CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

Canada

Efficient Recursive Arithmetic Algorithms for VLSI

Régis Cardin

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the requirements
for the degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

May, 1986

ISBN 0-315-35554-9

ABSTRACT

Efficient Recursive Arithmetic Algorithm for VLSI

Régis Cardin

An iterative cellular parallel multiplier based on multiplexers is proposed in Chapter 1. The structure accepts two N-bits two's complement numbers and gives the result in two's complement form. The structure can also be pipelined with a period complexity of O(1). The area complexity of the structure is $O(N^2)$. The proposed structure can be implemented to perform 4 single precision multiplications or 1 double precision multiplication. The multiplier is implemented with an array of equal macrocells and is suitable for automatic design and testing. The time complexity is O(N), but macrocells are designed in such a way that variables carrying a long delay propagate faster than other variables allowing to achieve practical advantages.

A new recursive algorithm for deriving the layout of parallel multipliers is proposed in Chapter 2. Based on this algorithm, a network for performing multiplication of two's complement numbers is proposed. The network can be implemented in a synchronous or an asynchronous way. If the factors to be multiplied have N bits, the area complexity of the network is $O(N^2)$ for practical value of N as in the case of cellular multipliers. Due to

the design approach based on a recursive algorithm, a time complexity O(log N) is achieved.

It is shown how the structure can be pipelined with a period complexity O(1) and used for single or double precision multiplication.

A combinational circuit for computing the binary logarithm of an N bits number is proposed in Chapter 3. The structure has a time complexity O(log N) and an area complexity $O(N^2)$. The same structure can be used for computing the antilogarithm of a number thus allowing to obtain the quotient of two binary numbers in O(log N) time complexity and $O(N^2)$ area complexity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF TABLES

INTRODUCTION

This thesis is divided into three chapters, two of them are concerned with multipliers and the other one with logarithm number generation. The two multipliers that will be presented are different in their conception even if they share some points in common.

Multipliers are fundamental components of computer arithmetic units and signal processing systems. In the last thirty years, the design of parallel multipliers has received considerable attention.

A fundamental contribution to the design of combinational or simultaneous multipliers has been given by Wallace [WAL64]. He proposed a network of Carry Save Adder (CSA) for adding Partial Product (PP) generated by two integer factors represented in binary code with N bits each and obtaining in O(log N) time complexity two addends whose sum is equal to the product of the factors. A pipelined version of this design has been implemented in commercially available machines.

Wallace's design was improved by Dadda [DAD76] who proposed to use Parallel Counters (PC) instead of CSA in order to reduce the cost.

In the late 1960s, a number of multiplier designs were proposed based on iterative arrays of equal cells [BUR70, DEA71, DEM69a, GUI69, HAB69, HOF68]. Those structures have an area complexity $O(N^2)$ and a time complexity O(N). The basic cell of the iterative multipliers was a gated full-adder. ***Cellularity*** was considered an advantage for Large Scale Integretion even throught the time complexity of cellular structures was O(N) rather than O(log N). In fact, their layout can be automatically generated by iteratively reproducing the layout of a single cell on a plane. Using the same cells, networks for multiplying signed numbers were proposed [BAN73, BAU73, DEE71, DEM72, MAJ71] having the same area and time complexity.

In order to increase the speed of the iterative multipliers having time complexity O(N), some macrocellular structures have also been proposed [DEA71, DEM69a, DEM75, KIN71, SPR70].

An extended review with interesting comments and contributions to the design of parallel multipliers can be found in the book of Hwang [HWA79].

Chapter 1 of this thesis proposes a new solution for macrocellar multipliers. A new macrocell design based on multiplexers will be proposed which allows to make the maximum delay of the multiplier array proportional to a ***fraction*** of N (the number of bits of the factors).

It is well known that cellular structures have big advantages from the point of view of automatic design, manufacturing and testing. These advantages can compensate the disadvantage of having a time complexity O(N) especially if N is not very large and if the technology based on Gallium Arsenide is used for implementation, since Gallium Arsenide requires simple design.

The structure that will be presented in Chapter 1 can be adapted to multiply two two's complement numbers with the same performances using the algorithm proposed by Baugh and Wooley [BAU73]. The design that will be proposed is based on modules whose connection can be easily programmmed in such a way that either four single precision multiplications or one double precision multiplication can be performed under the control of one specification variable to indicate single or double precision operation.

Cappello and Steiglitz [CAP83] propose a VLSI layout for parallel multipliers with an area complexity $A = O(N^2 \log N)$ and a time complexity $T = O(\log N)$. Cappello and Steiglitz compare the existing solutions on the basis of a VLSI figure of merit defined as follows:

$$FM_a = AT^2(PE)^2 \qquad (1)$$

where A is the area complexity, T is the time complexity and PE is the period complexity. Period complexity refers to the

number of clock pulses that have to be given to the circuit between the output of two successive results.

It should be noted that some other VLSI Figures of Merit have also been proposed:

$$FM_b = A(PE) \qquad (2)$$
$$FM_c = A(PE)T$$

Cappello and Steiglitz have shown that their multipliers can be pipelined with a period complexity PE = O(1) corresponding to a figure of merit:

$$FM_a = N^2\log^3 N$$
$$FM_b = N^2\log N \qquad (3)$$
$$FM_c = N^2\log^2 N$$

They also derived a Lower Bound Figure of Merit of (1) for parallel multipliers (LBFM): based on an area complexity of $O(N^2)$, a time complexity of O(log N) and a period complexity of O(1):

$$LBFM_a = N^2\log^2 N$$
$$LBFM_b = N^2 \qquad (4)$$
$$LBFM_c = N^2\log N$$

and reported that this lower bound were not reached by any solution published before their paper.

A new recursive algorithm for the layout generation of parallel multipliers will be presented in Chapter 2. The area complexity of multipliers obtained by the use of such algorithm is $O(N^2)$, for values of N not exceeding a threeshold greater than 100 (this will be justified in Chapter 2), while the time complexity is O(logN). The implementation of such a scheme requires essentially two types of cells: carry save adders and multiplexers. Positive as well as two's complement numbers can be handled. The structure can be pipelined with a period complexity O(1) thus reaching the lower bound (4) of FM as defined by (1) and (2).

Chapter 1 and 2 propose two solutions for multipliers. Chapter 1 proposes a structure that is simple and fast for small value of N. Chapter 2 mostly proposes an optimal multiplier from the point of view of operation speed, but for small values of N, the solution of Chapter 1 is faster and easier to implement.

Chapter 3 is concerned with the fast calculation of the base-two logarithm of a binary number which has been the object of investigation in the past twenty years. Recently, the advent of Very Large Scale Integration (VLSI) has attracted new interest in the conception of algorithms for numerical computation suitable to be integrated into microcircuit. Integration suggests new criteria for evaluating the merits of an algorithm. These criteria are based not only on ***time complexity*** whose optimality was

already a goal in early designs, but also on ***area complexity*** and ***period complexity***.

Almost all the solutions proposed for binary logarithm computation are based on piece-wise approximations.

Mitchell [MIT62] was one of the first to consider the logarithm approximation. He used a simple linear approximation with one segment in the interval [1,2] for computing the mantissa of the logarithm:

$$\log_2 x \approx x-1 \qquad (5)$$

A few years later, Combat et al. [COM65] have proposed piece-wise linear approximation of the logarithmic curve with two segments instead of one as proposed by Mitchell.

Marino [MAR72] proposed to use second order polynomial approximations with two segments. Sequential (serial) circuits were proposed for implementing the above methods. In many applications such as signal processing, it is important to compute a large number of logarithms in a short interval of time. This makes it attractive to consider purely combinational solutions that can be integrated into a single chip using parallel multipliers.

A design is proposed in this thesis which is based on a polynomial approximation of $\log_2(1 + x)$ in the interval:

$0 < x < 1$

An antilogarithm circuit can be built based on the design that will be presented in Chapter 3.

A binary division circuit can be obtained by computing the logarithms of the magnitudes of the dividend and the divisor. The logarithm of the quotient is obtained by subtracting the logarithm of the divisor from the logarithm of the dividend. The magnitude of the quotient can be obtained by computing an antilogarithm.

CHAPTER 1

ITERATIVE PARALLEL MULTIPLIERS BASED ON MULTIPLEXERS

INTRODUCTION

This chapter proposes a new solution for macrocellular multipliers. A macrocell design based on multiplexers is proposed which allows to make the maximum delay of the multiplier array proportional to a ***fraction*** of N (the number of bits of the factors).

It is well known that cellular structures have big advantages from the point of view of automatic design, manufacturing and testing. These advantages can compensate the disadvantage of having a time complexity O(N) especially if N is not very large and if a technology based on Gallium Arsenide is used for implementation.

The structure can be adapted to multiply two two's complement numbers with the same performances using the algorithm proposed by Baugh and Wooley [BAU73]. The design proposed is based on moduli whose connection can be easily programmmed in such a way that either four single precision multiplications or one double precision multiplication can be performed under the control of one specification variable.

THE ALGORITHM

In order to introduce the algorithm for designing multipliers with multiplexers, an iterative algorithm denoted Algorithm IM for multiplying positive binary integer numbers will first be presented. This algorithm leads to the implementation of cellular multipliers with time complexity O(N).

Algorithm IM (Iterative Multiplication)

Let

$$H = \sum_{i=0}^{N-1} h_i 2^i$$

and

$$K = \sum_{j=0}^{N-1} k_j 2^j$$

be the factors and

$$P = H * K = \sum_{r=0}^{2N-1} g_r 2^r \qquad (1-1)$$

be the product. h_i, k_j, g_r are binary variables.

Let N = LG with L and G integers.

Wording the factors H and K in the basis 2^G one gets:

$$H = \sum_{m=0}^{L-1} h_m 2^{mG} ,$$

$$K = \sum_{n=0}^{L-1} k_n 2^{nG} , \qquad (1\text{-}2)$$

with h_m and k_n integers less then 2^G and expressible in the binary code with G bits. The product P can be expressed by the following relation:

$$P = h_0k_0 + (h_1k_0 + k_1h_0)2^G + \ldots + h_{L-1}k_{L-1}2^{2(L-1)G} \qquad (1\text{-}3)$$

Let us consider the term h_0k_0; it can be written in the following manner:

$$h_0k_0 = 2^G U_{11} + V_{01}$$

where $U_{11} < 2^G$; $V_{01} < 2^G$. U_{11} and V_{01} are related to the product P in the following way:

$$V_{01} = \sum_{r=0}^{G-1} g_r 2^r ,$$

(1-4)

$$\sum_{r=0}^{G-1} g_r + G2^r = \left| H_1K_0 + H_0K_1 + U_{11} \right|_{2^G}$$

Where $|\quad|_{2G}$ indicates a modulo 2^G sum. The right-hand side of (1-4) can be developped as follows:

$$h_1k_0 = 2^GU_{21} + V_{11}$$
$$V_{11} + h_0k_1 + U_{11} = 2^GU_{22} + V_{12} \qquad (1\text{-}5)$$

$$\sum_{r=0}^{G-1} g_r + G2^r = V_{12}$$

Expression (1-2) can be computed by recursively applying the following relation:

$$V_{jf} + h_mk_n + U_{iw} = 2^GU_{kc} + V_{hd} \qquad (1\text{-}6)$$

until all the product bits are generated. Expression (1-2) can be computed using the structure shown in Figure 1-1 with L = 3. In this case the product P is obtained as follows:

$$P = V_{01} + V_{12}2^G + V_{23}2^{2G} + V_{33}2^{3G} + V_{43}2^{4G} + U_{53}2^{5G}$$

Notice that the structure shown in Figure 1-1 can be used for adding two numbers:

$$X = \sum_{a=0}^{N-1} x_a ,$$

$$Y = \sum_{b=0}^{N-1} y_b$$

to the product P=H*K, making the structure capable of performing the operation :

$$Z = H*K + X + Y, \qquad (1\text{-}7)$$

which is formally identical to the expression that has to be computed by expression (1-6). Dean [DEA71] has called such a structure, a ***full-multiplier***.

CELL DESIGN

Cell design is the main novelty of the solution proposed in this chapter of my thesis. The main idea of the design is based on the fact that a macrocell has some input variables (the Factors Bits : FB) that are all generated simultaneously and other input variables (the Additive Inputs : AI) which have to propagate throught the circuit and have a delay depending on the location of the macrocell that receives them.

The outputs of each macrocell depend both on FBs and AIs. A macrocell can be implemented using a VEM (Variable Entry Map) technique (see [FLE80] for details). With such an approach each output is generated by a multiplexer having AIs as address bits and suitable functions of the FBs at the inputs. The functions of the FBs can be generated in the most economic way because their delay contributes with a constant factor (independant from N) to the total delay of the multiplier. A design example for cell with G = 1 and G = 2 are reported in the following.

Cell design with G = 1

Let x_0, y_0 be the AIs and h_0k_0 be the FBs. The term x_0, y_0 and h_0k_0 have weight 2^0. All those terms have to be added together in order to give v_0 and u_0 where v_0 has weight 2^0 and u_0 has weight 2^1.

Let us assume that v_0 and u_0 are outputs of two 4-input multiplexer whose address selection is based on the following rule:

$$y_0 2^1 + x_0 2^0$$

Let $I_j v_0$ be the j^{th} input function for the multiplexer that produces v_0 at the output. Let $I_j u_0$ be the j^{th} input function that produces u_0 at the output ($0 \leqq j \leqq 3$). The input functions for each multiplexer can be derived using the following general equation:

$$u_0 2^1 + v_0 2^0 = h_0 k_0 + x_0 + y_0$$

The details of the algorithm performed by a cell is shown in Table 1-1. The 4 input functions for each multiplexer are the following:

$$I_0 v_0 = h_0 k_0$$

$$I_0 u_0 = 0$$

$$I_1 v_0 = \overline{h_0 k_0}$$

$$I_1 u_0 = h_0 k_0$$

$$I_2 v_0 = \overline{h_0 k_0}$$

$$I_2 u_0 = h_0 k_0$$

$$I_3 v_0 = h_0 k_0$$

$$I_3 u_0 = 1$$

Figure 1-2 shows the VEM maps for the two multiplexers and Figure 1-3 shows the logical structure of the cell.

If τ is the delay of a multiplexer, then the total delay T of a multiplier of N bit numbers implemented with cells with G = 1 is:

$$T = (2N-1)\tau + \alpha$$

where α is the delay of an AND gate.

Cell design with G = 2

Let x_0, x_1, y_0, y_1, be the AIs and h_0, h_1, k_0 , k_1 be the FBs. The term x_0, y_0, h_0k_0 have weight 2^0, and the term x_1, y_1, h_0k_1 and h_1k_0 have weight 2^1. The term h_1k_1 has weight 2^2. All these terms have to be added together producing four output variables v_0, v_1, u_0 and u_1. The term v_0 has weight 2^0, v_1 has weight 2^1, u_0 has weight 2^2 and u_1 has weight 2^3.

Let assume that v_0, v_1, u_0 and u_1 are the outputs of four 16 input multiplexers in which the input is selected by the following rule:

$$y_1 2^3 + x_1 2^2 + y_0 2^1 + x_0 2^0$$

There are 16 input functions for each multiplexer.

Let $I_j v_k$ be the j^{th} input function for the multiplexer that produces v_k at the output. Let $I_j u_k$ be the j^{th} input function that produces u_k at the output $(0 \leq j \leq 15)$ and $(0 \leq k \leq 1)$. The input functions for each multiplexer can be derived using the following general equation:

$$2^3 u_1 + 2^2 u_0 + 2 v_1 + v_0$$
$$= (2h_1 + h_0)*(2k_1 + k_0) + 2x_1 + x_0 + 2y_1 + y_0$$

The 16 input functions for each multiplexer are the following:

$$I_0V_0 = h_0k_0$$

$$I_0V_1 = h_0k_1 \oplus h_1k_0$$

$$I_0U_0 = h_1k_1\overline{h_0k_0}$$

$$I_0U_1 = h_1h_0k_1k_0$$

$$I_1V_0 = \overline{h_0k_0}$$

$$I_1V_1 = h_0k_1 \oplus h_1k_0 \oplus h_0k_0$$

$$I_1U_0 = h_1k_1 \oplus [h_0k_0k_1 + h_1k_0h_0]$$

$$I_1U_1 = h_0h_1k_0k_1$$

$$I_2V_0 = I_1V_0$$

$$I_2V_1 = I_1V_1$$

$$I_2U_0 = I_1U_0$$

$$I_2U_1 = I_1U_1$$

$$I_3v_0 = h_0k_0$$

$$I_3v_1 = h_0k_1 \oplus \overline{h_1k_0}$$

$$I_3u_0 = h_1k_1 \oplus [h_0k_1 + h_1k_0]$$

$$I_3u_1 = h_1h_0k_1 + h_1k_1k_0$$

$$I_4v_0 = h_0k_0$$

$$I_4v_1 = h_0k_1 \oplus \overline{h_1k_0}$$

$$I_4u_0 = h_1k_1 \oplus [h_0k_1 + h_1k_0]$$

$$I_4u_1 = h_1k_1h_0 + h_1k_1k_0$$

$$I_5v_0 = \overline{h_0k_0}$$

$$I_5v_1 = h_0k_1 \oplus \overline{h_1k_0} \oplus h_0k_0$$

$$I_5u_0 = h_1k_1 + [\overline{k_1}h_0k_0 + h_1h_0\overline{k_0} + h_0k_1\overline{h_1} + h_0\overline{k_1}k_0]$$

$$I_5u_1 = h_1\overline{k_1}[\overline{h_0}k_0 + h_0\overline{k_0}]$$

$$I_6v_0 = I_5v_0$$

$$I_6v_1 = I_5v_1$$

$$l_6u_0 = l_5u_0$$

$$l_6u_1 = l_5u_1$$

$$l_7v_0 = h_0k_0$$

$$l_7v_1 = h_0k_1 \oplus h_1k_0$$

$$l_7u_0 = h_1k_1 \oplus h_0k_1h_1k_0$$

$$l_7u_1 = h_1k_1$$

$$l_8v_0 = l_4v_0$$

$$l_8v_1 = l_4v_1$$

$$l_8u_0 = l_4u_0$$

$$l_8u_1 = l_4u_1$$

$$l_9v_0 = l_5v_0$$

$$l_9v_1 = l_5v_1$$

$$l_9u_0 = l_5u_0$$

$$l_9 u_1 = l_5 u_1$$

$$l_{10} v_0 = l_5 v_0$$

$$l_{10} v_1 = l_5 v_1$$

$$l_{10} u_0 = l_5 u_0$$

$$l_{10} u_1 = l_5 u_1$$

$$l_{11} v_0 = l_7 v_0$$

$$l_{11} v_1 = l_7 v_1$$

$$l_{11} u_0 = l_7 u_0$$

$$l_{11} u_1 = l_7 u_1$$

$$l_{12} v_0 = h_0 k_0$$

$$l_{12} v_1 = h_0 k_1 \oplus h_1 k_0$$

$$l_{12} u_0 = h_1 k_1 \oplus \overline{h_0 k_1 h_1 k_0}$$

$$l_{12} u_1 = h_1 k_1$$

$$l_{13} v_0 = \overline{h_0 k_0}$$

$$I_{13}v_1 = h_0k_1 \oplus h_1k_0 \oplus h_0k_0$$

$$\overline{I_{13}u_0} = h_1k_1 \oplus [h_0k_0(h_1 + k_1)]$$

$$I_{13}u_1 = h_1k_1 + h_0k_0[h_1 + k_1]$$

$$I_{14}v_0 = I_{13}v_0$$

$$I_{14}v_1 = I_{13}v_1$$

$$I_{14}u_0 = I_{13}u_0$$

$$I_{14}u_1 = I_{13}u_1$$

$$I_{15}v_0 = h_0k_0$$

$$\overline{I_{15}v_1} = h_0k_1 \oplus h_1k_0$$

$$\overline{I_{15}u_0} = h_1k_1 \oplus [h_0k_1 + h_1k_0]$$

$$I_{15}u_1 = h_1k_1 + h_0k_1 + h_1k_0$$

Figure 1-4 shows the VEM maps for the four multiplexers and Figure 1-5 shows the block diagram of the cell design for G = 2.

With this design, the multiplication delay becomes:

$$T = (2N-1)\tau + \alpha$$

Where τ is the delay of the multiplexer and α is the delay of the input functions. Figure 1-6 shows an example of the multiplication algorithm for N = 4 bits.

TIME COMPLEXITY

The proposed network have a time complexity of O(N). The longest path is $((2N - 1)\tau + \alpha)$, if $G = 1$. But if $G > 1$, the delay becomes $((2(N/G) - 1)\tau + \alpha)$ depending on the value of G, and the time complexity can be very small compared to other structures with O(N) time complexity. τ is the delay introduced by the select line of the multiplexer. α is the delay introduced by the input function of the multiplexer which does not depend on N.

In the proposed structure, τ is the delay of a three-NAND-gate cascade, which can be made very small with modern technologies.

AREA COMPLEXITY

The area complexity of the network can be easily derived. Considering the fact that $G = 1$, we would have N^2 cells. If $G > 1$, the number of cells of the network becomes $(N/G)^2$. A cell contains $2G$ multiplexers. Then the area complexity becomes $(N/G)^2 2G$ which is in the order of $Q(N^2)$.

PIPELINING

The structures presented so far are combinational but they can be converted into synchronous ones in several ways.

By providing some latches at the output of every cell, and some additional latches for synchronisation, the cellular structure can be made to work in a pipelined way. Figure 1-7 shows this structure.

MULTIPLICATION OF TWO'S COMPLEMENT NUMBERS

Performing multiplication of two's complement numbers is easy with iterative multipliers. The algorithm proposed by Baugh and Wooley [BAU73] can be used.

The algorithm can be summarized as follows in the case H and K are N bits two's complement numbers:

$$H = -h_{N-1}2^{N-1} + \sum_{i=0}^{N-2} h_i 2^i$$

$$K = -k_{N-1}2^{N-1} + \sum_{i=0}^{N-2} k_i 2^i$$

The product P can then be expressed as follows:

$$P = P_1 + P_2 + P_3 + P_4 + P_5 + P_6$$

$$P_1 = 2^{2N-1}$$

$$P_2 = (\overline{h_{N-1}} + \overline{k_{N-1}} + h_{N-1}k_{N-1})2^{2N-2}$$

$$P_3 = \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} h_i k_j 2^{i+j}$$

$$P_4 = \sum_{j=0}^{N-2} h_{N-1}\overline{k_j}\,2^{N-1+j}$$

$$P_5 = \sum_{i=0}^{N-2} \overline{h_i} k_{N-1} 2^{N-1+i}$$

$$P_6 = (h_{N-1} + k_{N-1}) 2^{N-1}$$

The term P_3 is the main product performed by the cells. The term P_4 can be added in the lowermost line of cells by simply inverting the input of the AND gate which has h_{N-1} at the other input. In the same way, the term P_5 can be added in the leftmost row of the cells.

The terms P_1 and P_2 can be created by some small additional hardware and added to the last row. The term in P_6 can be added using the spare inputs of the cells in the leftmost column.

Pipelining the structure for two's complement operation would required only the addition of a few delay elements for bringing P_1 and P_2 to the last row at the proper time interval. Figure 1-8 shows the implementation of two's complement multiplication.

4 SINGLE PRECISION OR 1 DOUBLE PRECISION MULTIPLICATION

Iterative cellular multipliers conceived with the method proposed here can be programmed to perform 4 single precision multiplications or 1 double precision multiplication. This implementation is done easily by using multiplexers. Figure 1-9 shows a structure for performing either double precision or single precision operations.

CONCLUDING REMARKS ON THE ITERATIVE SOLUTION

A network for performing multiplication of two two's complement numbers has been proposed. The network can be implemented in a synchronous or in an asynchronous way. If the factors to be multiplied have N bits, the area complexity of the networks is $O(N^2)$ as in the case of cellular multipliers.

With some additional circuits the multiplier can be used either independently for performing four separate single precision multiplications or connected to perform a single double precision multiplication.

Asynchrounous multipliers can be built based on this proposal. Input/output ports can be multiplexed in order to minimize the package pins needed: this technique was proposed by TRW [TRW78].

This chapter introduced a fast scheme for macro cellular iterative multipliers and a cell design method for reducing the delay introduced by a macrocell which reduce it to the switching time of a multiplexers. The approach will be suitable for implementations with very high speed technologies like Gallium Arsenide where the simplicity of a cellular structure is an imperative requirement.

TABLE 1-1

THE MULTIPLICATION ALGORITHM

```
procedure cell(h0k0, x0, y0, v0, u0);
begin
   case x0 2^0 + y0 2^1 of
      0:begin
            v0: = h0k0;
            u0: = 0;
         end;
      1:begin
            v0: = not(h0k0);
            u0: = h0k0;
         end;
      2:begin
            v0: = not(h0k0);
            u0: = h0k0;
         end;
      3:begin
            v0: = h0k0;
            u0: = 1;
         end;
   end;
end;
```

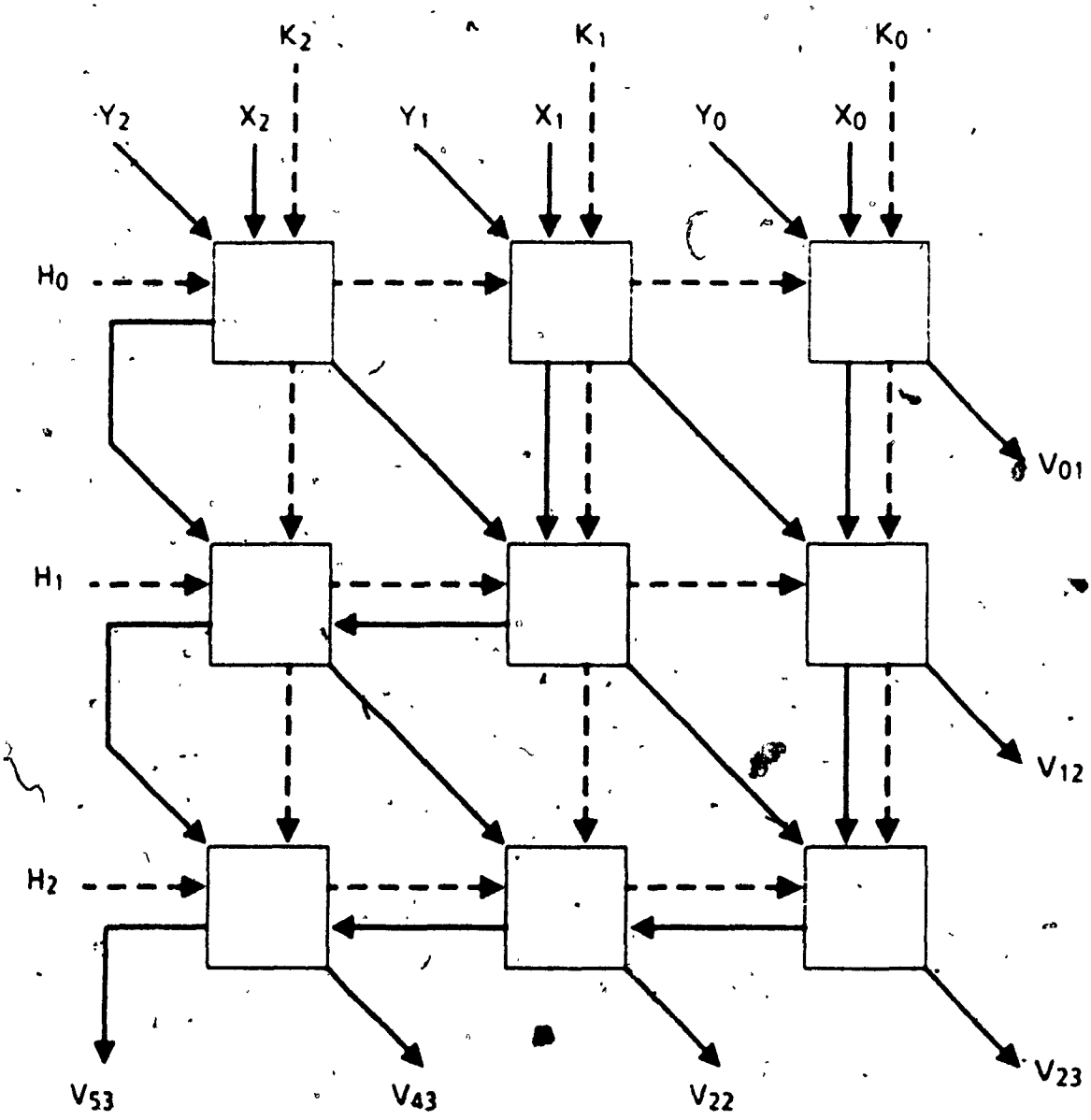
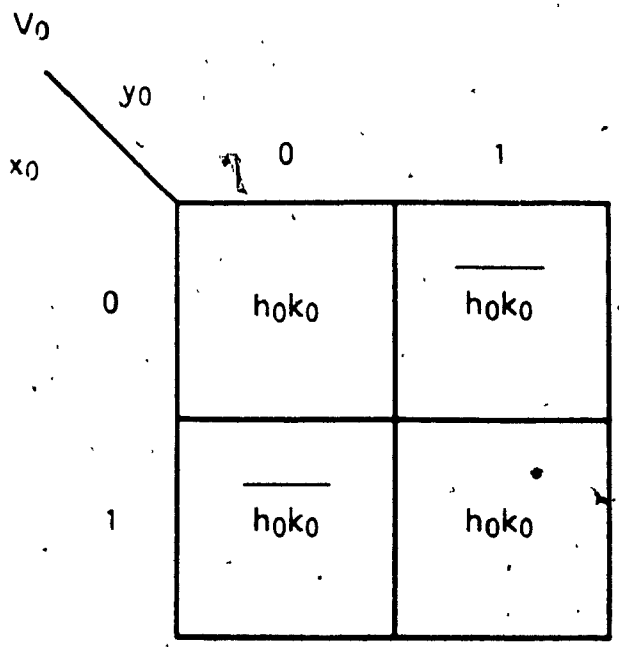


FIGURE 1-1. ***Iterative Multiplier***



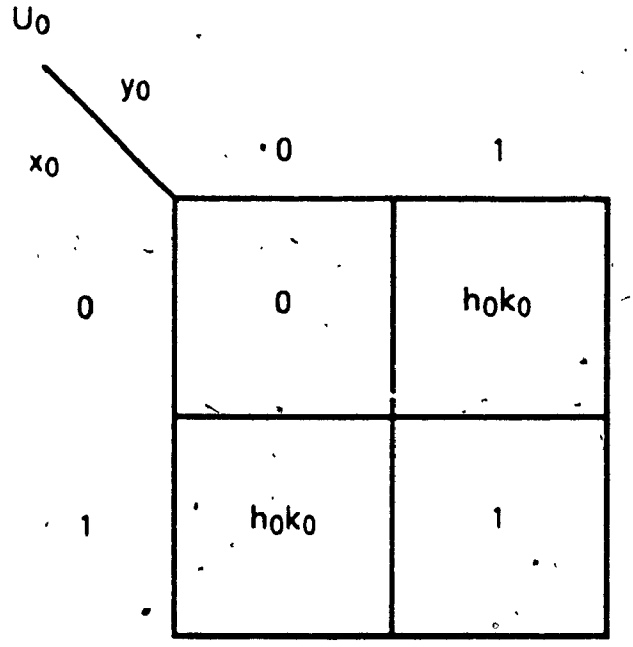


FIGURE 1-2. ***VEM Map for G = 1***

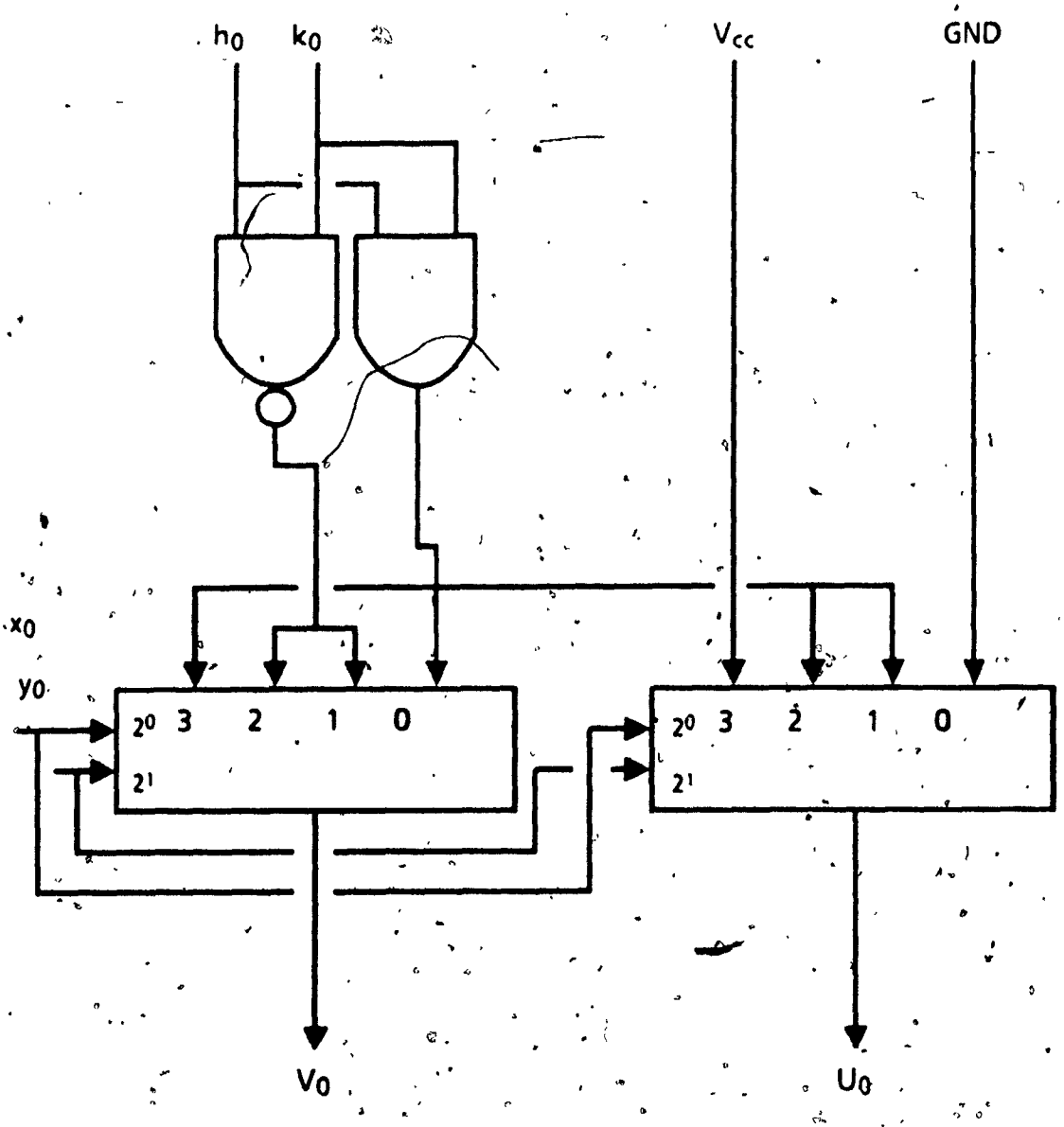


FIGURE 1-3. ***Logical Structure of the cell for G = 1***

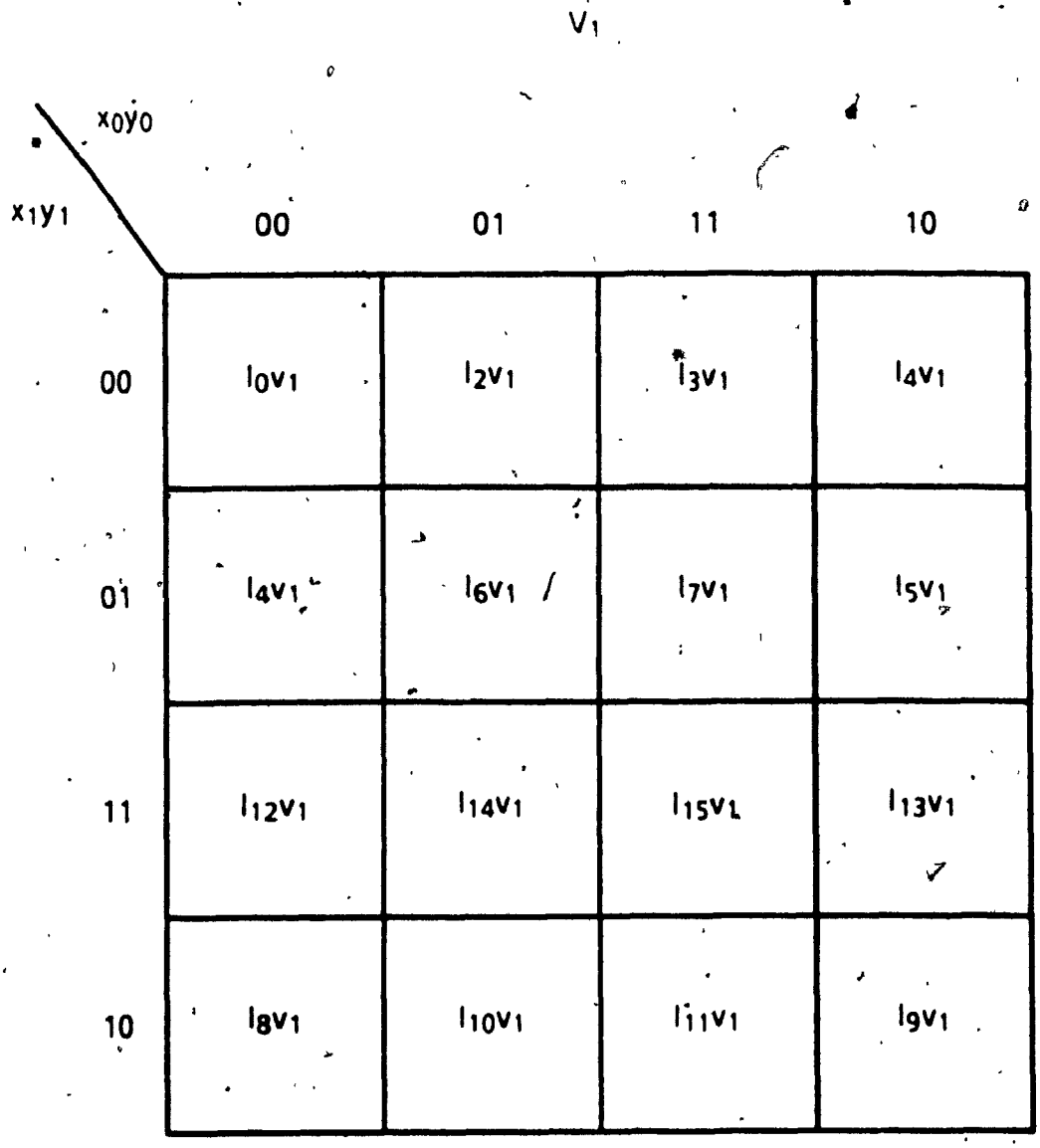


FIGURE 1-4. ***VEM Map for G = 2***

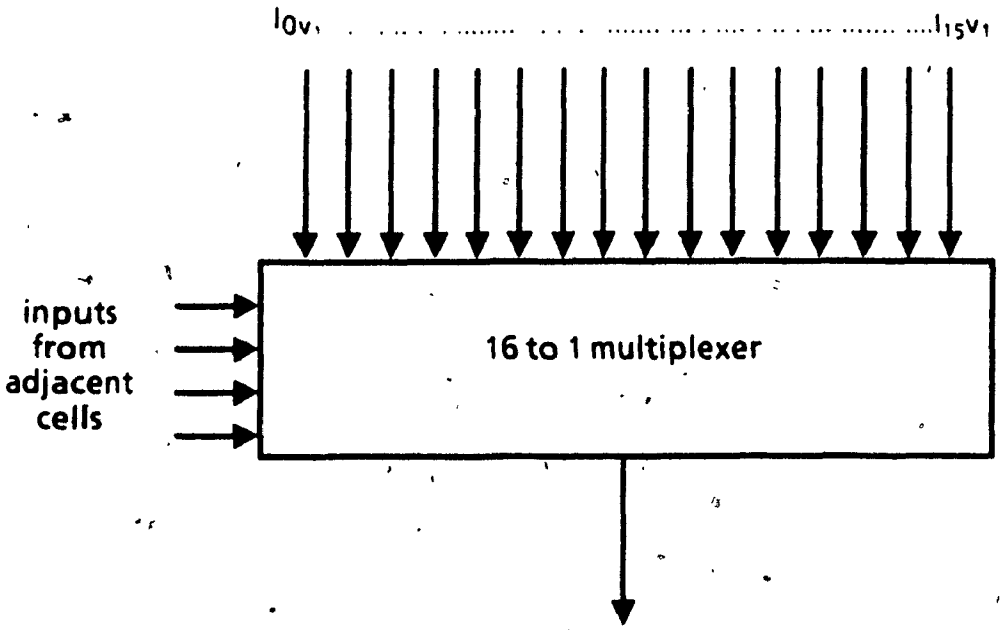


FIGURE 1-5. ***Logical Structure of the Cell for G = 2***

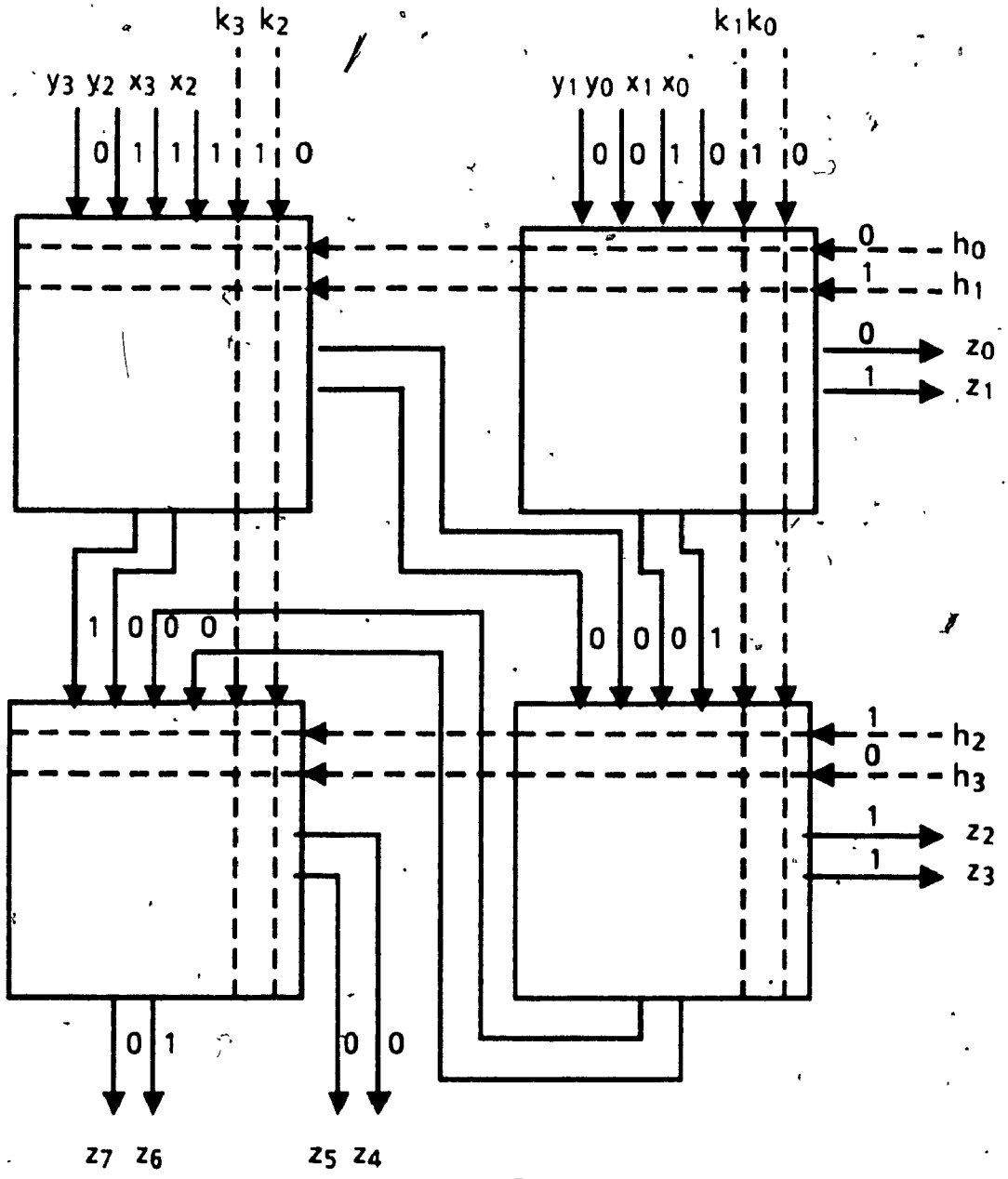


FIGURE 1-6. ***Example***

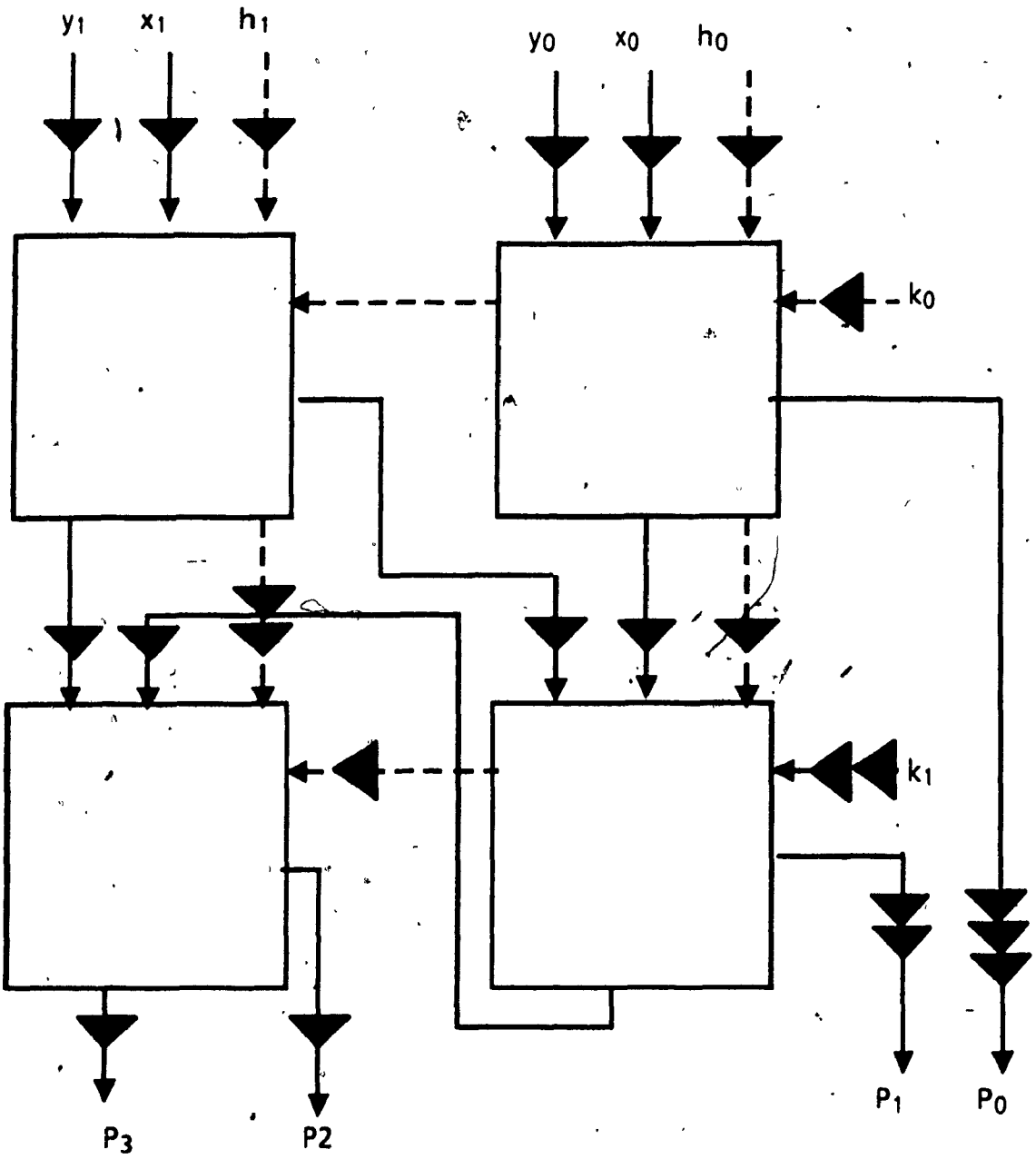


FIGURE 1-7. ***Pipelined Version***

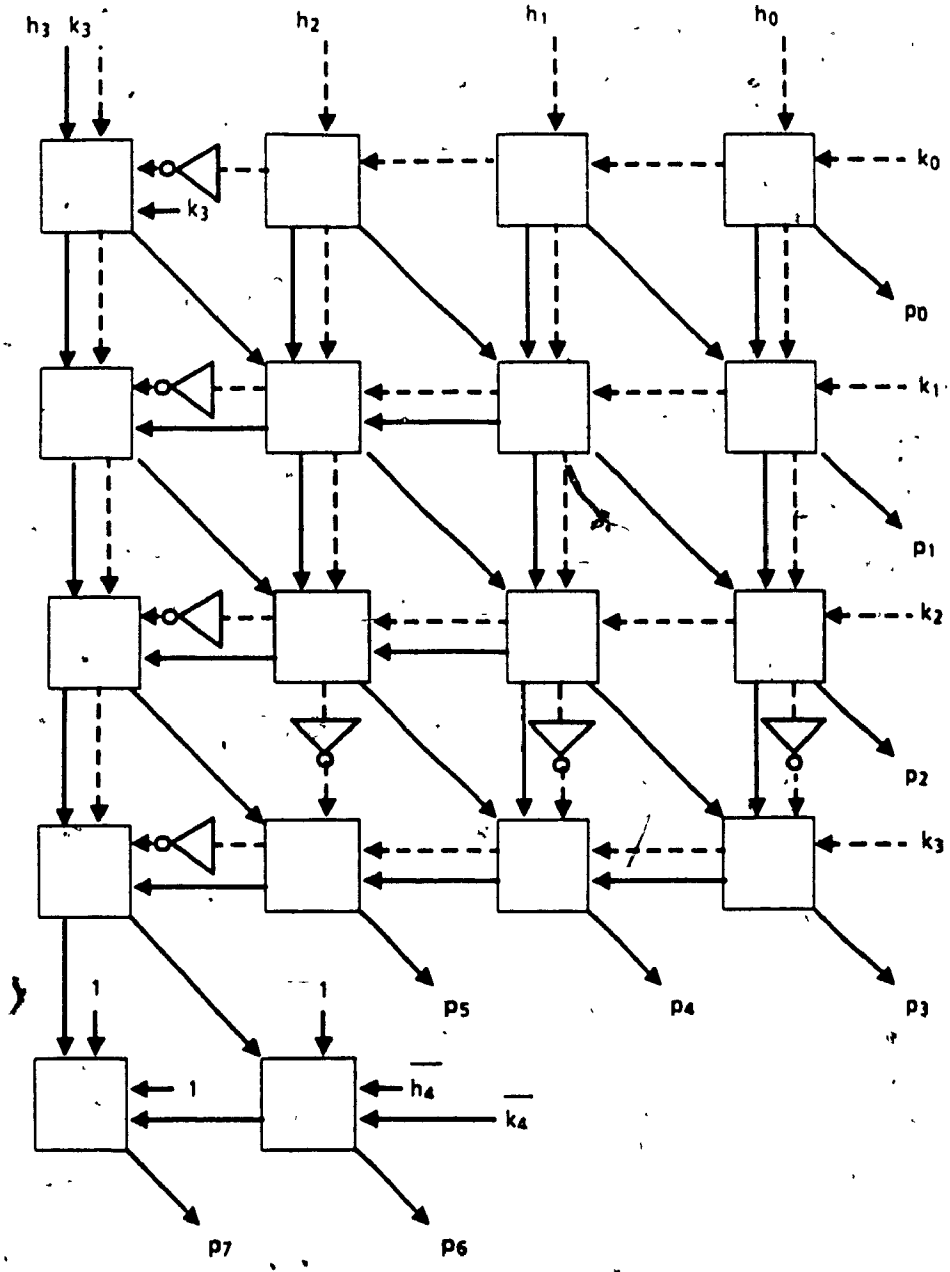
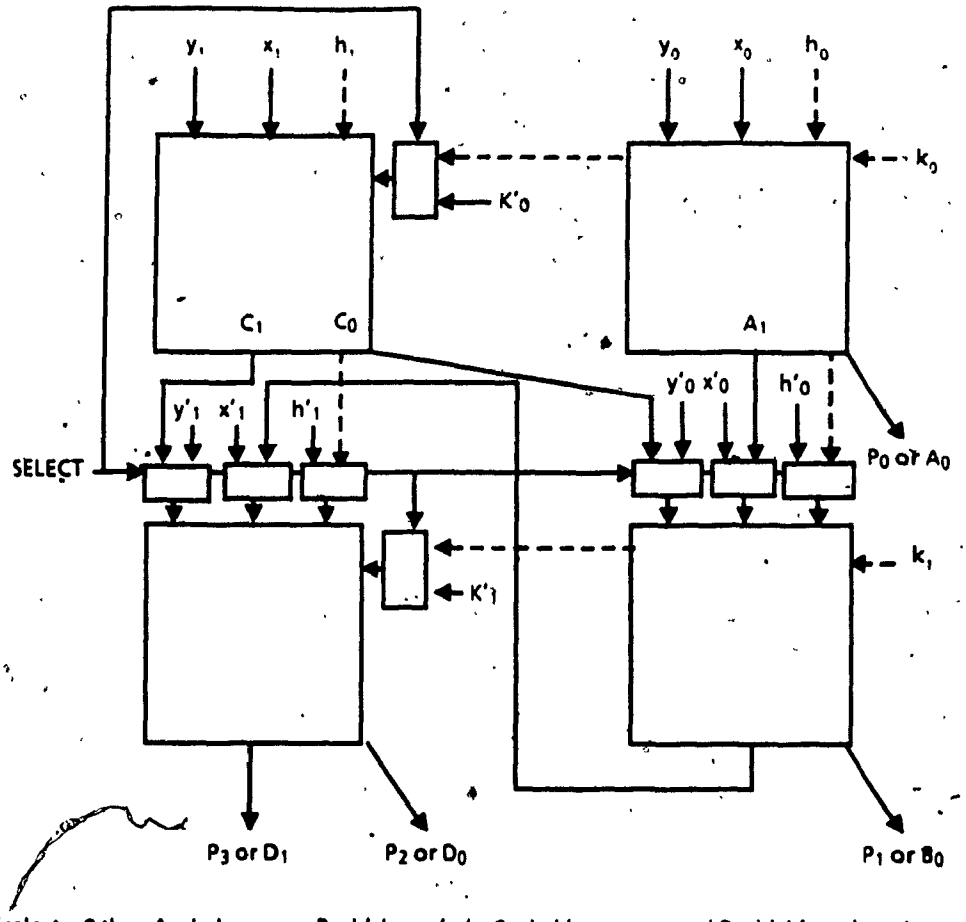


FIGURE 1-8. ***Two's Complement Implementation***



If select = 0 then $A = h_0k_0 + x_0y_0$, $B = h'_0k_1 + y'_0x'_0$, $C = h_1k'_0 + y_1 + x_1$ and $D = h'_1k'_1 + y'_1 + x'_1$

If select = 1 then $P = (h_0h_1)(k_0k_1) + (x_0x_1) + (y_0y_1)$

FIGURE 1-9. ***Circuit for Single and Double Precision Operation***

CHAPTER 2

A RECURSIVE ALGORITHM FOR BINARY MULTIPLICATION AND ITS IMPLEMENTATION

INTRODUCTION

In order to increase the speed of iterative multipliers having time complexity O(N), some macrocellular structures have been proposed in Chapter 1. A new solution for the macrocell designs based on multiplexers has been proposed in Chapter 1. It allows for the reduction of the maximum delay of the multiplier array proportional to a *fraction* of N.

A paper by Cappello and Steiglitz [CAP83] proposes a VLSI layout for parallel multipliers with an area complexity $A = O(N^2 \log N)$ and a time complexity $T = O(\log N)$. Cappello and Steiglitz compare the existing solutions on the basis of a VLSI Figure of Merit (FM) defined as follows:

$$FM = AT^2(PE)^2 \qquad (2\text{-}1)$$

where A is the area complexity, T is the time complexity and PE is the period complexity.

Cappello and Steiglitz [CAP83] have shown that their multipliers can be pipelined with a period complexity $PE = O(1)$ corresponding to a figure of merit:

$$FM_1 = N^2 \log^3 N \qquad (2\text{-}2)$$

They also derived a Lower Bound Figure of Merit of (2-1) for parallel multipliers (LBFM):

$$LBFM = N^2 \log^2 N \qquad (2\text{-}3)$$

and reported that this lower bound was not reached by any solution published before their paper.

A new recursive algorithm for the layout generation of parallel multipliers is presented in this chapter of my thesis. The area complexity of multipliers obtained by the use of such algorithm is $O(N^2)$ for values of N not exceeding a threeshold greater than 100, while the time complexity is $O(\log N)$. The implementation of such a scheme requires essentially two types of cells: carry-save adders and multiplexers. The circuit can easily be adapted to handle two's complement numbers. The structure can be pipelined with a period complexity $O(1)$ thus reaching the lower bound (2-3) of FM as defined by (2-1).

THE ALGORITHM FOR RECURSIVE MULTIPLIERS

In order to introduce the algorithm for designing recursive multipliers, an iterative algorithm, denoted Algorithm IM (Iterative Multiplication) for multiplying positive binary integer numbers will be first presented. This algorithm leads to the implementation of cellular multipliers with area complexity $A = O(N^2)$ and time complexity $T = O(N)$. A design for a basic cell of these multipliers is proposed in Chapter 1. These cells reduce the time complexity to a fraction of N allowing one to design fast multipliers for small values of N. Furthermore these cellular multipliers are implemented by repeating an elementary cellular structure which is a big advantage from the point of view of automatic design, manufacturing and testing. Iterative multipliers, also called ***array multipliers*** implemented as proposed in Chapter 1 are testable [DEM85] and can be pipelined with a period complexity O(1).

Algorithm IM (Iterative Multiplication)

Let

$$H = \sum_{i=0}^{N-1} h_i 2^i \qquad (2-4)$$

and

$$K = \sum_{j=0}^{N-1} k_j 2^j \qquad (2-5)$$

be positive integer factors and

$$P = H*K = \sum_{r=0}^{2N-1} p_r 2^r \qquad (2-6)$$

be the product. h_i, k_j, p_r are binary variables.

Let N = RG with R and G integers. Grouping the factors H and K in the basis 2^G one gets:

$$K = \sum_{n=0}^{R-1} K_n 2^{nG}$$

and (2-7)

$$H = \sum_{m=0}^{R-1} H_m 2^{mG}$$

with H_m and K_n integers less then 2^G and expressible in the binary code with G bits.

Figure 2-1 shows the structure of the iterative multiplier for the case G = 2 and N = 6. Some free inputs that can be used for adding two numbers are:

$$V = \sum_{i=0}^{N-1} v_i$$

and

$$W = \sum_{j=0}^{N-1} w_j$$

The input wires where V and W are applied are called ***additive input*** wires. The structure shown in Figure 2-1 is thus capable of performing the operation

$$Z = H^{*}K + V + W \qquad (2-8)$$

which is also the expression computed by the basic cell. Dean [DEA71] has called such a structure, a ***full-multiplier***.

A special cell design for G = 2 is proposed in Chapter 1. This cell design is based on four 16 input-1 output multiplexers where variables affected with the highest delay are applied at the ***select*** input. In this way the cell has a propagation delay equal to the switching time of a multiplexer.

The area complexity in this case is

$$A = O(\frac{N}{2})^2 = O(N^2) \qquad (2-9)$$

Let T_m be the switching time of a multiplexer; the structure has a total multiplication delay.

$$T_d = NT_m = O(N) \qquad (2-10)$$

Cells with G>2 can be designed following the approach proposed in Chapter 1 with multiplexers having 2G select inputs. This would reduce the multiplication delay to:

$$\frac{2N}{G} T_m$$

An algorithm has been proposed [DEM72] for using full multipliers of positive numbers in order to multiply two's complement binary fractions with the addition of front and back logic which does not affect the area, time, or the period complexity expressions. Another algorithm has been proposed by Baugh and Wooley for two's complement binary multiplication [BAU73]. It has been shown that the structure of Figure 2-1 can be adapted to perform multiplications of two's complement numbers with Baugh and Wooley's algorithm as shown in Chapter 1. Testability of two's complement multipliers based on iterative arrays of multiplexers has been investigated [DEM85].

In order to reduce the time complexity from O(N) to O(logN) by keeping the area complexity approximately $O(N^2)$, a new structure, called ***recursive multiplier*** is proposed in this chapter. This structure shares the following features with the iterative multiplier proposed in Chapter 1.

-modular layout,

-possibility of pipelining and multiplying two's complement factors,

-possibility of using the structure for a single or double precision multiplier or for four single precision multipliers.

An algorithm for designing recursive multipliers and for generating their layout will be given in the following.

ALGORITHM RM (Recursive Multiplication)

The RM algorithm will be introduced using the same assumptions and notations as in relations (2-4), (2-5), (2-6) and (2-7) of the IM algorithm.

For every step of the algorithm, the basic operations performed concurrently will be given. A device performing each basic operation will be defined and the details on how the device output is represented will be indicated.

Every output, represented by a capital letter, is supposed to be in pure binary code (1 bit per weight). The first step of the algorithm, called ***step 0***, consists in partitionning the N-bits of K and of H into adjacent G-tuples and in computing in parallel all the products between K_n ($1 \leq n \leq R$) and H_m ($1 \leq m \leq R$).

This product will be represented as:

$$P^0_{m,n} = H_m K_n 2^{(m+n)G}$$

where the superscript 0 shows that the product is the result of the computation performed at step 0. Products $P^0_{m,n}$ can all be computed with iterative arrays of macrocells based on

multiplexers. The delay introduced by these arrays can be reduced by allowing the most significant bits of the output to be 2 for each weight.

A layout for such arrays is shown in Figure 2-2a. There is one output for the first G least significant bits. Let:

$$L^0_{m,n}$$

be the binary number represented by these bits. On the contrary, there are two outputs per weight for the G most significant bits. These output bits can be considered as forming two binary numbers,

$$M^0_{1\,m,n} \quad and \quad M^0_{2\,m,n}$$

where each of these numbers has one bit per weight.

step 0 $(0 \le m < R, \quad 0 \le n < R)$
for every pair(m,n) **do**
cobegin

$$Compute: \qquad P^0_{m,n} = H_m K_n 2^{(m+n)G}$$

coend

Each concurent operation is performed by a device called a ***G_0-pseudomultiplier*** whose input-output characteristics are defined as follows:

G_0-pseudomultiplier(m,n)

Input: H_m, K_n

Output: $P^0_{m,n} = H_m K_n 2^{(m+n)G}$

The output is represented as follows:

$$P^0_{m,n} = L^0_{m,n} 2^{(m+n)G} + (M^0_{1m,n} + M^0_{2m,n}) 2^{(m+n+1)G}$$

$$(L^0_{m,n} < 2^G, M^0_{1m,n} < 2^G, M^0_{2m,n} < 2^G)$$

Figure 2-2a shows the layout of a G_0-pseudomultiplier with G = 8 using the cells whose design has been introduced in Chapter 1. The area complexity of this multiplier is

$$A_G = (\frac{G}{2})^2 \qquad (2-11)$$

and the time complexity is

$$T_G = GT_m \qquad (2-12)$$

Figure 2-2b shows a schematic representation of a G_0-pseudomultiplier.

Each cell in Figure 2-2a is, a 2 bits ***full multiplier***. In the Figures 2-2 the following assumptions have been made:

$$K_n = \sum_{i=0}^{7} k_{n+i} 2^i$$

$$H_m = \sum_{i=0}^{7} h_{m+i} 2^i$$

$$L^0_{m,n} = \sum_{i=0}^{7} o_{m+n+i} 2^i$$

$$M^0_{1m,n} = \sum_{i=0}^{7} a_{m+n+i+8} 2^i$$

$$M^0_{2m,n} = \sum_{i=0}^{5} b_{m+n+i+8} 2^i$$

Step 1 of the algorithm consists in grouping G_0-pseudomultipliers into squares of four each and in pseudo-adding the outputs using carry-save pseudoadders. Pseudo additions are performed on six numbers and produce 2 numbers whose sum is equal to the sum of the initial six. Pseudo-addition is a carry-save operation performed in a time that does not depend on the number of bits of the addends. The operation is repeated until two numbers whose sum is equal to the product are found.

Let ***step y*** be the generic pseudo-addition step and let ***step Y*** be the final step.

Step y

$$Let: \quad n_y = 0, 2^y, \ldots, q2^y, \ldots, R - 2^y.$$

$$m_y = 0, 2^y, \ldots, p2^y, \ldots, R - 2^y;$$

for every (m_y, n_y) **do**
cobegin

$$Compute: \quad P^y_{m_y,n_y} = P^{y-1}_{m_y,n_y} + P^{y-1}_{m_y+2^{y-1},n_y} + P^{y-1}_{m_y,n_y+2^{y-1}} + P^{y-1}_{m_y+2^{y-1},n_y+2^{y-1}}$$

coend

Each concurrent operation is performed by a device called a ***G_y-pseudomultiplier*** whose input-output characteristics are defined as follows:

G_y-pseudomultiplier (m_y, n_y)

input:

$$H_{m_y} = \sum_{m=m_y}^{m_y+2^y} H_m 2^{mG}$$

$$K_{n_y} = \sum_{n=n_y}^{n_y+2^y} K_n 2^{nG}$$

output :

$$P^y_{m_y,n_y} = H_{m_y} K_{n_y} 2^{(m_y+n_y)G}$$

The output is represented as follows:

$$P^y_{m_y,n_y} = (L^y_{1m_y,n_y} + L^y_{2m_y,n_y})2^{(m_y+n_y)G} + (M^y_{1m_y,n_y} + M^y_{2m_y,n_y})2^{(m_y+n_y+2^y)G} \quad (2-13)$$

The computation of two numbers:

$$P_1^Y = (L_1^Y + M_1^Y)2^{2RG}$$

$$P_2^Y = (L_2^Y + M_2^Y)2^{2RG} \quad (2\text{-}14)$$

such that:

$$P = P_1^Y + P_2^Y$$

is obtained by performing step #0 and then repeat step y until y = Y such that

$$2^Y = R \qquad (2-15)$$

There will be only one pseudomultiplier G_Y-(0,0) giving two output bits for each weight. These outputs can be added using a special adder having O(log N) time complexity [BRE80].

Figure 2-3 shows an example of the execution of a multiplication using the algorithm introduced so far. Numbers are represented in decimal code with one digit per weight for the sake of simplicity.

Although the algorithm proposed so far is iterative, a recursive algorithm can be introduced for defining a G_Y-pseudomultiplier in terms of G_y-pseudomultipliers ($0 \leq y < Y$). The recursive algorithm can be used for the generation of the multiplier layout.

```
G_y-pseudomultiplier(m,n)
begin
    if y = 0 then
        compute P^0_{m,n} using iterative arrays
    else
    cobegin
        G_(y-1)-pseudomultiplier(m,n);
        G_(y-1)-pseudomultiplier(m + 2^(y-1),n);
        G_(y-1)-pseudomultiplier(m,n + 2^(y-1));
```

$G_{(y-1)}$-pseudomultiplier($m + 2^{y-1}, n + 2^{y-1}$);

coend

compute $P^y_{m,n}$ as represented in ***step y*** using Pseudo-Adders (PA) and represent it by two numbers:

$$P^y_{1\,m,n} = (L^y_{1\,m,n} + L^y_{2\,m,n})\,2^{(m+n)G}$$

$$P^y_{2\,m,n} = (M^y_{1\,m,n} + M^y_{2\,m,n})\,2^{(m+n+2^y)G}$$

end

Notice that the size of a G_y-pseudomultiplier (defined as the number of factor bits involved in the operation) is 2^yG. Notice also that with the notation adopted here the index of the recursion is the subscript of G.

The algorithm of the entire multiplier can be described as:

multiplier(H,K);
inputs :H,K;
output :P;
begin
    y: = Y;
    G_Y-pseudomultiplier(0,0);
    $P := P^Y_1 + P^Y_2$;
end;

The addition of P^Y_1 and P^Y_2 that represent the output P^Y of the G_Y-pseudomultiplier is performed by a Special Adder. The way $P^Y_{m,n}$ is computed using the outputs of 4 $G_{(y-1)}$-pseudomultipliers is shown in Figure 2-4. PA stays for ***Pseudo-Adder***. Symbols in Figure 2-4 have been simplified for the sake of clarity. Some auxiliary variable OL_{ij} and OM_{ij} have been introduced ($1 \quad i, j \quad 2$) for representing pairs of numbers that are the results of partial pseudo-additions performed by PAs.

Notice that here pseudo-additions reduce, with a carry-save operation, six binary numbers to two binary numbers whose sum is equal to that of the six addends. The details of the operation performed by the PAs in Figure 2-4 are given in the following:

$$(OL^{y-1}_{21} + OL^{y-1}_{22})2^{(m+n+2^{y-1}G)} =$$

$$(M^{y-1}_{10} + M^{y-1}_{20} + L^{y-1}_{11} + L^{y-1}_{21} + L^{y-1}_{12} + L^{y-1}_{22})2^{(m+n+2^{y-1}G)}$$

$$(OM^{y-1}_{11} + OM^{y-1}_{12})2^{(m+n+2^{y}G)} =$$

$$(L^{y-1}_{13} + L^{y-1}_{23} + M^{y-1}_{12} + M^{y-1}_{22} + M^{y-1}_{11} + M^{y-1}_{21})2^{(m+n+2^{y}G)}$$

$$OL^{y-1}_{11} = L^{y-1}_{1\,m,n} \qquad (2-16)$$

$$OL^{y-1}_{12} = L^{y-1}_{2\,m,n}$$

$$OM^{y-1}_{21} = M^{y-1}_{1(m+2^{y-1}G),(n+2^{y-1}G)}$$

$$OM_{22}^{y-1} = M^{y-1}_{\cdot 2(m+2^{y-1}G),(n+2^{y-1}G)}$$

All the symbols used in (2-16) represent pure binary numbers having one bit per weight.

The representation of the output $P^y_{m,n}$, according to the scheme proposed by ***step y,*** is obtained by combining the outputs of Pseudo-Adders as follows:

$$2^{(m+n)}L^y_{1\,m,n} = OL_{11}^{y-1}\,2^{(m+n)} + OL_{21}^{y-1}\,2^{(m+n+2^{y-1}G)}$$

$$2^{(m+n)}L^y_{2\,m,n} = OL_{12}^{y-1}\,2^{(m+n)} + OL_{22}^{y-1}\,2^{(m+n+2^{y-1}G)} \qquad (2-17)$$

$$2^{(m+n+2^yG)}M^y_{1\,m,n} = OM_{11}^{y-1}\,2^{(m+n+2^yG)} + OM_{21}^{y-1}\,2^{(m+n+2^yG-2^{y-1}G)}$$

$$2^{(m+n+2^yG)}M^y_{2\,m,n} = OM_{12}^{y-1}\,2^{(m+n+2^yG)} + OM_{22}^{y-1}\,2^{(m+n+2^yG+2^{y-1}G)}$$

Figure 2-5 shows a schematic layout for a G_2-pseudomultiplier; connections between macrocells and PAs have been omitted for the sake of simplicity. The PAs are represented in Figure 2-5 by dashed lines. The maximum number of input addends in a PA is 6, and wires carrying pairs of addends are

already ordered in such a way that most of the wires carrying bits of the same weight are adjacent.

The Special-Adder (SA) layout occupies two sides of the square and is also represented by dashed lines.

The layout of a PA is shown in Figures 2-6. Figure 2-6a shows the structure of the Special Adder which uses four Carry-Save Adders for adding six numbers A_1, A_2, B_1, B_2, C_1, C_2 in order to produce two output O_1 and O_2 whose sum is equal to the sum of the six addends. Figure 2-6b shows the detailed implementation of the CSAs. Each cell is a full-adder, the sum and carry outputs are indicated by S and C. Capital letters in Figure 2-6a indicate binary numbers and lower case letters in Figure 2-6b indicate their bits.

The time complexity of the proposed PA structure is O(3). The area complexity of a PA is $O((2^{(y-1)}G)(\alpha 2^{(y-1)}G + 4)) = O(\alpha(2^{(y-1)}G)^2)$. Where α is the ratio between the area complexity of a pair of wires and the side of a G_0-pseudomultiplier divided by G.

TIME COMPLEXITY

Assuming that D is the delay introduced by an AND/OR circuit implementing the basic functions of a PA and the Special Adder (SA), assuming T(G) is the delay introduced by a G_0-pseudomultiplier, if Y recursions are applied, then the multiplication delay is the sum of three contributions due to the special adder, the chain of PAs and the G_0-pseudomultipliers. The contribution of the special adder is taken from [CAP83]. This global delay can be expressed as follows:

$$T = (\log 2N + 3Y)D + T(G) \qquad (2-18)$$

Logarithms are supposed to be base two unless specified otherwise.

Using the just described desing approach, a cellular G_0-pseudomultiplier of G bit can be designed with a delay:

$$T(G) = GD \qquad (2-19)$$

The G_0-pseudomultiplier could also be implemented with a Read Only Memory (ROM) making T(G) independant of G. In this case the (2-18) can be rewritten as :

$$T = (\log 2N + 3Y)D + T_{ROM}$$

A technologically acceptable solution could be a ROM with a number of bits less than 2^{12}.

In order to find a relation between Y and N from (2-14) ones gets:

$$Y = \lceil \log \frac{N}{G} \rceil \qquad (2-20)$$

Where $\lceil A \rceil$ means A if A is integer or the least integer greater than A.

In order to keep the time complexity of the structure logarithmic, different conditions on G can be imposed. A very simple one is the following:

$$G = \log N \qquad (2-21)$$

With this assumption the global delay can be expressed as:

$$T = (\log 2N + 3 \log \frac{N}{\log N} - \log N)D$$

and the overall time complexity is O(log N). Notice that for small values of N, 3log(N/log N) does not introduce a remarkable contribution to the overall delay.

AREA COMPLEXITY

The area complexity of the recursive multiplier can be computed by inspection of Figures 2-5 and 2-6 as follows:

$$O(A) = O(A(G_Y - pseudomultiplier) + A(Special\ Adder))$$

$$= O(4A(G_{Y-1} - pseudomultiplier)) +$$

$$+ O(N\log N) + area\ of\ 2(\frac{N}{2}\ bit\ pseudo - adders)$$

$$= O(16A(G_0 - pseudomultiplier)) +$$

$$area\ of\ 2(\frac{N}{2}\ bit\ pseudo - adders) +$$

$$area\ of\ 4(\frac{N}{4}\ bit\ pseudo - adders) + O(N\log N)$$

In general, there are $(N/G)^2$ G_0-pseudomultipliers whose area complexity is $O(N^2)$.

There are $Y \approx \log(N/\log N)$ levels of pseudo adders. At level Y there are two pseudo adders. Each one of them has an area complexity that can be computed from the scheme of Figure 2-6. The horizontal size is proportional to $2\alpha N/2$ and the vertical size is proportinal to N/2. α is the ratio between the area complexity of a wire and the area complexity of a size of a cell of the G_0-pseudomultiplier. Thus the area complexity at level Y is :

$$O(2^2 2a^2 \frac{N^2}{2^2}) = O(a\frac{N^2}{2})$$

Given the proposed structure and the cell design proposed in Chapter 1.

At level Y-1 there are 2^2 PAs handling $N/2^2$ wires. The area complexity at level Y-1 is:

$$O(2^2 2a^2 \frac{N^2}{2^{2*2}}) = O(a\frac{N^2}{2^2})$$

The overall area complexity of the PAs is:

$$A_{PA} = O(a.N^2(\sum_{i=1}^{Y}\frac{1}{2^i})) = O(aN^2) \qquad (2-22)$$

Equation (2-22) does not take into account the space that remains free in Figure 2-5. Part of this space is occupied by wires that connects pseudo-multipliers with pseudo-adders.

Taking this space into account in the evaluation of area complexity represents a worst-case situation in which no attempt is made for squeezing G_0-pseudomultipliers and pseudoadders in order to obtain an optimal layout with $O(N^2)$ area complexity. In the worst case situation, corresponding to the layout of Figure 2-

5, it is important to notice that each square containning four G_Y-pseudomultipliers has two strips whose sizes are:

$$\frac{N}{2^{Y-x}} \quad and \quad 2a\frac{N}{2^{Y-x+1}}$$

Thus, the overall area complexity can be computed as follows:

$$A_{PA} = 2*2a*N^2(\frac{1}{2^0*2^1} + 2^2\frac{1}{2^1*2^2} + 2^4\frac{1}{2^2*2^3} + \ldots.)$$

$$= 2aN^2(1+1+\ldots.)$$

$$= 2aN^2 logR$$

In the worst case, the area complexity of the G_Y-pseudomultiplier is:

$$A = O(N^2(1+2aY)) \qquad (2-23)$$

The area complexity of the proposed structure can be assumed to be $O(N^2)$ as far as:

$$log(\frac{N}{log\,N}) \leq \frac{1}{2a} \qquad (2-24)$$

Following the cell design proposed in Chapter 1, each cell of a G_0-pseudomultiplier contains four 16 input multiplexers and

circuits implementing functions of four variables. Based on the above considerations it can be assumed that $\alpha \approx 0.025$ which makes the (2-24) an acceptable condition for a large class of practical multipliers.

MULTIPLICATION OF TWO'S COMPLEMENT NUMBERS

From Figure 2-2a it appears that a G_0-pseudomultiplier can accept two additive inputs, a G bit number along the vertical inputs lines and a 2G bits number along the diagonal lines. In the layout sketched in Figure 2-5 each square represents a G_0-pseudomultiplier each one of which can accept two additive inputs. In particular, the lowermost row and the leftmost column of G_0-pseudomultipliers can accept two addends whose bit weights range from 2^{N-1} to 2^{2N-2}. We will show how these inputs can be used for multiplying two binary numbers with negative numbers represented in the two's complement notation using the Baugh and Wooley [BAU73] algorithm. For the sake of clarity the algorithm will be summarized in the following.

Let now assume that H and K are N bits two's complement numbers:

$$H = -h_{N-1}2^{N-1} + \sum_{i=0}^{N-2} h_i 2^i \qquad (2-25)$$

$$K = -k_{N-1}2^{N-1} + \sum_{i=0}^{N-2} k_i 2^i$$

The product P can then be expressed as follows:

$$P = P_1 + P_2 + P_3 + P_4 + P_5 + P_6$$

$$P_1 = 2^{2N-1}$$

$$P_2 = (\overline{h_{N-1}} + \overline{k_{N-1}} + h_{N-1}k_{N-1})2^{2N-2}$$

$$P_3 = \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} h_i k_j 2^{i+j}$$

(2 – 26)

$$P_4 = \sum_{j=0}^{N-2} h_{N-1}\overline{h_j}\,2^{N-1+j}$$

$$P_5 = \sum_{i=0}^{N-2} k_{N-1}\overline{k_i}\,2^{N-1+i}$$

$$P_6 = (h_{N-1} + k_{N-1})2^{N-1}$$

The term P_3 is the main product performed by the G_Y-pseudomultiplier. The term P_4 can be added in the leftmost column of cells of G_0-pseudomultipliers by introducing an array of cells made by an inverter and a two input AND gate fed by h_{N-1} and the complement of h_j. In the same way, the term P_5 can be added in the lowermost row.

The terms P_1, P_2 and P_6 can be created by some small additional hardware and added to SA.

Figure 2-7 shows the details of the modification introduced in the leftmost column and the lowermost line of G_0-pseudomultipliers in order to make the structure capable of multiplying two's complement numbers. By inspection of Figure 2-7 it is clear that the introduced modifications do not affect the delay nor the area complexities of the multiplier as evaluated previously.

PIPELINING

Let PE be the period of a multiplier, defined as the time between completion of successive multiplication instances. If the number of bits of the multiplier and the application for which the multiplier is designed are such that the combinational array introduced so far can complete an operation before a new operation is started, then registers can be introduced only for storing the factors and the product. In this case, the period complexity of the array would be

$$PE = 1 \qquad (2\text{-}27)$$

If the application requires a period between the output of two successive results to be less than the multiplication time, then the array can be ***pipelined*** for reducing the idle time of the circuit cells.

Concepts derived from retiming transformation [LEI83] can be applied to the recursive multiplier in order to implement pipelinning. Following a recent suggestion by Hawck et al. [HAW85], pipelining can be implemented by first adding registers on the inputs and then retiming to minimize the period.

The ***degree of pipelining*** can be defined as the maximum number of cells between any pair of registers. For our specific application there are two types of cells, normely, the cells of the

G_0-pseudomultiplier and the carry save adders used in the Pseudo Adders and in the Special Adder.

The most effective level of pipelining depends on many practical and technological considerations. A practically acceptable degree of pipelining given the actually available adders and multiplexers is four. This implies that registers can be placed at the output of the pseudo adders whose structure is proposed in Figure 2-6 but there is no need of providing registers inside them.

Figure 2-8 shows a rearrangement of the layout shown in Figure 2-5 where G_0-pseudomultiplier are represented by squares, pseudo adders are represented by rectangles and the wires carrying the bits of each of the binary numbers representing a PA's output are represented by a single arrow. A black arrow represent also an array of registers on the corresponding wires.

As far as $G<4$ there is no need of introducing registers inside the G_0-pseudomultipliers. It suffices to introduce registers at their outputs. From inspection of Figure 2-8 one can conclude that extra registers are required at the output of the PA feeding the SA in order to synchronize the appearence of the bits of the same product at the input of SA. The number of extra registers required is zero for the rightmost and the lowermost SA and increases by one going leftward and upward. These registers are not shown in

Figure 2-8 for the sake of simplicity. Pipelining SA has been discussed elsewhere [BRE80] and won't be discussed here.

Should the G_0-pseudomultiplier be pipelined, the same technique proposed by Hawck et al. [HAW85] can be used for placing the registers. In any case, a period complexity PE = O(1) can be achieved.

Several VLSI figures of merit have been proposed. The ones that have been mostly discussed are:

$$FM_a = AT^2(PE)^2$$
$$FM_b = A(PE)^2 \qquad (2\text{-}28)$$
$$FM_c = A(PE)T$$

For each one of them lower bounds for binary multipliers have been derived.

The recursive structure proposed here meets these lower bounds with the complexities:

$$A = N^2$$
$$T = \log N \qquad (2\text{-}29)$$
$$PE = 1$$

Given the following figures of merit

$$FM_a = N^2 \log^2 N$$
$$FM_b = N^2 \qquad (2\text{-}30)$$
$$FM_c = N^2 \log N$$

SOME APPLICATIONS OF THE RECURSIVE STRUCTURE

Figure 2-9 shows a layout for a structure based on G_γ-pseudomultipliers capable of performing a single double-precision multiplication or four single-precision multiplications. Binary numbers are represented with a single arrow.

S_1, S_2, S_3, S_4 represent single precision outputs, DP represents a double precision output. Essentially, four SAs are included for single precision with a global contribution to the whole area complexity of $A_{4SA} = 4N\log N$. These single-precision SAs are bypassed when double-precision multiplication has to be performed. External switches have to be added for sending the factor bits to the four G_γ-pseudomultipliers and for selecting the outputs to be stored. The complexity of these switches, which are not shown in Figure 2-9 for the sake of clarity, is O(N). Thus the overall area complexity remains $O(N^2)$. Also the time complexity remains unchanged.

CONCLUDING REMARKS ON THE RECURSIVE STRUCTURE

The recursive design approach introduces some new ideas that are suitable for an automatic layout design. Another potentially useful idea presented in this chapter is that of combining multiplexer-based macrocells with pseudo-adders. Macrocells based on multiplexers allows a fast propagation of changes of variables affected by a large delay by applying them to the address select inputs while using simpler and slower circuits applied at the multiplexer inputs for handling factor bits. Paths containing a few macrocells in cascade can be tolerated as far as their delay is not the predominant contribution to the overall multiplier delay. Efficient pseudo adders with constant delay can be used for adding up the outputs of chains of macrocells in order to produce two numbers whose sum is equal to the desired product.

A recursive procedure can systematically place pseudo adders in a regular layout of macrocells making the maximum delay of a chain of pseudo adders proportional to log N.

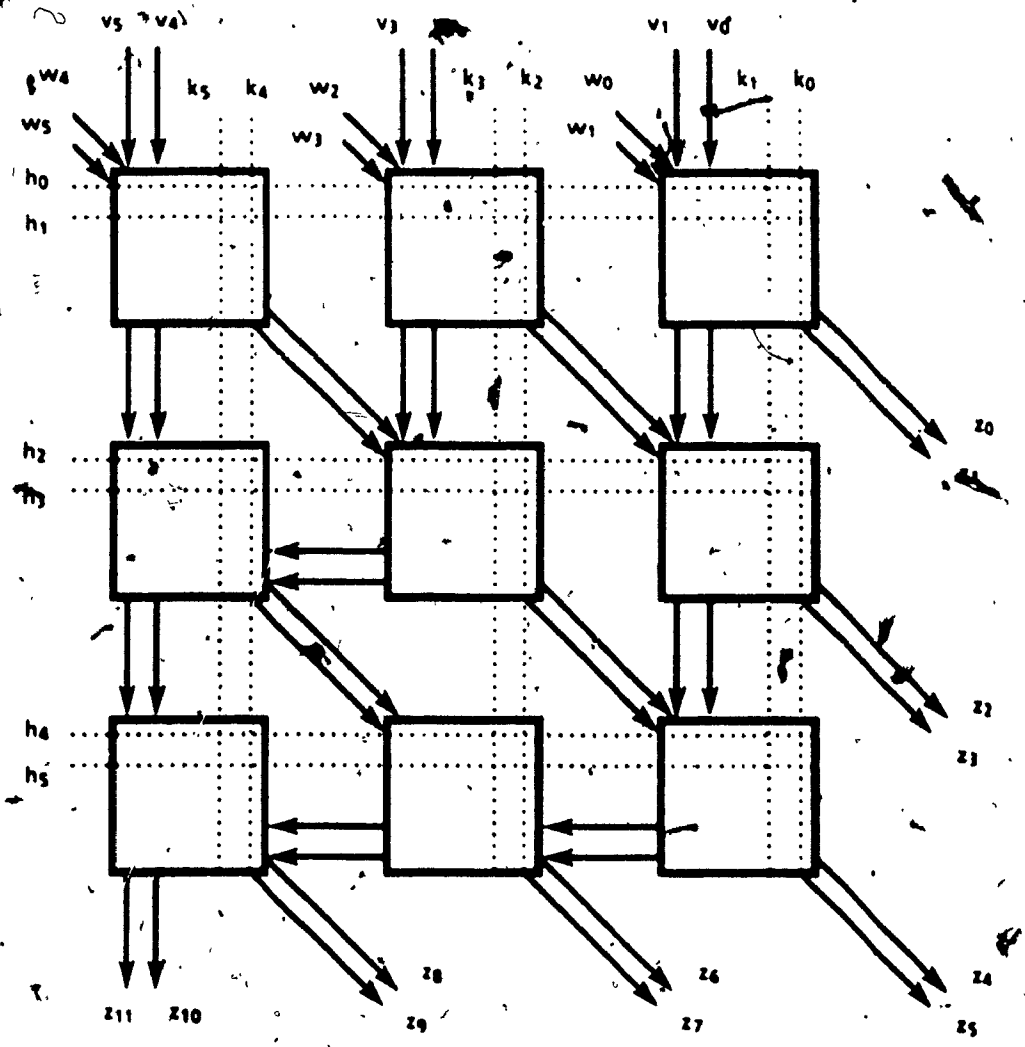


Figure 2-1. *Iterative multiplier*

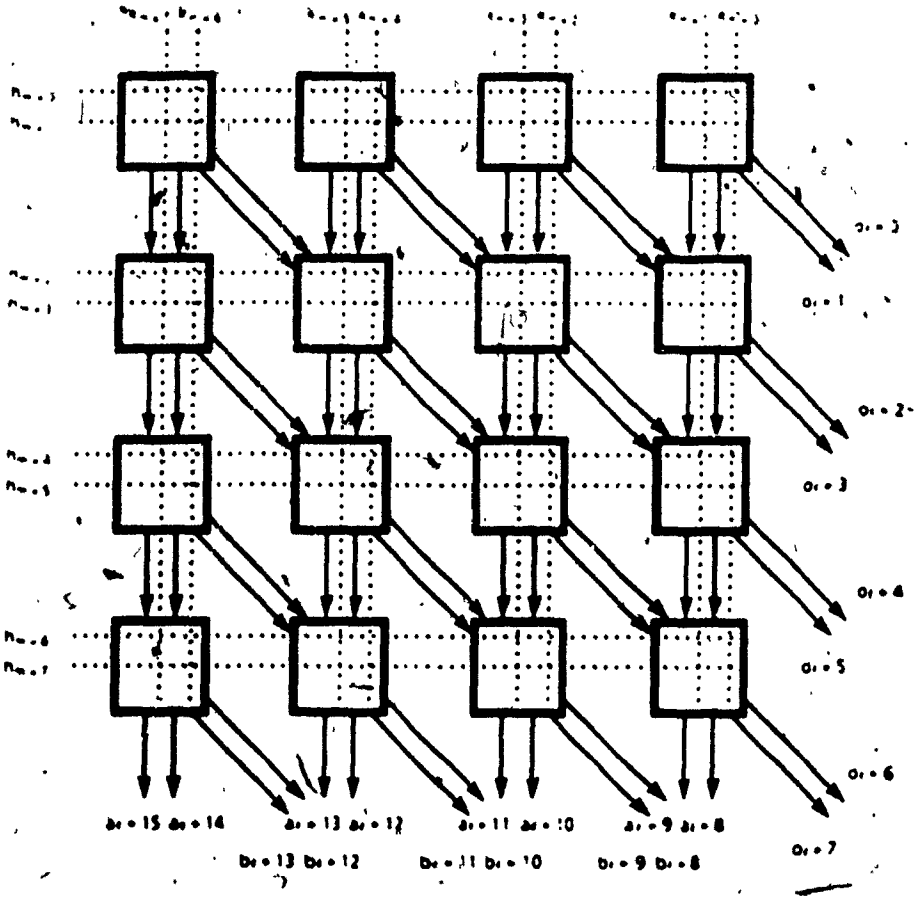


Figure 2-2a. *Iterative G_σ-pseudomultiplier (detailed scheme)*

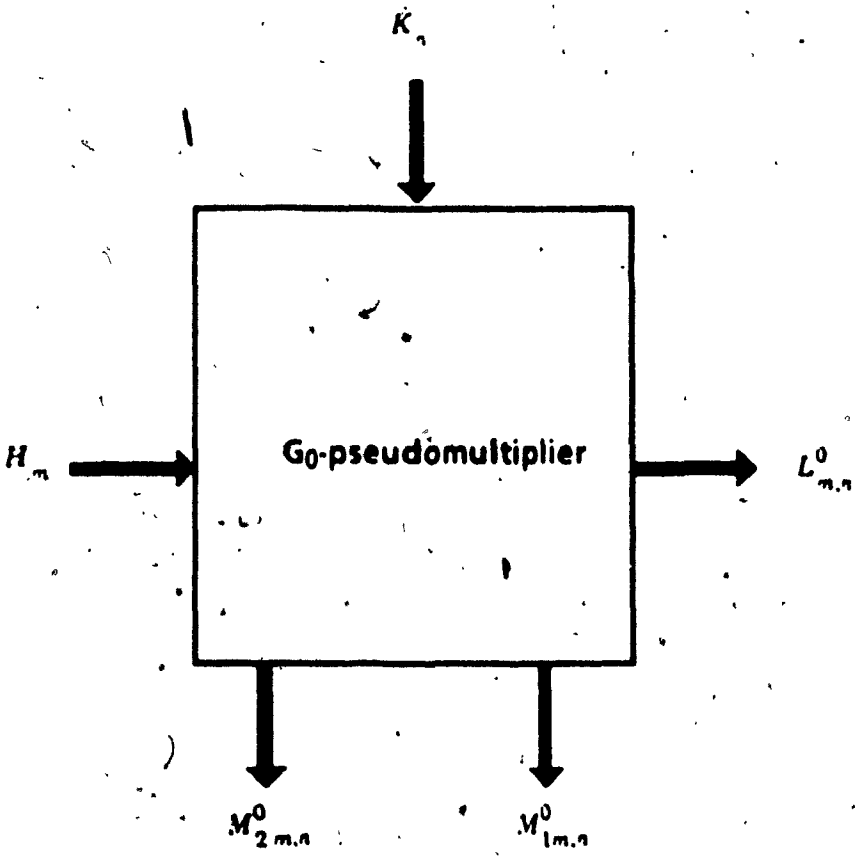


Figure 2-2b. *Iterative G_0-pseudomultiplier*

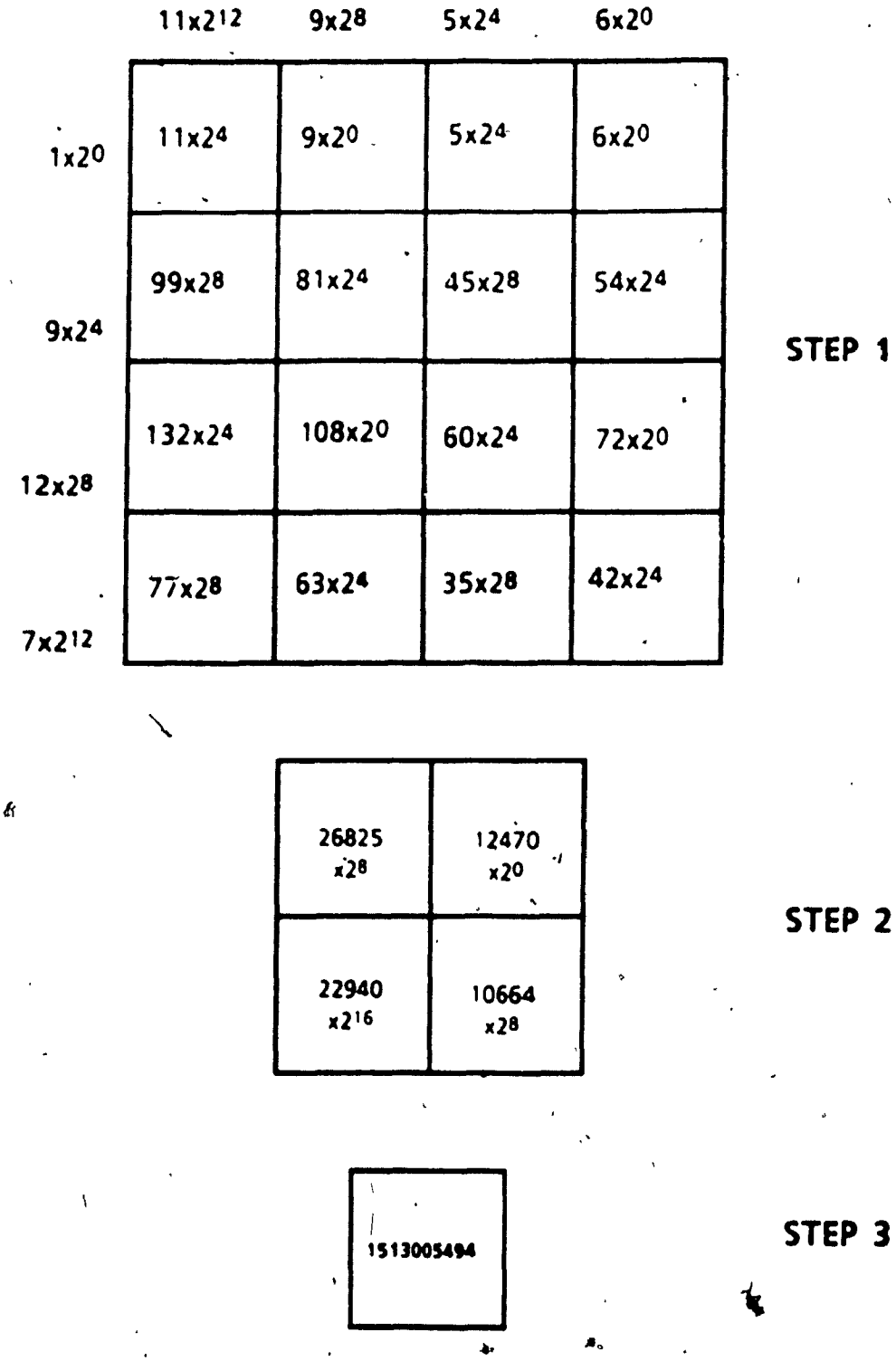


Figure 2-3. ***Recursive multiplication example***

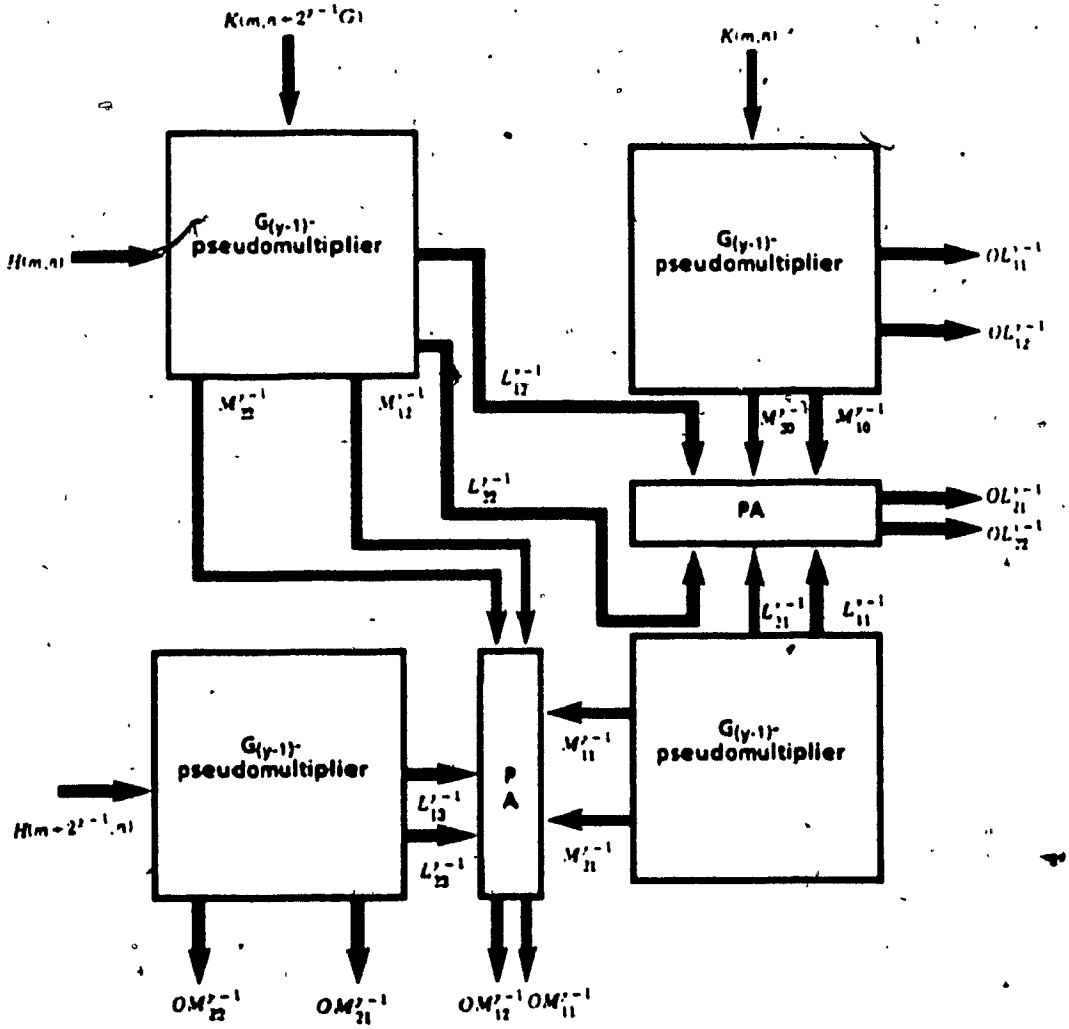


Figure 2-4. G_y-pseudomultiplier

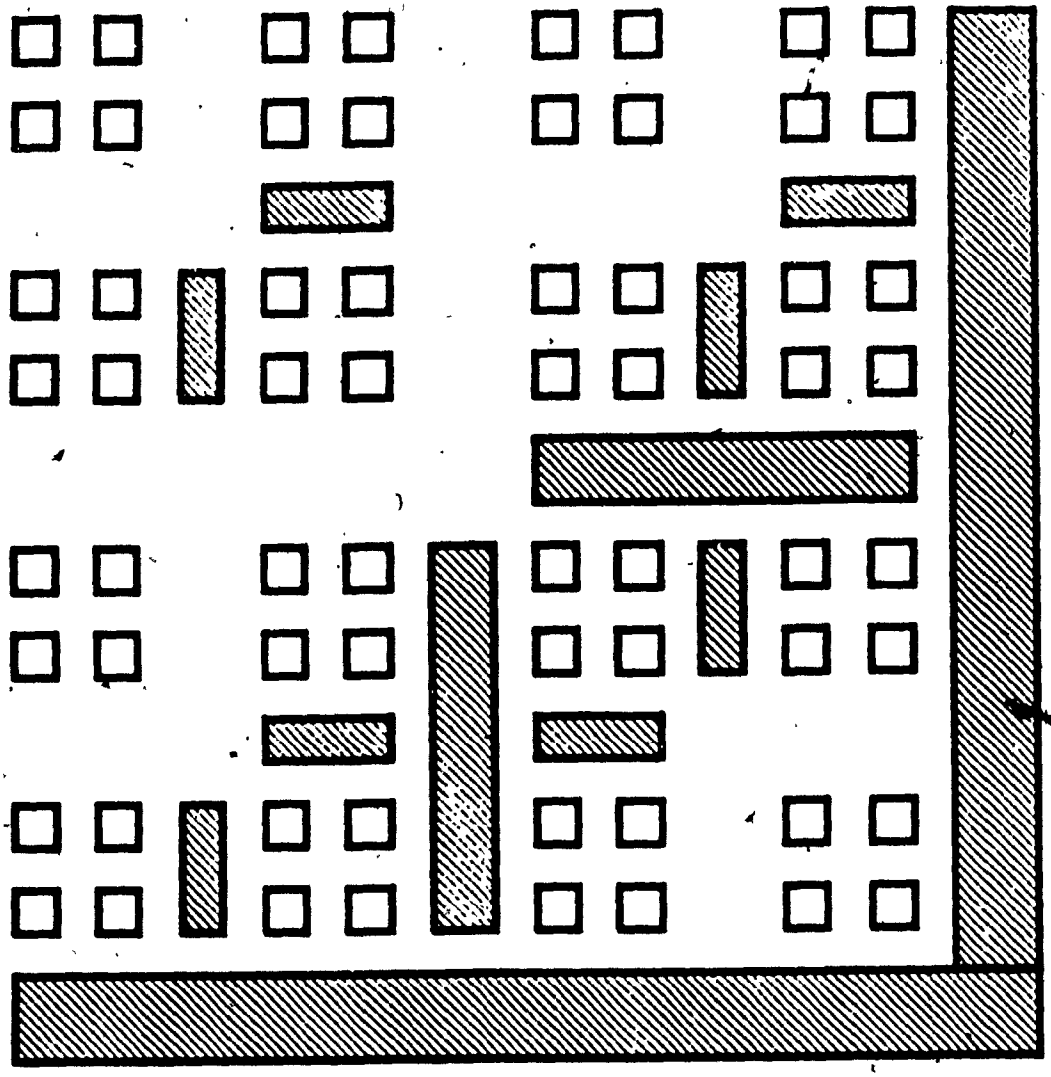


Figure 2-5. ***Recursive multiplier layout***

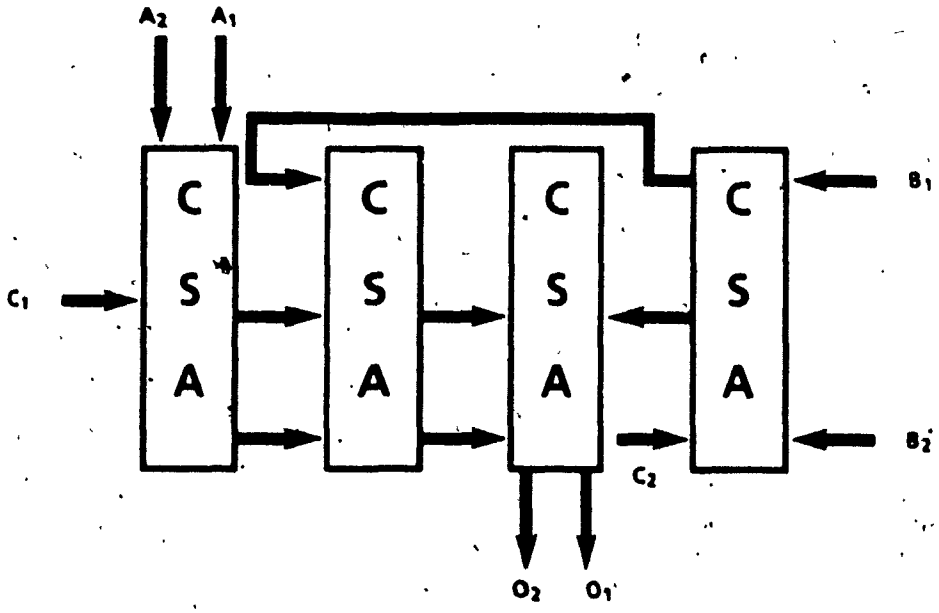


Figure 2-6a. ***Pseudo adder scheme (global scheme)***

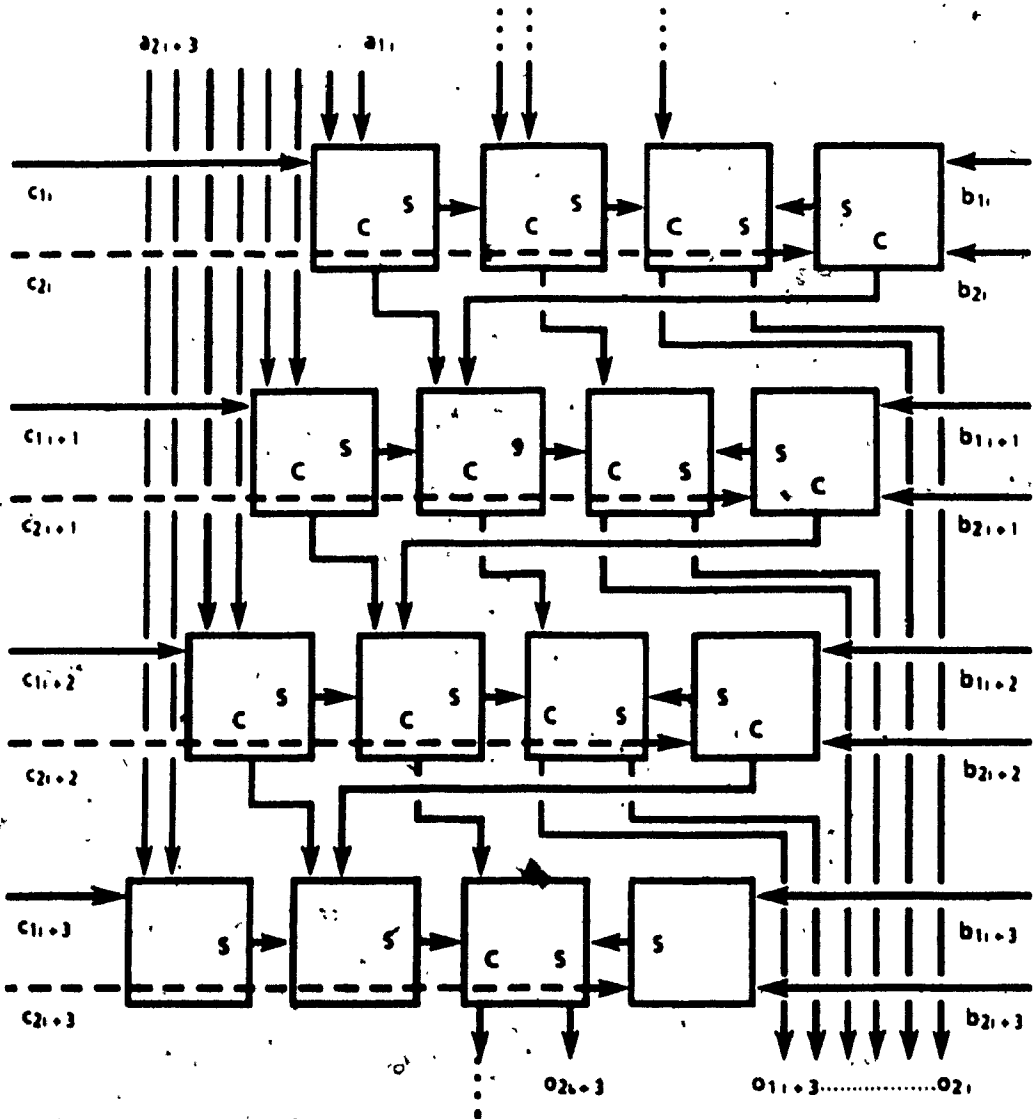


Figure 2- 6b. ***Pseudo adder scheme (detailed scheme)***

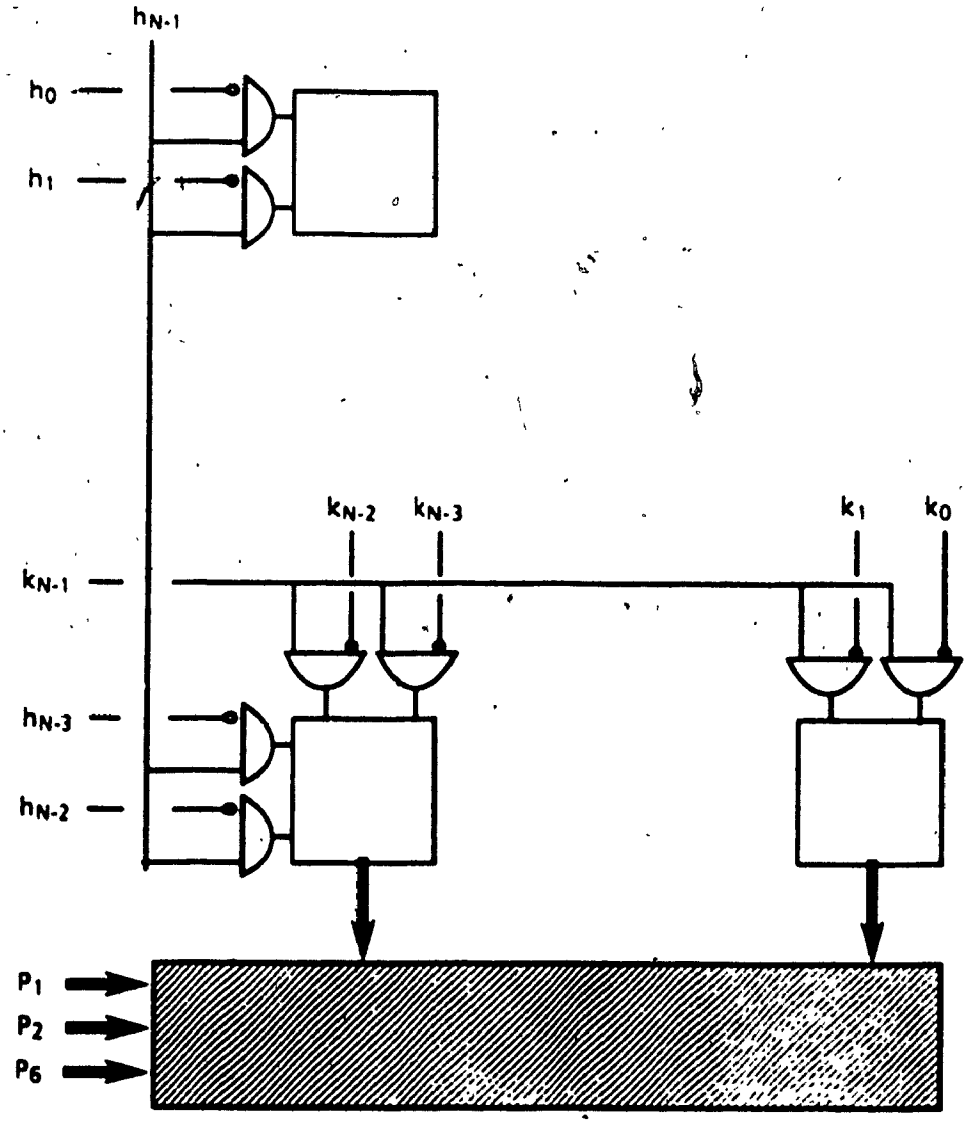


Figure 2-7. ***Scheme for two's complement multiplication***

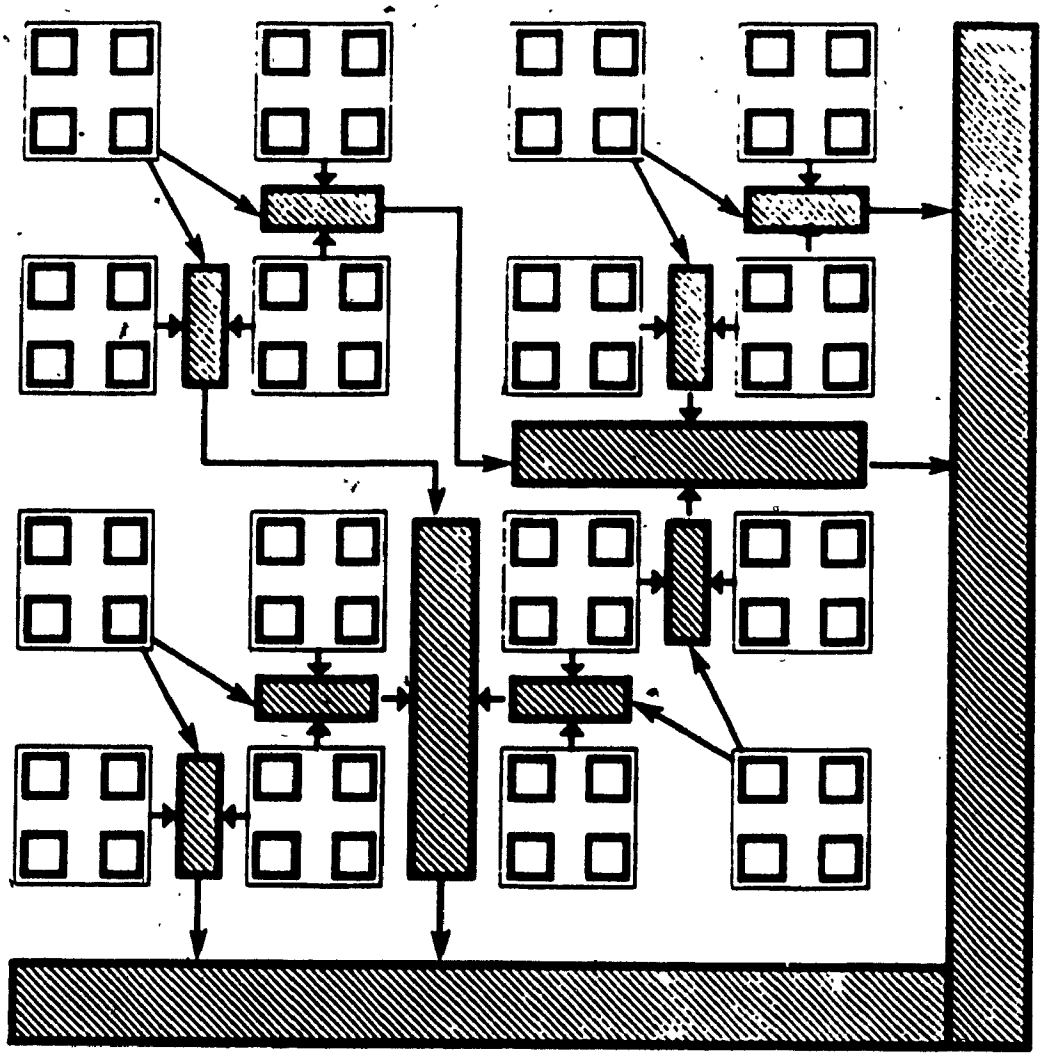


Figure 2-8. *Pipelined multiplier layout*

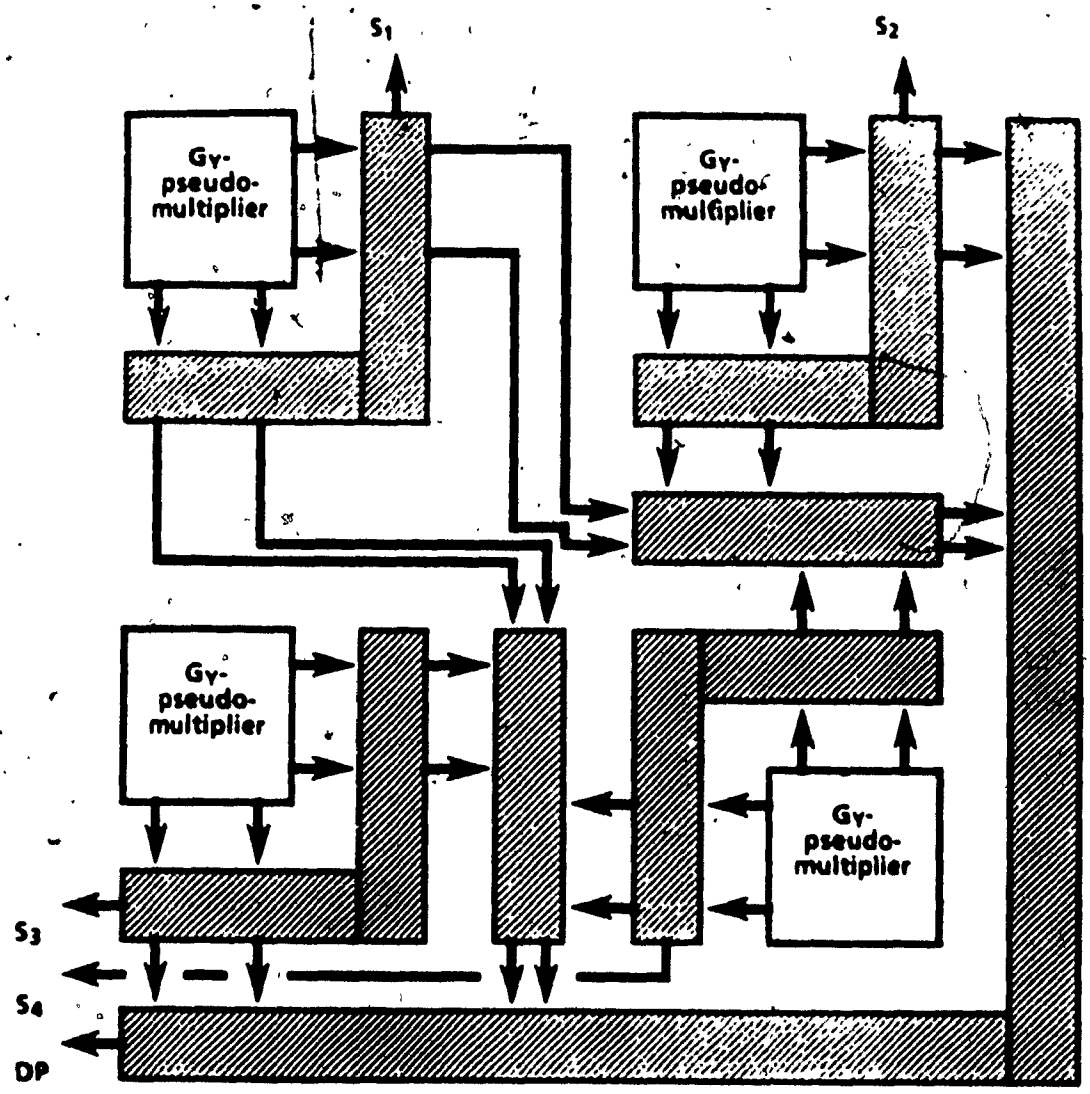


Figure 2-9. ***Single and double precision operation***

CHAPTER 3

A NEW DESIGN APPROACH TO BINARY LOGARITHM AND BINARY DIVISION COMPUTATION

INTRODUCTION

The fast calculation of the base-two logarithm of a binary number has been the object of investigation in the past twenty years. Recently, the advent of Very Large Scale Integration (VLSI) has attracted new interest in the conception of algorithms for numerical computation suitable to be integrated into microcircuits. Integration suggests new criteria for evaluating the merits of an algorithm. These criteria are based not only on ***time complexity*** whose optimality was already a goal in early designs, but also on ***area complexity*** and ***period complexity***. Period complexity refers to the number of clock pulses that have to be issued to the circuit before the next input can be applied.

Almost all the approaches for binary logarithm computation are based on piece-wise approximations.

A positive number N is represented in floating point format by a pair of integers (x,E) such that:

$$N = x\,2^{E}$$
$$(0 \leq x < 1)$$

The base-two logarithm of N can be obtained as follows:

$$\log_2 N = \log_2 x + E$$

Where E is the integer part and $\log_2 x$ is the fractional part called the mantissa. As the integer part E is already available in the number representation, the calculation of $\log_2 N$ consists essentially in the calculation of $\log_2 x$.

Mitchell [MIT62], one of the pioneer in designing circuits for binary logarithm calculation, proposed to use a linear approximation in the interval [1,2] for computing the mantissa of the logarithm:

$$\log_2 x = x-1 \qquad (1 \leq x \leq 2) \qquad (3\text{-}1)$$

The error introduced by assuming a linear approximation of the mantissa is:

$$e = \log_2 x - (x-1) \qquad (3\text{-}2)$$

so that:

$$de/dx = 1.443 * 1/(x-1),$$
$$de/dx = 0 \quad when \quad x = 1.443. \qquad (3\text{-}3)$$

Thus:

$$e_{max} = \log_2(1.443) - 0.443 = 0.086$$

is the maximum error introduced by Mitchell's method. The approximation and error curves for the linear approximation of the base-two logarithm mantissa are shown in Figure 3-1a and Figure 3-1b.

The symbol * represents here arithmetic multiplication.

A few years later, Combat et al. [COM65] have proposed piece-wise linear approximation of the logarithmic curve following the principle shown in Figure 3-2a and Figure 3-2b.

Marino [MAR72] proposed to use second order polynomial approximations for the error function of Figure 3-1b with two curves. An approximation of the binary logarithm is obtained with this approach by adding the approximated error value given by the thick curve in Figure 3-3a to the linear approximation of the logarithm as shown in Figure 3-1a. The overall error curve is represented in Figure 3-3b. The maximum approximation error is $e_{max} = 0.004$.

Sequential (serial) circuits were proposed for implementing the above methods. In many applications such as signal processing, it is important to compute a large number of logarithms in a short interval of time. This makes it attractive to consider purely combinational solutions that can be integrated into a single chip using parallel multipliers. High speed parallel multipliers of N bit factors are now available with time complexity

O(log N) and area complexity $O(N^2)$ as described in Chapter 2. The advent of Very Large Scale Integration (VLSI) suggests to consider designs of combinational circuits for logarithm computation. A solution that is optimal from the point of view of time and area complexity is proposed in this chapter. It is based on parallel multipliers, adders, carry-save adders (CSA), multiplexers and Read Only Memories (ROM).

It will be shown how a great saving in total area complexity and operation time can be obtained by using components for designing recursive multipliers introduced in Chapter 2.

A design is proposed in this thesis which is based on a polynomial approximation of $\log_2(1 + x)$ in the interval:

$$0 \leqq x \leqq 1 \qquad (3\text{-}4)$$

The interval (3-4) can be subdivided into subintervals:

$$([X_0,X_1],[X_1,X_2] \ldots [X_{i-1},X_i] \ldots [X_{l-1},X_l]).$$

The error function in the interval (3-4) is expressed by :

$$e(x) = \log_2(1 + x) - x \qquad (3\text{-}5)$$

A piece-wise polynomial approximation of the logarithm mantissa is obtained using curves of the type:

$$Z_i(x) = A_i x^2 + B_i x + C_i + x \qquad (3\text{-}6)$$

$$(1 \leq i \leq l)$$

In the combinational solution proposed in this chapter, for each interval $[X_{i-1}, X_i]$, e(x) is approximated by the function:

$$Q_i(x) = A_i x^2 + b_i x + c_i.$$

The previously proposed solutions can be seen as a particular cases of the one proposed here. Mitchell proposed l = 1, Marino proposed l = 2, Combet et al. proposed l = 4. Other design approaches suitable for a sequential implementation are proposed in [DEA68a, DEA68b, MAJ73, NIC71]. The choice of a small number of break-points was imposed by hardware constraints especially the one of keeping simple the design of the address selection scheme for coefficients (A_i, B_i, C_i). Garcia and Kubitz [GAR83] have pointed out how this address selection scheme becomes prohibitively complex in general if l is large and the subintervals between two successive break-points have unequal, non-increasing or decreasing size.

In the new design approach proposed here, coefficients A, B, C are stored into Read Only Memory (ROM). An algorithm for ROM address computation is proposed that allows to retrieve the coefficients (A_i, B_i, C_i) with an address selection scheme that is

simple, fast and suitable for VLSI integration even if I is large and the intervals between two successive break-points have unequal, non-increasing or decreasing size. With the approach proposed here an approximation of the logarithmic function with any desired accuracy can be obtained.

It will be shown in this chapter how binary logarithms can be computed with any accuracy using a combinational network with

$$T = O(\log N) \qquad (3\text{-}7)$$

time complexity and with an area complexity

$$A = O(N^2) \qquad (3\text{-}8)$$

Area and time complexities are important features for computing figures of merits of VLSI circuits.

It will be also shown how the combinational network can be transformed into a pipelined version with period complexity

$$P = O(1) \qquad (3\text{-}9)$$

Period complexity is also an important feature for defining the figure of merit of VLSI circuits.

The circuit proposed here will require one clock period with a period proportional to $\log_2 N$ for computing the mantissa of a logarithm of N bits. The solutions proposed so far require a number of clock pulses proportional to the number of segments approximating the logarithm curve and each clock pulse must have a duration that is at best proportional to log N in order to allow a binary multiplication to be executed. In order to make the clock period shorter it is possible to pipeline multipliers and adders in order to get a clock period which is proportional to the switching time of a few AND/OR gates.

The scheme proposed here for computing binary logarithms can be used also for computing antilogarithms with similar figures for time and area complexities.

A binary division circuit can thus be obtained by computing the logarithms of the magnitudes of the dividend and the divisor. The logarithm of the quotient is obtained by subtracting the logarithm of the divisor from the logarithm of the dividend. The magnitude of the quotient can be obtained by computing an antilogarithm.

Brent and Kung [BRE80] have shown that binary addition (and subtraction) can be performed with a circuit having O(log N) time complexity, O(N log N) area complexity and O(1) period complexity. It will be shown how a binary divider consisting of a chain of two logarithm extractors, an adder and a circuit for

antilogarithm computation can be implemented with O(log N) time complexity, $O(N^2)$ area complexity and O(1) period complexity.

PIECE-WISE POLYNOMIAL APPROXIMATION

The problem of obtaining a piece-wise polynomial approximation of the logarithmic curve with any specified accuracy will be considered in this Section.

The curve to be approximated is $\log_2(1 + x)$ in the interval specified in equation (3-4). It will be approximated by taking a linear approximation and adding to it a piece-wise polynomial approximation of the error function (3-5).

The interval specified in equation (3-4) will be subdivided in such a way that a piece- wise polynomial approximation of $\log_2(1 + x)$ will be represented by functions of the type (3-6).

As a result of the approximation, the interval (3-4) will be subdivided into subintervals:

$$([X_0,X_1],[X_1,X_2],\ldots,[X_{i-1},X_i],\ldots,[X_{l-1},X_l]) \qquad (3\text{-}10)$$

where:

$$X_0 = 0 \quad and \quad X_l = 1$$

For each interval $[X_{i-1},X_i]$, $\log_2(1 + x)$ is approximated by the function $Z_i(x)$.

The coefficients A_i, B_i and C_i are stored in three ROMs. Given x, the access time to the coefficients of $Z_i(x)$ corresponding to the interval $[X_{i-1},X_i]$ where x falls into is an important contribution to the total delay of the circuit. Notice that the fastest search would require a number of comparisons proportional to $\log_2 I$ and that each comparison would required a time proportional to $\log_2 N$, where N is the number of bits with which x is represented. This means a contribution to the overall time complexity of $O(\log_2 I * \log_2 N)$ which would prevent us from obtaining a time complexity $O(\log_2 N)$. In order to achieve a fast access to the approximation coefficients, an address decoder scheme will be introduced later in this chapter and has a time complexity of $O(\log_2 N)$.

The break-points of the piece-wise approximation and the coefficients (A_i, B_i, C_i) of the approximating function $Z_i(x)$ in the subinterval (X_{i-1},X_i), $1 \leq i \leq I$, can be obtained by the following bisection algorithm (***AL1***).

ALGORITHM AL1

```
procedure bisection(var bp:array[1..2] of integer;
                    var coefficient:array[1..3] of integer);
var l1,l2 : integer;
begin
    l1: = bp[1];
    l2: = bp[2];
    while l1 <> l2 do
    begin
        leastsquare(coefficient,bp[1],((l1 + l2) div 2));
        if check(coefficient,bp[1], ((l1 + l2) div 2)) then
            l1: = (l1 + l2) div 2
        else
            l2: = (l1 + l2) div 2;
    end;
    bp[2]: = l2;
end;

begin {main program}
    bp[1]: = X0;
    repeat
        bp[2]: = X1;
        bisection(bp,coefficient);
        store(bp,coefficient);
        bp[1]: = bp[2];
    until bp[2] = X1;
end.
```

The algorithm uses the procedure ***leastsquare*** and the function ***check***. The procedure leastsquare gives the coefficients A_i, B_i, C_i of (3-6) for a given subinterval. The function ***check*** returns the value ***true*** if for the given coefficients and subinterval, the

approximation has the desired accuracy. The approximation $Z_i(x)$ of the mantissa is represented by an integer binary number. For this purpose, the coefficients obtained by algorithm ***AL1*** are multiplied by the desired accuracy and converted to integer numbers.

The break-points bp[1] and bp[2] are passed to the procedure and bp[2] is modified by the procedure. The break-point bp[1] is equal to X_{i-1}. The first time the procedure is executed, i is set equal to 1. At the beginning of the procedure execution, bp[2] is always equal to X_I. The procedure finds a value for bp[2] that corresponds to the end X_i of the maximum length interval $[X_{i-1},X_i]$ in which an approximation with the desired accuracy has been found.

If $X_i < X_I$ then the procedure is called again with $i = i + 1$, $bp[1] = x_{i-1}$ and $bp[2] = X_I$.

Table 3-1 contains the break-points and coefficients for computing the logarithm of a number represented by N = 16 bits with an accuracy of the order of magnitude of the least-significant-bit. The values are represented by integer numbers in the decimal code. In order to compute the true values of the break-points, the numbers in Table 3-1 have to be divided by 32768. Figure 3-4a and Figure 3-4b show the logarithmic curve and its piece-wise approximation obtained with the above described method.

More details on bisection algorithms can be found in [BUR78].

An approximation $Z_1(x)$ of $\log_2(1 + x)$ can be computed by a combinational network described by the following concurrent algorithm ***AL2***.

Algorithm AL2

```
function log(x:real):real;
var address:integer;
    Y1, Y2, Y3, Y4, Y5:real;
begin
    cobegin
        address: = ADR(x);
        Y1 = x*x;
    coend;
    cobegin
        Y2: = Y1*A[address];
        Y3: = B[address]*x;
        PSA(Y4,Y5,Y2,Y3,x,C[address]);
    coend;
    log: = Y4 + Y5;
end;
```

The function ***ADR***(x) computes the addresses of the coefficients A_i, B_i and C_i for a given x. The circuit implementing it will be described in the next section.

A scheme of a circuit implementing algorithm ***AL2*** is shown in Figure 3-5. The network requires 4 levels of computation independently from the number of break-points.

The coefficients (A_i, B_i, C_i) can be negative, thus some of the multipliers and adders have to be able to handle two's-complement numbers.

The procedure ***PSA*** represents a Pseudo-Adder, that is a circuit with 4 inputs, in our case Y_3, Y_2, x, C and two outputs Y_4 and Y_5, related by the following equation:

$$Y_4 + Y_5 = Y_2 + Y_3 + x + C \qquad (3\text{-}11)$$

It is well known that this operation can be performed by a network of Carry-Save Adders (CSA) with a delay that is independent from N.

It has been shown in Chapter 2 that a considerable saving in area complexity with great improvement in the operation speed of a parallel multiplier can be obtained by using Pseudo-Multipliers (PM) of two's complement numbers. PMs are binary multipliers with two output numbers whose sum is equal to the product. PMs can be used for computing two numbers Y_{21} and Y_{22} whose sum is Y_2 and other two numbers Y_{31} and Y_{32} whose sum is Y_3, i. e.:

$$\begin{aligned} Y_{21} + Y_{22} &= Y_2 \\ Y_{31} + Y_{32} &= Y_3 \end{aligned} \qquad (3\text{-}12)$$

If pseudo-multipliers are used, then two numbers are obtained for each multiplication and PSA has to compute the following sum:

$$Y_4 + Y_5 = Y_{21} + Y_{22} + Y_{31} + Y_{32} + X + C \qquad (3\text{-}13)$$

Chapter 2 proposes a circuit based on CSAs for computing the (3-13) with a time delay equal to three times the delay of a CSA which is independent from N.

With such an approach, the part of Figure 3-5 below the dashed line can be substituted by the circuit sketched in Figure 3-6.

A detailed description of the design of PMs can be found in Chapter 2.

As described in Chapter 2, PMs have an area complexity $O(N^2)$ and a time complexity O(logN), PSA has an area complexity O(4N) and a time complexity O(3). According to [8RE80] ADDER can be implemented with a circuit having an area complexity O(N log N) and a time complexity O(log N).

Chapter 2 also describes the use of two's complement notation and shows how MULTIPLIER-1 can be implemented with a PM and an ADDER with a time complexity O(logN) and an area complexity $O(N^2)$.

ADDRESS DECODER

The address generator design is an interesting contribution of this thesis and will be described in detail in this Section. It will be shown how this design approach does not deteriorate the area complexity figure $O(N^2)$ nor the time complexity $O(\log N)$ of the other components.

The sign of the following differences are computed in parallel:

$$S_i = \text{sign}(x - X_i) \qquad (3\text{-}14)$$

$$i = (1, \ldots, I\text{-}1)$$

A positive sign is represented by a 0. A negative sign is represented by a 1.

The address is obtained as follow:

$$ADR(x) = \sum_{i=1}^{I-1} \overline{S_i} \qquad (3-15)$$

assuming that the first interval has address 0.

One possible solution consists in computing, as shown in Figure 3-7a, the address of coefficients (A_i, B_i, C_i) by I-1 carry-

lookahead networks and a parallel binary counter that can be implemented with a tree of full-adder cells.

Since a carry-lookahead network has an area complexity O(N log N), and I of these circuits are required, the overall area complexity is O(IN log N). As this complexity can be greater then $O(N^2)$, a better solution has to be found.

The bits S_i ($1 \leq i \leq I-1$) can be obtained by comparisons between x and the break-points X_i. As the values of the break-points X_i are fixed, the design of comparators can be simplified. Let us call ***Pseudo-Comparator*** (PC) a simplified comparator having a N bit input number and one bit output. PC compares a variable number x with a fixed number X_i. Furthermore, the value x can be compared with all X_i simultaneously and ADR(x) can be obtained by a circuit that performs a logarithmic search on a binary tree where the nodes are pseudo-comparators.

The design of special fast pseudo-comparators will be now considered.

Let us assume, as an example, that a break-point BP = X_i is a 16 bit number:

$$BP = b_{15}b_{14} \ldots b_0 \qquad (3\text{-}16)$$

Let divide BP into 4 groups of 4 bits each:

$$BP = B_3B_2B_1B_0 \qquad (3\text{-}17)$$

where

$$B_3 = b_{15}b_{14}b_{13}b_{12}$$
$$B_2 = b_{11}b_{10}b_9b_8$$
$$B_1 = b_7b_6b_5b_4 \qquad (3\text{-}18)$$
$$B_0 = b_3b_2b_1b_0$$

In a similar way, let subdivide x into four groups as follows:

$$x = W_3W_2W_1W_0 \qquad (3\text{-}19)$$

Notice that x and BP are positive binary fractions in this particular application. The design of PCs starts with the design of cells that compare W_i with B_i and generate outputs L_i and E_i according to the following algorithm PC1.

ALGORITHM PC1

```
begin
    if W_i < B_i then
        L_i = 1
    else
        L_i = 0;
    if W_i = B_i then
        E_i = 1
    else
        E_i = 0;
end
```

L stays for ***Lower***, E for ***Equal***.

In order to find if x is less than BP, the following expression is used:

$$x \text{ is less than or equal to } BP = L_3 + E_3 \cdot L_2 + E_3 \cdot E_2 \cdot L_1 + E_3 \cdot E_2 \cdot E_1 \cdot L_0 + E_3 \cdot E_2 \cdot E_1 \cdot E_0 \quad (3\text{-}20)$$

+ represents logical disjunction and · represents logical conjunction.

The circuits for implementing L_i and E_i will be presented in Appendix 3-A.

The pseudo-comparators have constant time and area complexity. Their outputs have to be interconnected in such a way that they implement the function (3-20).

The 4-bit pseudo-comparators can be connected with the scheme of Figure 3-7b.

The area complexity A(PC) of such a scheme grows with N because gates have limited ***fan-in*** and each PC handles a fixed number of bits. Area complexity can be computed as follows:

$$A(PC) = \frac{N}{4}A_{pc} + \frac{N}{4}(A_{and} + A_{wires}) + \frac{N}{8}(A_{or} + A_{and} + A_{wires}) + \ldots =$$

$$= \frac{N}{4}(A_{pc} + A_{and} + A_{wires}) + (A_{or} + A_{and} + A_{wires})(\frac{1}{8} + \frac{1}{16} + \ldots + \frac{1}{N})$$

$$= \frac{N}{4}(A_{pc} + A_{and} + A_{wires}) + (A_{or} + A_{and} + A_{wires})(2^{-3} + 2^{-4} + \ldots + 2^{-\log N})$$

$$\alpha \triangleq \frac{A_{or} + A_{and} + A_{wires}}{A_{pc} + A_{and} + A_{wires}} \ll 1 \qquad (3-21)$$

$$A(PC) = N(A_{pc} + A_{and} + A_{wires})(\frac{1}{4} + \alpha \sum_{k=3}^{\log N} 2^{-k}) \qquad (3-22)$$

The complexity A(PC) is O(N) as far as:

$$\frac{1}{4} + \alpha \sum_{k=3}^{\log N} 2^{-k} \leq 1 \qquad (3-23)$$

Under the condition (3-23) that is true for practical value of N, the area complexity of I comparators is:

$$A_{comp} = O(IN) < O(N^2) \qquad (3-24)$$

The time complexity of each comparator is:

$$T_{comp} = O(\log_6 N). \qquad (3-25)$$

The base of the log in (3-25) depends on the ***fan-in*** of an or gate. A practical value of 6 can be assumed for the ***fan-in***. As all the comparators operate in parallel, the (3-25) represents the contribution of the comparators to the overall time complexity.

Let cmp_j be the output of the i^{th} comparator, ADR(x) can be obtained as follows:

$$ADR(x) = \sum_{i=0}^{l-1} cmp_j \qquad (3-26)$$

The bits adr_j of ADR can be obtained by l combinational circuits derived by Table 3-2. From Table 3-2, the following functions are derived:

$$adr_2 = cmp_3$$

$$adr_1 = cmp_1 \cdot \overline{cmp_3}$$

$$adr_0 = \overline{cmp_0} \cdot \overline{cmp_1} + \overline{cmp_2} \cdot \overline{cmp_3} + cmp_4$$

The functions ADR are particularly simple because the cmp_i have to respect the constraint that all the ones and the zeros must be contiguous.

The time complexity contribution introduced by these functions is constant with I and N and the area complexity contribution is O(log I).

A more systematic approach for large value of N consists in computing the adr bits with a network of multiplexers (MUX) as shown in Figure 3-10.

Each network of multiplexers has (I-1) inputs consisting of the cmp_j bits. Two cmp_j feed the first MUX, other 4 cmp_j bits feed the second MUX and so on. The correspondence between MUX inputs and cmp_j is determined by the algorithm ***order*** given in the following:

```
ALGORITHM order (N : integer);
var step, i, j, sum : integer;
begin
    step : = N;
    for i: = 1 to log2N do
    begin
        sum : = step div 2;
        for j: = 1 to 2^(i-1) do
        begin
            writeln(sum: 5);
            sum : = sum + step;
        end;
        step : = step div 2;
    end;
end;
```

The successive values assumed by ***sum*** make a sequence of indices for the cmp_i that have to be applied to the multiplexer inputs in Figure 10. For N = 16, the sequence $(i_1, i_2,...,i_{15})$ of cmp indices is the following (8, 4, 12, 2, 6, 10, 14, 1, 3, 5, 7, 9, 11, 13, 15).

The overall area complexity of the multiplexer networks is $O(I^2)$ which is less than O(IN).

COMPLEXITY

Multipliers and PMs can be implemented with an area complexity $O(N^2)$, a time complexity $O(\log N)$ and a period complexity $O(1)$ according to the design approach proposed in Chapter 2.

Using the special design of the pseudo-comparator, it is possible to design the address generator with an area complexity of (IN), where I is the number of break-points in the approximated curves. Since I is less than N, the overall area complexity of the circuit is still $O(N^2)$. The time complexity of the address generator is $O(\log I)$ and the time complexity of the entire circuit is $O(\log N)$.

PIPELINING

The network can be easily pipelined by introducing memory elements as shown in Figure 3-8 and Figure 3-9. From the inspection of Figure 3-8 and Figure 3-9, it can be easily derived that the period complexity of the network is $P = O(1)$.

COMPUTATION OF THE ANTILOGARITHM AND OF BINARY DIVISION

Let us consider two numbers X and Y represented in sign and magnitude with the magnitude represented in floating point:

$$X = S_x M_x 2^{E_x} \tag{3-27}$$

$$Y = S_y M_y 2^{E_y}$$

Where M stays for Mantissa and E for Exponent.

The logarithm of the magnitude of the quotient:

$$Z = X/Y \tag{3-28}$$

can be obtained as follows:

$$\log |Z| = (E_x - E_y) + (\log M_x - \log M_y) \tag{3-29}$$

While the signs of Z, S_z, can be obtained by the following boolean equation:

$$S_z = S_x \oplus S_y$$

Figure 3-11a and Figure 3-11b shows a scheme for binary division. Notice that it is not necessary to perform the final addition for log M_x and log M_y. Rather the two numbers representing each logarithm can be added togheter in a PSA. As M_y has to be subtracted and all the numbers are in two's complement at the output of the PMs, each bit of each output of the PM for log M_y is complemented and two ones are added to the PSA. In this way the two numbers whose sum is log M_y are two's complemented before being added to the numbers whose sum is log M_x.

Suitable shifts can be performed before extracting the logarithms in order to ensure that $M_x \geq M_y$.

The antilogarithm of log M_z can be computed using the same approach and hardware introduced for the computation of the logarithm.

The approach consist in using piece-wise polynomial approximation of the error function

$$a(m) = 2m - 2^m \qquad (3\text{-}30)$$

Break-point and coefficients for the approximation of a(m) when N = 16 are given in Table 3-3.

CONCLUDING REMARKS ON THE LOGARITHM SOLUTION

A combinational circuit for computing the binary logarithm of an N bits number has been proposed with an area complexity $O(N^2)$. The structure has a time complexity O(log N) and can be pipelined with a period complexity O(1).

Several figures of merit (FM) have been proposed the ones that have mostly been discussed are:

$$FM_a = AT^2(PE)P^2$$
$$FM_b = A(PE)$$
$$FM_c = A(PE)T$$

The combinational circuit proposed here, have the following figures of merit:

$$FM_a = N^2 \log^2 N$$
$$FM_b = N^2 \qquad (3\text{-}31)$$
$$FM_c = N^2 \log N$$

The same approach can be used for computing the antilogarithm. Two circuits for computing the logarithm and one circuit for computing the antilogarithm can be used for performing a binary division with the same figures of merit as equations (3-31).

TABLE 3-1

Coefficients and break-points for computing the logarithm of 16 bit numbers.

SEGMENT	RANGE	X	COEFFICIENT	$\log_2(1 + x)$
0	0	0.0000	$A_0 = -21679$	0.00000
	2923	0.0892	$B_0 = 14438$	0.12327
			$C_0 = 0$	
1	2923	0.0892	$A_1 = -17983$	0.12327
	6718	0.20502	$B_1 = 13756$	0.26906
			$C_1 = 33$	
2	6718	0.20502	$A_2 = -14611$	0.26906
	11153	0.34036	$B_2 = 12367$	0.42262
			$C_2 = 178$	
3	11153	0.34036	$A_3 = -11794$	0.42262
	16148	0.49280	$B_3 = 10450$	0.57802
			$C_3 = 506$	
4	16148	0.49280	$A_4 = -9520$	0.57802
	21644	0.66052	$B_4 = 8213$	0.73164
			$C_4 = 1058$	

5	21644	0.66052	$A_5 = -7694$	0.73164
	27755	0.84702	$B_5 = 5803$	0.88520
			$C_5 = 1855$	
6	27755	0.84702	$A_6 = -6393$	0.88520
	32768	1.00000	$B_6 = 3625$	1.00000
			$C_6 = 2768$	

TABLE 3-2

Address generation table.

$cmp_0 \ldots cmp_{l-1}$	$adr_2 \ldots adr_0$
00000	000
10000	001
11000	010
11100	011
11110	100
11111	101

TABLE 3-3

Break-points and coefficients for a polynomial approximation of $a(m) = 2m - 2^m$.

SEGMENT	RANGE	X	COEFFICIENT	2^x
0	0	0.0000	$A_0 = -16504$	1.0000
	4443	0.1356	$B_0 = 20151$	1.0985
			$C_0 = 0$	
1	4443	0.1356	$A_1 = -18301$	1.0985
	9767	0.2981	$B_1 = 20662$	1.2295
			$C_1 = -38$	
2	9767	0.2981	$A_2 = -20354$	1.2295
	14499	0.4425	$B_2 = 21876$	1.3590
			$C_2 = -219$	
3	14499	0.4425	$A_3 = -22321$	1.3590
	18495	0.5644	$B_3 = 23603$	1.4788
			$C_3 = -599$	
4	18495	0.5644	$A_4 = -24519$	1.4788
	23375	0.7133	$B_4 = 26113$	1.6346
			$C_4 = -1317$	

5	23375	0.7133	A_5 = -27083	1.6346
	27900	0.8514	B_5 = 29765	1.8043
			C_5 = -2619	
6	27900	0.8514	A_6 = -29699	1.8043
	32096	0.9795	B_6 = 34215	1.9718
			C_6 = -4513	
7	32096	0.9795	A_6 = -31264	1.9718
	32768	1.0000	B_6 = 37214	2.0000
			C_6 = -5950	

APPENDIX 3-A

Table 3-A1 is a truth table for L_i and E_i as functions of four bits of X. There are 16 such functions for each L_i and E_i. The pair of functions used in a particular circuit depends on 4 bits of the constant that has to be compared with X.

TABLE 3-A1

CONSTANT

X

	0000		0001		0010		0011		0100		0101		0110		0111		1000		1001		1010		1011		1100		1101		1110		1111	
	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E	L	E
0000	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0001	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0010	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0011	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0100	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0101	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0110	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0
1010	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1	0
1011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0	1
1100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1
1101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1
1110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
1111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

For each combination of bits of the constant acting as specification bits, the following logic functions have been derived.

0000

$$L_0 = 0$$

$$E_0 = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot \overline{X_0}$$

0001

$$L_1 = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot \overline{X_0}$$

$$E_1 = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot X_0$$

0010

$$L_2 = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1}$$

$$E_2 = \overline{X_3} \cdot \overline{X_2} \cdot X_1 \cdot \overline{X_0}$$

0011

$$L_3 = \overline{X_3} \cdot \overline{X_2} \cdot (\overline{X_1} + \overline{X_0})$$

$$E_3 = \overline{X_3} \cdot \overline{X_2} \cdot X_1 \cdot X_0$$

0100

$$L_4 = \overline{X_3} \cdot \overline{X_2}$$

$$E_4 = \overline{X_3} \cdot X_2 \cdot \overline{X_1} \cdot \overline{X_0}$$

0101

$$L_5 = \overline{X_3} \cdot (\overline{X_2} + \overline{X_1} \cdot \overline{X_0})$$

$$E_5 = \overline{X_3} \cdot X_2 \cdot \overline{X_1} \cdot X_0$$

0110

$$L_6 = \overline{X_3} \cdot (\overline{X_2} + \overline{X_1})$$

$$E_6 = \overline{X_3} \cdot X_2 \cdot X_1 \cdot \overline{X_0}$$

0111

$$L_7 = \overline{X_3} \cdot (\overline{X_2} + \overline{X_1} + \overline{X_0})$$

$$E_7 = \overline{X_3} \cdot X_2 \cdot X_1 \cdot X_0$$

1000

$$L_8 = \overline{X_3}$$

$$E_8 = X_3 \cdot \overline{X_2} \cdot \overline{X_1} \cdot \overline{X_0}$$

1001

$$L_9 = \overline{X_3} + \overline{X_2} \cdot \overline{X_1} \cdot \overline{X_0}$$

$$E_9 = X_3 \cdot \overline{X_2} \cdot \overline{X_1} \cdot X_0$$

1010

$$L_{10} = \overline{X_3} + \overline{X_2} \cdot \overline{X_1}$$

$$E_{10} = X_3 \cdot \overline{X_2} \cdot X_1 \cdot \overline{X_0}$$

1011

$$L_{11} = \overline{X_3} + \overline{X_2} \cdot (\overline{X_1} + \overline{X_0})$$

$$E_{11} = X_3 \cdot \overline{X_2} \cdot X_1 \cdot X_0$$

1100

$$L_{12} = \overline{X_3} + \overline{X_2}$$

$$E_{12} = X_3 \cdot X_2 \cdot \overline{X_1} \cdot \overline{X_0}$$

1101

$$L_{13} = \overline{X_3} + \overline{X_2} + \overline{X_1} \cdot \overline{X_0}$$

$$E_{13} = X_3 \cdot X_2 \cdot \overline{X_1} \cdot X_0$$

1110

$$L_{14} = \overline{X_3} + \overline{X_2} + \overline{X_1}$$

$$E_{14} = X_3 \cdot X_2 \cdot X_1 \cdot \overline{X_0}$$

1111

$$L_{15} = \overline{X_3 \cdot X_2 \cdot X_1 \cdot X_0}$$

$$E_{15} = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot \overline{X_0}$$

Figure 3-A1 shows the implementation of functions L_{11} and E_{11}.

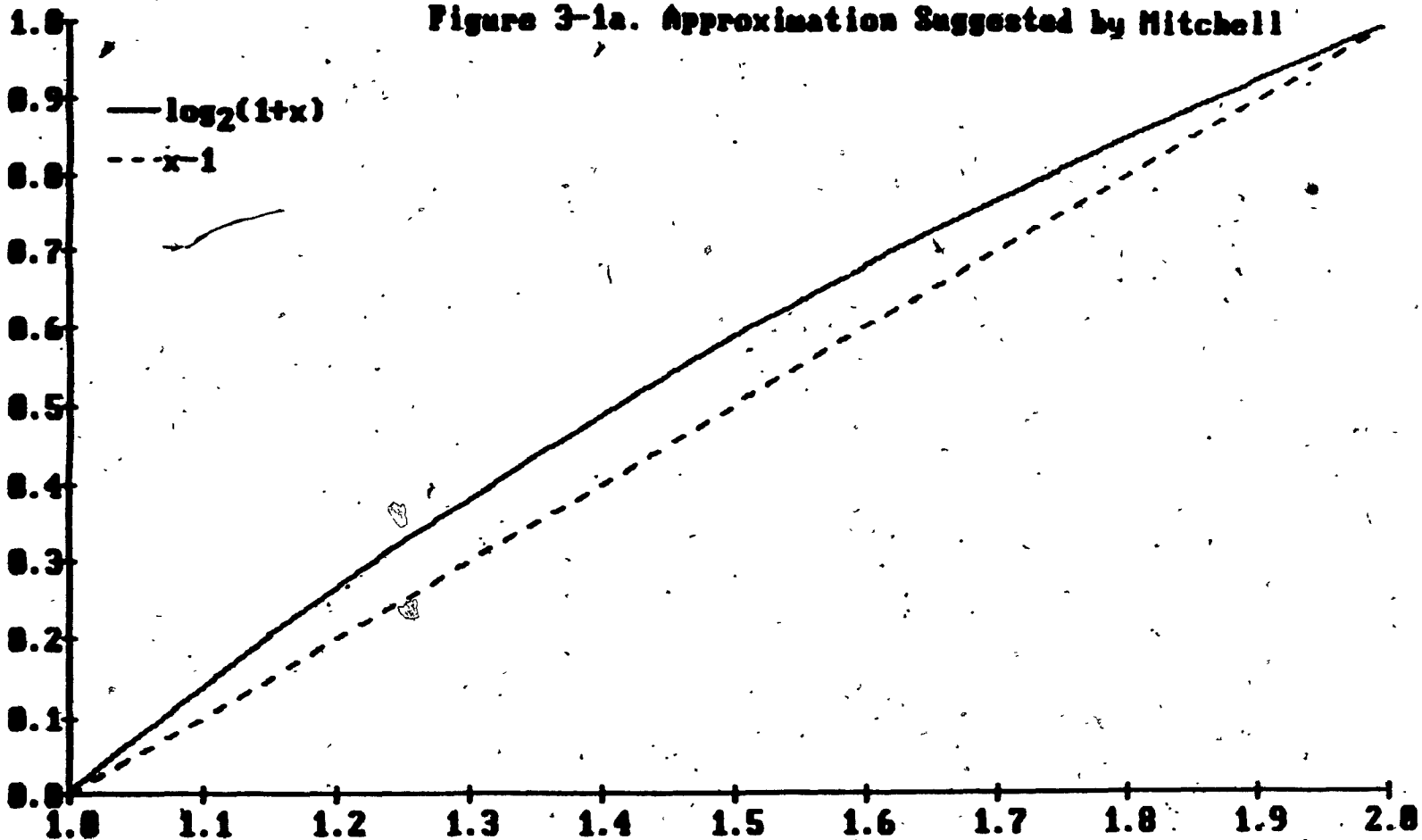


Figure 3-1a. Approximation Suggested by Mitchell

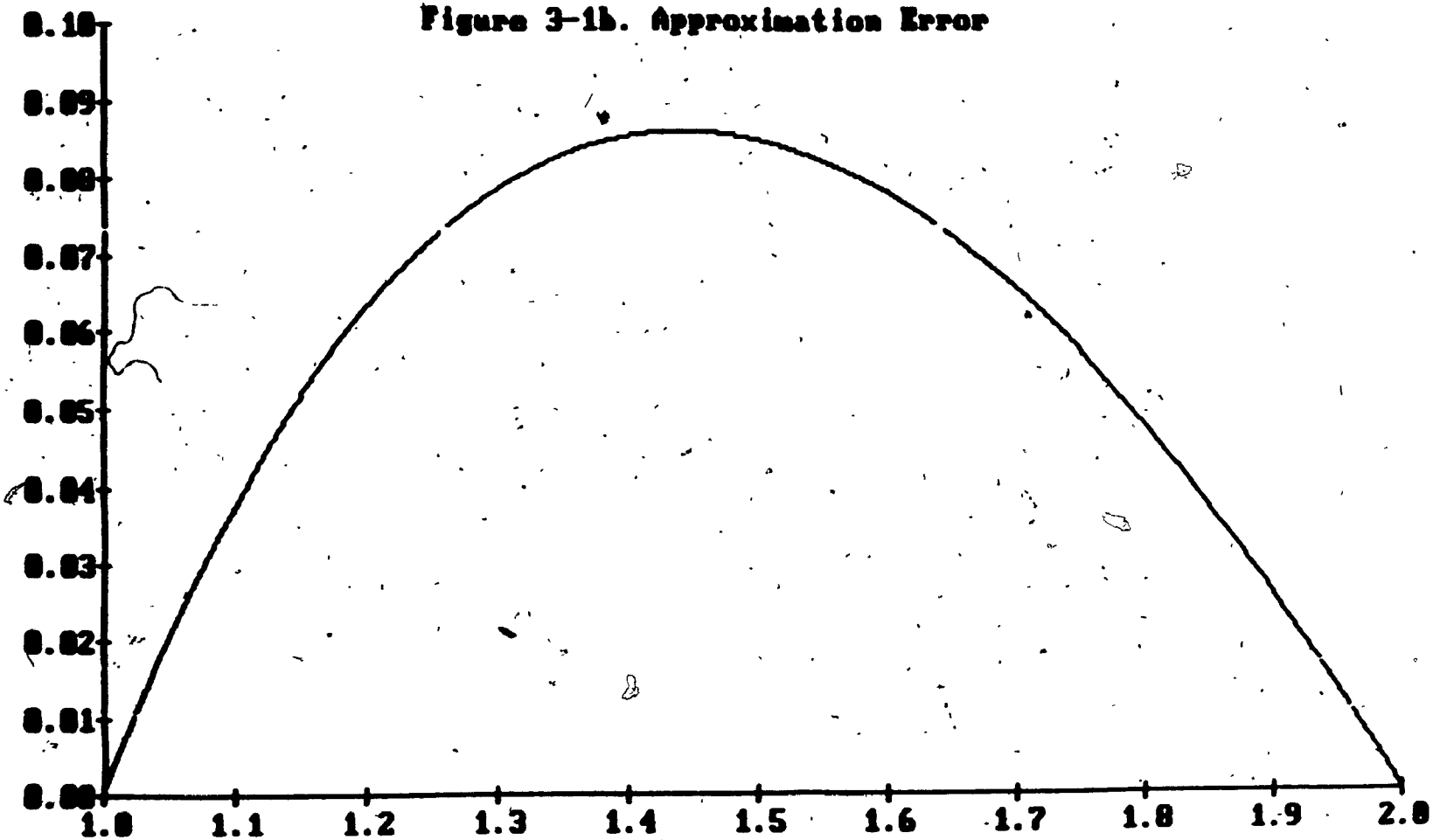
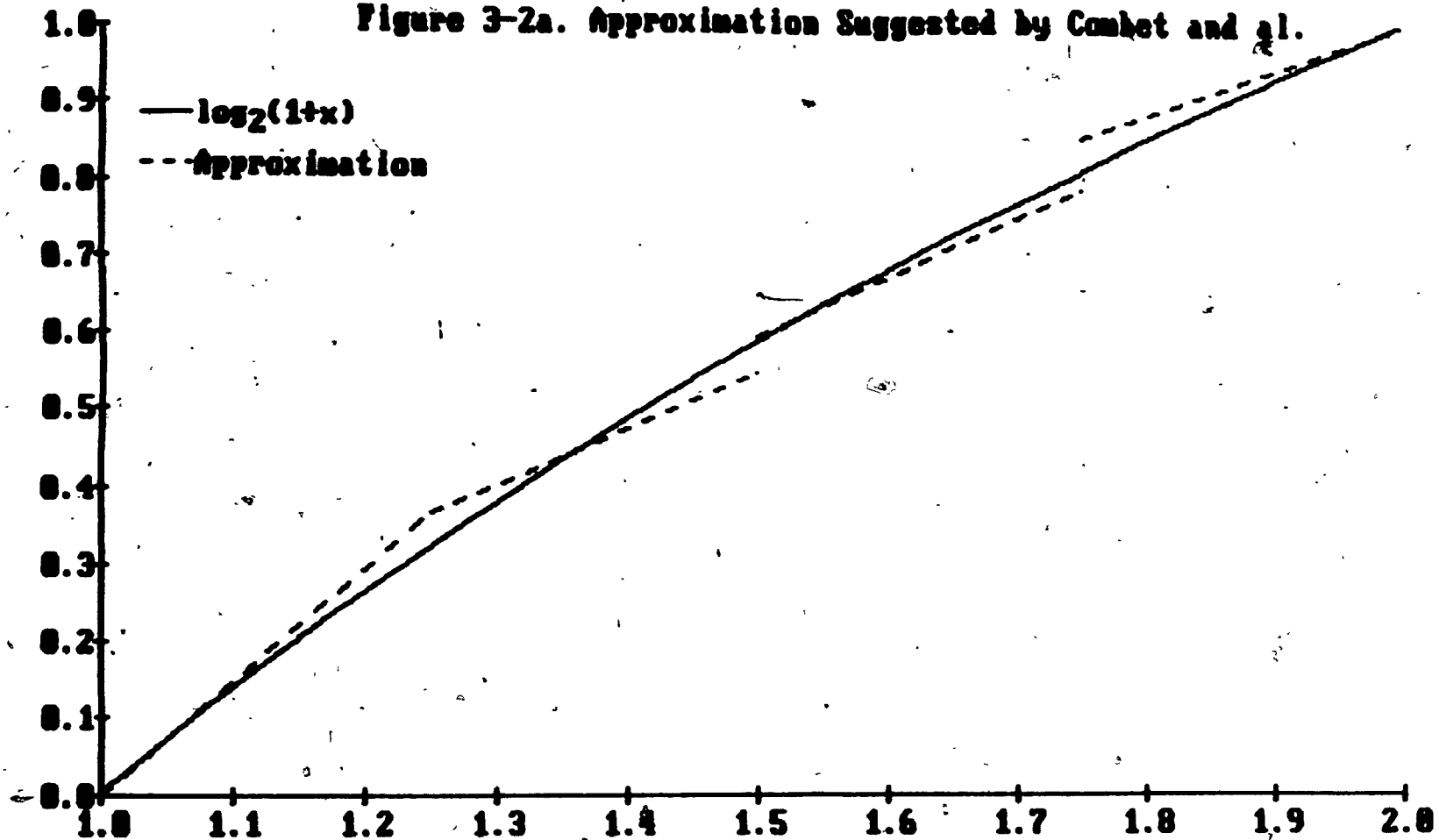


Figure 3-1b. Approximation Error

Figure 3-2a. Approximation Suggested by Combet and al.



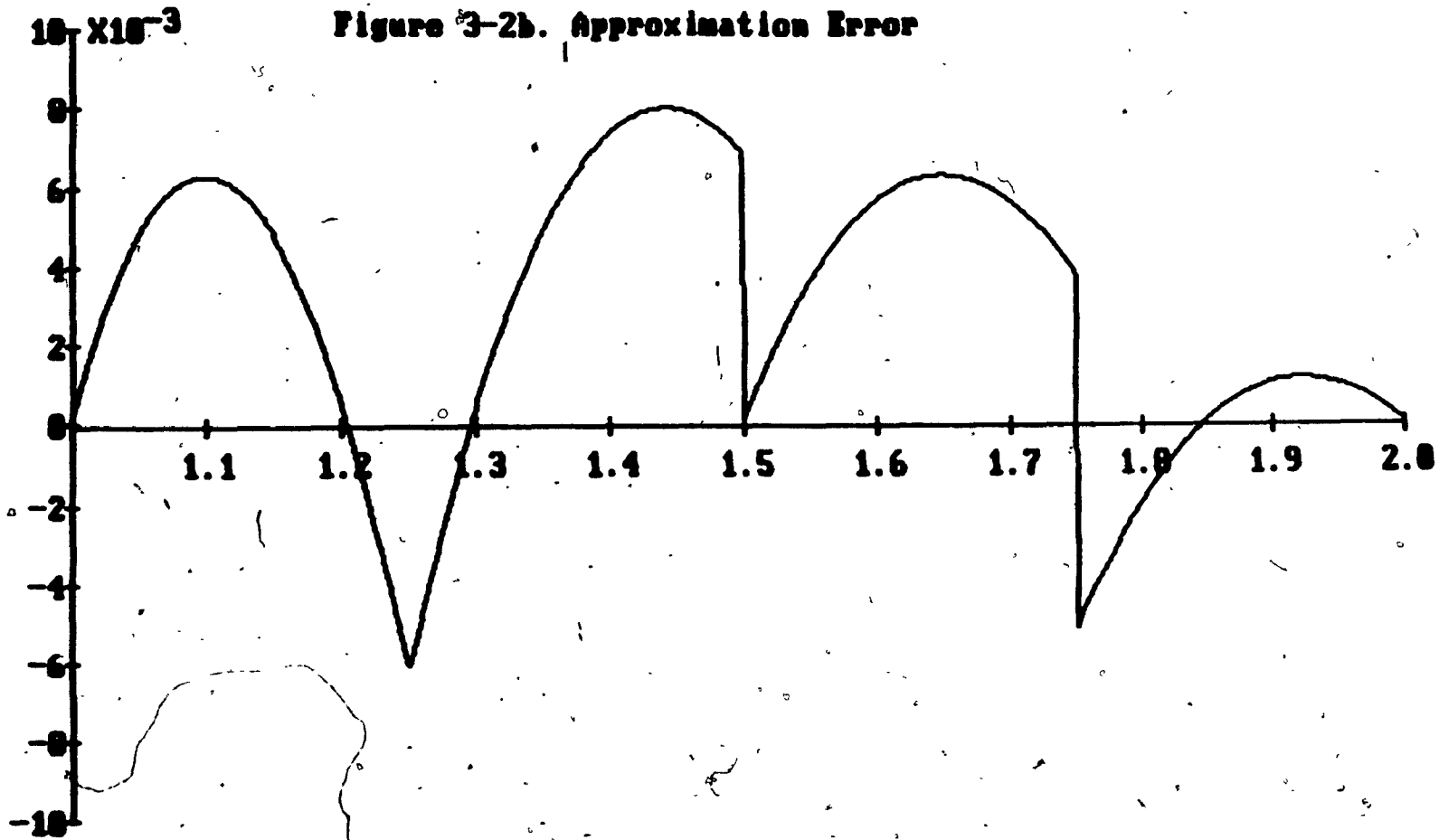


Figure 3-2b. Approximation Error

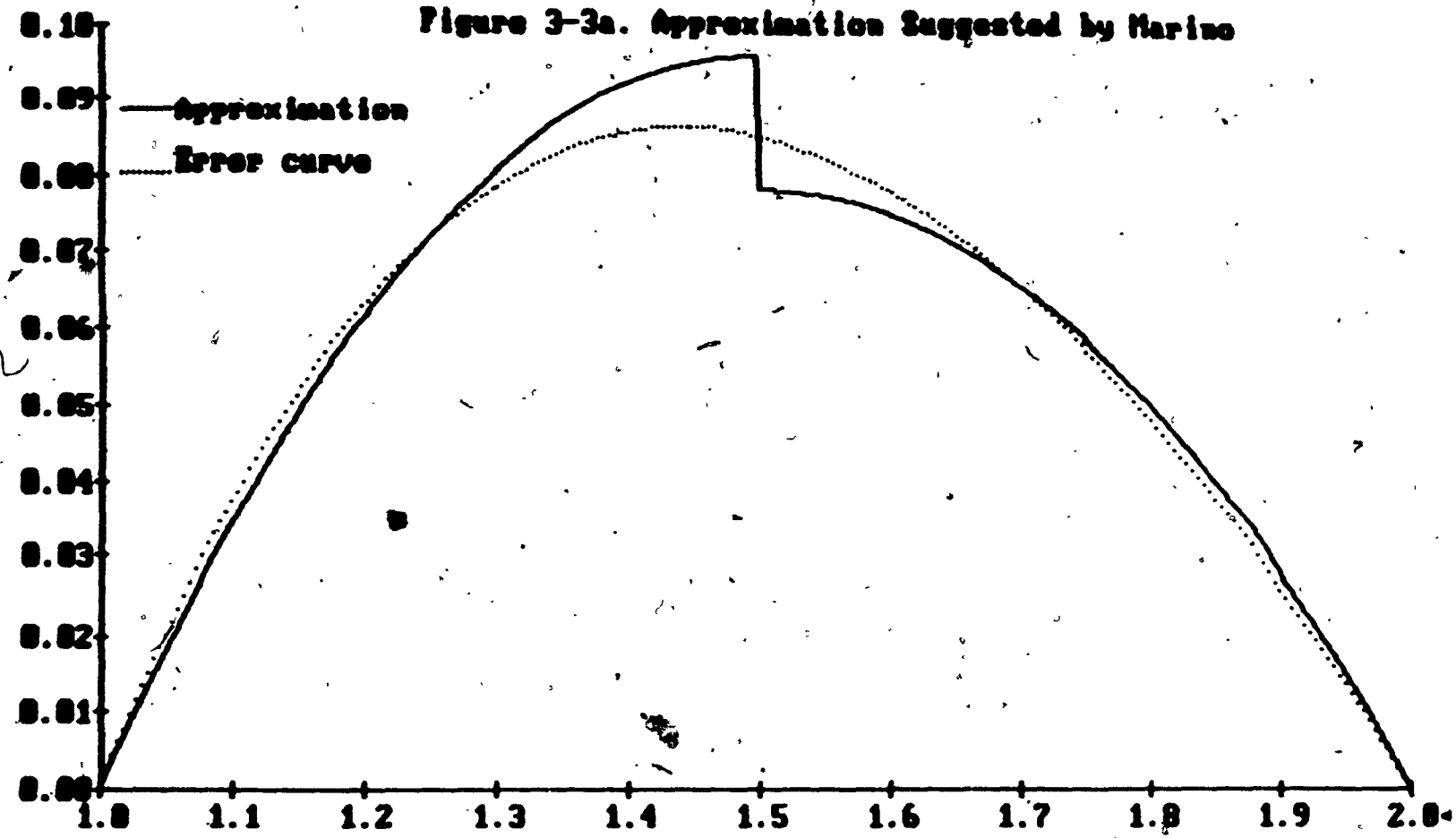


Figure 3-3a. Approximation Suggested by Marino

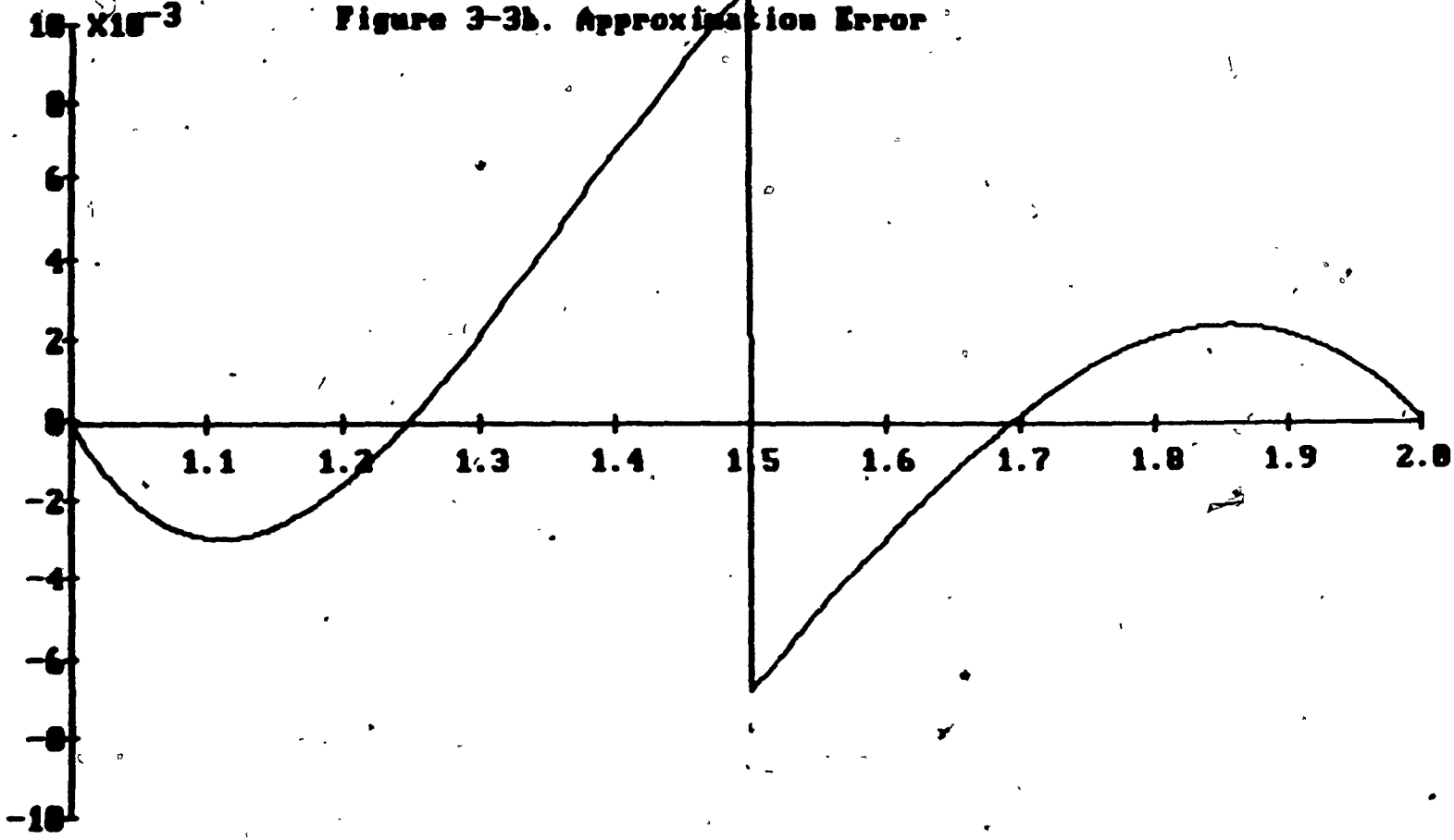
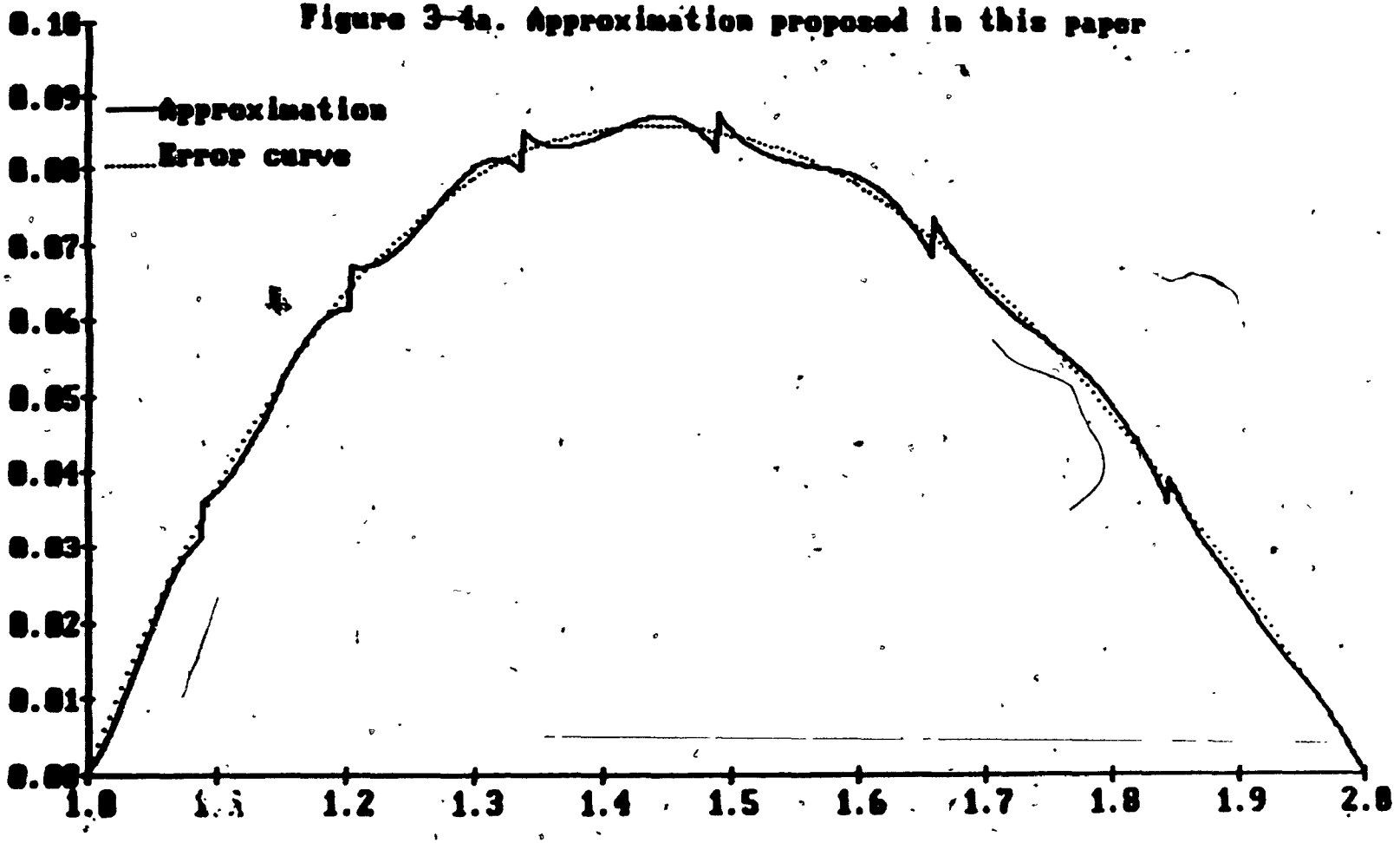


Figure 3-3b. Approximation Error

Figure 3-4a. Approximation proposed in this paper



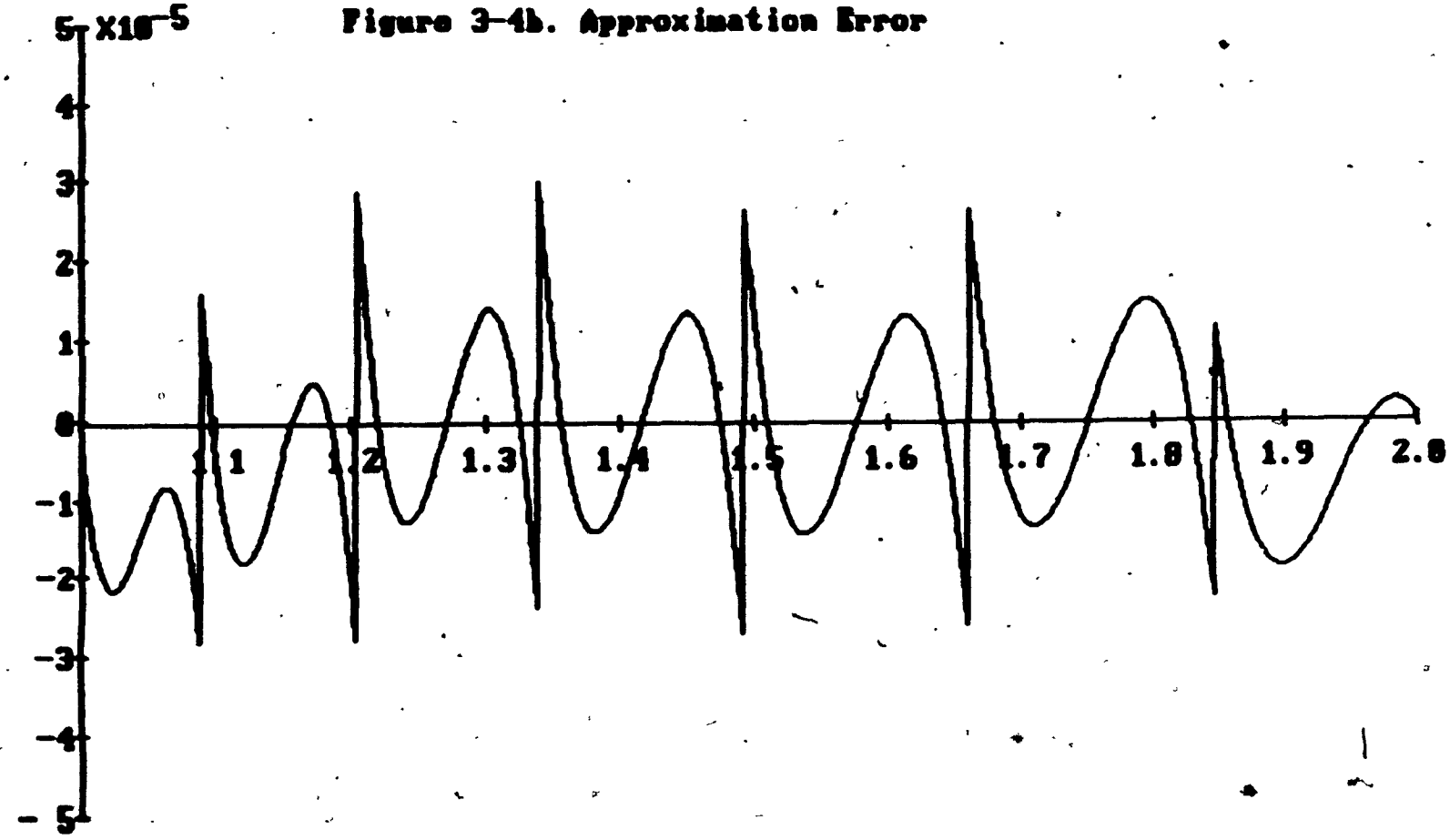


Figure 3-4b. Approximation Error

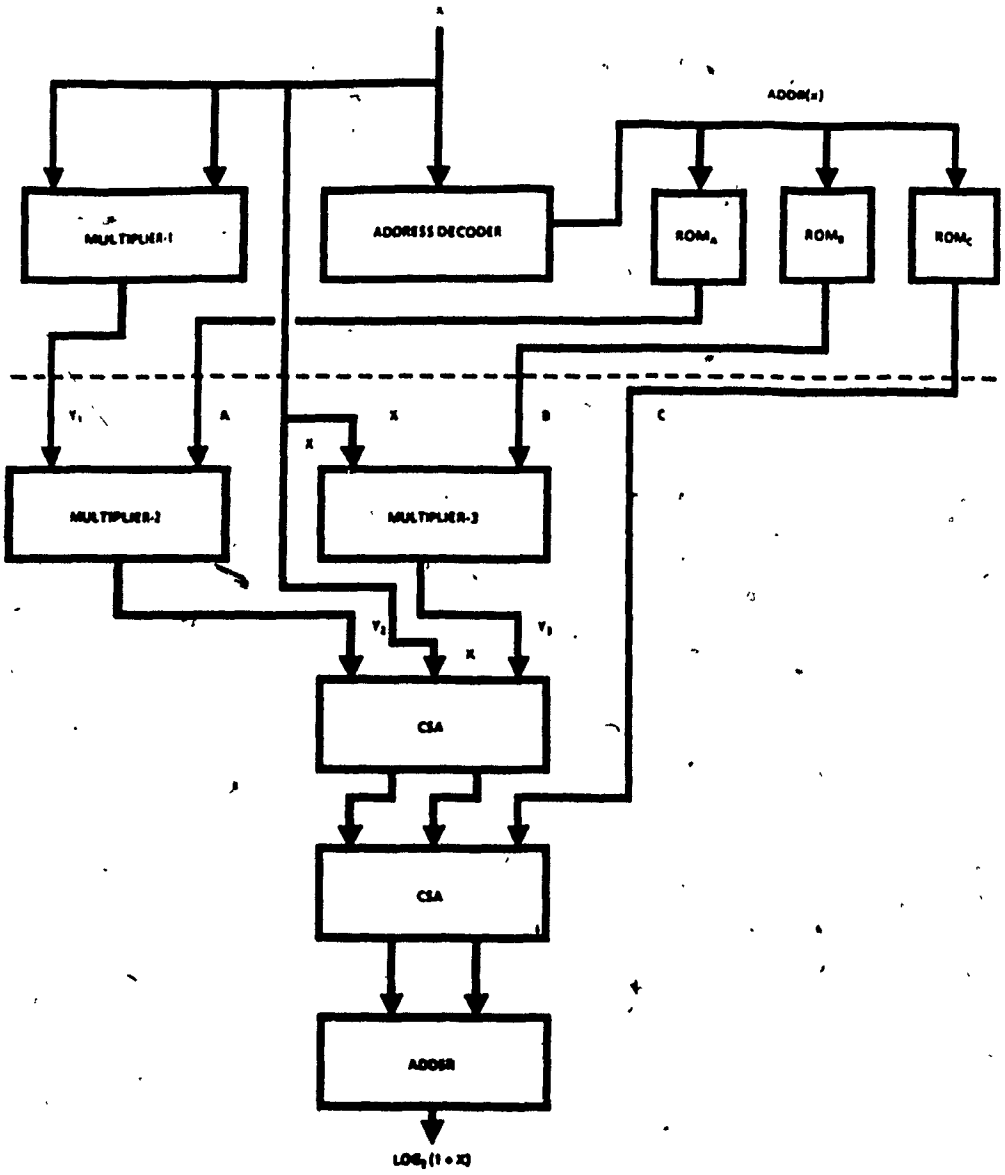


Figure 3-5. ***Combinational circuit***

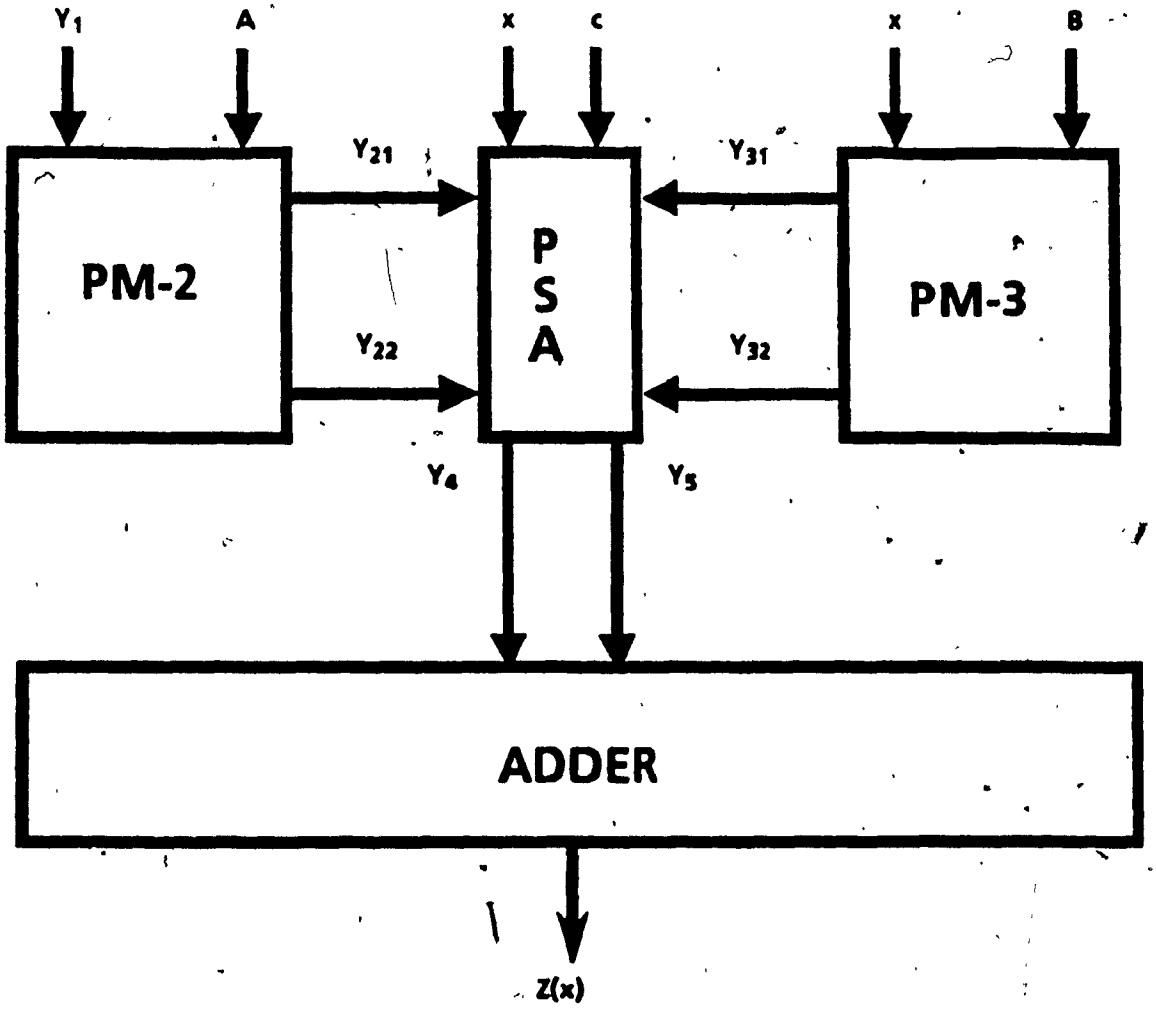


Figure 3-6. ***Pseudo-Multiplier Circuit***

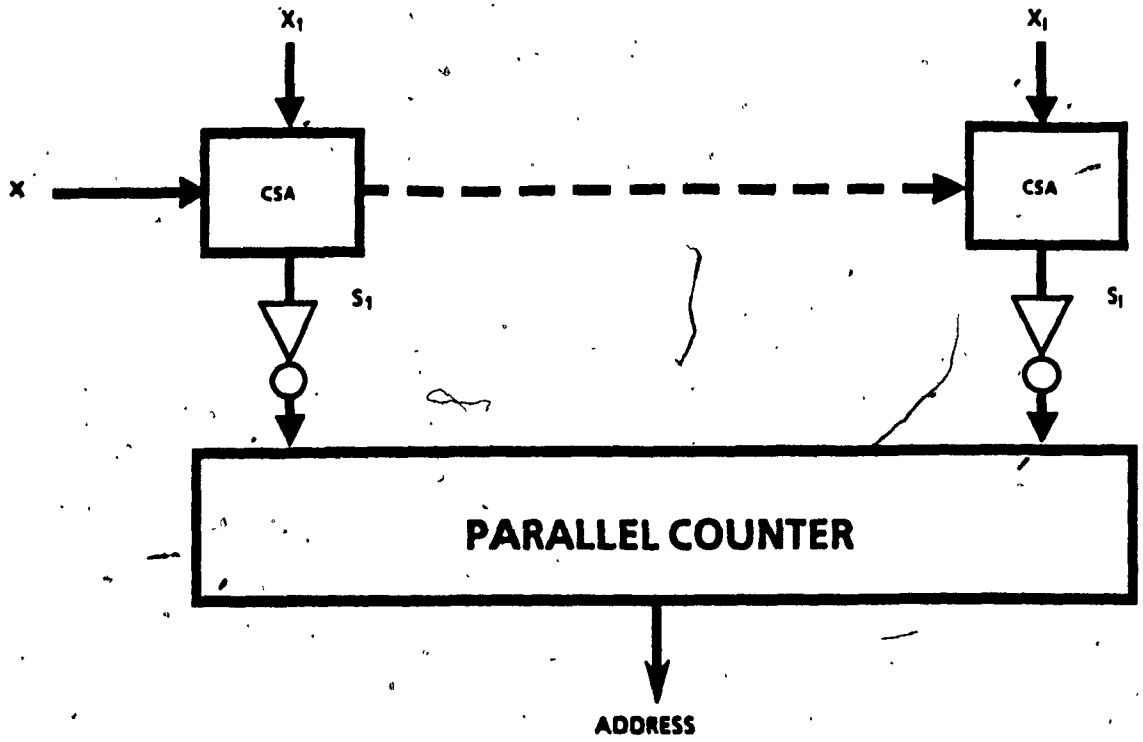


Figure 3-7a. *Address Decoder*

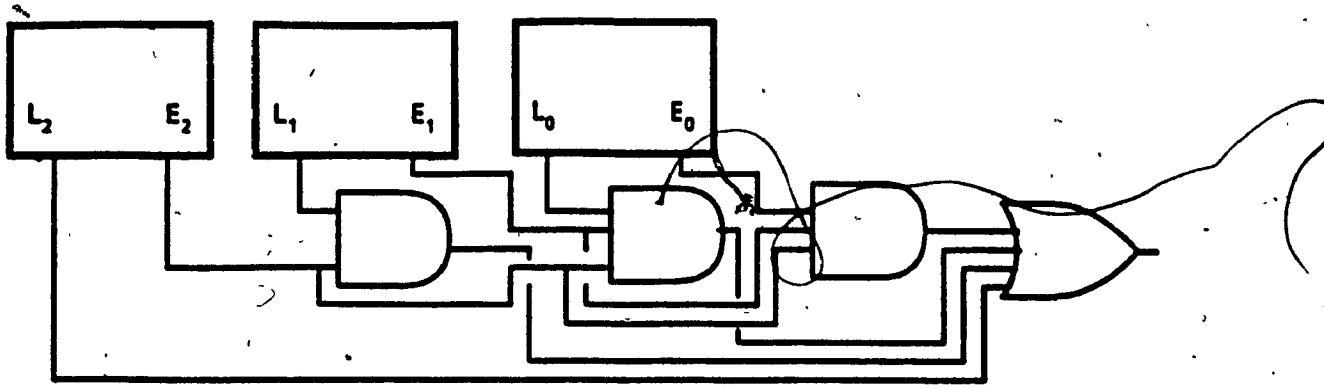


Figure 3-7b. ***Pseudo-Comparator***

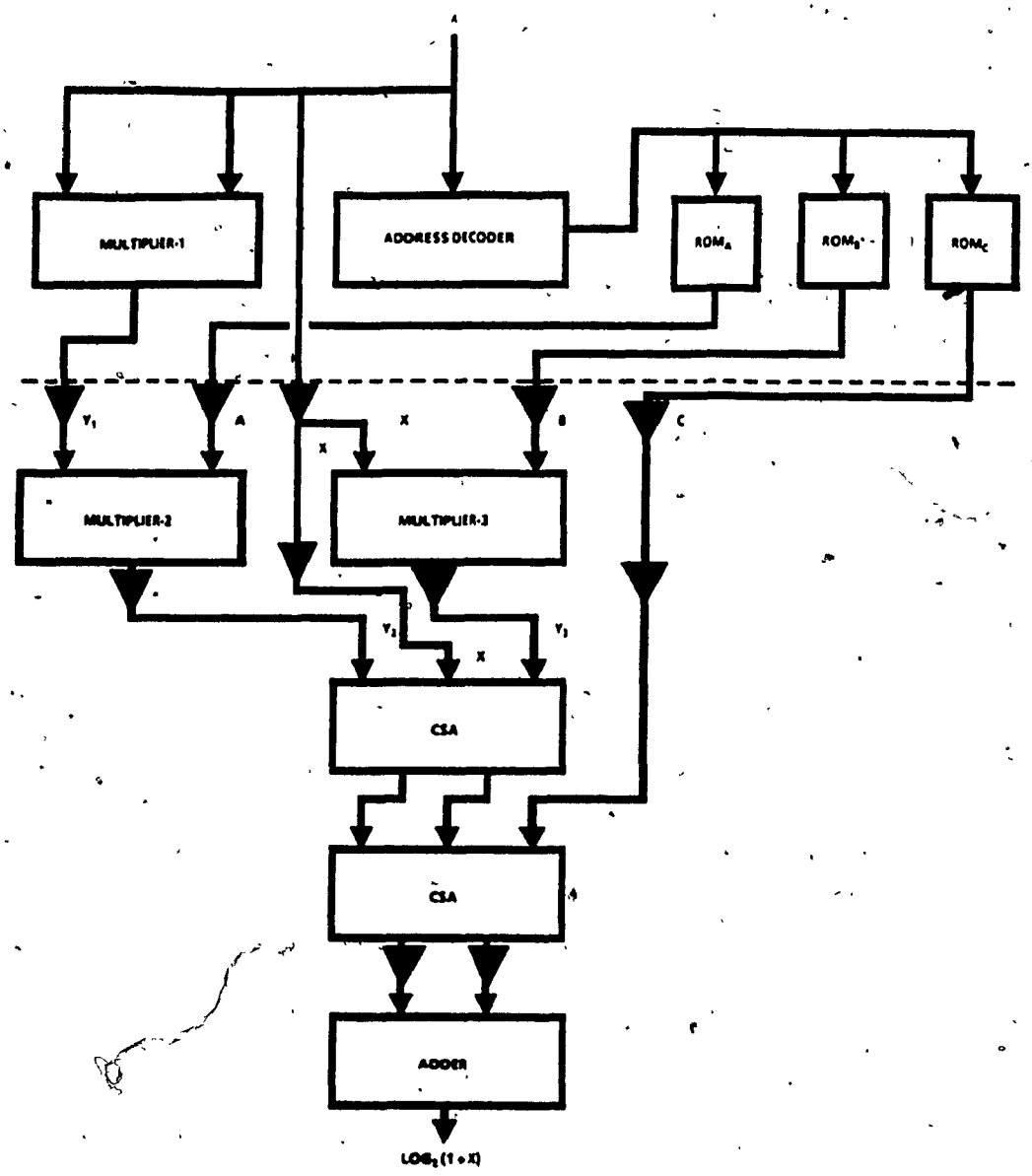


Figure 3-8. ***Pipelined Circuit***

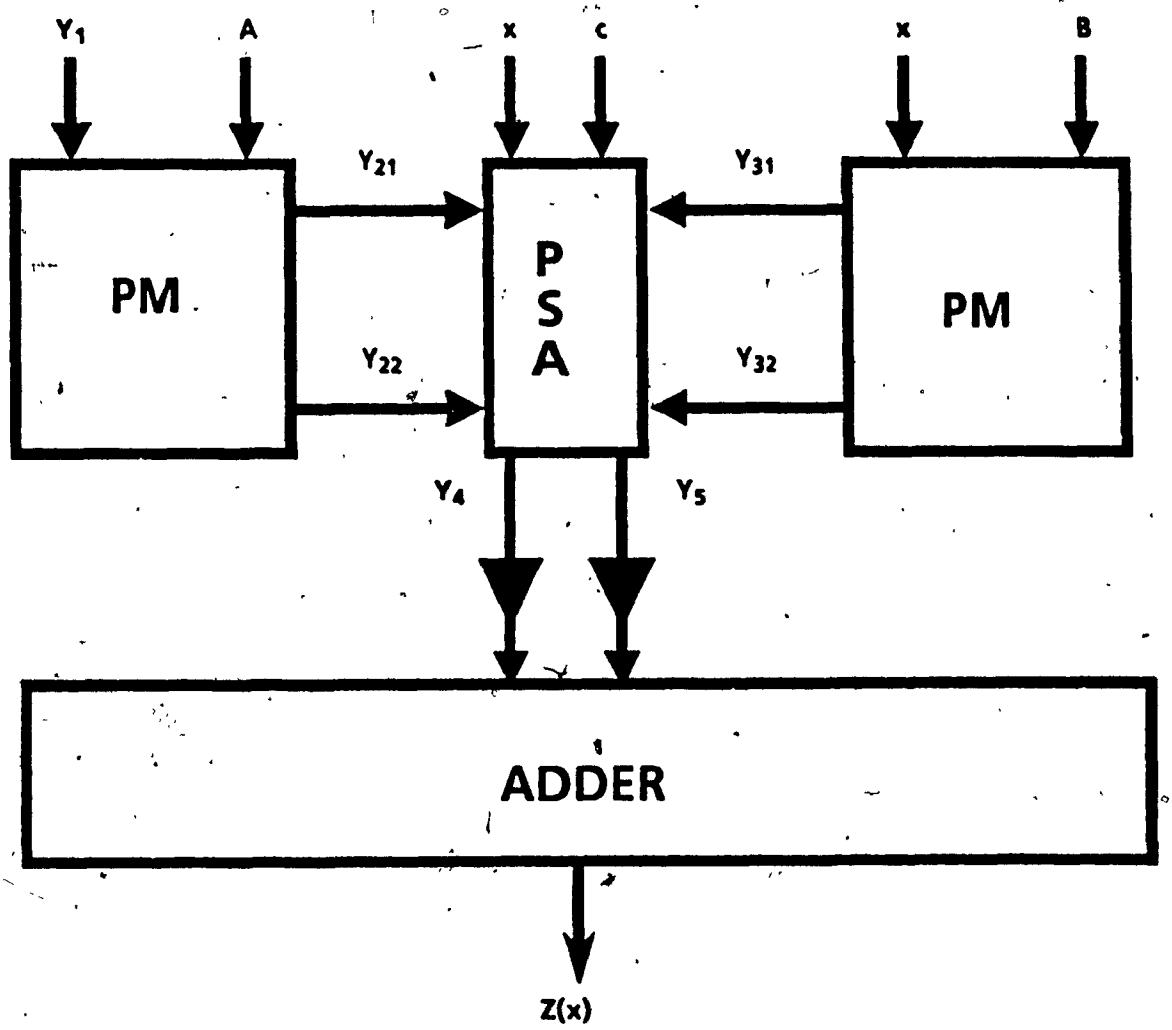


Figure 3-9. ***Pipelined addition of the outputs of pseudo-multipliers***

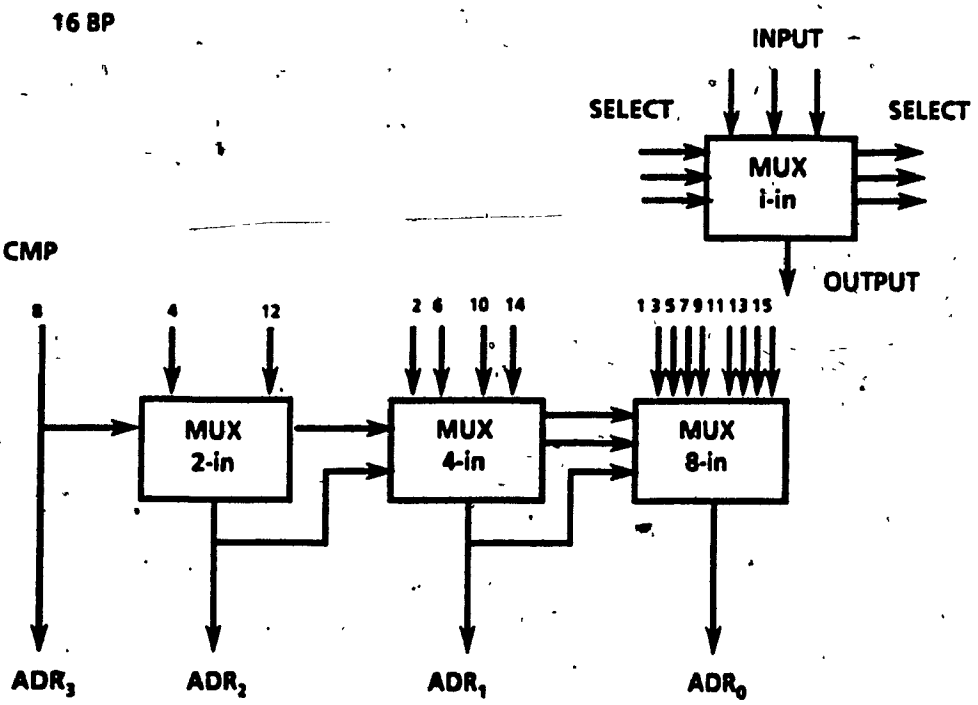


Figure 3-10. ***Network of multiplexers for computing approximation coefficients address***

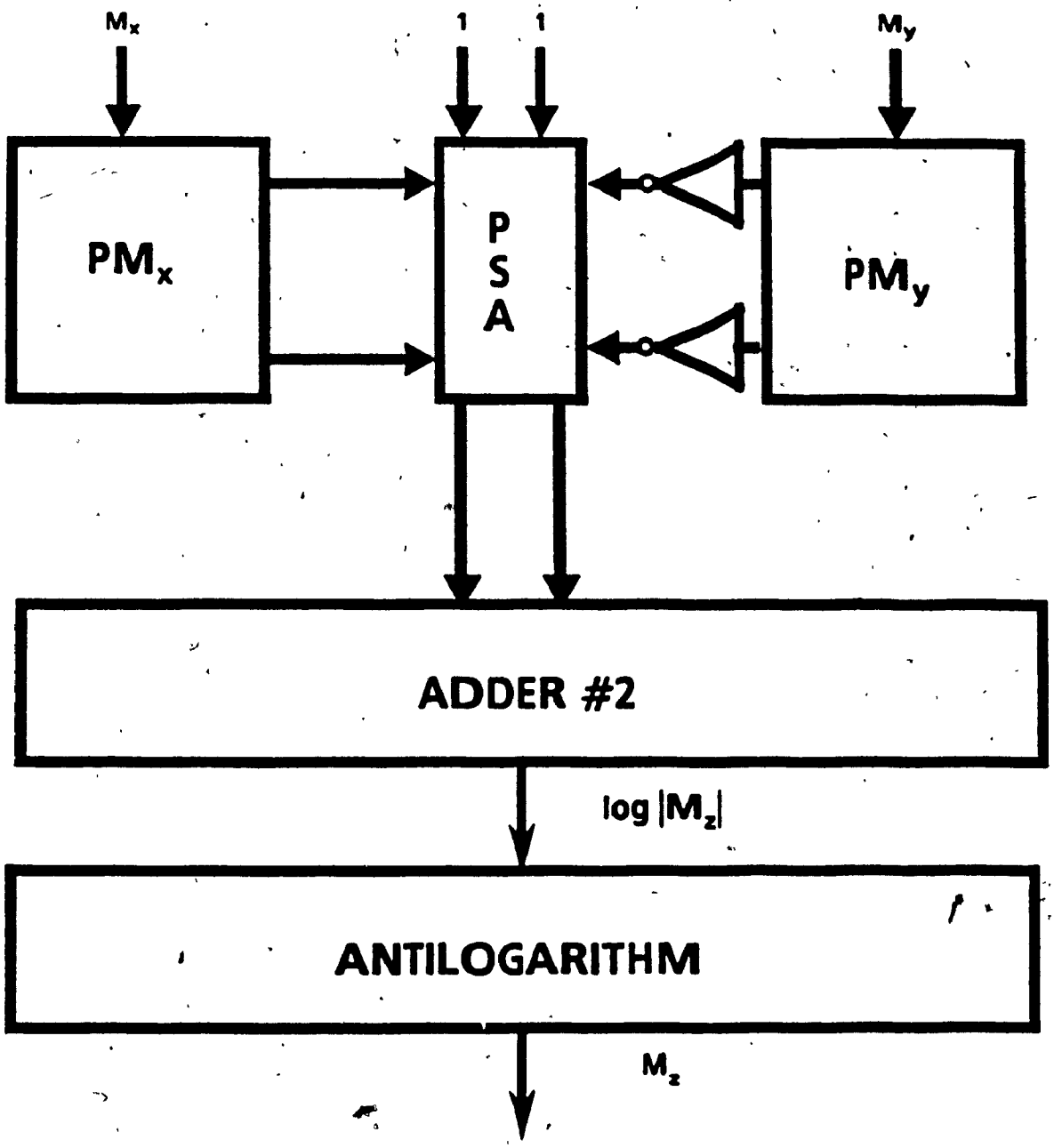


Figure 3-11a. ***A Scheme for binary division (Mantissa)***

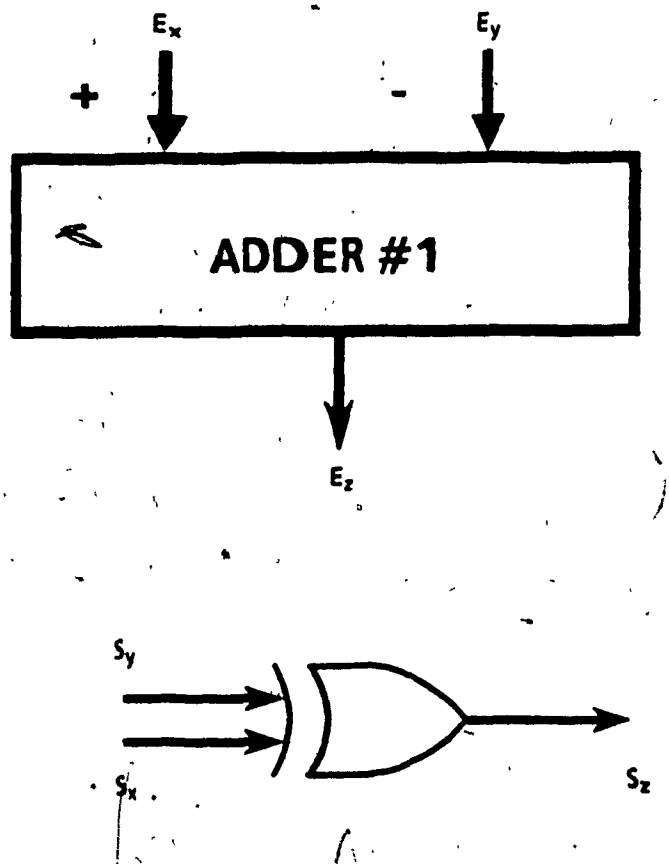


Figure 3-11b. ***A scheme for binary division (Exponent + Sign)***

1011

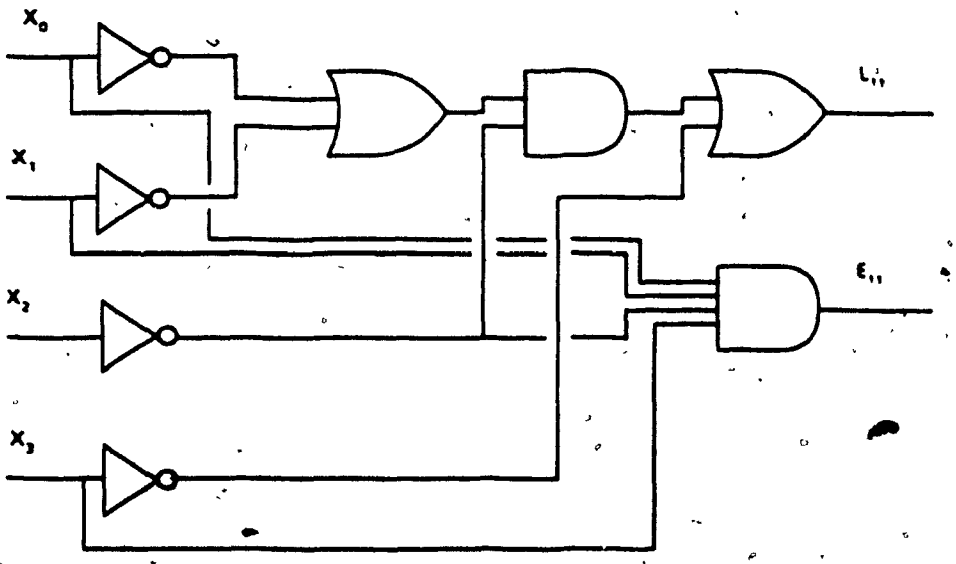


Figure 3-A1. *Example of Pseudo-Comparator*

CONCLUSION

Networks for performing multiplication of two two's complement numbers have been proposed in Chapter 1 and 2. Those networks can be implemented in a synchronous or an asynchronous way. If the factors to be multiplied have N bits, the area complexity of the networks is $O(N^2)$ as in the case of cellular multipliers.

With some additional circuits those multipliers can be used either independently for performing four separate single precision multiplications or connected to perform a single double precision multiplication.

Asynchronous multipliers can be built on this proposal. Input/output ports can be multiplexed in order to minimize the package pins needed using the same technique proposed by TRW [TRW78].

Chapter 1 introduced a fast scheme for macro cellular iterative multipliers and a cell design method for reducing the delay introduced by a macrocell. The approach will be suitable for implementations with very high speed technologies like Gallium Arsenide where the simplicity of a cellular structure is an imperative requirement.

Several Figures of Merit (FM) have been proposed the ones that have been mostly discussed are:

$$FM_a = AT^2(PE)^2$$
$$FM_b = A(PE)$$
$$FM_c = A(PE)T$$

The structure proposed in Chapter 1 has an area complexity of $O(N^2)$, a time complexity of $O(N)$ and a period complexity of $O(1)$. Those complexities correspond to:

$$FM_a = N^4$$
$$FM_b = N^2$$
$$FM_c = N^3$$

The recursive algorithm design approach introduces some new ideas that can be useful for an automatic layout design. Another potentially useful idea presented is the combination of multiplexer based macrocells with pseudo-adders. Macrocells based on multiplexers allow a fast propagation of changes of variables affected by a large delay by applying them to the address select inputs while using simpler and slower circuits applied at the multiplexers inputs. Paths containning a few macrocell in cascade can be tolerated as far as their delay is not a predominant contribution to the overall multiplier delay. Efficient pseudo adders with a constant delay can be used for adding up

the outputs of chains of macrocells in order to produce two numbers whose sum is equal to the desired product.

A recursive procedure can systematically place adders in a regular layout of macrocells making the maximum lenght of a chain of pseudo adders proportional to log N.

The recursive structure has an area complexity of $O(N^2)$, a time complexity of O(log N) and a period complexity of O(1) which correspond to the following figures of merit:

$$FM_a = N^2 \log^2 N$$
$$FM_b = N^2$$
$$FM_c = N^2 \log N$$

A combinational circuit for computing binary logarithm of an N bit number has been proposed in Chapter 3 with an area complexity $O(N^2)$. The structure has a time complexity O(log N) and can be pipelined with a period complexity O(1).

The combinational circuit proposed in Chapter 3 has the following figures of merit:

$$FM_a = N^2 \log^2 N$$
$$FM_b = N^2$$
$$FM_c = N^2 \log N$$

The same approach can be used for computing the antilogarithm. Two circuits for computing the logarithm and one circuit for computing the antilogarithm can be used for performing a binary division with the same performance.

REFERENCES

[BAN72] - Bandyopadhay, S., Basu, S. and Choudhury, A.K., ***An iterative array for multiplication of signed binary numbers.*** IEEE Trans. Comp., vol. C-21, pp. 921-922, 1972.

[BAU73] - Baugh, C.R. and Wooley, B.A., ***A two's complement parallel array multiplication algorithm,*** IEEE trans. Comp., vol C-22, pp. 1045-1047, 1973.

[BRE80] - Brent,R.P. and Kung, H.T., ***The chip complexity of binary arithmetic,*** In proc. 12 Annual ACM symp. Theory of computing, Los Angeles, Cal., April 28-30, ACM, new York, pp. 190-200, 1980.

[BUR78] - Burden, R.L., Faires, J.D. and Reynolds, A.C., ***Numerical analysis,*** Prindle, Weber and Schimidt, Boston, 1978.

[BUR70] - Burton, D.P., Byrne, P.C. and Noaks, D.R., ***A multiplier for complex binary numbers,*** Electronic Eng., pp. 71-73, 1970.

[CAP83] - Cappello, P.R. and Steiglitz, K., ***A VLSI layout for a pipelined Dadda multiplier,*** ACM Trans. Comp. Sys., vol.1, PP. 157-174, 1983.

[COM65] - Combet, M. Van Zonneveld, H. and Verbeek, I., ***Computation of the base two logarithm of binary numbers***, IEEE Trans. Elec. Comp., Vol EC-14, pp. 863-867, 1965.

[DAD76] - Dadda, L. ***Some schemes for parallel multipliers***, Alta frequenza, pp. 349-356, 1965. Reprinted in Computer Design Developpement, E.E. Swartzlander, Jr., Ed. Hayden book, Rochelle Park, N.J., 1976.

[DEA71] - Dean, K.J., ***Cellular multiplier subarray: a critical-path approach to propagation time***, Electronics Lett., vol. 7, pp. 75-77, 1971.

[DEA68a] - Dean, K.J., ***Binary logarithms***, Electronic Engineering, pp. 560-562, 1968.

[DEA68b] - Dean, K.J., ***Design of binary logarithm generators***, Proc. IEE, vol. 115, No. 8, pp.1118-1120, 1968.

[DEE71] - Deegan, I.D., ***Cellular multiplier for signed binary numbers***, Electronics Lett., vol. 7, pp. 436-437, 1971.

[DEM69a] - De Mori, R., ***Suggestion for an IC fast parallel multiplier***, Electronics Lett., vol. 5, pp. 50-51, 1969.

[DEM84] - De Mori, R. and Cardin, R., ***Iterative parallel multipliers based on multiplexers***, Signal Processing, Vol. 6, pp. 213-223, 1984.

[DEM85]. De Mori, R. and Cardin, R., ***A recursive algorithm for binary multiplication and its implementation***, Vol. 3, No. 4, pp. 294-314, 1985.

[DEM72] - De Mori, R. and Serra, A., ***A parallel structure for signed-number multiplication and addition***, IEEE Trans. Comp., vol. C-21, pp. 1453-1454, 1972.

[DEM69b] - De Mori, R., ***Cellular structure for parallel multiplier***, Proc. Int. Symp. On Systemes Logiques, pp. 1076-1098, Bruxelles, 1969.

[DEM75] - De Mori, R., Riviora, S. and Serra, A., ***A special-purpose computers for digital processing***, IEEE trans. Comp., vol. C-24, pp. 1202-1211, 1975.

[DEM85] De Mori, R. and Venkatesh, K. ***On the testability of macro cellular multipliers***, Concordia University, Internal report, May 1985.

[FLE80] - Fletcher, W.I., ***An Engineering approach to digital design***, Englewood Cliffs, Prentice-Hall, 1980.

[GAR83] - Garcia, G.H. and Kubitz, W.J., ***Minimum mean running time function generation using read-only-memory***, IEEE Trans. Comp., vol. C-32, pp. 147-156, 1983.

[GUI69] - Guild, H.H., ***Fully iterative fast array for binary multiplication and fast addition***, Electronics Lett., vol 5, pp. 263, 1969.

[HAB70] - Habibi, A. and Wintz, P.A., ***Fast multipliers***, IEEE Trans. Comp., vol C-19, pp. 153-157, 1970.

[HOF68] - Hoffman, J.C., Lacaze, B. and Csillag, p., ***Multiplieur parallele a circuit logiques iteratifs***, Electronics Lett., vol. 4, pp. 153-157, 1968.

[KIN71] - Kingsbury, N.G., ***High Speed binary multiplier***, Electronics Lett., vol. 7, pp. 277-278, 1971.

[HWA79] - Hwang, H., ***Computer arithmetic: Principales, Architecture and Design***, New York, Wiley, 1979.

[HWA85] Hawck, C.E., Baniji, C.S. and Allen, J.,***The systematic exploration of pipelined array multiplier performance***, Proc. IEEE Int. Conference on Acoustic

Speech and Signal Processing ICASSP-85, Tampa, Fla, 1985, pp.1461-1464.

[LEI83] Leiserson, C., Rose, F., and Saxe, J., ***Optimizing synchronous circuitry by retiming.,*** Third Calteck conference on VLSI, Pasadena, CA, 1983.

[LIN70] - Ling, H., ***High speed computer multiplication using a multiple-bit decoding algorithm,*** IEEE Trans. Comp., vol C-19, pp. 706-709, 1970.

[LUK81] Luk, W. K., ***A regular layout for parallel multiplier of O(log²N) time.,*** In Kung, H. T., Sproull, R. F. and Steele, G. L. Jr. (editor), ***VLSI systems and Computations,*** pages 317-326. Computer-Science Department, Carnegie-Mellon University, Computer Science Press, Inc., October, 1981.

[MAJ73] - Majithia, J.C. and Levan, P., ***A note on base 2 logarithm computation,*** IEEE Proc. 61, pp. 1519-1520, 1973.

[MAJ71] - Majithia, J.C. and Kitai, R., ***An iterative array for multiplication of signed binary numbers,*** IEEE Trans. Comp., vol. C-20, pp. 214-216, 1971.

[MAR72] - Marino, D., ***New algorithm for approximate evaluation in hardware of binary logarithm and elementary functions***, IEEE Trans. Comp., vol C-21, pp. 1416-1421, 1972.

[MIT62] - Mitchell, J.N., ***Computer multiplication and division using binary logarithm***, IRE Trans. on Elec. Comp., vol EC-11, pp. 512-517, 1962.

[NIC71] - Nicoud, J.D. and Dessoulavy, R., ***Logarithm converter***, Electronics Letters, vol. 7, No. 9, pp. 230-231, 1971.

[PRE83] - Preparata, F.P., ***A mesh-connected Area-time Optimal VLSI multiplier of large integers***, IEEE Trans. Comp., vol c-32, pp. 214-216, 1983.

[SIN73] - Singh, S. and Waxman, R., ***Multiple operand addition and multiplication***, IEEE Trans. Comp., vol C-22, pp. 113-120, 1973.

[SPR70] - Springer, J. and Alfke, P., ***Parallel multiplier gets boost from IC iterative logic***, Electronics, vol. 43, pp. 89-93, 1970.

[STE77] - Stenzel, W.J., Kubitz, W.J. and Garcia, G.H., ***A compact high-speed parallel multiplication scheme,*** IEEE trans. Comp., vol C-26, pp. 948-957, 1977.

[SWA75] - Swartzlander, E.E.Jr, and Alexopoulos, A.G., ***The signllogarithm number system,*** IEEE Trans. Comp., vol. C-24, pp. 1238-1242, 1975.

[TRW78] - TRW, LSI products, ***LSI multipliers,*** Redondo Beach, CA, Aug 1978.

[WAL64] - Wallace, C.S., ***A suggestion for a fast multiplier,*** IEEE Trans. Electron. Comp., pp. 14-17, Feb 1964.