AN ABSTRACT OF THE THESIS OF

Phillip Dennis Siemens (Name) for the M.S. (Degree) in Electrical and Electronic Engineering (Major)

Date thesis is presented 8-23-66

Title OPTIMUM LOGIC DESIGN FOR A FAST PARALLEL MULTIPLIER

Abstract approved Redacted for Privacy (Major Professor)

This thesis discusses a method of fast multiplication by parallel addition of summands. A logical element that performs this parallel addition is defined, and examples of the element realized with threshold logic are shown. Relations between the type of logical element used, and the speed and cost of the multiplier are discussed. The optimum type of logical element is defined, and two examples of a multiplier using this optimum element are discussed. By assuming some hypothetical propagation times for the various elements, a multiply time of 500 ns is predicted for an eighty-bit multiplier.

OPTIMUM LOGIC DESIGN FOR A FAST PARALLEL MULTIPLIER

by

PHILLIP DENNIS SIEMENS

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

MASTER OF SCIENCE

June 1967

APPROVED:

Redacted for Privacy

Professor of Electrical and Electronic Engineering
In Charge of Major

Redacted for Privacy

Head of Department of Electrical and Electronic Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented 8-23-66

Typed by Jan Lewis

ACKNOWLEDGMENTS

The author wishes to express his appreciation for the guidance and criticism given by Mr. L. N. Stone during the preparation of this thesis.

He would also like to thank his wife for her patience, understanding, and help during the preparation of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

OPTIMUM LOGIC DESIGN FOR A FAST PARALLEL MULTIPLIER

INTRODUCTION

With the development of today's large computer systems, a major aspect of design criteria has been the speed of operation. Economically it is advantageous to have a system operate as fast as possible, as long as the increase in cost of the hardware does not offset the gain in computing speed. Generally, work in this area has been along the lines of decreasing the propagation time through a logical gate. As a result, the clock rate of the computing systems has been increasing. However, it is believed that improvement along these lines must soon reach a plateau, due to the physical limitation of the speed of propagation of an electrical signal. Thinking of the multiplication process in this light, it is evident that for large scientific applications, the multiplication time could be a serious limitation. Therefore, a scheme faster than the so-called "conventional" method of multiplication, which is achieved by repeated addition and shifting, needs to be found.

Lehman (5) has proposed a short-cut multiplication process that requires both addition and subtraction. The process, which requires a comparison between two adjacent multiplier bits to determine the operation to be performed and which still requires repeated shifting, takes advantage of sequences of "1's" or "0's" appearing

in the multiplier. Lehman predicted an average reduction in multiplication time from the "conventional" shift-add method of thirty percent.

Green (3) suggested another method of binary multiplication. Green's multiplier consisted of three stages, the first of which was a matrix of AND gates to simultaneously form all the summands. The second stage was a simultaneous adder that gave as an output the number of inputs energized. The adder proceeded position by position, from the least to the most significant bit position. The higher ordered positions were delayed until the next lower ordered position was processed. A carry was generated when necessary and fed back to the input, after being delayed one bit time. The output from the simultaneous adder was fed to the third stage of the multiplier, a shift register. While this method would be faster than "conventional" multiplication, basically it still required n operations, where n was the length of the multiplier.

A fast method of parallel multiplication has been suggested by Wallace (10). Since multiplication is basically the addition of a number of summands, Wallace suggested three criteria to speed up the process: (1) reduce the number of summands, (2) accelerate the formation of summands, and (3) accelerate the addition of summands.

The first of these suggestions will require the recoding of the

multiplier and multiplicand into a number system other than the binary code. The second suggestion can easily be achieved with the necessary amount of hardware. It is the third suggestion with which most of the work in this thesis will be concerned.

Essentially, the question Wallace asks is: why add the summands sequentially? Why not add more hardware and add some of the summands in parallel? Conceivably, it would be possible to break the summands into sub-groups of two each and add, in parallel, the two summands in each sub-group. This same process would then be repeated on the results, etc., until the final sum was reached. Note, however, that a carry propagating adder would be required for each sub-group. The gain in multiplication time would probably be disproportionately small compared to the increase in cost. Another possibility would be to add more than two summands to get a single result, but this becomes complex due to the need for propagation of a carry.

Wallace indicates that it would be possible to add three numbers and as a result get two numbers whose sum is the sum of the original three (see Table I for the truth table of a logical element that accomplishes this). In fact, a conventional full adder accomplishes exactly this. This idea can be extended in the following manner: add five numbers and as a result get three numbers whose sum is the sum of the original five (see Table II for the truth table

of a logical element that accomplishes this). In general, n numbers can be added resulting in (n+1)/2 numbers whose sum is the sum of the original n numbers. The logic block that will accomplish this will be designated an n-input pseudo-adder. Thus, a full adder can be referred to as a three-input pseudo-adder. The implementation of an n-input pseudo-adder, where n is greater than three, becomes very complex, if not impossible, using conventional NAND/NOR logic. The use of generalized threshold logic, however, greatly simplifies the design of a pseudo-adder, and it will be shown that the use of such a pseudo-adder will simplify the design of a parallel multiplier.

The thesis investigation was involved in finding which type of pseudo-adder gave the simplest multiplier, which type gave the fastest multiplier, and which type gave the optimum multiplier.

MULTIPLIER DESIGN

As mentioned previously, multiplication is nothing more than the addition of a number of summands. In order to achieve fast multiplication times, this addition should be done in parallel, with as little carry propagation as possible. A logic block, termed a pseudo-adder, which is capable of implementing this type of addition, will now be defined.

A three-input pseudo-adder has an input of three numbers and an output of two numbers whose sum is equal to that of the input numbers. The truth table of a three-input pseudo-adder, which is identical to that of a full adder, is given in Table I. A five-input pseudo-adder is merely an extension of this--it has a sum and two carry outputs. Its truth table is given in Table II.

The truth tables for pseudo-adders with more than five inputs become unmanageable. Fortunately, the sum and carry functions are symmetric, and thus the truth tables are not needed. More will be said on this subject later. Note that three-input pseudo-adders eliminate one summand and five-input pseudo-adders eliminate two summands. In general, an n-input pseudo-adder eliminates $(n-1)/2$ summands.

Note also, that only pseudo-adders with an odd number of inputs have been discussed. While pseudo-adders with an even

Table I. Truth Table for a Three-Input Pseudo-Adder.

X_1	X_2	X_3	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table II. Truth Table for a Five-Input Pseudo-Adder.

X_1	X_2	X_3	X_4	X_5	SUM	$CARRY_1$	$CARRY_2$
0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0
0	0	0	1	0	1	0	0
0	0	0	1	1	0	1	0
0	0	1	0	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0
0	0	1	1	1	1	1	0
0	1	0	0	0	1	0	0
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	0	1	1	1	1	0
0	1	1	0	0	0	1	0
0	1	1	0	1	1	1	0
0	1	1	1	0	1	1	0
0	1	1	1	1	0	0	1
1	0	0	0	0	1	0	0
1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	0	1	1	1	1	0
1	0	1	0	0	0	1	0
1	0	1	0	1	1	1	0
1	0	1	1	0	1	1	0
1	0	1	1	1	0	0	1
1	1	0	0	0	0	1	0
1	1	0	0	1	1	1	0
1	1	0	1	0	1	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	1	1	0
1	1	1	0	1	0	0	1
1	1	1	1	0	0	0	1
1	1	1	1	1	1	0	1

number of inputs are possible to design, they offer no advantages. In fact, they have a disadvantage which can be illustrated as follows: a four-input pseudo-adder must have three outputs--a sum and two carries. Thus, it eliminates only one summand. A five-input pseudo-adder also has three outputs and thus eliminates two summands. By extension, then, it can be seen that pseudo-adders with an odd number of inputs are generally best. Any reference made to a pseudo-adder in the rest of this thesis will assume an odd number of inputs.

For the present, suppose that all the summands are available simultaneously. Consider the arrangement of pseudo-adders in a multiplier. Take, as an example, the realization of a five-bit word length multiplier with three-input pseudo-adders. Initially, with five summands to add, the best method would be to group them into groups of three summands each. This would give one group of three summands, with two summands left over. An output of two new summands (sum and carry functions) would result from the first level (see Figure 1). Combining these new summands with the two left over would give four summands. In the second level of logic, these four are regrouped, giving one group of three and one left over. The three summands remaining (two out of the second level and the one left over) are grouped together and fed into a third level of pseudo-adders. Two summands result from this third level, and

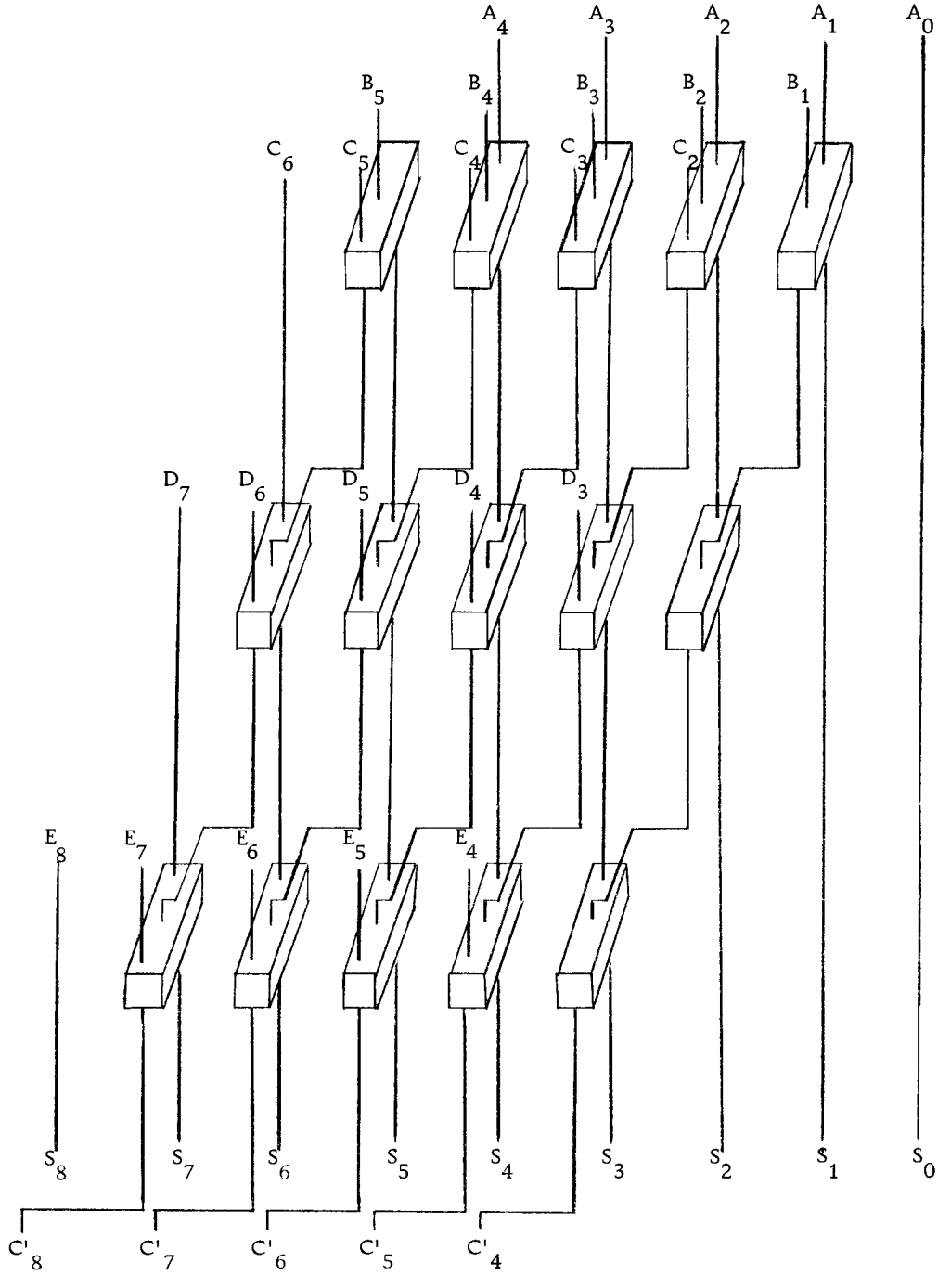


Figure 1. Five-Bit Multiplier Using Three-Input Pseudo-Adders

must be added in a carry propagating adder.

Realizing this same word length multiplier using five-input pseudo-adders gives an entirely different arrangement (see Figure 2). The five summands are grouped into five-input pseudo-adders for the first level. Resulting from this level are three new summands, which are grouped into three-input pseudo-adders. Out of the second level are two summands which are added in a carry propagating adder. From this it would appear that using pseudo-adders with more inputs simplifies the realization of a multiplier.

At this point a small detail must be considered. Analyzing a ten-bit word length multiplier using seven-input pseudo-adders shows that after taking seven of the summands and grouping them into seven-input pseudo-adders, there are three summands left. The remaining three summands could be added in at the next level, or they could be grouped into three-input pseudo-adders. The latter method is the best, since it is most economical in usage of pseudo-adders and it requires fewer levels of logic.

Note that when there are fewer summands than pseudo-adder inputs, the number of inputs should be decreased just enough so that one group of pseudo-adders can handle all of the summands. This was illustrated in the example of a five-bit word length multiplier using five-input pseudo-adders. In the second level only three summands remained, so three-input instead of five-input pseudo-adders were used.

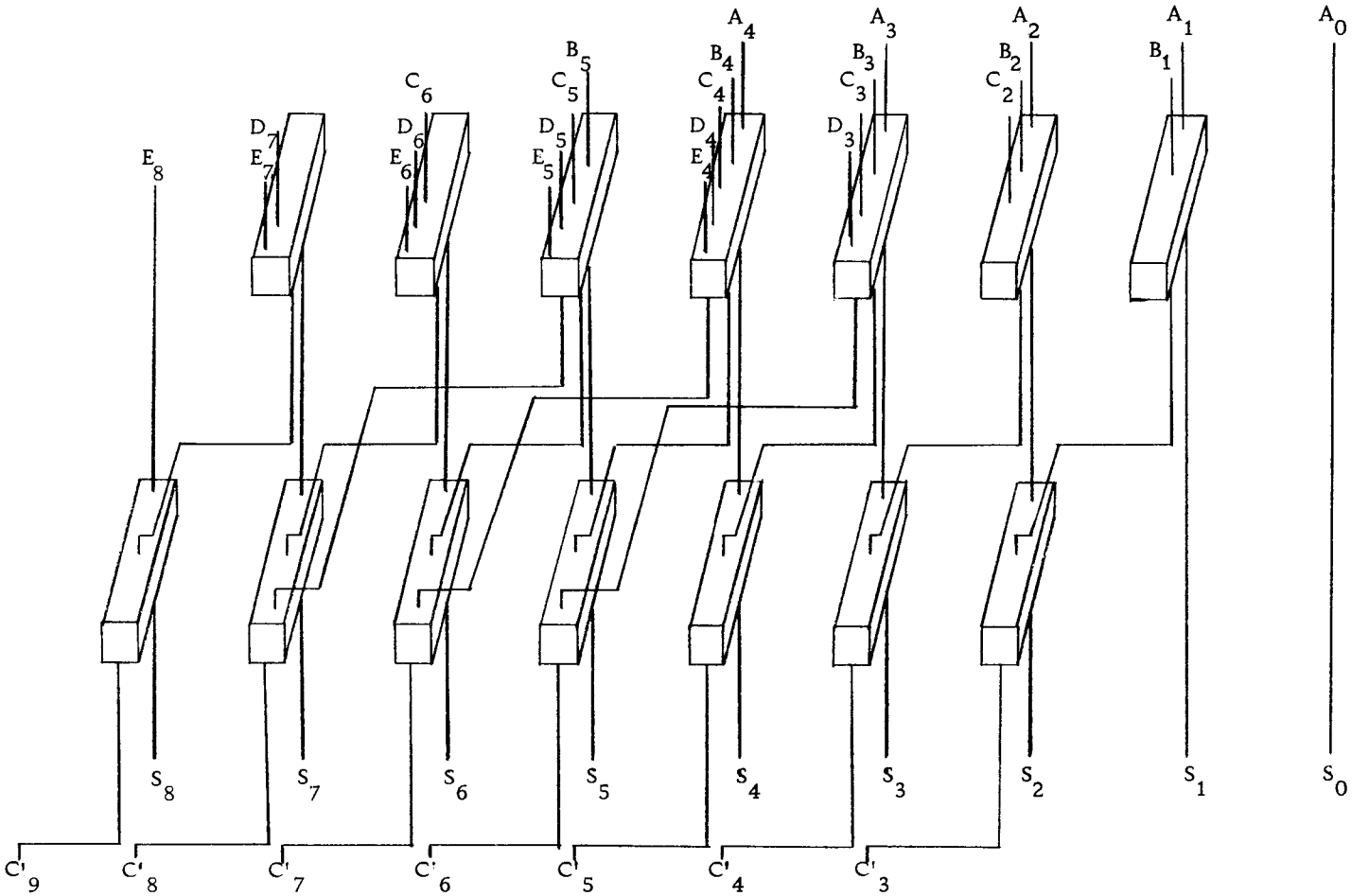


Figure 2. Five-Bit Multiplier Using Five-Input Pseudo-Adders

PSEUDO-ADDER DESIGN

An n-input pseudo-adder, having been definied, must now be designed using available logic. First, consider a three-input pseudo-adder, which as previously mentioned is merely a full adder. This can be realized with conventional NAND logic as shown in Figure 3. Note that the full adder designed in this manner requires eight gates, and the longest signal path is through four gates and one inverter. Three-input majority logic is also a possibility. Using this type of logic, only three gates are needed, and the longest signal path is through two gates and one inverter (see Figure 4). Finally, there is the choice of threshold logic. A threshold gate is defined as having n binary inputs $(x_i \quad i=1, 2, 3, \ldots, n)$ with positive or negative weights $(w_i \quad i=1, 2, 3, \ldots, n)$. The output of the gate is a binary one if the weighted sum of the inputs is greater than or equal to a specified threshold. Otherwise, the output is a zero. This is symbolically represented as

$$\sum_{i=1}^{n} w_i x_i \geq T$$

for a binary one output.

Using threshold logic to realize a three-input pseudo-adder requires only two gates (see Figure 5). Thus, it would seem that threshold logic gives the simplest logic design for a pseudo-adder.

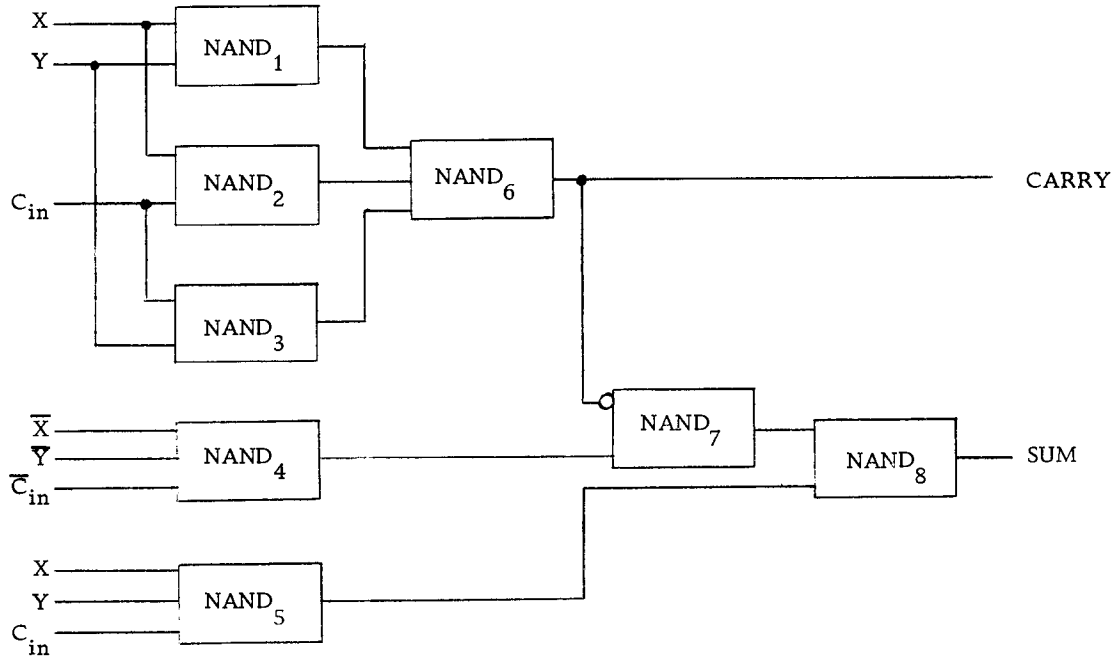


Figure 3. Three-Input Pseudo-Adder Realized With NAND Logic

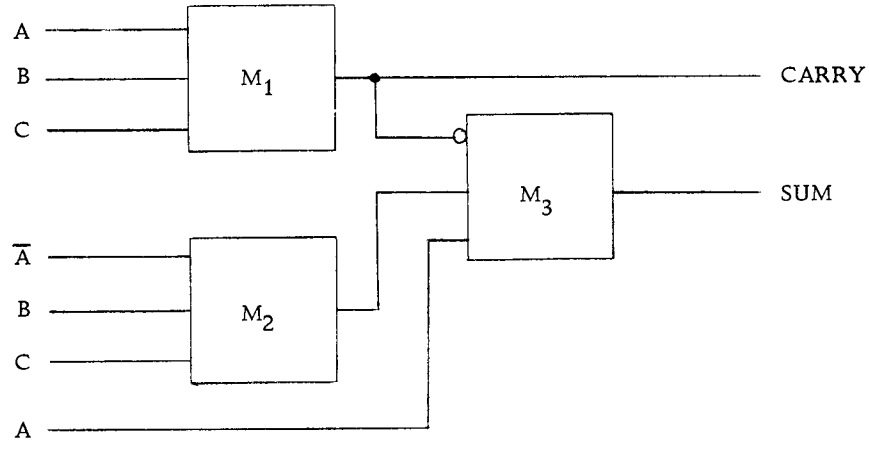


Figure 4. Three-Input Pseudo-Adder Realized With Three-Input Majority Logic

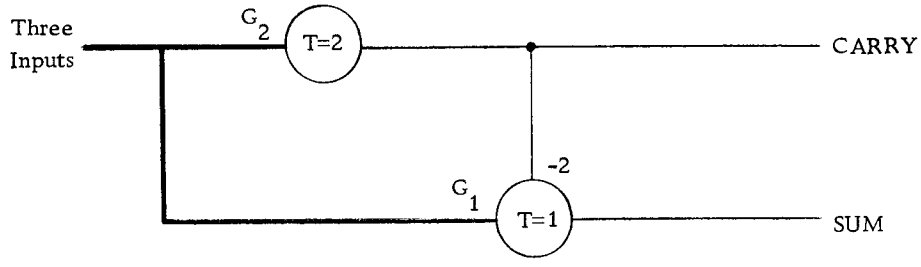


Figure 5. Threshold Realization of a Three-Input Pseudo-Adder

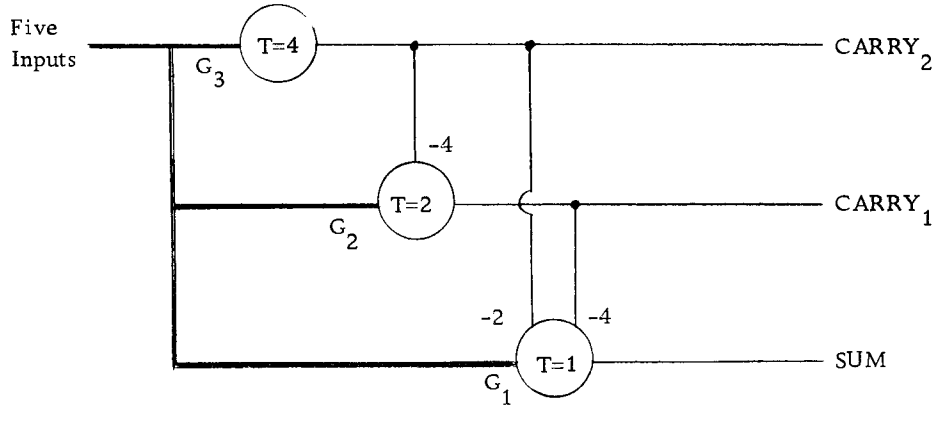


Figure 6. Threshold Realization of a Five-Input Pseudo-Adder

A five-input pseudo-adder realized with threshold logic is shown in Figure 6.

These realizations were done with a design method proposed by Sheng (9) for symmetric Boolean functions. A symmetric function of n variables is one which remains invariant to all permutations of all n variables. Sheng represents a symmetric function as follows: a symmetric function of n variables with n tuples is represented as $S_{(i, j, k, \ldots, m)}(x_1, x_2, x_3, \ldots, x_n)$ where the number of uncomplemented variables with n tuples is $i, j, k, \ldots, m$. Thus, for the full adder, or three-input pseudo-adder,

$$\text{SUM} = S_{(1, 3)}(x_1, x_2, x_3)$$

and

$$\text{CARRY} = S_{(2, 3)}(x_1, x_2, x_3) \ .$$

For the five-input pseudo-adder, the functions are:

$$\text{SUM} = S_{(1, 3, 5)}(x_1, x_2, x_3, x_4, x_5) \ ,$$

$$\text{CARRY}_1 = S_{(2, 3)}(x_1, x_2, x_3, x_4, x_5) \ ,$$

and

$$\text{CARRY}_2 = S_{(4, 5)}(x_1, x_2, x_3, x_4, x_5) \ .$$

In general, for an n-input pseudo-adder the functions are:

$$\text{SUM} = S_{(1, 3, 5, \ldots, n)}(x_1, x_2, x_3, \ldots, x_n) \ ,$$

$$\text{CARRY}_1 = S_{(2, 3)}(x_1, x_2, x_3, \ldots, x_n) \ ,$$

$$\text{CARRY}_2 = S_{(4, 5)}(x_1, x_2, x_3, \ldots, x_n) \ ,$$

$$\vdots$$

$$\text{CARRY}_{m-1} = S_{(m-1, m)}(x_1, x_2, x_3, \ldots, x_n)$$

where $m = (n+1)/2$.

Referring to Figure 6, it can be seen that the SUM is the output from gate G_1 and $CARRY_2$ is the output from gate G_3. Following this example, the design of a pseudo-adder generalizes as follows.

For an n-input pseudo-adder, where n is odd, there are m gates, where $m = (n+1)/2$. The negative weights between gates are represented as w_{jk}, where j is the number of the gate from which the signal comes, and k is the number of the gate to which the signal goes.

Then, the weights are defined as follows:

$w = -2(j-1)$ where $j = 2, 3, 4, \ldots, m$,

$w = -2(j-k+1)$ where $j = k+1, k+2, k+3, \ldots, m$
$2 \leq k \leq m-1$,

$m = (n+1)/2$ where n equals the number of pseudo-adder inputs.

It can be noted here that the longest signal path is through the total number of threshold gates in the pseudo-adder.

OPTIMIZATION OF THE PSEUDO-ADDER TREE

Having proposed the use of pseudo-adders to form a fast parallel multiplier, questions that now might be asked are: for a fixed multiplier word length, which type pseudo-adder (how many inputs) gives the fastest operation, which type gives the cheapest multiplier, and which type gives the optimum realization?

To answer these questions, the summands must be formed into groups, and these groups formed into levels just as in the examples of the five-bit word length multiplier. A computer program was developed to help solve this problem. The propagation time was found by keeping track of how many threshold gates the signal had to pass through. A unit of time was taken to be the time required for the signal to propagate through one threshold gate. The cost of the multiplier was computed on the following basis. It was assumed that an n-input pseudo-adder would be considered as a single package with unit cost. With this view, the cost of a pseudo-adder would be constant. In other words, a three-input pseudo-adder would cost the same as a five-input pseudo-adder. The optimum type of pseudo-adder was the one which gave the fastest operation for the least cost. This was found by computing the cost-propagation time product and the pseudo-adder type that gave a minimum product was the optimum one.

These three quantities were computed for the multipliers, realized with three-, five-, seven-, nine-, eleven-, and thirteen-input pseudo-adders, with word lengths of four to one-hundred-twenty bits in steps of two.

Only the results of the three- and five-input pseudo-adders are shown, as the other cases were very similar. As can be seen in Figure 7, the propagation time increased with word length, and the multiplier realized with five-input pseudo-adders was slower than the multiplier realized with three-input pseudo-adders. The other cases showed this same effect--the more inputs on the pseudo-adders, the longer the propagation time through the pseudo-adder tree. This was very interesting, because the program results also showed that as the number of inputs on the pseudo-adders was increased, the number of pseudo-adder levels decreased. However, the propagation time through a single level increased, and hence canceled any gain. The adder cost of the multiplier behaved opposite to the propagation time. Increasing the number of inputs on the pseudo-adders decreased the cost. This is plainly indicated in Figure 8. It was noted, however, that the cost difference between, say, multipliers using eleven- and thirteen-input pseudo-adders was not as great as the difference between multipliers realized with three- and five-input pseudo-adders. This will be indicated more clearly later.

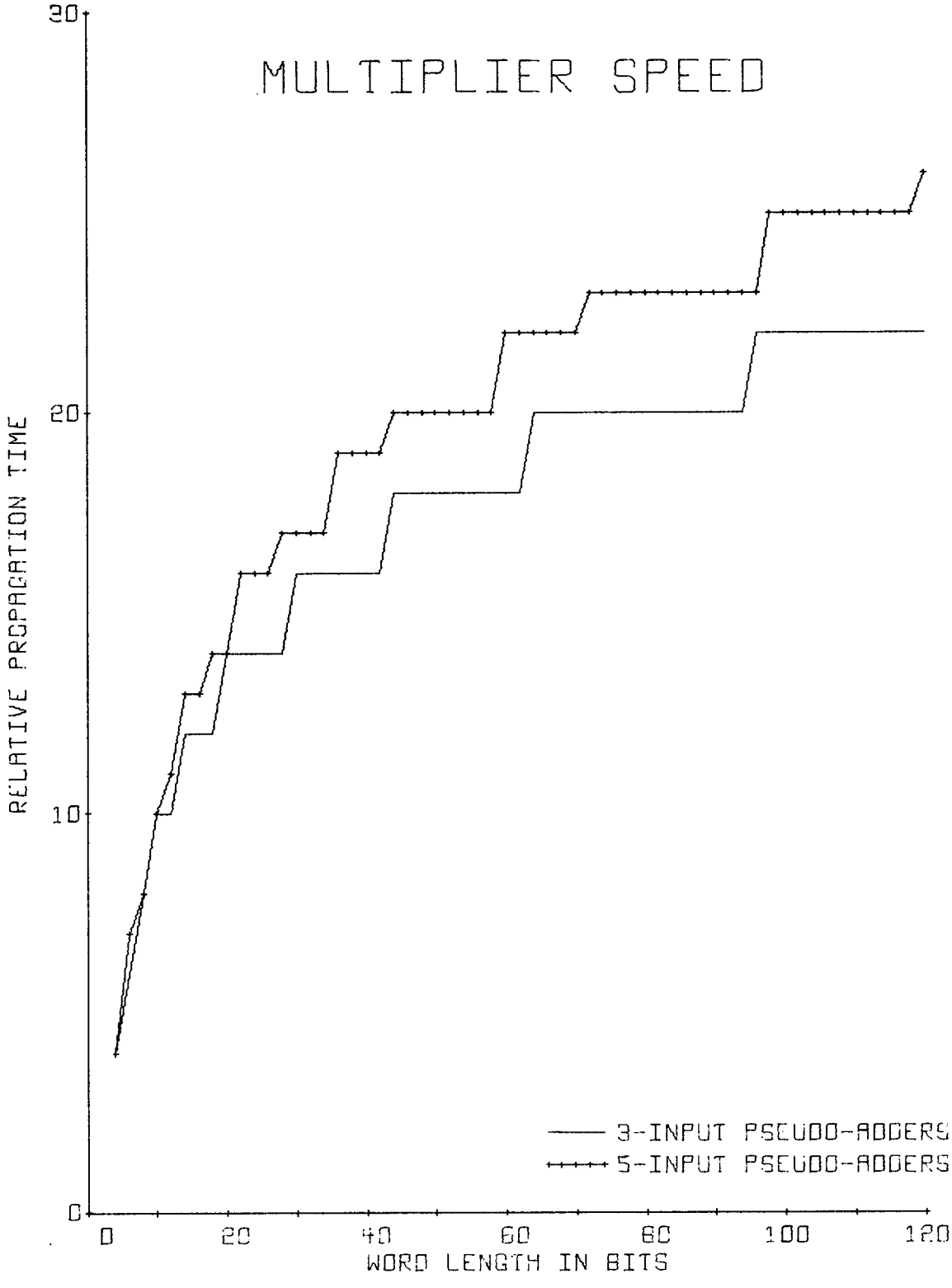


FIGURE 7. RELATIVE PROPAGATION TIME FOR MULTIPLIERS USING THREE- AND FIVE-INPUT PSEUDO-ADDERS

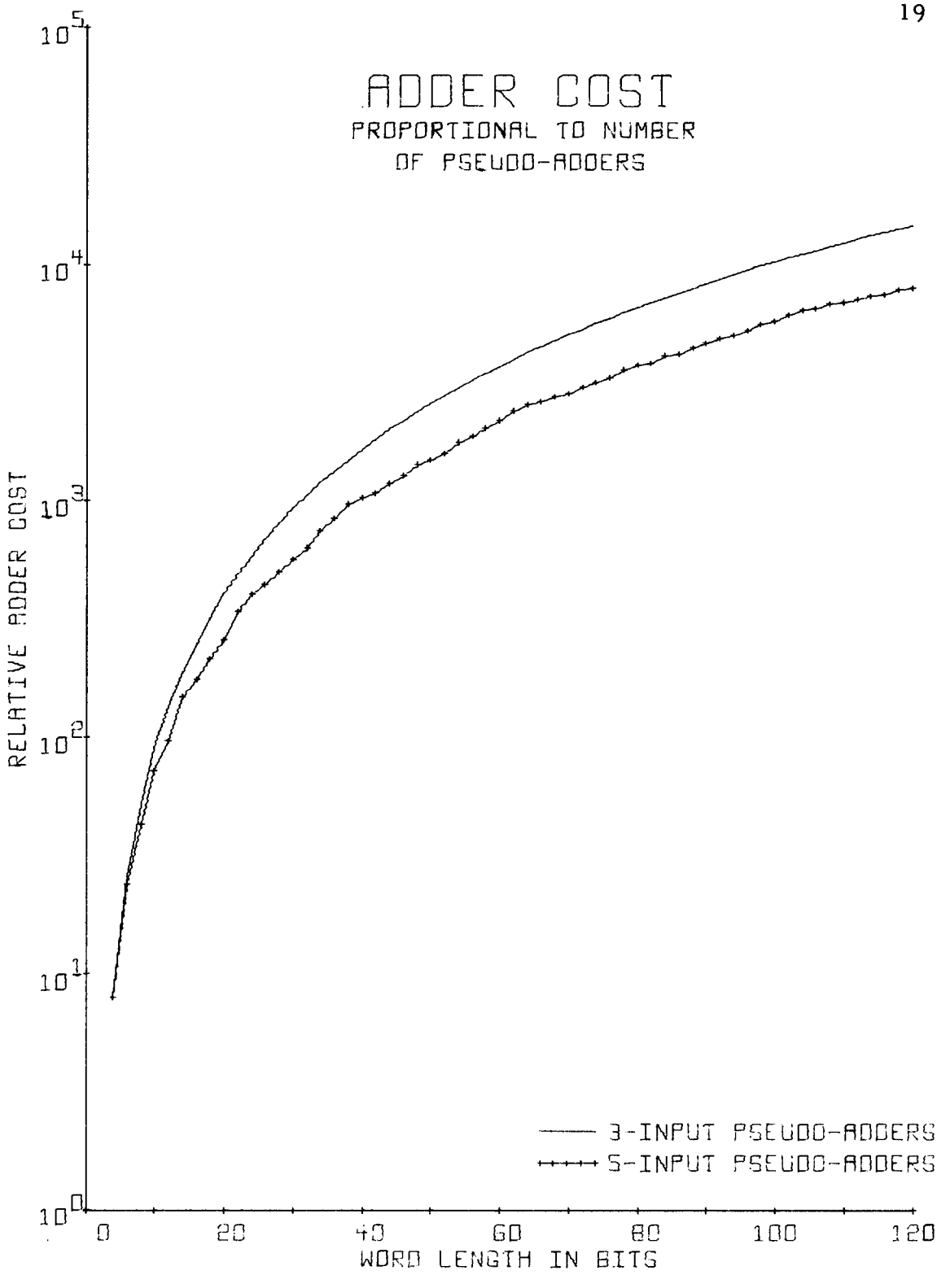


FIGURE 8. RELATIVE COST OF ADDER TREE USING THREE- AND FIVE-INPUT PSEUDO-ADDERS

Having found the cost and the propagation time, it was then possible to find the minimum cost-time product and the associated pseudo-adder type--whether three, five, seven, nine, eleven, or thirteen inputs. This is shown in Figure 9. Note that, generally, as the word length increased, the number of pseudo-adder inputs became larger. The obvious discreteness here is due to the discrete jumps made by the propagation time and the cost.

For a different type of analysis, a fixed word length multiplier was chosen, and the propagation time, adder cost, and cost-time product were computed for various types of pseudo-adders. This analysis was performed for both a forty- and an eighty-bit word length multiplier. The graphs of propagation time clearly show the increase in propagation time as the number of pseudo-adder inputs was increased (see Figures 10 and 13). The adder cost showed a decrease in cost as the number of inputs was increased (see Figures 11 and 14). Here again, it is plain to see that as the number of inputs gets large, the gain in cost reduction is minimal. It is apparent that speed and cost are opposing factors--the fastest multiplier is the most expensive, and the cheapest is the slowest.

The graphs of the cost-time product indicate that there could be more than one choice of an optimum pseudo-adder type (see Figures 12 and 15). For example, for the forty-bit multiplier, an eleven-input pseudo-adder gives the minimum cost-time product.

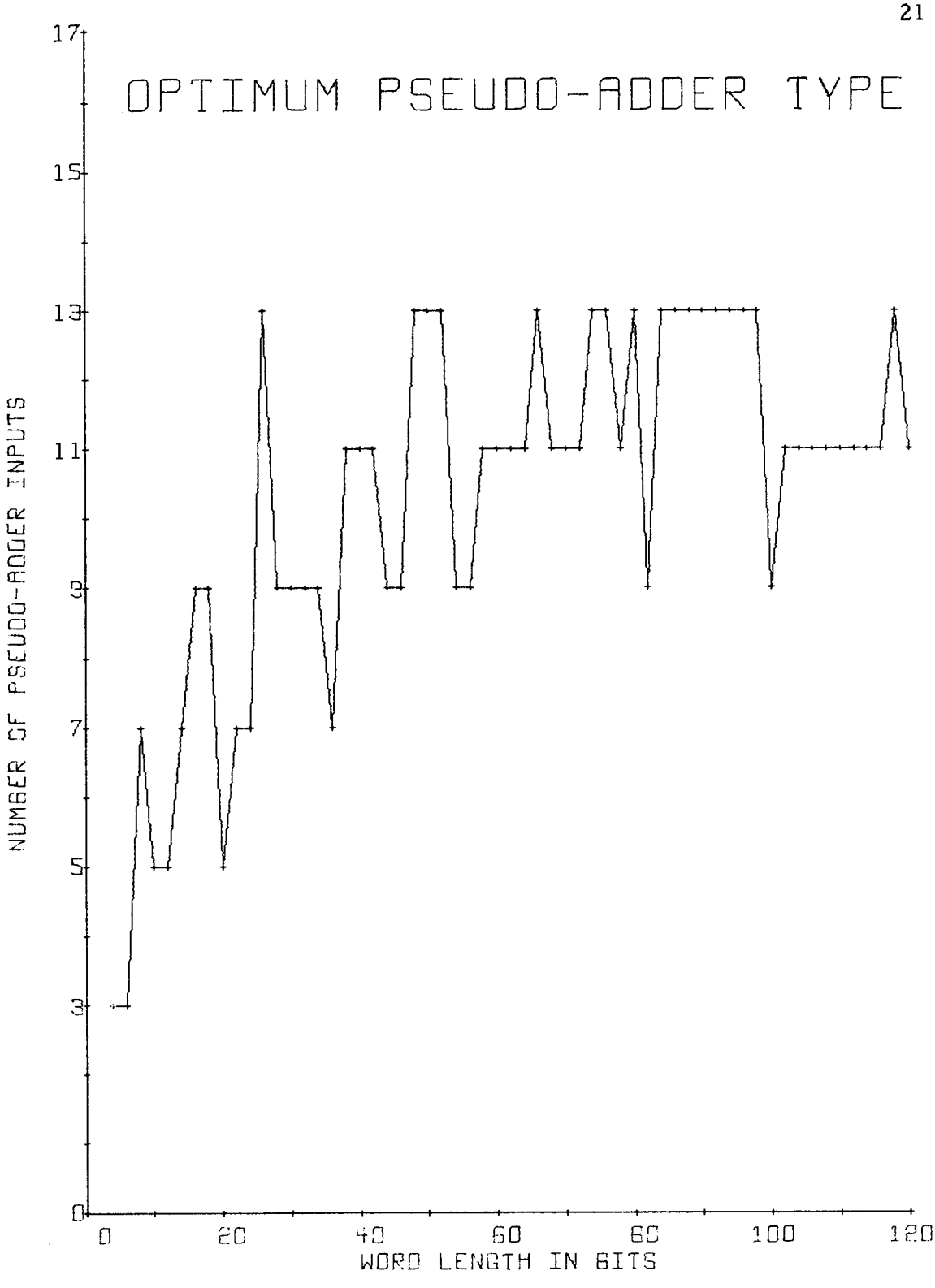


FIGURE 9. RELATION SHOWING THE NUMBER OF PSEUDO-ADDER INPUTS THAT GIVE THE OPTIMUM MULTIPLIER FOR VARIOUS WORD LENGTHS

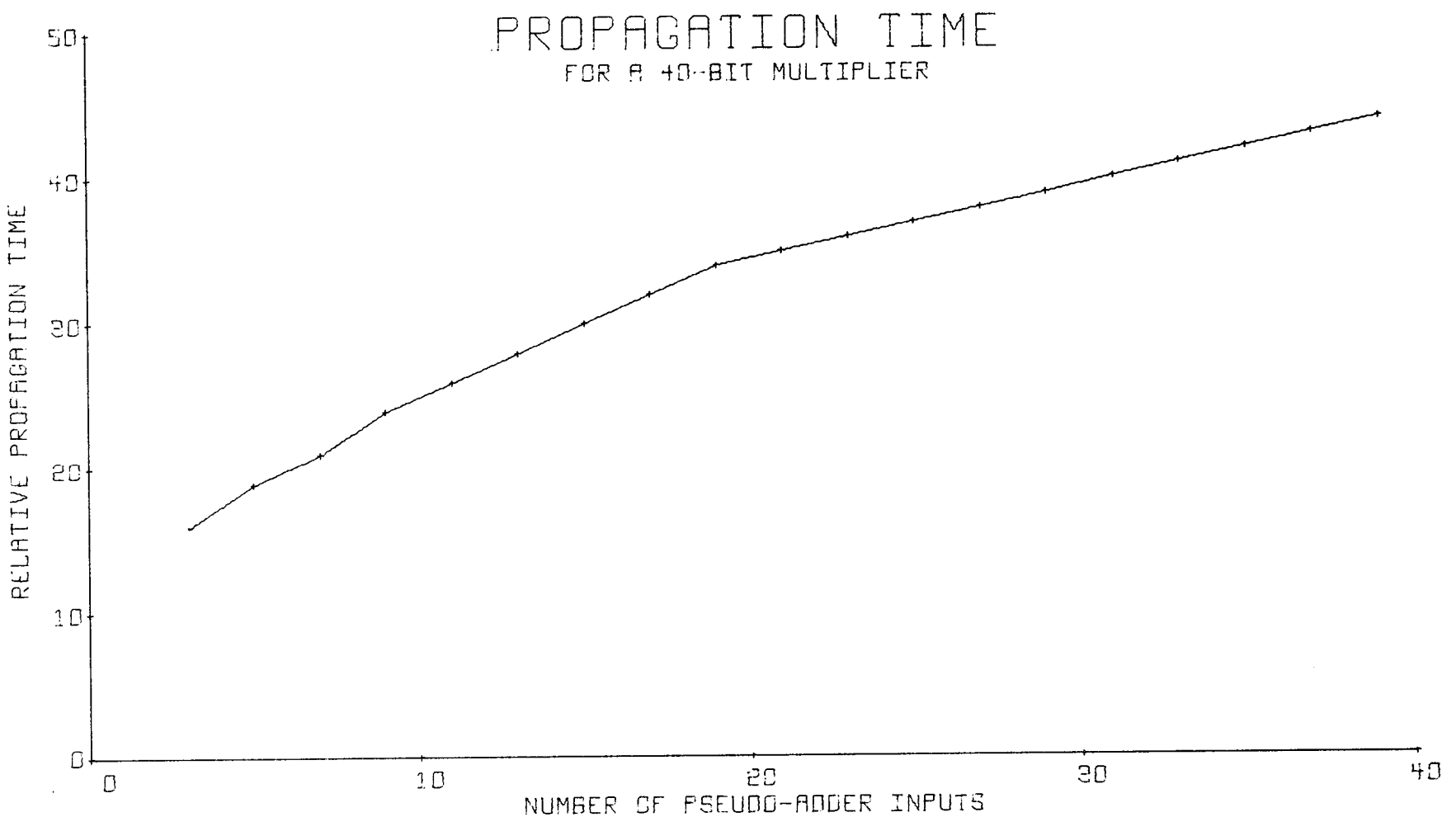


FIGURE 10. RELATIVE PROPAGATION TIME FOR A FORTY-BIT MULTIPLIER AS A FUNCTION OF PSEUDO-ADDER INPUTS

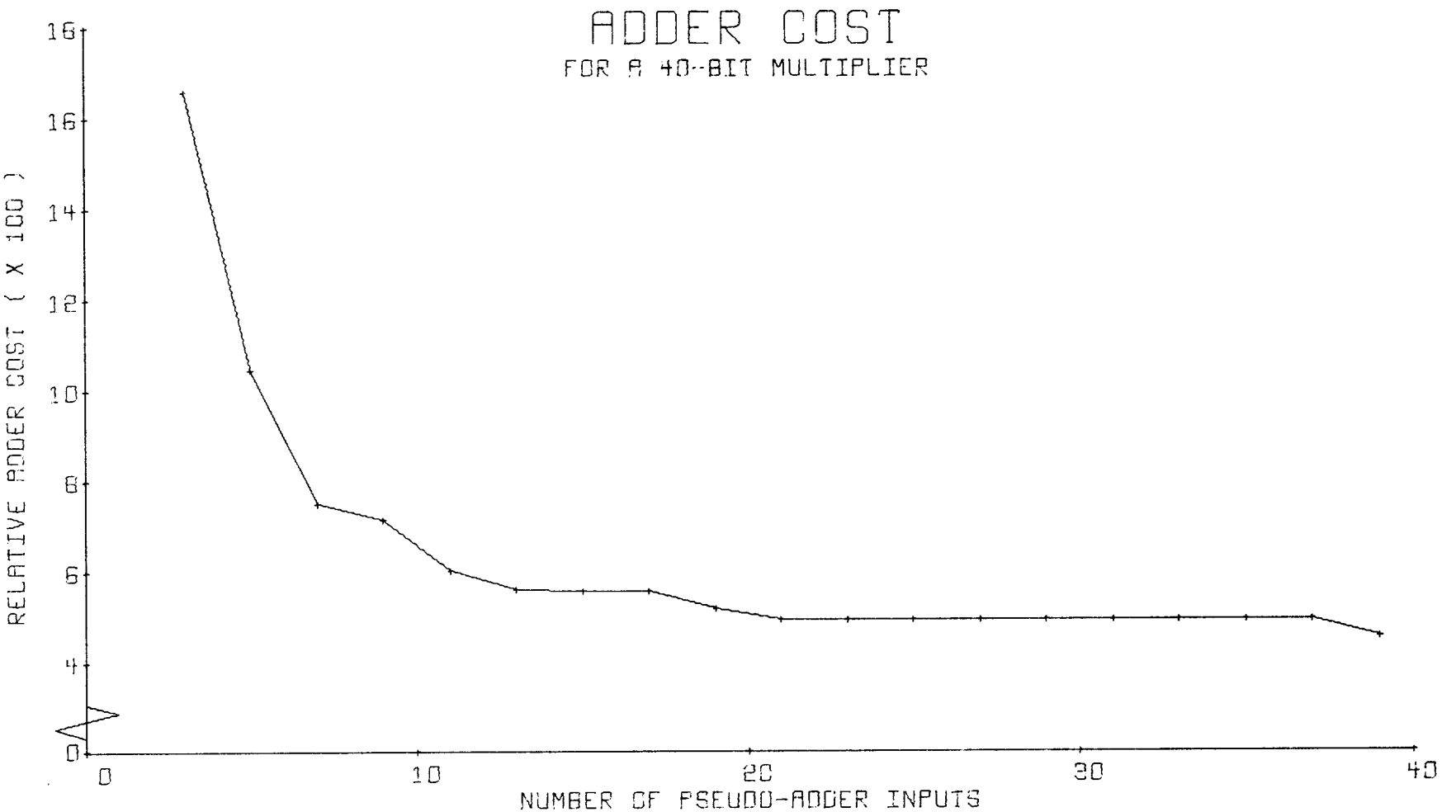


FIGURE 11. RELATIVE ADDER-TREE COST FOR A FORTY-BIT MULTIPLIER AS A FUNCTION OF PSEUDO-ADDER INPUTS

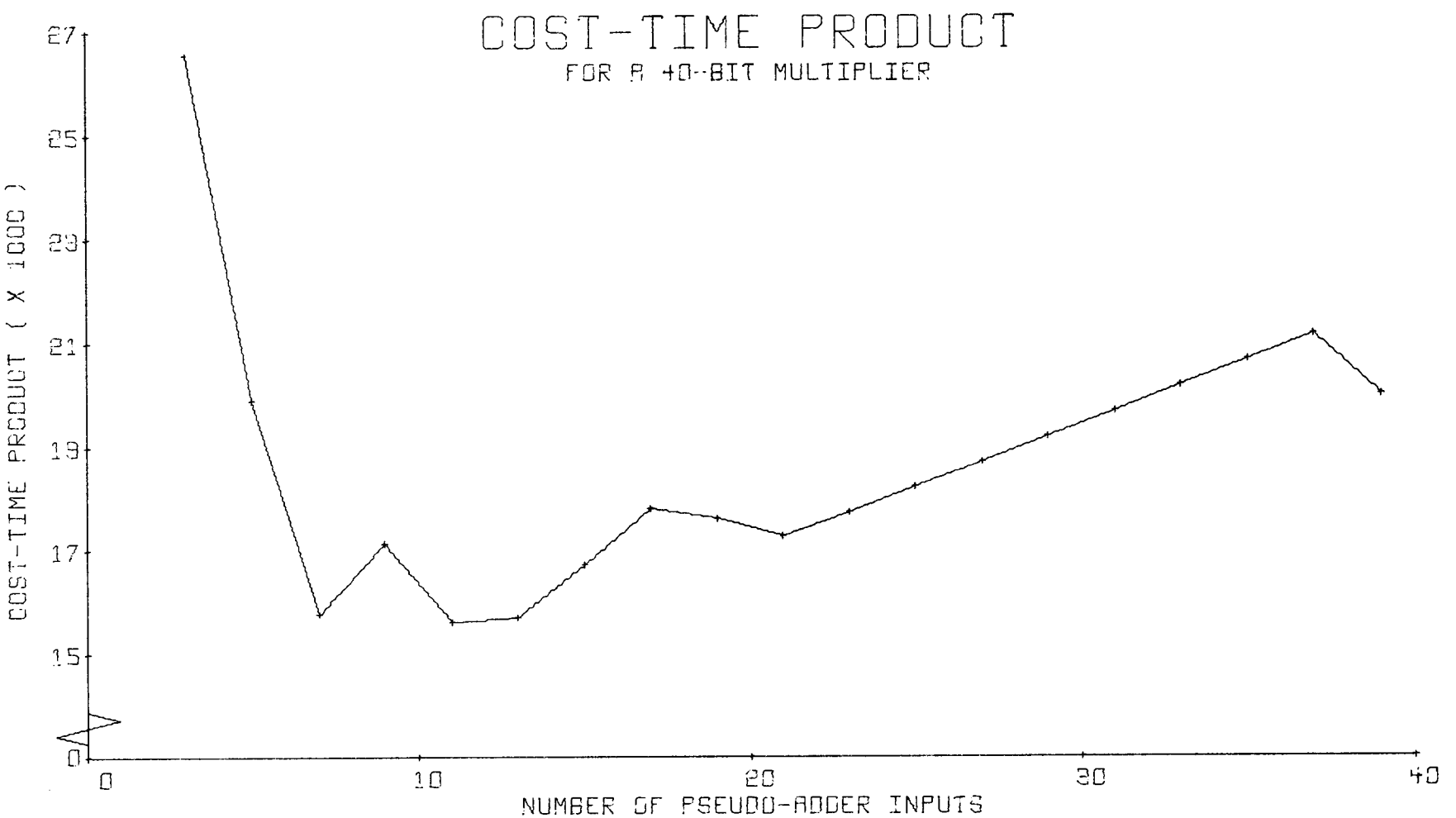


FIGURE 12. COST-TIME PRODUCT FOR A FORTY-BIT MULTIPLIER AS A FUNCTION OF PSEUDO-ADDER INPUTS

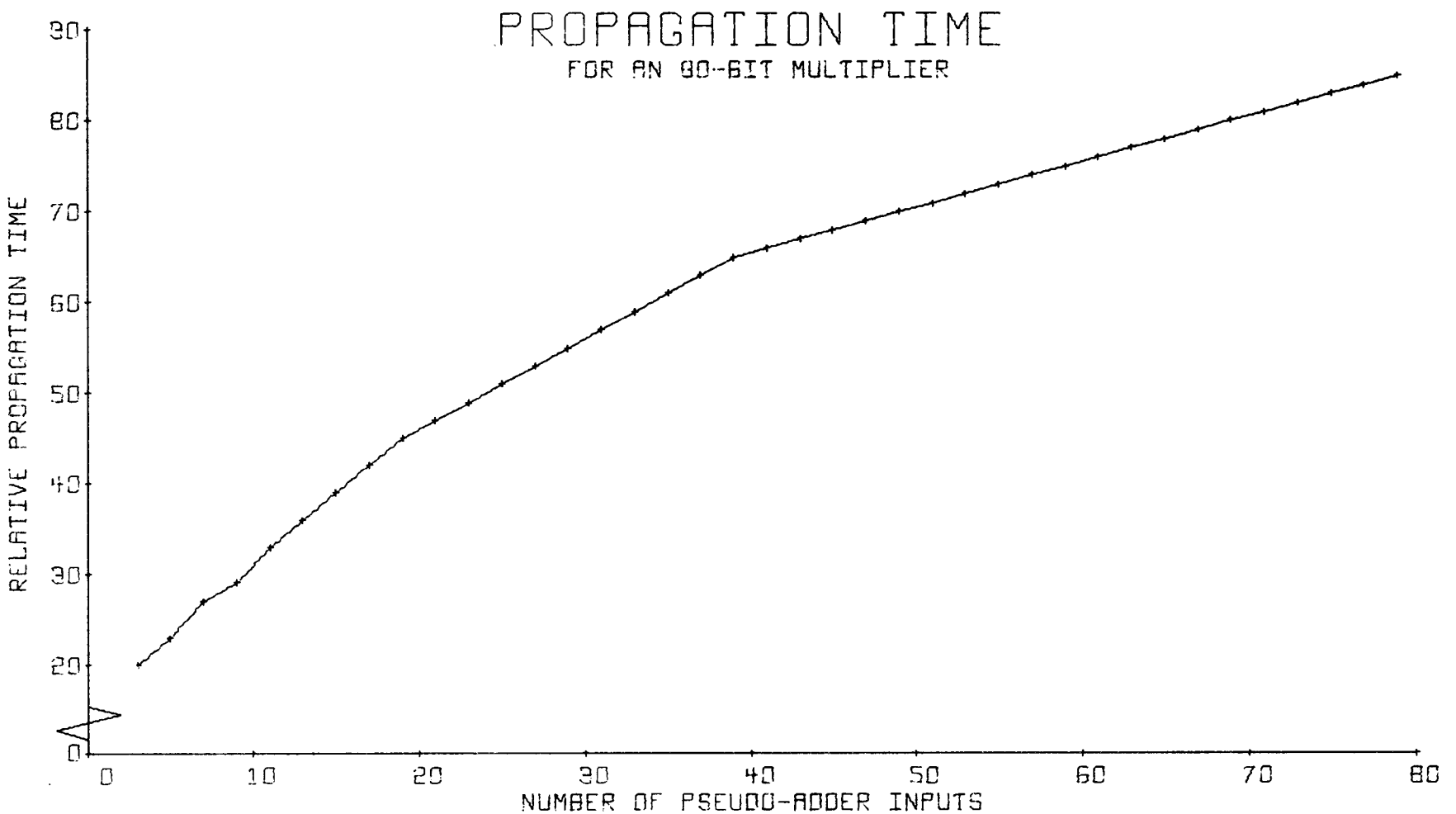


FIGURE 13. RELATIVE PROPAGATION TIME FOR AN EIGHTY-BIT MULTIPLIER AS A FUNCTION OF PSEUDO-ADDER INPUTS

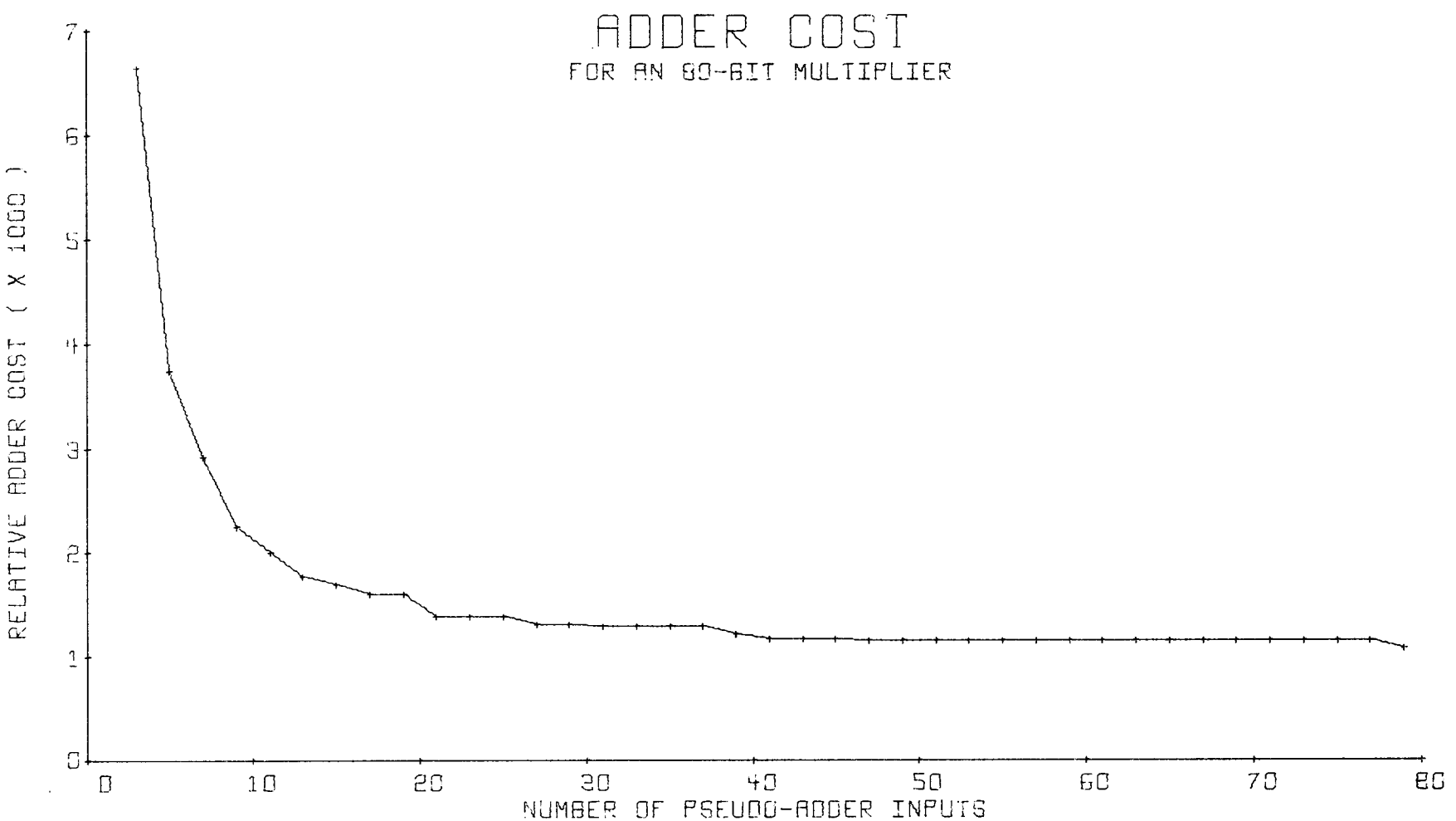


FIGURE 14. RELATIVE ADDER-TREE COST FOR AN EIGHTY-BIT MULTIPLIER AS A FUNCTION OF PSEUDO-ADDER INPUTS

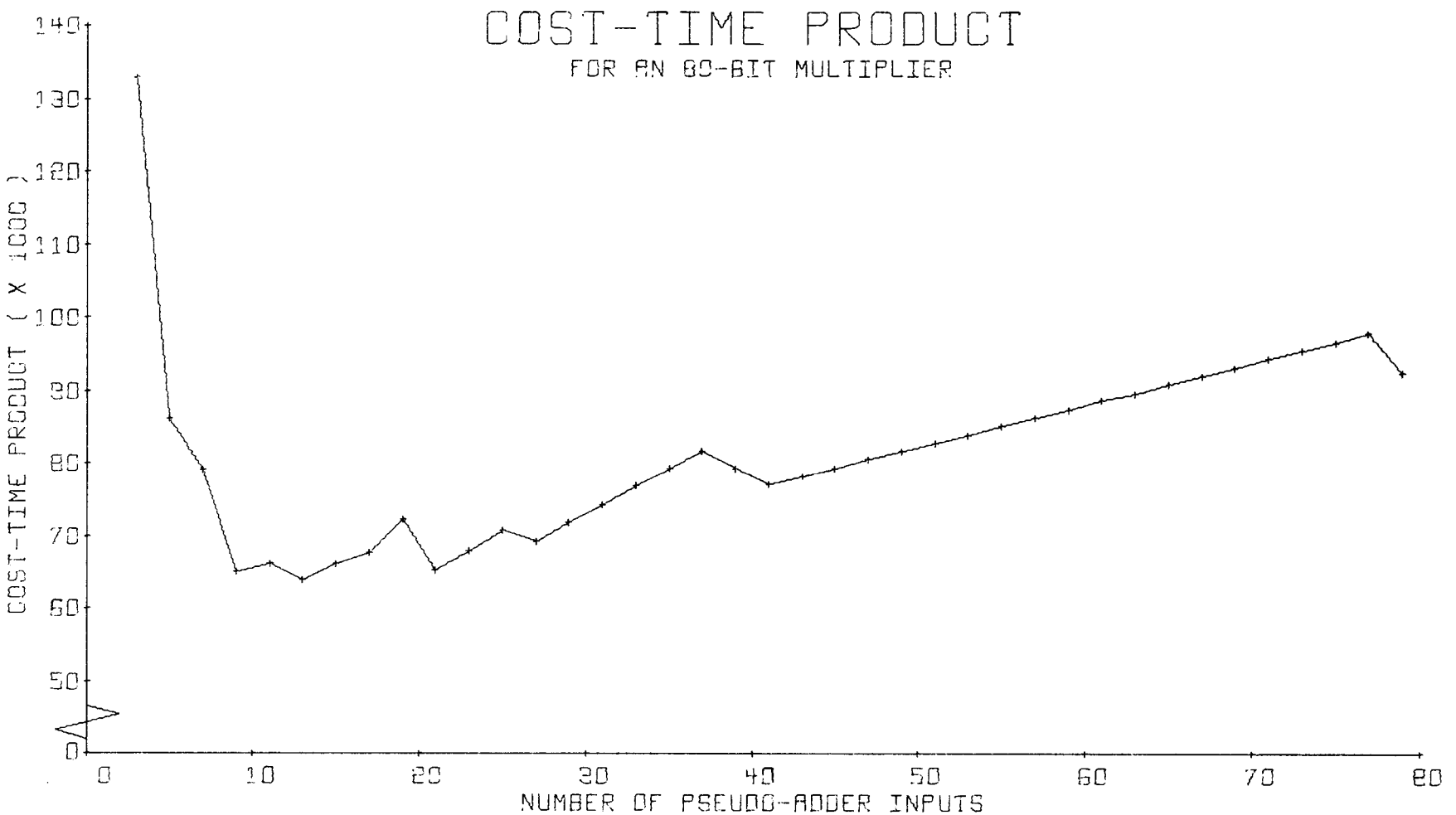


FIGURE 15. COST-TIME PRODUCT FOR AN EIGHTY-BIT MULTIPLIER AS A FUNCTION OF PSEUDO-ADDER INPUTS

However, by paying a little more money for a faster multiplier, seven-input pseudo-adders could be chosen with little sacrifice in optimization. Likewise, by sacrificing a little speed for a cost savings, thirteen-input pseudo-adders would give nearly optimum performance.

FORMATION OF SUMMANDS AND CARRY PROPAGATING ADDER

Besides the pseudo-adder tree, a complete multiplier requires two more parts, a unit to form the summands from the multiplier and multiplicand, and a carry propagating adder to form the product from the two outputs from the adder tree.

Formation of the summands, which must be done quickly, can be accomplished by using an array of AND gates as shown in Figure 16. Using this method, the summands are formed simultaneously, with the propagation time due to one level of AND gates.

The carry propagating adder is one of the most important considerations in the design of a parallel multiplier. If the adder is not extremely fast, most of the multiply time may be used in the addition of the final two summands.

Fast carry propagating adders have been suggested by Gilchrist (2), Kilburn (4), and Salter (8). However, none of these were asynchronous. Bernhardt (1) has pointed out that an asynchronous adder has a speed advantage over most fixed time adders, especially for longer word lengths. (Recall that in the multiplier, the word length of the adder must be twice as long as the word length of the multiplier.) As Reitweisner (7) has shown, the average maximum carry length resulting from the addition of two randomly

Multiplicand $X_4X_3X_2X_1$

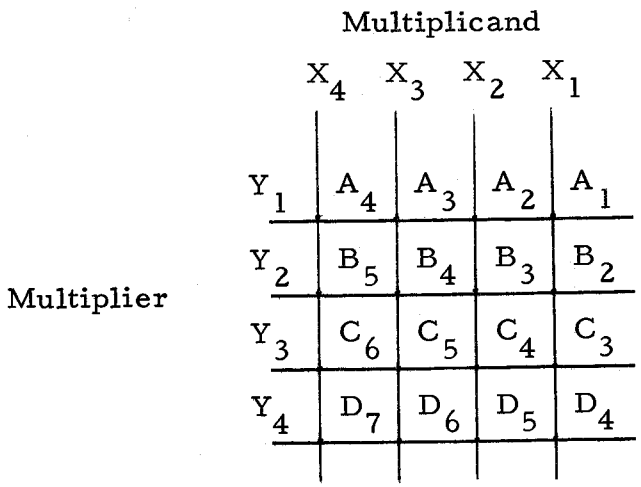
Multiplier $Y_4Y_3Y_2Y_1$

$A_4A_3A_2A_1$

$B_5B_4B_3B_2$

$C_6C_5C_4C_3$

$D_7D_6D_5D_4$



At each intersection there is an AND gate.

$$A_1 = X_1Y_1$$
$$A_2 = X_2Y_1$$
$$A_3 = X_3Y_1$$
etc.

Figure 16. Arrangement of AND Gate Matrix to Form Summands

arranged operands can be approximated by $Lavg = Log_2(5N/4)$ where N is the number of bits in the operands and $Lavg$ is the average length of the carry in bit positions. $Lavg$ for a fifty-bit adder is approximately six, while the maximum delay would be fifty. Considering a one hundred-bit adder, $Lavg$ would be approximately seven and the maximum carry length would be one hundred. Thus, the larger the adder, the more time can be saved, on the average, by making the adder asynchronous.

This comparison, of course, is between asynchronous and ordinary ripple carry adders. There are other methods of speeding up the addition operation. MacSorely (6) compares several of these methods. He illustrates a fixed time adder that is nearly as fast as the average asynchronous operation. This adder, which utilizes full carry look-ahead, is broken into groups of five bits each. Within each group carry look-ahead logic is employed. Rather than let the carry propagate successively through each group, the groups are divided into sections--five groups to a section. Carry look-ahead logic is then employed among the groups in a section, and between sections, carry speed-up logic is used. The result is an adder that is approximately seven times faster than a ripple carry adder, assuming the length to be one hundred bits.

The full carry look-ahead adder and an asynchronous double rail completion recognition adder (described by both MacSorely (6)

and Bernhardt (1)) have approximately the same complexity, hence cost the same. Which type adder to use, then, appears to be the designer's choice.

THE MULTIPLIER SYSTEM

The three major parts of the multiplier are now tied together into a complete system. Referring to Figure 17, operation of the multiplier is started when the multiplier and multiplicand are gated into the AND matrix. Here, all of the summands are formed and immediately enter the pseudo-adder tree. After allowing a sufficient amount of time for the signals to propagate through the pseudo-adder tree, the two outputs from it are gated into the carry propagating adder. At the completion of the addition, the adder contains the product.

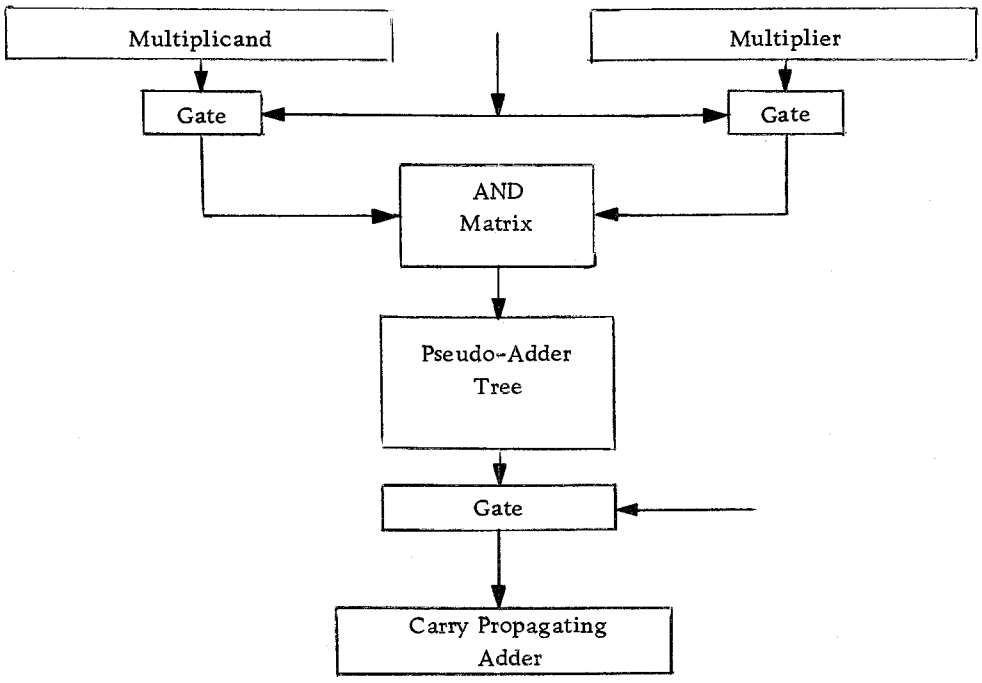


Figure 17. Complete Multiplier System

SUMMARY

Recognizing the need for faster operating speeds in computer systems, the advantage of using a multiplication process faster than the "conventional" shift-add method is obvious. Of the several methods proposed, the one suggested by Wallace (10) appears to be the fastest. This method used a tree of full adders to add in parallel the n summands of an n bit multiplier. By using threshold logic it is possible to design logic blocks, termed pseudo-adders, which are more powerful than full adders. By using different types of pseudo-adders, it is possible to design a pseudo-adder tree with different characteristics. It was demonstrated that the fastest multiplier would be realized with three-input pseudo-adders. Also, the cheapest realization was possible by using pseudo-adders with as many inputs as possible. Somewhere between these two extremes is the optimum realization, or the one which gives the fastest operation for the least cost. The type of pseudo-adder that gives the optimum multiplier was found for a forty- and an eighty-bit multiplier, and could easily be found for any other length multiplier.

The summands could be formed simultaneously from the multiplicand and multiplier by a matrix of AND gates. Concerning the carry propagating adder, which forms the final sum from the output of the pseudo-adder tree, a definite requirement of speed

was established. The carry propagating adder, if not extremely fast, will be the determining factor in how quickly the product can be formed.

By assuming some propagation times for the different elements in the multiplier, it should be possible to compute a multiply time for an eighty-bit multiplier. Assume a delay time through the AND gate matrix and through a threshold gate of 5 ns. For this multiplier assume that the optimum pseudo-adder (thirteen inputs) is used (recall that the fastest multiplier would use three-input pseudo-adders). Using this type pseudo-adder, there are thirty-three threshold gates in the longest path through the pseudo-adder tree. The propagation time through the AND gate matrix and the pseudo-adder tree requires 170 ns. Assuming a maximum add time in the carry propagating adder of 250 ns gives a total of 420 ns. Thus, it seems reasonable to predict that an eighty-bit multiplier with a multiply time of 500 ns is possible using thirteen-input pseudo-adders.

BIBLIOGRAPHY

1. Bernhardt, D. E. A fast carry binary adder. Master's thesis. Corvallis, Oregon State University, 1964. 57 numb. leaves.

2. Gilchrist, B., J. H. Pomerene and S. Y. Wong. Fast carry logic for digital computers. Institute of Radio Engineers Transactions on Electronic Computers EC-4:133-136. 1955.

3. Green, Abraham. Binary multiplication in digital computers. Proceedings of the Institute of Radio Engineers 47:1159-1160. 1959.

4. Kilburn, T., D. B. G. Edwards and D. Aspinall. A parallel arithmetic unit using a saturated-transistor fast-carry circuit. The Proceedings of the Institution of Electrical Engineers 107-B:573-584. 1960.

5. Lehman, M. Short-cut multiplication and division in automatic binary digital computers. Proceedings of the Institution of Electrical Engineers 105-B:496-504. 1958.

6. MacSorley, O. L. High speed arithmetic in binary computers. Proceedings of the Institute of Radio Engineers 49:67-91. 1961.

7. Reitweisner, G. W. The determination of carry propagation length for binary addition. Institute of Radio Engineers Transactions on Electronic Computers EC-9:35-38. 1960.

8. Salter, Forrest. High speed transistorized adder for a digital computer. Institute of Radio Engineers Transactions on Electronic Computers EC-9:461-464. 1960.

9. Sheng, C. L. A graphical interpretation of realization of symmetric Boolean functions with threshold logic elements. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers EC-14:8-18. 1965.

10. Wallace, C. S. A suggestion for a fast multiplier. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers EC-13:14-17. 1964.