Wallace (1964) technique is used for recoding the multiplier as shown in **Table 4.** The two non-trivial summands, $2 \times M$ and $4 \times M$, are obtained at the start of the multiplication process by using both halves of the adder. A new method of summand reduction in a tree structure is proposed for ternary. Four summands are reduced to two at each level. The sum of four ternary digits can have a maximum value of 8, which can just be represented by two ternary digits, each of value 2, one in the same digit position as the four summand digits and the other in the next higher digit position. It is convenient to call these the sum and carry-out digits, respectively. This 4 to 2 summand reduction process makes the reduction tree smaller than in the binary case, and in addition the ternary equivalent of any binary multiplier has fewer summands from the start.

This brief summary of the design technique for ternary multipliers will now be expanded in terms of the switching functions needed in the various subunits, and a 16-digit multiplier will be used as a concrete example. Its schematic is shown in **Fig. 2.** The incoming carry value $C_{i/2-1}$, to the digit pair position $d_i d_{i-1}$, $i = 2, 4, \ldots, 16$, which is simply the adjacent bit, d_{i-2}, in binary, is somewhat more complicated in ternary. In general, it is a function of all preceding multiplier digits, very analogous to the way in which carries are determined in parallel binary adders. This is the motivation for the term carry as used here. There is an incoming carry if the previous digit pair, considered as a 2-digit ternary coded integer has a value equal to or greater than 5 or has a value of 4 with a carry in to its position. Some notation is helpful at this point. Let

$$(Z_{J,K})_m = \overset{J}{d_i^{\leftarrow 2}}\ \overset{K}{d_{i-1}^{\leftarrow 2}}$$

for $J, K \in \{0, 1, 2\}$, $2m = i$, represent '1-out-of-9' decoding of the multiplier digit pairs. Note that the $(Z_{J,K})_m$ variables only attain the values 2 or 0, since they each represent the presence or absence (respectively) of specific digit pair values.

The summand selection variables Q are given by

$$\begin{aligned} Q_{1,m} &= (Z_{0,0})_m C_{m-1} + (Z_{0,1})_m C_{m-1}{}^2 \\ Q_{-1,m} &= (Z_{2,1})_m C_{m-1} + (Z_{2,2})_m C_{m-1}{}^2 \\ &\vdots \\ Q_{-4,m} &= (Z_{1,1})_m C_{m-1} + (Z_{1,2})_m C_{m-1}{}^2 \end{aligned}$$

for

$$C_m = G_m + P_m G_{m-1} + \ldots + P_m P_{m-1} \ldots P_2 G_1$$

Table 4 A ternary multiplier recoding scheme

DIGIT PAIR d_i	d_{i-1}	INCOMING CARRY $C_{i/2-1}$	SUMMAND SELECTED
0	0	0	$0 \times M$
0	0	2	1
0	1	0	1
0	1	2	2
0	2	0	2
0	2	2	3
1	0	0	3
1	0	2	4
1	1	0	4
1	1	2	−4
1	2	0	−4
1	2	2	−3
2	0	0	−3
2	0	2	−2
2	1	0	−2
2	1	2	−1
2	2	0	−1
2	2	2	0

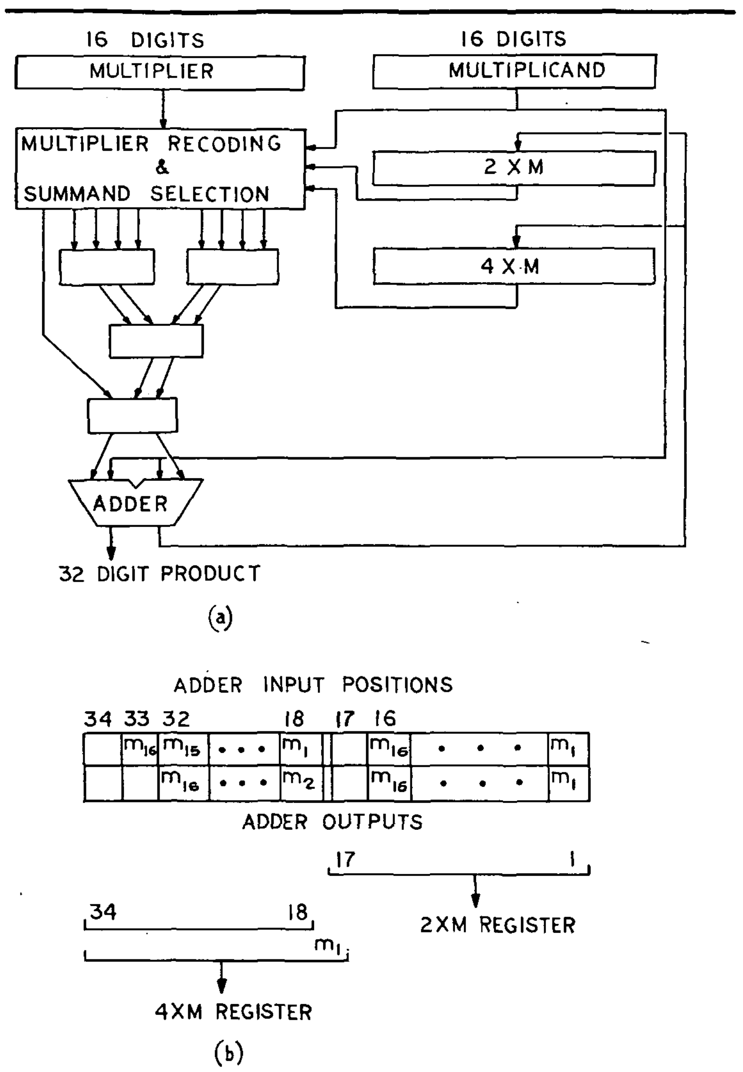

Fig. 2. Ternary multiplier for 16-digit operands. (*a*) Overall schematic. (*b*) Adder details for summand generation

where

$$P_m = (Z_{1,1})_m$$

and

$$G_m = (Z_{1,2})_m + (Z_{2,0})_m + (Z_{2,1})_m + (Z_{2,2})_m$$

The variables G_m, P_m, C_m are used to form the gating variables $Q_{x,m}$, hence they are 'binary-like', attaining the values 0 and 2 only. If $T_{m,i}$ denotes the ith digit position of the mth summand input to the ternary tree then

$$T_{m,i} = M_{1,i}Q_{1,m} + M_{-1,i}Q_{-1,m} + \ldots + M_{-4,i}Q_{-4,m}$$

where $M_{1,i}$ is the ith digit position of $1 \times M$, $M_{-1,i}$ is the ith digit position of $-1 \times M$, etc. An example helps to visualise how these equations (synonymous with their logic gate implementation) select the proper tree input summands. Suppose

$$D = 2011022101211210$$

Digit pair $d_{12}d_{11} = 02$, therefore $(Z_{0,2})_6 = 2$ and all other Z variables with subscript 6 are zero. Since $d_{10}d_9 = 21$ has a value of $7 \geq 5$, the incoming carry C_5 must be present, i.e. have a logic value of 2. Note that $C_5 = 2$, because $(Z_{2,1})_5 = 2$, causing G_5 to have a value of 2. From the pattern of the above equations for the $Q_{x,6}$ variables, which directly implement the selection procedure of Table 4 for the $m = 6$ digit pair, it is clear that $Q_{3,6}$ is the only one that has a value of 2. It is set to 2 by the term $(Z_{0,2})_6C_5$. Since $Q_{3,6} = 2$, tree inputs $T_{6,i}$ attain the values of the summand variables $M_{3,i}$ which is exactly the action called for by Table 4. It should be noted that when a negative summand is to be used, 3's-complement arithmetic is used, that is, the 2's-complement of the summand is introduced into the tree with a 1 to be added in the low-order position. From the above equations, it is easily shown that the tree inputs are available after 11τ time units. This assumes that all versions of M are available at the outputs of their registers after 9τ time units, as explained below.

Fig. 2(*b*) shows schematically how both halves of the 34-digit adder are used simultaneously to form $2 \times M$ and $4 \times M$. The right-most 17 positions form $2 \times M = M + M$ by gating digits m_{16} through m_1 into both adder inputs of positions 16 through 1. Note that although sum digit 17 may or may not be 0, the carry out from position 17 is always 0, so that adder positions 34 through 18 can be used to form $4 \times M = M + 3M$ by gating m_{16} through m_1 into positions 33 through 18 on one side of the adder and gating m_{16} through m_2 into positions 32 through 18 on the other side of the adder. Adder outputs 34 through 18 are then set into positions 18 through 2 of the $4 \times M$, with m_1 being set into position 1; and outputs 17 through 1 are set into the corresponding positions of the $2 \times M$ register. This is a convenient point to complete the discussion of the adder. A first-level lookahead type adder has been assumed similar to the standard procedure as in Flores (1963). The general fan-in limit of 8 implies a group size of 8 for lookahead purposes and this results in a worst case delay of 8τ time units in producing all sum digits. The delay is derived as follows. The propagate and generate functions for each digit position i are formed in 3τ as

$$p_i = a_ib_i + a_i^{\rightarrow 1} + b_i^{\rightarrow 1}$$

$$g_i = (a_ib_i)(a_i + b_i)^{\leftarrow}$$

followed by formation of all carries in 4τ using standard binary logic. A further 1τ is needed to compute the sum function

$$s_i = (a_i^{\overset{b_i}{\rightarrow}})^{\overset{c_{i-1}}{\rightarrow}}$$

where $(a_i^{\overset{b_i}{\rightarrow}})$ is formed before the carries are available.

A description of the ternary tree summand reduction logic remains. Letting the inputs to one position of 4 to 2 reduction logic be a, b, c, and d, the sum and carry-out functions are

$$s = a^{\overset{b}{\rightarrow}\overset{c}{\rightarrow}\overset{d}{\rightarrow}}$$

$$c = abc + abd + bcd + p(abcd)^{22} + p(a + b + c + d)^{\leftarrow}$$

where

$$p = (a + b + c)(a + b + d)(a + c + d)(b + c + d)\,.$$

The delay through a 4 to 2 reduction level is clearly 5τ and thus through the complete tree of Fig. 2(*a*) is 15τ. The above discussion shows a 34τ delay through the complete ternary

Table 5 16-digit ternary multiplier cost

	GATES			
SUBUNIT	INVERTER	CYCLING	AND/OR	INPUTS
Multiplier recoding and summand selection	55	32	1,523	4,114
Tree	112	331	916	3,880
Adder	66	165	330	1,596
$2 \times M$, $4 \times M$ registers and adder input gating	—	71	172	340
Totals	233	599	2,941	9,930

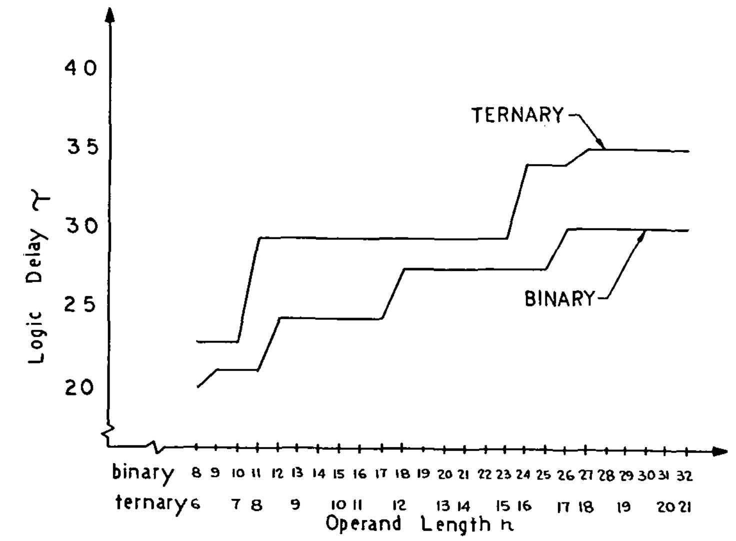

Fig. 3. Multiplier speed comparisons

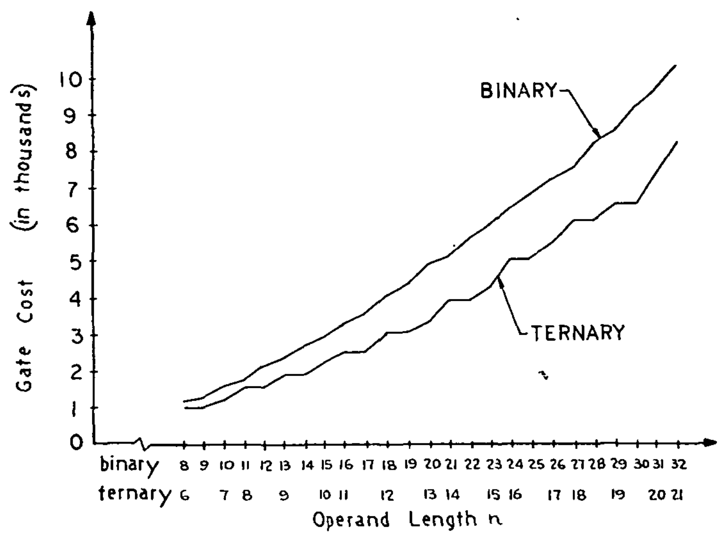

Fig. 4. Multiplier gate cost comparisons

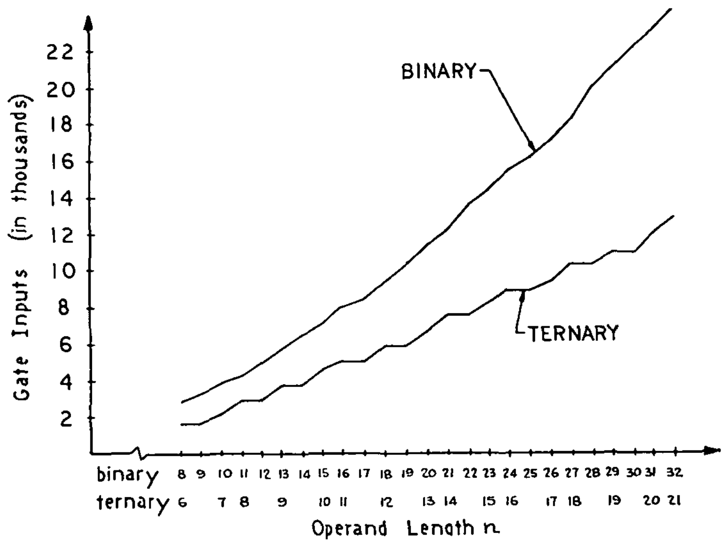

Fig. 5. Multiplier gate inputs comparisons

multiplier. The cost in gates and inputs is given in **Table 5.** Again, as in the binary case, similar computations were made for other operand lengths and the comparative results appear in Section 4.

4. Comparative analysis

When comparing any arithmetic units that operate on number representations in different bases, operand lengths should be such that about the same range of values is accommodated in each unit. It was arbitrarily decided to plot the comparison of binary multipliers in the operand range 8 through 32 with the corresponding ternary designs that are just large enough to include the binary range for each operand length. In general, if D ternary digits are to represent values representable by d binary digits, then

$$3^D - 1 \geq 2^d - 1$$

or

$$D = \ulcorner d \log 2/\log 3 \urcorner \doteq \ulcorner 0{\cdot}631\, d \urcorner$$

In the examples of the previous two sections we illustrated the designs for $d = 24$ and $D = \ulcorner 0{\cdot}631 \times 24 \urcorner = \ulcorner 15{\cdot}144 \urcorner = 16$. In terms of value ranges, $2^{24} - 1 = 16{,}777{,}215$ and $3^{16} - 1 = 43{,}046{,}720$. This particular example illustrates that the comparison method chosen is in fact biased against ternary quite strongly for certain operand lengths.

Fig. 3 shows the speed comparison in terms of logic delay τ. Whether or not delay through a 3-valued logic gate is comparable to delay through a 2-valued gate from a practical standpoint is clearly technology dependent and likely to change as circuit techniques evolve.

It is more difficult to decide upon measures for cost comparisons. One of the more conventional techniques is to use the total number of gates and inputs as the two measures, and that has been done here. The total number of inputs to gates is a straightforward measure of the combination of circuit package pin count and intra-package circuit connection complexity. The difficulty is in arriving at a way of counting gate costs. The basic assumption made here is that each binary gate has a cost of one unit. Ternary AND, OR and INVERTER gates are also assumed to have unit cost since they have circuit implementations very similar to the binary gates. The two versions of ternary CYCLING gates (mod 3 adder and mod 3 subtractor) are more expensive to implement than any of the other gates, hence a weight of 3 units has been assigned to each cycling gate. The effect of this choice is not too critical since cycling gates form a reasonably small portion of the total number of gates, less than 20% in the $n = 16$ example detailed in Table 5. **Figs. 4 and 5** show the comparisons. Using the three standard binary operand lengths of 16, 24, and 32, it is instructive to calculate some relative percentage comparisons between the binary and ternary designs. At those binary operand lengths, where the comparable ternary lengths are 11, 16, and 21, respectively, the binary versions are about 17% faster on the average. The gate cost in ternary is about 22% less than in binary, while input count is about 38% lower.

5. Conclusions

An attempt was made to design ternary parallel multipliers with propagation delays along the longest logic path comparable to those of their binary equivalents. Actual designs show the delays slightly favouring the binary case.

Circuit cost in terms of gate count is lower in ternary designs although not significantly enough to be of major importance.

The key result is exhibited by Fig. 5, showing considerable reduction in the number of inputs in ternary cases. This may be sufficient to permit physical implementation of ternary multipliers at a size level where construction of the binary equivalents is currently not feasible because of wiring interconnection complexity.

References

Allen, C. M., and Givone, D. D. (1968). A Minimisation Technique for Multiple-Valued Logic Systems, *IEEE Trans. Comp.*, Vol. C-17, No. 2, pp. 182-184.

ANDERSON, S. F., EARLE, J. G., GOLDSCHMIDT, R. E., and POWERS, D. M. (1967). The IBM System/360 Model 91: Floating Point Execution Unit, *IBM Journal of R. and D.*, pp. 34-53.
CONTROL DATA CORPORATION (1966). 6600 Central Processor Description, Control Data Institute.
FLORES, I. (1963). *The Logic of Computer Arithmetic*, Prentice-Hall, Englewood Cliffs, N.J.
HABIBI, A., and WINTZ, P. A. (1970). Fast Multipliers, *IEEE Trans. on Computers*, Vol. C-19, No. 2, pp. 153-157.
PEZARIS, S. D. (1971). A 40-ns 17-bit by 17-bit Array Multiplier, *IEEE Trans. on Comp.*, Vol. C-20, No. 4, pp. 442-447.
PUGH, A. (1967). Application of Binary Devices and Boolean Algebra to Realisation of 3-valued Logic Circuits, *Proc. IEE*, Vol. 114, pp. 225-228.
RAMAMOORTHY, C. V., and ECONOMIDES, S. C. (1969). Fast Multiplication Cellular Arrays for LSI Implementation, 1969 FJCC, Vol 35, pp. 89-98.
VRANESIC, Z. G., LEE, E. S., and SMITH, K. C. (1970). A Many-Valued Algebra for Switching Systems, *IEEE Trans. on Comp.*, Vol. C-19, No. 10, pp. 964-71.
WALLACE, C. S. (1964). A Suggestion for a Fast Multiplier, *IEEE Trans. on Elec. Comp.*, pp. 14-17.

Correspondence

To the Editor
The Computer Journal

Sir,

The generalised Euler transformation

A recent paper by Wynn (1971) discusses the generalised Euler transformation

$$\sum_{s=0}^{\infty} u_s \Rightarrow \frac{1}{1-z} \sum_{s=0}^{\infty} \left(\frac{z}{1-z}\right)^s \Delta v^s_0 ,$$

where $u_s = z^s v_s$. An important matter in the use of this transformation is the choice of a suitable value for z, and Wynn suggests that z be chosen as the limit as $s \to \infty$ of the ratio u_{s+1}/u_s (provided that this limit exists). In fact this does not generally lead to the best value of z.

Consider, for instance, the well-known series

$$1 - \tfrac{1}{2} + \tfrac{1}{3} - \tfrac{1}{4} + \ldots,$$

in which $u_s = (-1)^s(s + 1)^{-1}$. The ratio u_{s+1}/u_s tends to -1, and putting $z = -1$ gives the transformed series

$$\tfrac{1}{2}[1 + \tfrac{1}{2}(\tfrac{1}{2}) + \tfrac{1}{3}(\tfrac{1}{2})^2 + \tfrac{1}{4}(\tfrac{1}{2})^3 + \ldots] .$$

This converges more rapidly than the original series; but we can do better still by putting $z = -\tfrac{1}{2}$, which gives

$$\tfrac{2}{3}[1 + 0 + \tfrac{1}{3}(\tfrac{1}{3})^2 + 0 + \tfrac{1}{5}(\tfrac{1}{3})^4 + \ldots] .$$

A more startling example is given by taking

$$u_s = \tfrac{1}{5}[(\tfrac{2}{5})^s + (\tfrac{4}{5})^s] .$$

In this case $u_{s+1}/u_s \to \tfrac{4}{5}$, and the transformed series with $z = \tfrac{4}{5}$ is

$$2 - 2 + 2^2 - 2^3 + 2^4 - \ldots,$$

which is divergent. Yet by taking $z = \tfrac{3}{5}$ we get the rapidly convergent series

$$1 + 0 + (\tfrac{1}{2})^2 + 0 + (\tfrac{1}{2})^4 + \ldots.$$

A theoretical method for finding the optimum value of z was given by me in an earlier paper (Scraton, 1969). This method cannot be used, however, if one knows nothing about the terms u_s except their numerical values, and as far as I am aware no computational algorithm has yet been devised for finding the optimum value of z in these circumstances.

Yours faithfully,
R. E. SCRATON

Department of Mathematics
University of Bradford
Bradford 7
14 March 1972

References

SCRATON, R. E. (1969). A note on the summation of divergent power series, *Proc. Camb. Phil. Soc.*, Vol. 66, p. 109.
WYNN, P. (1971). A note on the generalised Euler transformation, *The Computer Journal*, Vol. 14, p. 437.

To the Editor
The Computer Journal

Sir,

In his letter on high level languages in Vol. 15, No. 1, 1972 of this Journal, J. Palme gives an example of a spelling mistake which he says would be detected in ALGOL but not in FORTRAN.

Provided the incorrectly written variable did not also occur on the left hand side of an assignment statement it would be detected by a good FORTRAN compiler as an undefined variable.

Yours faithfully,
H. W. BRADLY

3 Belleville Drive
Oadby
Leicester LE2 4HA
24 March 1972

To the Editor
The Computer Journal

Sir,

G. M. Bull's article 'Dynamic debugging in BASIC' (February, 1972) was chiefly valuable in spreading the gospel about the great advantages of interpretive compilers for use in time sharing. However, his implementation contains little that is new; all his facilities except breakpoints and the trace feature have existed for several years on the Conversational Programming System (CPS) running on 360/40's and up. CPS, with a choice of PL/1 or BASIC, remains (until TSO proves otherwise) the best general purpose time sharing system available on IBM computers.

Yours faithfully,
DAVID SILBER

Reference

IBM MANUAL GH20-0758. Conversational Programming System (CPS) Terminal User's Manual.

P.S. Speaking of time sharing, does it not seem odd to anyone else that there has been no demand to form a BCS Specialist Group on Time Sharing?

Erratum

There was an error in the paper 'File design fallacies' by S. J. Waters (this *Journal*, Vol. 15, No. 1, p. 1). The formula on the 9th line of page 2 should read:

Track Hit Ratio = 1 − (1 - Record Hit Ratio) Number of Records in Track so that the multiplier should be a power.