OPTIMIZATION OF ONE BIT FULL ADDERS EMBEDDED IN REGULAR STRUCTURES

Kazuo Iwano
Kenneth Steiglitz

Dept. of Computer Science
Princeton University
Princeton, NJ 08544

TR-031-86

Optimization of One-Bit Full Adders Embedded in Regular Structures[†]

Kazuo Iwano
Kenneth Steiglitz

Dept. of Computer Science
Princeton University
Princeton, NJ 08544

Abstract

We study the problem of optimizing the transistor sizes in the one-bit nMOS full adder either isolated or embedded in a regular array.

A local optimization method that we call the *critical-path optimization method* is developed. In this method two parameters at a time are changed along the critical path until a locally optimal choice of transistor sizes is found. The critical-path optimization method uses the Berkeley VLSI tools and the hierarchical layout language ALLENDE developed at Princeton.

First, we optimize the isolated one-bit full adder implemented in three ways: as a PLA, Data Selector, and with Random Logic. The details of the critical-path optimization method and Power-Time tradeoff curves are illustrated here.

Second, we optimize the one-bit full adder embedded in a simple array multiplier. The entire 3×3, 4×4, 8×8 and 10×10 multipliers are optimized and their local optima are compared. Because the optimization of the entire circuit becomes less practical when the circuit becomes larger, we develop a method that makes use of circuit regularity. We prove that some small array of one-bit full adders, called the *canonical configuration*, has the same local optima as the $n\times n$ multiplier for large n, with the criterion of minimizing the delay time T. Hence we can greatly reduce the computation load by optimizing this canonical configuration instead of optimizing the entire circuit. Experimental results confirm the effectiveness of this approach.

1. Introduction

Regular arrays of cells are used often in custom chips for digital signal processing. Such regular arrays lead to designs that are easy to lay out efficiently and have high throughput. For example, bit-parallel and bit-serial multipliers can

[†] This work was supported in part by NSF Grant ECS-8414674, U. S. Army Research-Durham Contract DAAG29-85-K-0191, DARPA Contract N00014-82-K-0549, ONR Grant N00014-83-K-0275, and IBM-Japan.

be constructed from one- and two-dimensional arrays of one-bit full adders, as well as a wide variety of pipelined FIR and IIR filters (see, for example, [1-5, 9, 12, 16, 25]). This paper is aimed at the problem of optimizing such large arrays. The technology used throughout is 4 μ nMOS, but the general approach described is applicable to other technologies as well.

We will develop what we call the *critical-path optimization method.* This is a heuristic method for finding a locally optimal choice of transistor sizes by using systematic variation of the parameters along the critical path. The optimization loop uses the Berkeley tools [15] CRYSTAL (for timing) [19], POWEST (for estimating the power consumption), and the constraint-based high level layout language ALLENDE, developed at Princeton [11, 13]. We illustrate the critical-path optimization method and resulting Power-Time tradeoff curves by optimizing an isolated one-bit full adder implemented in three topologies, namely: the PLA, Data Selector, and with Random Logic.

Investigating the optimization of the one-bit full adder embedded in an array multiplier, we will next study how to take advantage of a circuit's regularity to reduce the optimization workload. Here we develop the *canonical configuration method,* which is shown to be practical for optimizing large regular structures. We will analyze the critical paths of the $n \times n$ multiplier using a finite automaton, and extract its *canonical configuration.* Then we will prove that the optimization of this *canonical configuration* provides, in the limit of large n, the same local optima as the $n \times n$ multiplier.

The problem of optimizing the transistor sizes has been studied by several authors. ANDY, developed by Trimberger [17], sizes transistors in a symbolic description of a chip to match the load the transistors are driving, then performs power optimization off the critical path. L. A. Glasser and L. P. J. Hoyte [6] developed what they called *macro models* of VLSI circuits and optimize the transistor sizes in a critical path. However their macro model is sometimes inaccurate and leads them to errors of as much as 70% when compared with the SPICE [15] circuit simulation. Matson [14] improved their macro models to be more accurate and computationally faster, and used them for nonlinear optimization of transistor sizing. Other related work is reported by Strojwas, Nassif and Director [18] and N. Jouppi [23]. Our efforts will concentrate on taking advantage of circuit regularity to make practical the optimization of large arrays.

2. Critical-path optimization method

The design of VLSI chips often involves the difficult task of effecting tradeoffs among three important measures, that is: the delay time T; the peak or average power dissipation $P_{\max}$ or P_{ave}; and the area A. There are many circuit choices which can be used for controlling these tradeoffs: For example, we can

control the choice of an appropriate topology, use precharging, super buffers, insert or delete logic stages to control the appropriate fanout factor, etc. However we will concentrate in this paper on the choice of pulldown diffusion widths, because the problem of sizing transistors is important but very tedious work for chip designers, and lends itself well to efficient solution by automated methods.

The constraint-based high-level layout language ALLENDE [13] enables us to parameterize a circuit; that is, ALLENDE accepts a circuit parameter vector π and produces the layout $C(\pi)$. Since the circuit performance (such as the delay time T, the power dissipation P, the area A, etc.) is determined by the circuit, the vector (P, T, A) can be expressed as a function of C. Since the circuit C is parameterized as $C(\pi)$, the vector (P, T, A) can finally be expressed as a function $g(\pi)$.

In general, therefore, our optimization can be formalized as follows:

$$\min_{\pi} f(P,T,A) = f(g(\pi))$$

subject to constraints on P, T, *and/or* A.

Here $f(\cdot)$ is the cost function to be optimized , and π is a circuit parameter vector $\pi = (d_1, d_2, \ldots, d_n)$. Since we optimize the transistor sizes of pulldowns, we treat each pullup/pulldown pair as a *node*. Typically, each node represents an inverter, NAND, or NOR gate. Each layout is characterized by the parameter vector $\pi = (d_1, d_2, \ldots, d_n)$, which means that the pulldown diffusion width of node i is $d_i\lambda$. We also use the vector $\kappa = (k_1, k_2, \ldots, k_n)$ to mean that the pullup to pulldown ratio of node i is k_i [22]. The vector κ is fixed for each circuit.

The choice of the cost function $f(\cdot)$ and constraints depends on the design issues. For example, in one application the clock period may be fixed at a known value T_0, and it would therefore be senseless to make the the cell faster. On the other hand, peak power may be a real constraint because of heat dissipation limitations. At the same time it may be important to keep the area small so as to fit as many cells on one chip as possible. We might, therefore, try to minimize some measure of the peak power and area (the product, $P_{\max}T$, for example), while enforcing the constraint $T \leq T_0$. In other applications speed may be critical, and it may be important to minimize T while observing constraints on P and A, and so on. In general, we would like to have enough information about the tradeoffs among the measures P, T and A to make intelligent design decisions. As we will see, the P–T tradeoff is often of most interest, since the area is often a less sensitive function of design parameters (at least for fixed topology).

3. Implementation of the critical-path optimization

We use a heuristic optimization method based on a critical path, and we will call our optimization method a *critical-path optimization method.* The general concept of the method is shown in Fig. 1. A circuit $C(\pi)$ is generated based on a circuit parameter π. The cost function $f(C)$ of the circuit $C(\pi)$ is computed next. Then a desired variation $\delta(\pi)$ of the parameter vector π is computed, based on the critical path. Finally $\delta(\pi)$ is added to π and the new parameter $(\pi + \delta\pi)$ replaces π if its cost is better. This is repeated until a local optimum is found. Fig. 2 shows a detailed flowchart of our implementation, which uses the tools ALLENDE, MEXTRA, CRYSTAL, POWEST. A short description of each follows below:

1) **ALLENDE** This procedural constraint-based VLSI layout language produces an integrated circuit layout in Caltech Intermediate Form (CIF) corresponding to the specified circuit parameter π [14].

2) **MEXTRA** MEXTRA reads CIF and extracts the nodes to create a circuit description for further analysis [15].

3) **CRYSTAL** CRYSTAL is used for finding the critical path and the delay time of the circuit [15, 19].

4) **POWEST** POWEST is used for finding the average and maximum power consumption of the circuit [15].

The basic approach we take will be to search for local improvements from random initial designs. The search strategy will be to consider all single or double changes of the current parameter vector π along the critical path. The idea is that the critical path indicates which parameters are most important to performance at any given point in the analysis. The "k-change" method is described below in general, with simple and double change corresponding to $k = 1$ and 2.

Given a current parameter vector $\pi = (d_1, d_2, \ldots, d_n)$ and the critical path nodes which are on a critical path, say $cpn = (d_{i_1}, d_{i_2}, \ldots, d_{i_m})$, the "$k$-change" method picks k nodes from cpn, say d_{j_1}, d_{j_2}, ..., and d_{j_k}, then changes each of d_{j_l}, $1 \leq l \leq k$ by one unit and keeps the others the same. For example, "2-change" produces $\binom{m}{2} \times 2^2 = 2m(m-1)$ sets of parameters. Then each parameter is analyzed in a fixed order. When the first cost improvement is met, the current parameter is picked as the parameter for the next iteration. That is, the first improvement found is adopted.

4. Full-adder circuit implementations

We used the one-bit full adder circuit as an example for experiments because it is relatively simple, but is a basic arithmetic logic circuit and has a wide variety of uses. The one-bit full-adder circuit can be implemented in many ways. We chose three kinds of circuits: the PLA, Data Selector, and Random Logic. All implemented circuits have been verified by **ESIM** [15] or **SIMULATE** [13].

1) **PLA** Fig. 3a shows the full-adder circuit diagram implemented by a programmable logic array (PLA). The π and κ of the PLA are as follows:

$\pi = (d_{\text{and}_1}, \ldots, d_{\text{and}_7}, d_{or_1}, d_{or_2}, d_{in_1}, d_{in_2}, d_{in_3}, d_{out_1}, d_{out_2})$

$k = (4,4,4,4,4,4,4,4,4,4,4,4,4,4)$

2) **Data Selector** Fig. 3b shows the full-adder circuit diagram of a Data Selector implementation [20]. The following truth table is used.

C_i	S_i	S_o	C_o
0	0	A	C_i (or S_i)
0	1	$\overline{A}$	A
1	0	$\overline{A}$	A
1	1	A	C_i (or S_i)

This circuit selects inputs (A, $\overline{A}$, or C_i) instead of calculating S_o and C_o. Here C_i is the input carry signal, C_o is the output carry signal, and S_o is the output sum signal. A and S_i denote the two other inputs. This layout has the following 7 parameters.

$\pi = (d_A, d_{S_i}, d_{C_i}, d_1, d_2, d_{C_o}, d_{S_o})$

$\kappa = (4,4,4,8,4,8,8)$

3) **Random Logic** Fig. 3c shows the circuit diagram of the Random Logic implementation [21]. This layout has the following 4 parameters: one node for computing $\overline{carry}$, one for $\overline{sum}$, one for *carry*, and one for *sum*.

$\pi = (d_{\overline{C_o}}, d_{\overline{S_o}}, d_{C_o}, d_{S_o})$

$\kappa = (8,12,4,4)$

The following logical equations describe the circuit:

$C_o = A \cdot (S_i + C_i) + C_i \cdot S_i,$

$S_o = \overline{C_o} \cdot (A + S_i + C_i) + C_i \cdot S_i \cdot A.$

5. Computational results of the full-adder circuits optimization

Table 1 shows a comparison of the performance of our implementations. Each row represents one point locally optimal with respect to T. The units of A, P_{ave}, $P_{\max}$, T, APT and PT are $10^3 \cdot \lambda^2$, (mW), (mW), ns, $(10^{-12} \cdot \lambda^2 \cdot W \cdot ns)$ and $(10^{-8} \cdot W \cdot ns)$ respectively in all tables.

type	A	P_{ave}	$P_{\max}$	T	APT	PT
PLA	21.2	7.2	10.1	9.7	2074	980
	21.3	7.5	10.6	9.7	2185	1026
Data Selector	8.2	3.3	5.2	14.7	628	760
	8.4	3.5	5.6	14.7	699	830
Random Logic	9.3	1.7	2.4	7.2	161	173
	9.3	1.8	2.6	7.3	179	197

Table 1. Performance comparison (one-bit full adder)

Fig. 4 shows $P_{\max}$ *vs* T curves of the one-bit full adder for different topologies when minimizing T. These $P_{\max}$ *vs* T tradeoff curves are obtained as follows: During a critical-path optimization process, every time a current parameter π shows an improvement in cost (in this case, T), we plot the associated values of $P_{\max}(\pi)$ and $T(\pi)$. We then obtain a trajectory from each initial random starting point to a locally optimal point. By drawing an envelope of these points on many trajectories, we finally obtain an approximate $P_{\max}$ *vs* T tradeoff curve.

As shown in Fig. 4, the $P_{\max}$ *vs* T tradeoff curve of the Random Logic circuit is below that of the Data Selector circuit and that of the PLA.

type	A	P_{ave}	$P_{\max}$	T	APT	PT
PLA	228	424	421	135	1288	566
Data Selector	88	194	217	204	390	439
Random Logic	100	100	100	100	100	100

Table 2. Normalized performance comparison (1-bit full adder)

Table 2 shows a normalized performance comparison of the best locally optimal point for each layout, minimizing T. The Random Logic seems to be the best choice in all respects except A. The product $P_{\max}T$ of the Random Logic is about 1/4.4 that of the Data Selector, while it is about 1/5.7 that of the PLA.

Table 3 below is the performance comparison table between the T-locally optimal circuit and the circuit designed using the minimum sizes (2λ). Our optimization shows a good improvement of the delay time (improvement from

55% to 73%) in any implementation. Matson optimized the same Random Logic circuit and obtained a delay time of 8.0 ns in [14], while our locally optimal circuit has 7.2 ns, providing an independent check of the effectiveness of our optimization method.

type	cost	PLA	Data Selector	Random Logic
min size (ns)	T	21.7	53.2	26.9
T-opt (ns)	T	9.7	14.7	7.2
improvement	T	-55.2%	-72.3%	-73.2%

Table 3. Performance improvement ratio

6. Full optimization of n×n multipliers

In this section we take up the problem of optimizing the one-bit full adder when it is embedded in a regular array, using the array multiplier illustrated in Fig. 5a for 4 bits.

Complete layouts of the 3×3, 4×4, 8×8, and 10×10 multipliers were optimized using the critical-path optimization method. Fig. 6 shows the possible tradeoff of power against delay time obtained in the same way as Fig. 4.

Table 4 gives the results of starting from 11 different initial parameter vectors. They are yielding only four distinct local optima, corresponding to the parameters $\pi_1 = (4,16,8,8,8)$, $\pi_2 = (8,16,8,8,8)$, $\pi_3 = (12,12,8,8,8)$, and $\pi_4 = (12,16,8,8,8)$. Note that in Table 4, ** indicates that the associated π is not a local optimum.

As we can see in Table 5, the running time of this optimization method increases quickly with the size of the array, growing approximately as the number of basic cells in the circuit, or n^2 for an $n \times n$ multiplier. This means that optimizing the entire circuit at once is a very costly operation, practical for a relatively small circuit, but not for large circuits. We will see later how to take advantage of the circuit regularity to reduce the computational workload.

type	cost	π_1	π_2	π_3	π_4
3 x 3	T	78.9	78.6	74.1	80.7(**)
	$P_{\max}$	30.1	32.9	32.9	32.9
	P_{ave}	18.4	20.5	20.5	20.5
	A	82.7	84.0	82.4	87.9
4 x 4	T	118.9	118.1	110.2	120.3(**)
	$P_{\max}$	54.6	60.2	60.2	60.2
	P_{ave}	33.4	37.6	37.6	37.6
	A	152.3	154.7	151.5	162.0
8 x 8	T	279.6	277.1	377.9(**)	283.5
	$P_{\max}$	224.7	251.0	224.6	251.0
	P_{ave}	138.4	158.1	138.3	158.1
	A	636.2	646.6	632.9	678.8
10 x 10	T	359.2	355.9	487.2(**)	364.1
	$P_{\max}$	353.0	395.3	352.9	395.3
	P_{ave}	217.7	249.4	217.6	249.4
	A	1001.7	1018.1	996.5	1069.3

Table 4. Local optima (** means that this is not a local optimum.)

	m33	m44	m88	m1010
#(basic cells)	6	12	56	90
cpu (one point)	2 min.	3 min.	14 min.	22 min.
average # of (iterations)	5.0	5.1	5.0	6.0
average # (points searched)	61	84	61	100

Table 5. Optimization computing cost

In the next section we will define a class of circuits consisting of rectangular arrays of one-bit full adders. In succeeding sections, we will analyze the critical paths of their circuits, and show that they can be constructed from information obtained by optimization of a small "representative" circuit, called a *canonical configuration.* We will then prove that this optimization also yields the same locally optimal one-bit full adders as direct optimization of large circuits.

Finally, we give experimental results which confirm this fact, and show the utility of this approach.

7. A class of circuits

We discuss a class of circuits consisting of one-bit full adders which have four nodes as shown in Fig. 3c. They are $N_{\overline{C_o}}$, N_{C_o}, $N_{\overline{S_o}}$, and N_{S_o}, which yield the $\overline{carry}$, *carry*, $\overline{sum}$, and *sum* signals respectively. Note that there are inputs C_i and S_i to the nodes $N_{\overline{C_o}}$ and $N_{\overline{S_o}}$. The relations among nodes are shown in Fig. 7. Next we define a class of circuits as follows:

Definition $C_{FA} \equiv \{\ C \mid$ The circuit C has the following three properties. $\}$

1) The circuit C is an $m \times n$ subarray of the two-dimensional infinite array of identical one-bit full adders shown in Fig. 8, for some m and n.
2) The one-bit full adder cell used in the array is the random logic circuit shown in Fig. 3c. Note that the input $A = a \cdot b$ is created from two other inputs $\overline{a}$ and $\overline{b}$ by a NOR circuit. The one-bit full adder is characterized by the circuit parameter $\pi = (\ d_{a \cdot b}, d_{\overline{C_o}}, d_{\overline{S_o}}, d_{C_o}, d_{S_o}\ )$. Since we use identical one-bit full adders in the entire array as mentioned above, we regard the circuit parameter π of the one-bit full adder as the circuit parameter of the entire circuit C.
3) The interconnection scheme of the one-bit full adders is shown in Fig. 9.

In Fig. 8 each cell is a one-bit full adder with coordinates (x, y). We measure x to the left and y down, and $A_{x,y}$ designates a cell located in position (x, y). Each cell has two inputs (S_i: the sum input and C_i: the carry input) and two outputs (S_o: the sum output and C_o: the carry output) as shown in Fig. 9. Precisely speaking, $A_{x,y}$ has two more inputs $\overline{a}$ and $\overline{b}$, as shown in Fig. 5b, but we do not include these inputs in our model because they will not appear in any critical path. In other words, we assume that two input $\overline{a}$ and $\overline{b}$ are available to any cell when a critical path reaches that cell. As shown in Fig. 9, the carry output C_o of $A_{x,y}$ is propagated to the carry input of $A_{x+1,y}$, while the sum output S_o of $A_{x,y}$ is propagated to the sum input S_i of $A_{x-1,y+1}$.

Here we analyze an $n \times n$ multiplier which has n columns and $(n-1)$ rows of full adders. Hence the $n \times n$ multiplier corresponds to the rectangle bounded by the corner cells $A_{0,0}$, $A_{n-1,0}$, $A_{n-1,n-2}$, and $A_{0,n-2}$ in Fig. 8.

8. Definition of critical paths

In this section we define a critical path between two cells in a circuit $C \in C_{FA}$ and analyze the behavior of signals on a critical path.

Suppose that a path α exits the one-bit full adder cell $A_{x,y}$. The path α exits the cell $A_{x,y}$ from either the carry output C_o or the sum output S_o with high or low signal. Hence in order to identify the state of the path α at the exit of the cell $A_{x,y}$, we can use the representation (A, a) where A is either *carry* or *sum*, and a is either *high* or *low*. Let (C, 0) denote the state in which a path exits from the carry output with a low signal, and let (C, 1), (S, 0), and (S, 1) be defined analogously. Then the behavior of the path α can be represented by a sequence of states. For example, the expression (C, 0) $\rightarrow$ (C, 0) $\rightarrow$ (S, 1) $\rightarrow$ (S, 0) represents the path which exits $A_{0,0}$ with the low carry signal, exits $A_{1,0}$ with the low carry signal, exits $A_{2,0}$ with the high sum signal, and finally exits $A_{1,1}$ with the low sum signal as shown in Fig. 8. We can thus represent the behavior of the path by the state transition diagram D as shown in Fig. 10a. Note that in this diagram, the states C, D, S, and T are used instead of (C, 0), (C, 1), (D, 0), and (D, 1) respectively.

Lemma 1 The state transition diagram D in Fig. 10a correctly describes the behavior of the signal along any path in the array of Fig. 8.

Proof In Fig. 7, each time a path passes through a node, the signal on that path changes from high (low) to low (high). For example, suppose we have a path α which exits from the carry output C_o with a high signal. The path α must pass through the nodes $N_{\overline{C_o}}$ and N_{C_o}. Since a signal changes from low (high) to high (low) when a path passes through a node, the $\overline{C_o}$ signal is low, and the input on the path into the node $N_{\overline{C_o}}$ is high. Hence the state (C, 1) can be reached either from the state (C, 1) or (S, 1).

In the same way we can determine other state transitions. Note that a path to the sum output S_o results from either $N_{\overline{C_o}} \rightarrow N_{\overline{S_o}} \rightarrow N_{S_o}$ or $N_{\overline{S_o}} \rightarrow N_{S_o}$. □

From the above discussion, we can define a finite automaton M that represents the state transitions along a path.

Definition The finite automaton M is defined as follows:

$$\mathrm{M} = (Q, \Sigma, \delta, q_0, F),$$

where Q is the set of states, Σ is the alphabet, δ is the state transition function, q_0 is the initial state, and F is the set of final states.

$$Q = \{q_0, C, D, S, T\},$$

$$\Sigma = \{ c, d, s_0, s_1, t_0, t_1 \}, \text{ and } F = Q,$$

where the states C, D, S, and T represent the states (C, 0), (C, 1), (S, 0), and (S, 1) respectively. The symbol c (d) indicates the transition from the node $N_{\overline{C_o}}$ with a low (high) input to the node N_{C_o} with a high (low) output, while the symbol s_0 (t_0) indicates the transition from the node $N_{\overline{C_o}}$ with a high (low) input to the node N_{S_o} with a low (high) output. And the symbol s_1 (t_1) indicates the transition from the node $N_{\overline{S_o}}$ with a low (high) input to the node N_{S_o} with a low (high) output. The transition function δ is shown in Fig. 10a where q_0 and transitions from q_0 such as $\delta(\, q_0,\, c\,) = C$, $\delta(\, q_0,\, d\,) = D$, $\delta(\, q_0,\, s_0\,) = S$, and $\delta(\, q_0,\, t_0\,) = T$ are not shown.

Since we use the fixed one-bit full adder shown in Fig. 7, there is a fixed delay time associated with each state transition when the circuit parameter π is fixed. Hence we can define a delay-time function w as follows:

Definition Given a circuit parameter π and a finite automaton $M = (\, Q, \Sigma, \delta, q_0, F\,)$ defined above, the *delay-time function* w_π is defined as follows: $w_\pi : Q \times Q \rightarrow R$ such that $w_\pi(\, q_1, q_2\,)$ is the delay time for calculating the corresponding output signal and propagating this signal when the transition $(\, q_1, q_2\,)$ is in δ. When the transition $(\, q_1, q_2\,)$ is not defined in δ, $w_\pi(\, q_1, q_2\,)$ is not defined. We use w instead of w_π when a circuit parameter π is fixed in the discussion. Since from the definition of the symbols, any transition with the same symbol has the same delay time, we also use the symbol to mean its delay time. Note that $s_0 > s_1$ and $t_0 > t_1$, since the transition from the node $N_{\overline{S_o}}$ to the node N_{S_o} is a part of the transition from the node $N_{\overline{C_o}}$ to the node N_{S_o} through $N_{\overline{S_o}}$.

Let $L(\, M\,)$ be the language accepted by the finite automaton M. The language $L(\, M\,)$ corresponds to the set of all possible paths in our two-dimensional array. Let $L_{mn} = \{\, \alpha \in L(\, M\,) \mid\, |\, \alpha\,|_c + |\, \alpha\,|_d = m, |\, \alpha\,|_{s_0} + |\, \alpha\,|_{s_1} + |\, \alpha\,|_{t_0} + |\, \alpha\,|_{t_1} = n\,\}$ where $|\, \alpha\,|_a$ indicates the number of times the symbol "a" in the string α. Thus L_{mn} corresponds to the set of paths from $A_{0,0}$ to $A_{m-n-1,m-1}$, or in another words, L_{mn} is the set of paths that have m carry stages and n sum stages.

We use the notation $C_{FA,m,n}$ for representing the set of circuits $C \in C_{FA}$ whose critical paths are in L_{mn}. We define the delay-time function w on a string in $L(\, M\,)$ as follows: Let $\alpha \in L(\, M\,)$ and $\alpha = a_1 a_2 \cdots a_n$ where $a_i \in \{\, c, d, s_0, s_1, t_0, t_1\,\}$ for $1 \leq i \leq n$. Then $w(\, \alpha\,) \equiv \sum_{i=1}^{n} w(\, a_i\,)$.

We can now define a *critical path* in our terms.

Definition We call $\alpha \in L_{mn} \subseteq L(\, M\,)$ a *critical path* when $w(\, \alpha\,) \geq w(\, \beta\,)$ for all $\beta \in L_{mn}$. Define CPN as the set of all critical paths of all subcircuits in C_{FA} and $\mathrm{CPN}_{mn} = \mathrm{CPN} \bigcap L_{mn}$. We say that two paths are *equivalent* when their

delay times are equal. When $w(\alpha) < w(\beta)$ ($w(\alpha) > w(\beta)$, $w(\alpha) = w(\beta)$) for two path α and β, we denote that by $\alpha <_w \beta$ ($\alpha >_w \beta$, $\alpha =_w \beta$, respectively.)

Our problem of finding a critical path in the $m \times n$ multiplier then corresponds to the problem of finding a critical path from $A_{0,0}$ to another cell $A_{m-1,n-2}$, as shown in Section 7.

Lemma 2 Every critical path of the $n \times n$ multiplier has ($2n-3$) carry calculation stages and ($n-1$) sum calculation stages.

Proof Every time a sum signal appears in a path the y-coordinate increases by 1, while x-coordinate decreases by 1. Every time a carry signal appears in a path the x-coordinate increases by 1, while y-coordinate remains the same. Hence a path from the cell $A_{0,0}$ with a_s sum stages and a_c carry stages reaches the cell $A_{a_c-a_s,a_s}$. In the $n \times n$ multiplier, a critical path starting from the cell $A_{0,0}$ finally comes out of the cell $A_{n-1,n-2}$ with the sum signal. Hence $n-1 = a_c - a_s$ and $n-2 = a_s$. Thus $a_c = n-1+a_s = 2n-3$. Hence we proved that every critical path α of the $n \times n$ multiplier has ($2n-3$) carry calculation stages, and ($n-2$) sum calculation stages, plus another sum calculation in the cell $A_{n-1,n-2}$. □

9. Analysis of critical paths

In Section 8, we saw that the behavior of the signals on the path can be described by the weighted state transition diagram M. In this section, we investigated the problem of finding the critical paths effectively, given the weighted state transition diagram.

Since the longest path between two nodes can be computed given the delay time of all paths between two nodes, we have the following theorem.

Theorem 1 Given a fixed parameter π and the delay-time function w_π: $Q \times Q \rightarrow R$, we can effectively construct the set CPN.

We next describe a more practical way of constructing a critical path in L_{mn}. We use $R(a_1, a_2, \ldots, a_k)$ to represent the pair (i, a_i) where $a_i = max(a_1, a_2, \ldots, a_k)$; that is, R tells us which argument is maximum.

Theorem 2 Given a circuit parameter π, knowledge of

$$R(c, d),\ R(s_0, t_0),\ R(s_0+t_0, s_0+s_1, t_0+t_1),\ \text{and}$$

$$R(s_0+t_0, 2s_1, 2t_1)$$

is both necessary and sufficient to construct a critical path in L_{mn} for $m \geq 0$ and $n \geq 0$.

We need the following lemmas to prove Theorem 2. The first lemma shows that we can simplify our finite automaton.

Lemma 3 Let L be the set of strings accepted by the finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ defined in Section 8 and shown in Fig. 10a. Let L_1 be the set of strings accepted by the finite automaton $M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$ shown in Fig. 10b, where $Q_1 = \{ q_0, A, B \}$. $F_1 = \{ A, B \}$, and δ_1 is shown in Fig. 10b. Then $L(M) = L(M_1)$.

Proof Use the state minimization algorithm in [24]. □

From now on, we will concentrate our attention on the reduced state automaton M_1. Next we will characterize the strings accepted by M_1.

Lemma 4 Let E be the regular expression of the strings accepted by M_1. Then

$$E = a_0 a_1^*(t_0 b_1^* s_0)^* a_1^*(\epsilon + t_0) + b_0 b_1^*(s_0 a_1^* t_0)^* b_1^*(\epsilon + s_0)$$

where $a_i = s_i + c$, $b_i = t_i + d$ for $i = 0, 1$ and ϵ is the empty input.

Proof Let r_A (r_B) be the regular expressions of strings starting from and ending at state A (B). Then $r_A = a_1^*(t_0 b_1^* s_0)^* a_1^*$ and $r_B = b_1^*(s_0 a_1^* t_0)^* b_1^*$. Let E_A (E_B) be the regular expressions of strings accepted at state A (B). Then $E_A = a_0 r_A + b_0 r_B s_0$ and $E_B = b_0 r_B + a_0 r_A t_0$. Therefore $E = E_A + E_B$ is the desired regular expression. □

Definition For two regular expressions E_1 and E_2, we define $E_1 \sim E_2$ iff for any $e_1 \in E_1$ ($e_2 \in E_2$), there exists $e_2 \in E_2$ ($e_1 \in E_1$) such that $e_1 =_w e_2$, that is, $T(e_1) = T(e_2)$. We also use $e_1 \sim e_2$ when $e_1 =_w e_2$.

Lemma 5 $E \sim (s_0 t_0 + c t_0 + d s_0 + s_0 + c + d) s_1^* c^*(s_0 + t_0)^* t_1^* d^*$.

Proof Note that for $a, b \in \Sigma$, $(a + b)^* \sim a^* b^*$, $ab \sim ba$. Hence $a_1^* = (s_1 + c)^* \sim s_1^* c^*$, $b_1^* = (t_1 + d)^* \sim t_1^* d^*$, $(t_0 b_1^* s_0)^* \sim (s_0 t_0)^* b_1^*$, and $(s_0 a_1^* t_0)^* \sim (s_0 t_0)^* a_1^*$. From Lemma 4, we can obtain desired result. □

Let $y_3 = max(s_0 + t_0, s_0 + s_1, t_0 + t_1)$ and $y_4 = max(s_0 + t_0, 2s_1, 2t_1)$. Then define z_1, z_2, z_3, and z_4 as follows:

$$z_1 = \max (c, d), \quad z_2 = \max (s_0, t_0),$$

$$z_3 = \begin{cases} s_0 t_0 & \text{if } y_3 = s_0 + t_0 \\ s_0 s_1 & \text{if } y_3 = s_0 + s_1 \\ t_0 t_1 & \text{if } y_3 = t_0 + t_1 \end{cases}, \quad z_4 = \begin{cases} s_0 t_0 & \text{if } y_4 = s_0 + t_0 \\ s_1^2 & \text{if } y_4 = 2s_1 \\ t_1^2 & \text{if } y_4 = 2t_1 \end{cases}$$

Lemma 6 Let α be a critical path in $L_{m,n}$. Then

$$\alpha \leq_w \begin{cases} z_1^m z_2 z_4^k & \text{if } n = 2k + 1 \\ z_1^m z_3 z_4^{k-1} & \text{if } n = 2k \end{cases}$$

Proof Let β be a path in $L_{m,n}$. From Lemma 5, $\beta \sim c^{x_1}d^{x_2}\gamma$ where $x_1 + x_2 = m$ and $\gamma \in (s_0 + t_0 + s_1 + t_1)^n$. Clearly, $\beta \leq_w z_1^m\gamma = max(c, d)^m\gamma$. Hence we only have to think about a critical path in $L_{0,n}$. Let β be a path in $L_{0,n}$. From Lemma 5, $\beta \sim (s_0t_0 + s_0 + t_0)s_1^{x_1}t_1^{x_2}(s_0t_0)^{x_3}$.

We take the following two cases: 1) $n = 2k + 1$ and 2) $n = 2k$.

1) $n = 2k + 1$: $\beta \sim \gamma s_1^{x_1}t_1^{x_2}(s_0t_0)^{x_3}$ where $\gamma \in (s_0t_0 + s_0 + t_0)$.

1.a) If $\gamma = s_0t_0$, then either x_1 or x_2 is odd. Without loss of generality, we assume that $x_1 = 2k_1 + 1$ and $x_2 = 2k_2$. Since $s_1 <_w s_0$, $s_1^2 \leq_w z_4$, $t_1^2 \leq_w z_4$, $s_0t_0 \leq_w z_4$, and $s_0 \leq_w z_2$, we have

$$\beta \sim s_0t_0s_1s_1^{2k_1}t_1^{2k_2}(s_0t_0)^{x_3} <_w z_4s_0z_4^{k_1}z_4^{k_2}z_4^{x_3} \leq_w z_2z_4^k$$

1.b) If $\gamma = s_0$ or t_0, then $\beta \sim \gamma s_1^{x_1}t_1^{x_2}(s_0t_0)^{x_3}$ and $x_1 \equiv x_2 \ (mod\ 2)$. Note that $\gamma \leq_w z_2$. If $x_1 = 2k_1 + 1$ and $x_2 = 2k_2 + 1$, then

$$\beta \sim \gamma(s_1t_1)s_1^{2k_1}t_1^{2k_2}(s_0t_0)^{x_3} \leq_w z_2(s_0t_0)z_4^{k_1}z_4^{k_2}z_4^{x_3} \leq_w z_2z_4^k.$$

If $x_1 = 2k_1$ and $x_2 = 2k_2$, then $\beta \sim \gamma s_1^{2k_1}t_1^{2k_2}(s_0t_0)^{x_3} \leq_w z_2z_4^k$.

Hence $\beta \leq z_2z_4^k$.

2) $n = 2k$: $\beta \sim \gamma s_1^{x_1}t_1^{x_2}(s_0t_0)^{x_3}$ where $\gamma \in (s_0t_0 + s_0s_1 + t_0t_1 + s_0t_1 + s_1t_0)$. Since $n = 2 + x_1 + x_2 + 2x_3$ is even, we know $x_1 \equiv x_2 \ (mod\ 2)$. Since $s_0t_1 \leq_w s_0t_0$ and $s_1t_0 \leq_w s_0t_0$, we have $\beta \leq_w z_3s_1^{x_1}t_1^{x_2}(s_0t_0)^{x_3}$. In the same way as in 1.b), $s_1^{x_1}t_1^{x_2}(s_0t_0)^{x_3} \leq_w z_4^{k-1}$. Thus $\beta \leq_w z_3z_4^{k-1}$. □

Definition A string $\beta \in \Sigma^*$ is said to be *constructible* when there exists a path $\alpha \in L(M_1)$ such that $\alpha \sim \beta$.

Lemma 7 For any integers $m \geq 0$ and $k \geq 1$, the strings $z_1^mz_2z_4^k$ and $z_1^mz_3z_4^{k-1}$ are constructible, and therefore the upper bounds in Lemma 6 are attained.

Proof We will prove this for $n = 2k + 1$. The proof for n even is similar. Without loss of generality, we assume that $z_2 = s_0$. Now we consider the constructibility of $z_2z_4^k$. We will find a string $\alpha_{0,n} \in L_{0,n}$ such that $\alpha_{0,n} \sim z_2z_4^k$. We have the following three cases for z_4:

1) $z_4 = s_0t_0$: Let $\alpha_{0,n} = s_0(t_0s_0)^k \in L_{0,n}$. Then $\alpha_{0,n} \in L(M_1)$ and $\alpha_{0,n} \sim z_2z_4^k$.

2) $z_4 = s_1^2$: Let $\alpha_{0,n} = s_0s_1^{2k} \in L_{0,n}$. Then $\alpha_{0,n} \in L(M_1)$ and $\alpha_{0,n} \sim z_2z_4^k$.

3) $z_4 = t_1^2$: Since $s_0 + t_0 \leq 2t_1 < t_0 + t_1$, we have $s_0 < t_1 < t_0$. However we assumed that $z_2 = \max (s_0, t_0) = s_0$. Thus this case does not happen.

From 1), 2), and 3) we showed that the string $z_2z_4^k$ is constructible.

Now we prove that $z_1^m z_2 z_4^k$ is constructible. Let $y_1 = s_0 c^m$ if $z_1 = c$, or let $y_1 = d^m s_0$ if $z_1 = d$. Let $\alpha_{0,n} = s_0 y_2$. Let $\alpha = y_1 y_2$. Here we know that $\alpha \in L_{m,n}$ and $\alpha \sim z_1^m z_2 z_4^k$. Thus we proved that $z_1^m z_2 z_4^k$ is constructible. □

The following example shows how to construct a critical path β in $L_{10,5}$ from knowledge of R-functions, given a fixed parameter π = (12,16,12,8,8). By computation we obtain a critical path $\alpha = cct_0s_0$ in $L_{2,2}$, $w_c = 10.3\ ns$, $w_{t_0} = 18.7\ ns$, and $w_{s_0} = 15.2\ ns$. Since $w(\ cct_0s_0\ ) \geq w(\ dds_0t_0\ )$, we have $max(\ c,\ d\ ) = c$. Since $cct_0s_0 \geq_w cct_0t_1$, we have $s_0 + t_0 \geq t_0 + t_1 > 2t_1$. Since $max(\ s_0,\ t_0\ ) = t_0$, we know $s_0 + t_0 \geq 2s_0 > 2s_1$ and $t_0 \geq s_0 > s_1$. Thus $max(\ s_0 + t_0,\ 2\ s_1,\ 2\ t_1\ ) = s_0 + t_0$ and $max(\ s_0 + t_0,\ s_0 + s_1,\ t_0 + t_1\ ) = s_0 + t_0$. Then from Lemma 7, we can construct a critical path $\beta = c^{10}(\ t_0s_0\ )^2t_0 \in L_{10,5}$ and calculate the delay time $w(\beta) = 189.5$ ns. In fact, actual computation shows that a critical path of $L_{10,5}$ is $\gamma = c^7(\ t_0s_0\ )c^2(\ t_0s_0\ )ct_0$ and its delay time w(γ) is 189.3 ns. Note that the critical paths γ and β are equivalent in the sense that $w(\beta) = w(\gamma) = 10c + 3t_0 + 2s_0$.

Lemma 8 Given a critical path α in $L_{1,3}$, we know the values of the R-functions in Theorem 2.

Proof Let α be a critical path in $L_{1,3}$. $\alpha \sim \beta\gamma$ where $\beta \in (\ c + d\ )$ and $\gamma \in (\ s_0 + s_1 + t_0 + t_1\ )^3$. Clearly $\beta = max(\ c,\ d\ ) = z_1$. Since $s_0s_1t_0 <_w s_0t_0s_0$ and $s_0t_0t_1 <_w t_0s_0t_0$, we have $\gamma \sim (\ s_0s_1^2 + s_0t_0s_0 + t_0s_0t_0 + t_0t_1^2\ )$. Suppose $\gamma \sim s_0s_1^2$. Then we can find the values of the R-functions in Theorem 2 as follows. Since $s_0s_1s_1 \geq_w s_0t_0s_0$, we have $s_0 + t_0 \leq 2s_1 < s_1 + s_0$. Thus $t_0 < s_1 < s_0$. Hence $max(\ s_0,\ t_0\ ) = s_0$, $max(\ s_0 + t_0,\ s_0 + s_1,\ t_0 + t_1\ ) = s_0 + s_1$ and $max(\ s_0 + t_0,\ 2s_1,\ 2t_1\ ) = 2s_1$. We can also compute the R-functions in the other cases as above. □

Now we prove Theorem 2.

Proof of Theorem 2 Suppose we know the values of R-functions in Theorem 2. From Lemma 7, there is a path β in $L_{m,n}$ such that $\beta \sim \begin{cases} z_1^m z_2 z_4^k & \text{if } n = 2k+1 \\ z_1^m z_3 z_4^{k-1} & \text{if } n = 2k. \end{cases}$ From Lemma 6, the delay time of the path β is worse than any path in $L_{m,n}$. This means that the path β itself is a critical path. Conversely, we can find the desired R-functions from a critical path in $L_{1,3}$ from Lemma 8. □

From Theorem 2 and Lemma 8, we have found a new way to implement our critical-path optimization method, as shown in Fig. 11. By analyzing the small configuration (the circuit in $C_{FA,1,3}$), we can avoid analyzing the entire circuit. This means that our optimization workload will be reduced significantly when the circuit is large.

10. A canonical configuration for the $n \times n$ multiplier

Although we found an effective way of computing a critical path of L_{mn} and its delay time, we still have a major question left. That is, does there exist an effective way to find a locally optimal parameter for the large $n \times n$ multiplier? In this section we prove that the answer to the question is yes and, furthermore, we will show a stronger result by introducing the idea of the *canonical configuration.*

Definition A circuit CC $\in C_{FA}$ is called the *canonical configuration* of the $n \times n$ multiplier iff the optimization of CC yields the same parameters as the optimization of the $n \times n$ multiplier, for all sufficiently large n.

We now consider the following problem:

Problem 1 Is there a canonical configuration CC of the $n \times n$ multiplier? If a CC exists, what is it? How can it be found?

Let $T_\pi(n)$ denote the delay time of the critical path of the $n \times n$ multiplier given a circuit parameter π. We use $T(n)$ instead of $T_\pi(n)$ when π is fixed in the discussion. We can now give an explicit formula $T_\pi(n)$ for the $n \times n$ multiplier.

Theorem 3 Given a circuit parameter π, then $T_\pi(n)$ can be represented as follows:

$$T_\pi(n) = kx_1 + x_2, \tag{1}$$

where $k = \lfloor n/2 \rfloor - 1$,

$$x_1 = 4\ max(c, d) + max(t_0 + s_0, 2s_1, 2t_1)$$

$$x_2 = \begin{cases} max(c, d) + max(t_0, s_0) & \text{if } n = 2k \\ 3\ max(c, d) + max(t_0 + s_0, s_0 + s_1, t_0 + t_1) & \text{if } n = 2k+1 \end{cases}$$

Proof Let z_1, z_2, z_3, and z_4 be as defined in Section 9. From Lemma 2, a critical path α of $n \times n$ multiplier is in $L_{2n-3,\,n-1}$. And from Lemma 6, we know that

$$\alpha \sim \begin{cases} z_1^{2n-3} z_2 z_4^k & \text{if } n-1 = 2k+1 \quad n = 2k+2 \\ z_1^{2n-3} z_3 z_4^{k-1} & \text{if } n-1 = 2k \quad \text{or} \quad n = 2k+1 \end{cases}$$

$$\alpha \sim \begin{cases} z_1^{4k-3} z_2 z_4^{k-1} & \text{if } n = 2k \\ z_1^{4k-1} z_3 z_4^{k-1} & \text{if } n = 2k+1 \end{cases} \text{ Thus,}$$

$$\alpha \sim \begin{cases} (z_1^4 z_4)^{k-1}(z_1 z_2) & \text{if } n = 2k \\ (z_1^4 z_4)^{k-1}(z_1^3 z_3) & \text{if } n = 2k+1 \end{cases}$$

Note that $k = \lfloor n/2 \rfloor - 1$. □

Corollary 1 Given a circuit parameter π, then $T_\pi(n)$ is asymptotically proportional to n.

Proof This is clear from Eq. (1). □

In fact, from Table 4 in Section 5, we can obtain empirically the formula $T_{\pi_1} = 40(n\text{–}1)$ when $\pi_1 = (4,16,8,8)$.

Corollary 2 The circuit $C_1 \in C_{FA,4,2}$ shown in Fig. 12 is a canonical configuration of the $n \times n$ multiplier. The critical path of C_1 has four carry stages and two sum stages.

Proof From Theorem 3, we have $T(n) = \alpha_{4,2}\lfloor n/2 \rfloor + \beta$, where $\alpha_{4,2}$ is the delay time of four carry stages and two sum stages and β is the constant delay time. Hence an optimal parameter of $L_{4,2}$ is, asymptotically for large n, also an optimal parameter of L_{mn}. □

From Lemma 2 in Section 7, there are $(2n-3)$ carry stages and $(n-1)$ sum stages in any critical path of the $n \times n$ multiplier. Thus we might expect that the circuit $C_0 \in C_{FA,2,1}$, whose critical path has two carry stages and one sum stage, is a canonical configuration of the $n \times n$ multiplier. However, as we saw in this section, the circuit $C_1 \in C_{FA,4,2}$ is a canonical configuration, but the circuit $C_0 \in C_{FA,2,1}$ is not a canonical configuration. The reason is that optimization of the circuit C_0 cannot determine $max(s_0 + t_0, 2s_1, 2t_1)$.

11. The optimization of a subcircuit

In this section, we optimize the two circuits C_0 and C_1 discussed in the previous section, verifying that the circuit C_1 works well as a canonical configuration, but the circuit C_0 does not. The circuit C_1 is indicated by a solid line in Fig. 12, while the circuit C_0 is indicated by a dotted line. The critical path from cell $A_{0,0}$ to cell $A_{2,2}$ is analyzed for C_1 in Fig. 12, while the critical path from cell $A_{0,0}$ to cell $A_{1,1}$ is analyzed for C_0.

Table 6 shows which parameters are locally optimal for m33, m44, m88, m1010, C_0 and C_1. The symbol $\times$ indicates a local optimum. The same set of 11 random initial parameters were used for each circuit, and only these 5 distinct local optima were obtained.

The most important result is that every local optimum of m88 and m1010 is also a local optimum of C_1. This is not true for C_0, nor is it true for the smaller circuits m33 and m44. Thus we can say that the circuit C_1 is indeed appropriate as a representative subcircuit of the $n \times n$ multiplier. In this way, Corollary 2 in Section 9 is confirmed very well by numerical experiments.

	π_1	π_2	π_3	π_4	π_5
C_0	×				×
C_1	×	×		×	
m1010	×	×		×	
m88	×	×		×	
m44	×	×	×		
m33	×	×	×		

Table 6. Locally optimal parameters for each circuit

12. Conclusions

We have described a general approach for sizing the transistors in a cell that is embedded in a regular array, using local search along the critical path. The simplest, most regular array multiplier structure was used as an example, with delay time (not throughput) as a criterion. No attempt was made to incorporate intermediate clocking, precharging, or superbuffers.

We quickly encountered the problem that the running time of the optimization increases rapidly when we increase the size of the multiplier. We therefore tried to make our optimization method more practical by making use of the circuit's regularity and developed what we call the *canonical configuration method.* In this method we locally optimize the small circuit C_1 instead of the entire circuit. We showed how to extract this canonical configuration C_1 for the $n \times n$ multiplier, and gave experimental results that illustrate the savings offered by this method.

The following problems are the subject of future investigation: Do there exist canonical configurations in more general regular arrays? If the canonical configuration exits for some regular array, how can we extract it?

References

1. P. R. Cappello and K. Steiglitz, "Digital Signal Processing Applications of Systolic Algorithms," *CMU Conference on VLSI Systems and Computations,* H.T. Kung, Bob Sproull, and Guy Steele (eds.), Computer Science Press, Rockville, Md., 1981.

2. P. R. Cappello and K. Steiglitz, "Completely Pipelined Architectures for Digital Signal Processing," *IEEE Trans. on Acoustics, Speech, and Signal*

Proc., vol. ASSP-31, no. 4, pp. 1016-22, August 1983.

3. P. R. Cappello and K. Steiglitz, "A Note on 'Free Accumulation' in VLSI Filter Architectures," *IEEE Trans. on Circuits and Systems*, in press.

4. C. Caraiscos and B. Liu, "Bit Serial VLSI Implementations of FIR and IIR Digital Filters," *Proc. IEEE Int. Symp. on Circuits and Systems,* May 1983.

5. P. B. Denyer and D. J. Myers, "Carry-Save Arrays for VLSI Signal Processing," in *VLSI 81: Very Large Scale Integration,* John P. Gray (ed.), Academic Press, London, 1981. (*Proceedings of the First International Conference on Very Large Scale Integration,* University of Edinburgh, August 18-21, 1981.)

6. L. A. Glasser and L. P. J. Hoyte, "Delay and Power Optimization in VLSI Circuits," *Proc. IEEE 21st Design Automation Conference*, 1984, pp. 529-535.

7. K. Iwano and K. Steiglitz, "Some Experiments in VLSI Leaf-cell Optimization," 1984 IEEE Workshop on VLSI Signal Processing, pp. 387-395, University of Southern California, Nov. 12-14, 1984.

8. K. Iwano and K. Steiglitz, "Time-Power-Area Tradeoffs for the nMOS VLSI Full-adder," *1985 Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, pp 1453-56, Tampa, Florida, Mar., 1985.

9. H. T. Kung, L. M. Ruane, and D. W. L. Yen, "A Two-Level Pipelined Systolic Array for Convolutions," *CMU Conference on VLSI Systems and Computations,* H.T. Kung, Bob Sproull, and Guy Steele (eds.), Computer Science Press, Rockville, Md., 1981.

10. E. Lawler, *Combinatorial Optimization*, Holt, Renehart and Winston, New York, N. Y., 1976.

11. R. J. Lipton, S. C. North, R. Sedgewick, J. Valdes, and G. Vijayan, "VLSI Layout as Programming," *ACM Trans. on Programming Languages and Systems*, July 1983.

12. R. F. Lyon, "A Bit-Serial VLSI Architecture Methodology for Signal Processing," in *VLSI 81: Very Large Scale Integration,* John P. Gray (ed.), Academic Press, London, 1981. (*Proceedings of the First International Conference on Very Large Scale Integration,* University of Edinburgh, August 18-21, 1981.)

13. J. Mata, "ALLENDE: A Procedural Language for the Hierarchical Specification of VLSI Layouts," *Proc. IEEE 22nd Design Automation Conference*, 1985.

14. M. D. Matson, "Macromodeling and Optimization of Digital MOS VLSI Circuits," *Ph.D thesis, Department of EECS, M. I. T.*, Jan., 1985.

15. R. N. Mayo, J. K. Ousterhout, and W. S. Scott, "1983 VLSI Tools," Report No. UCB/CSD 83/115, Computer Science Division (EECS), University of California, Berkeley, Calif., March 1983.

16. J. V. McCanny, J.G. McWhirter, J. B. G. Roberts, D. J. Day, and T. L. Thorp, "Bit Level Systolic Arrays," *Proc. 15th Asilomar Conf. on Circuits, Systems, and Computers*, Nov., 1981.

17. S. Trimberger, "Automated Performance Optimization of Custom Integrated Circuits," in *VLSI 83: VLSI Design of Digital Systems*, F. Anceau and E. J. Aas (eds.), North Holland, Amsterdam, 1983. (*Proceedings of the IFIP International Conference on Very Large Scale Integration*, Trondheim, Norway, August, 1983.)

18. A. J. Strojwas, S. R. Nassif, and S. W. Director, "Optimal Design of VLSI Minicells using a Statistical Process Simulator," *Proc. IEEE International Conference on Circuit and System*, 1983, pp. 202-205.

19. J. K. Ousterhout, "Switch-Level Delay Models for Digital MOS VLSI," *Proc. IEEE 21st Design Automation Conference*, 1984, pp. 542-547.

20. D. J. Myers, "Multipliers for LSI and VLSI Signal Processing Applications," Masters Degree Thesis, Edinburgh University, Edinburgh, England, Sept. 1981.

21. J. Allen, "VLSI Architectures for Signal Processing," in *VLSI architecture*, B. Randell and P. C. Treleaven (eds.), pp. 242-254, Prentice-Hall Inc., Englewood Cliffs, N. J., 1983.

22. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Co. Menlo Park, Ca., 1980.

23. N. Jouppi, "Timing Analysis for nMOS VLSI," *Proceedings 20th Design Automation Conference*, IEEE, June 1983, pp. 411-418.

24. J. E. Hopcroft and J. P. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Co. Menlo Park, Ca., 1979.

25. C. W. Wu, P. R. Cappello, and M. Saboff, "An FIR Filter Tissue," *1985 Proc. 19th Asilomar Conference on Circuit, Systems, and Computers*, Pacific Grove, CA, Nov., 1985.

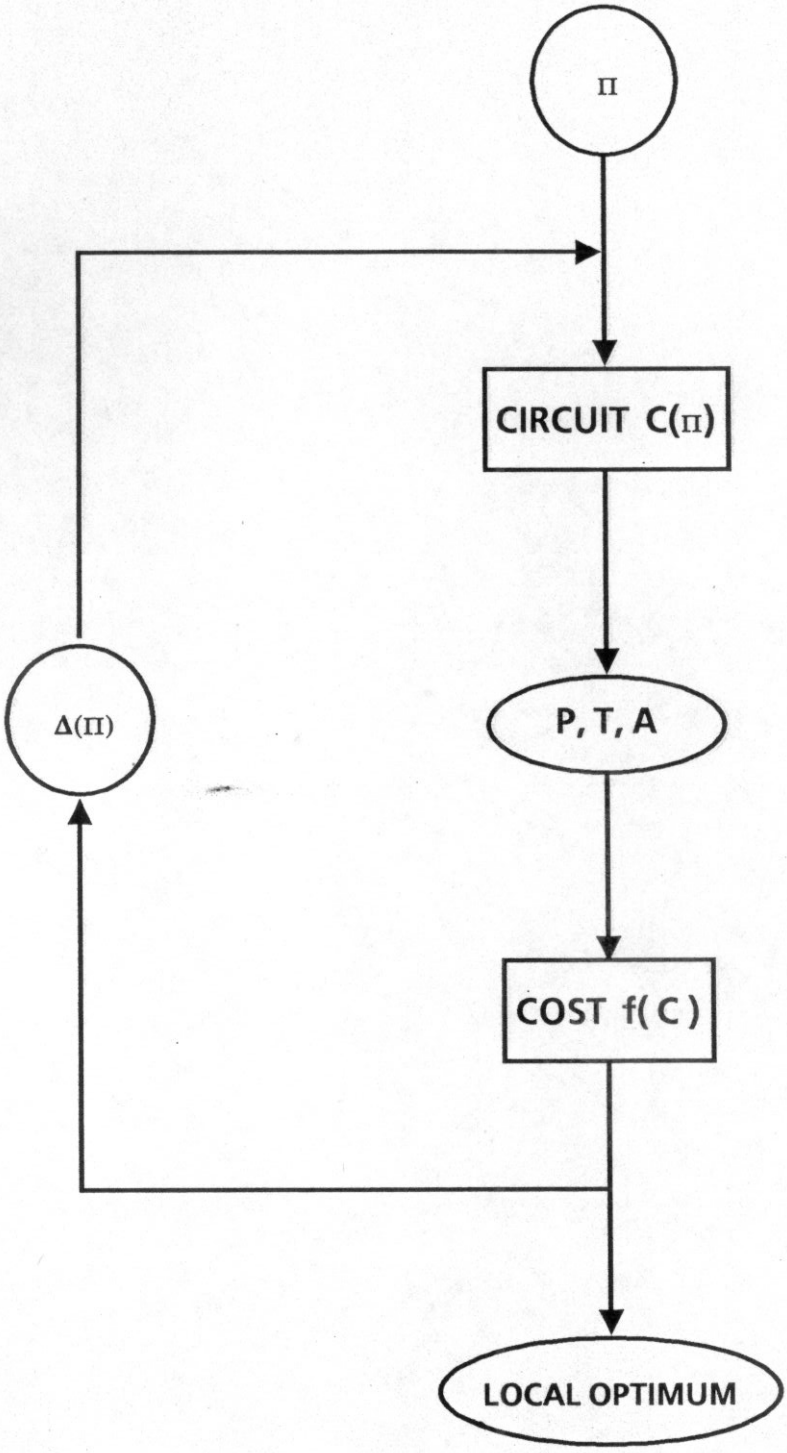

Figure 1. Critical-path optimization

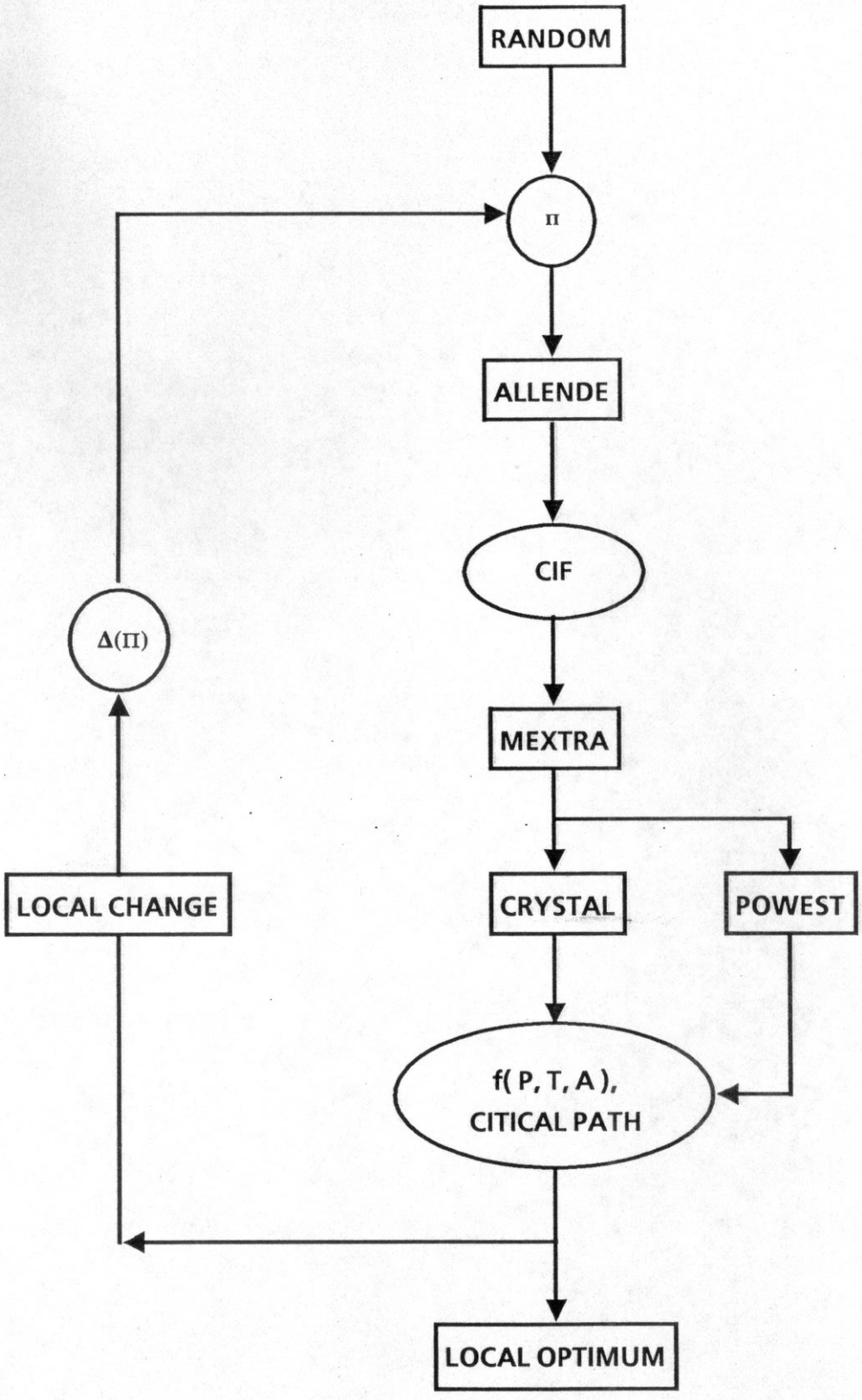

Figure 2.
Detailed flowchart of the critical-path optimization method

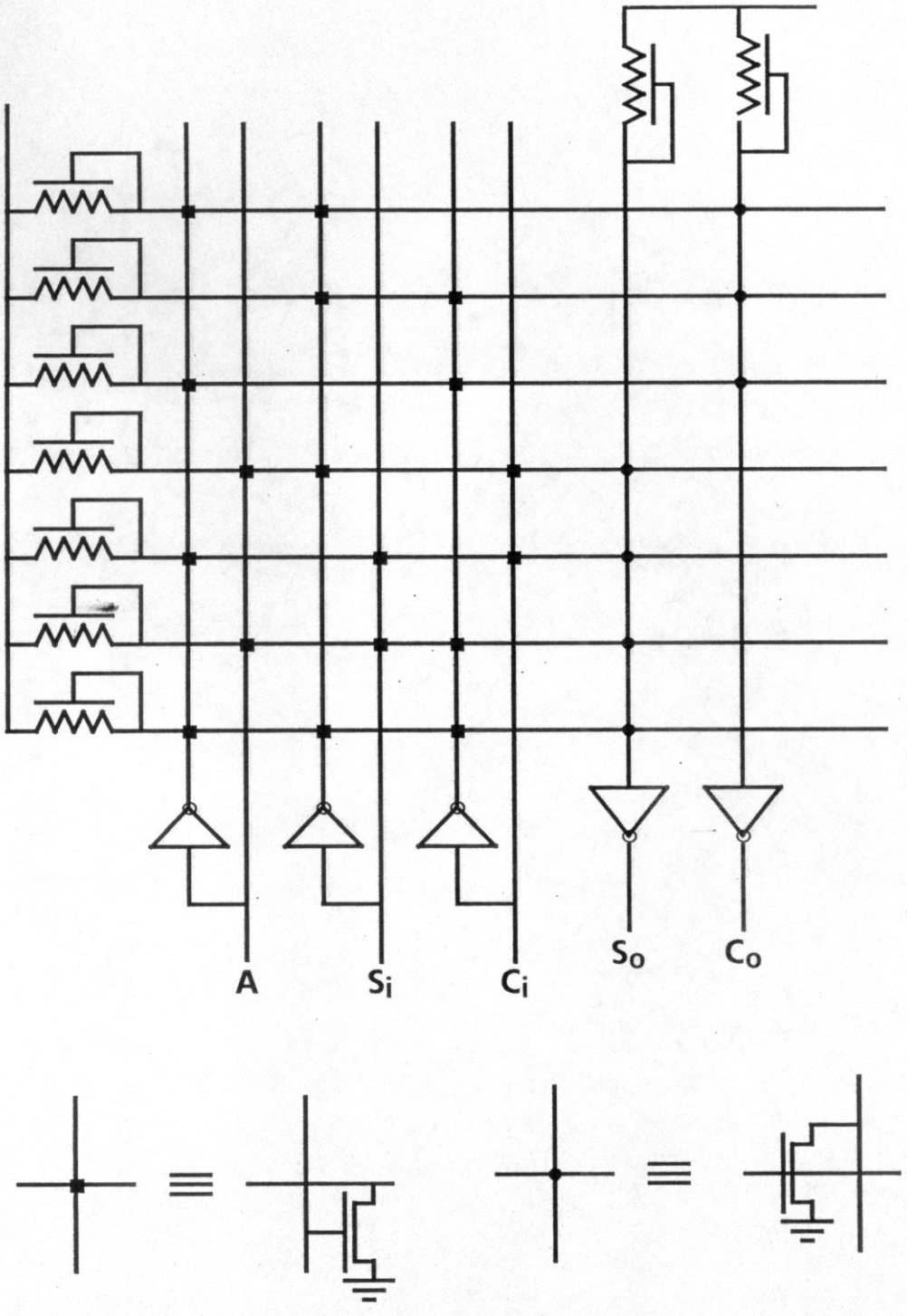

Figure 3a. Circuit diagram of the PLA

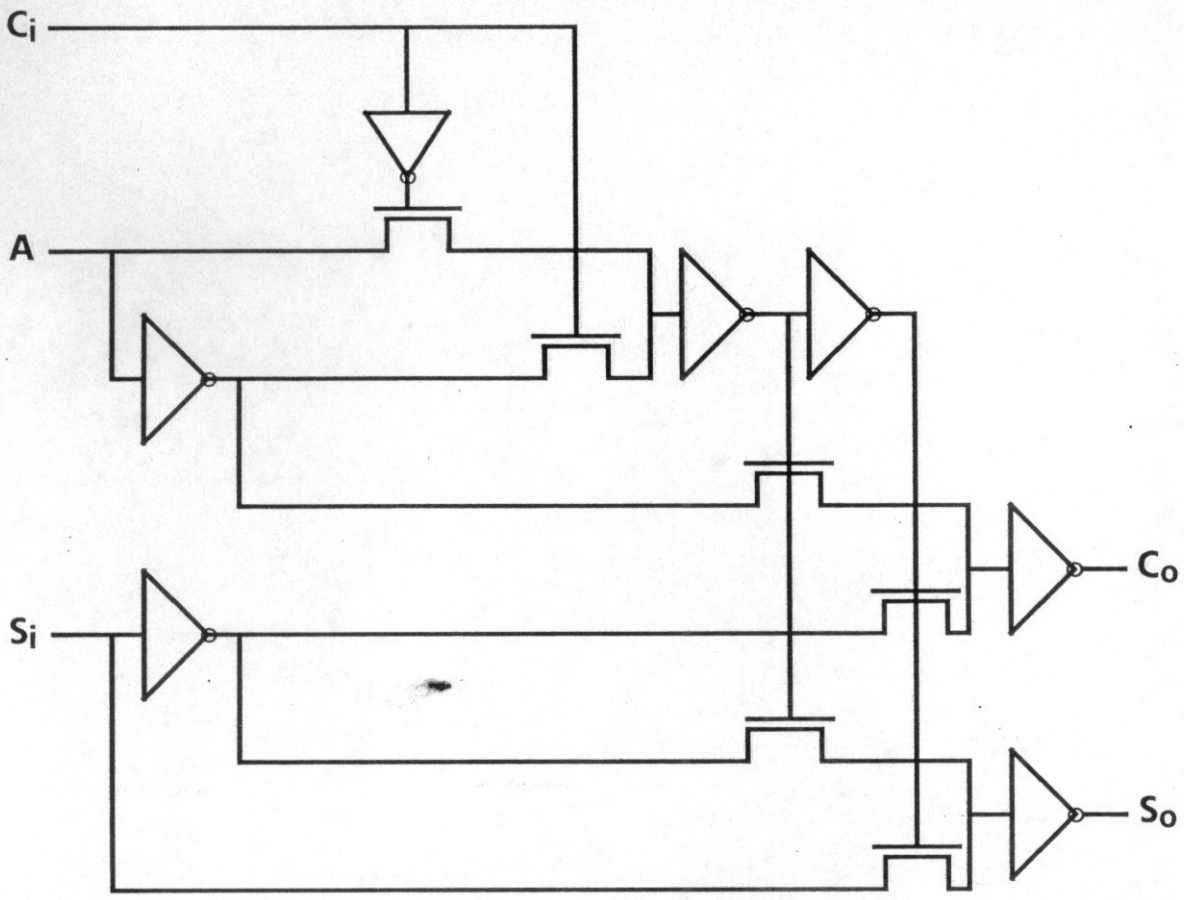

Figure 3b. Circuit diagram of the Data Selector

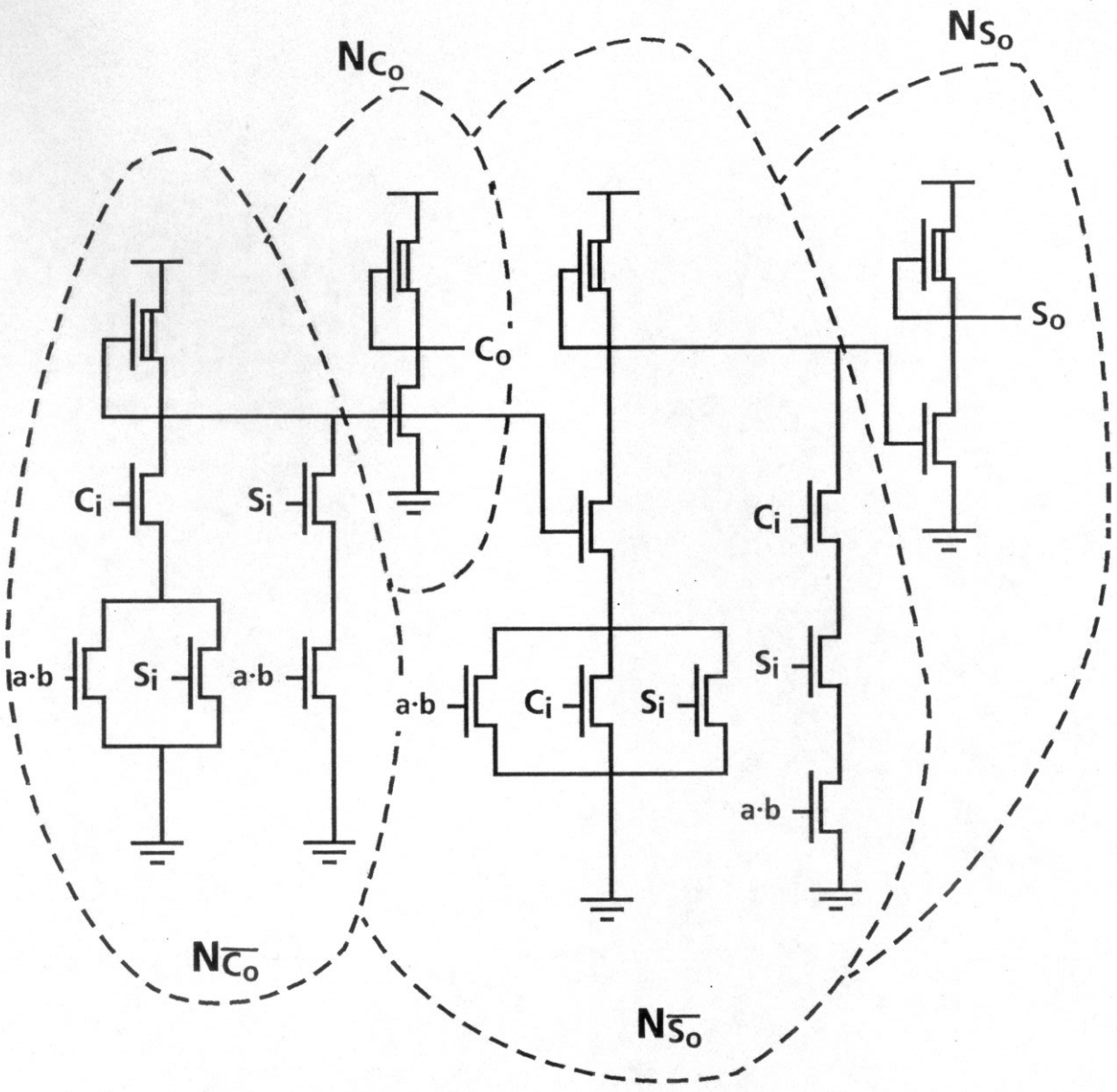

Figure 3c. Circuit diagram of the Random Logic circuit

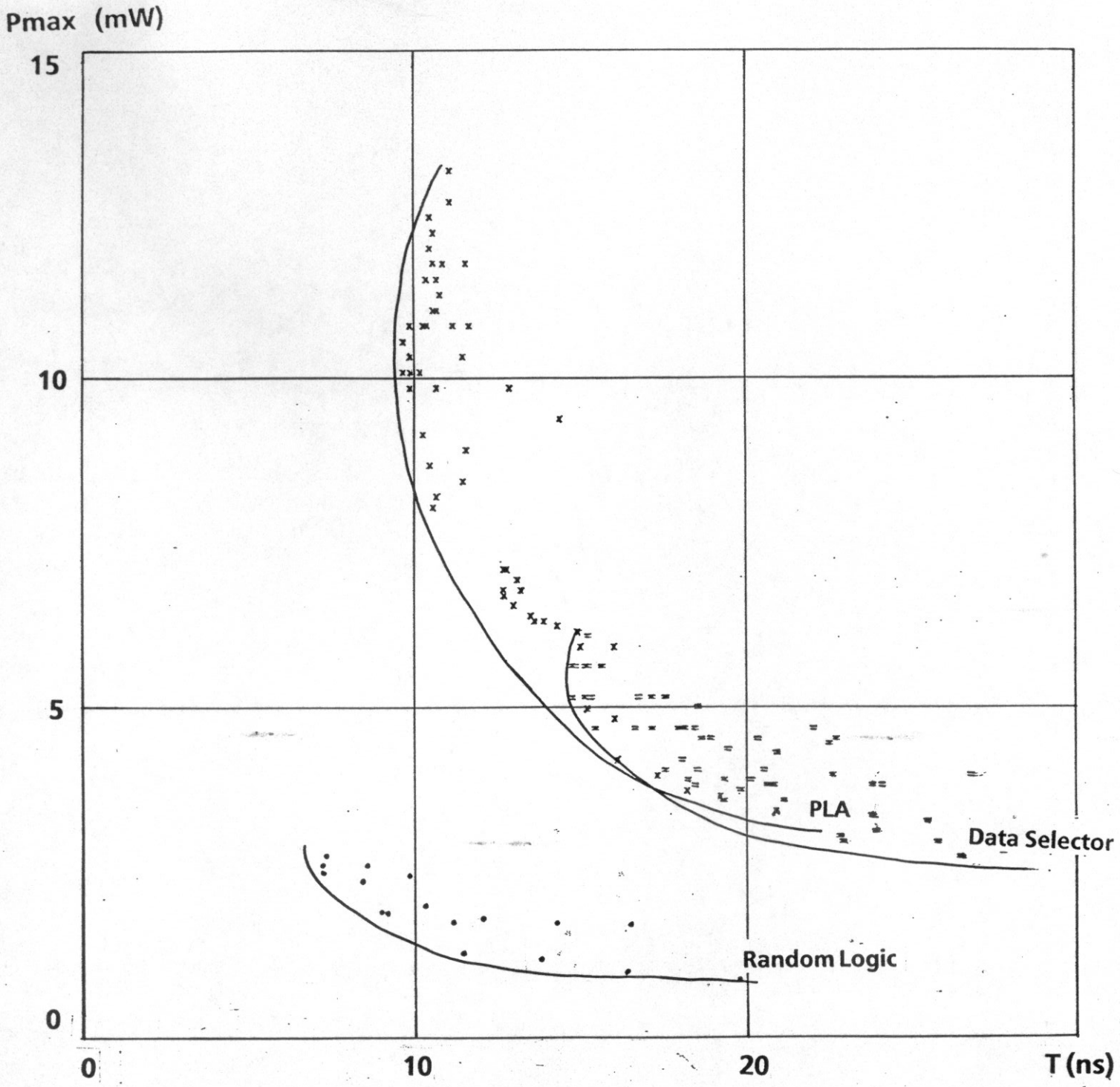

Figure 4.
The P-T trade off curves of one-bit full adder circuits

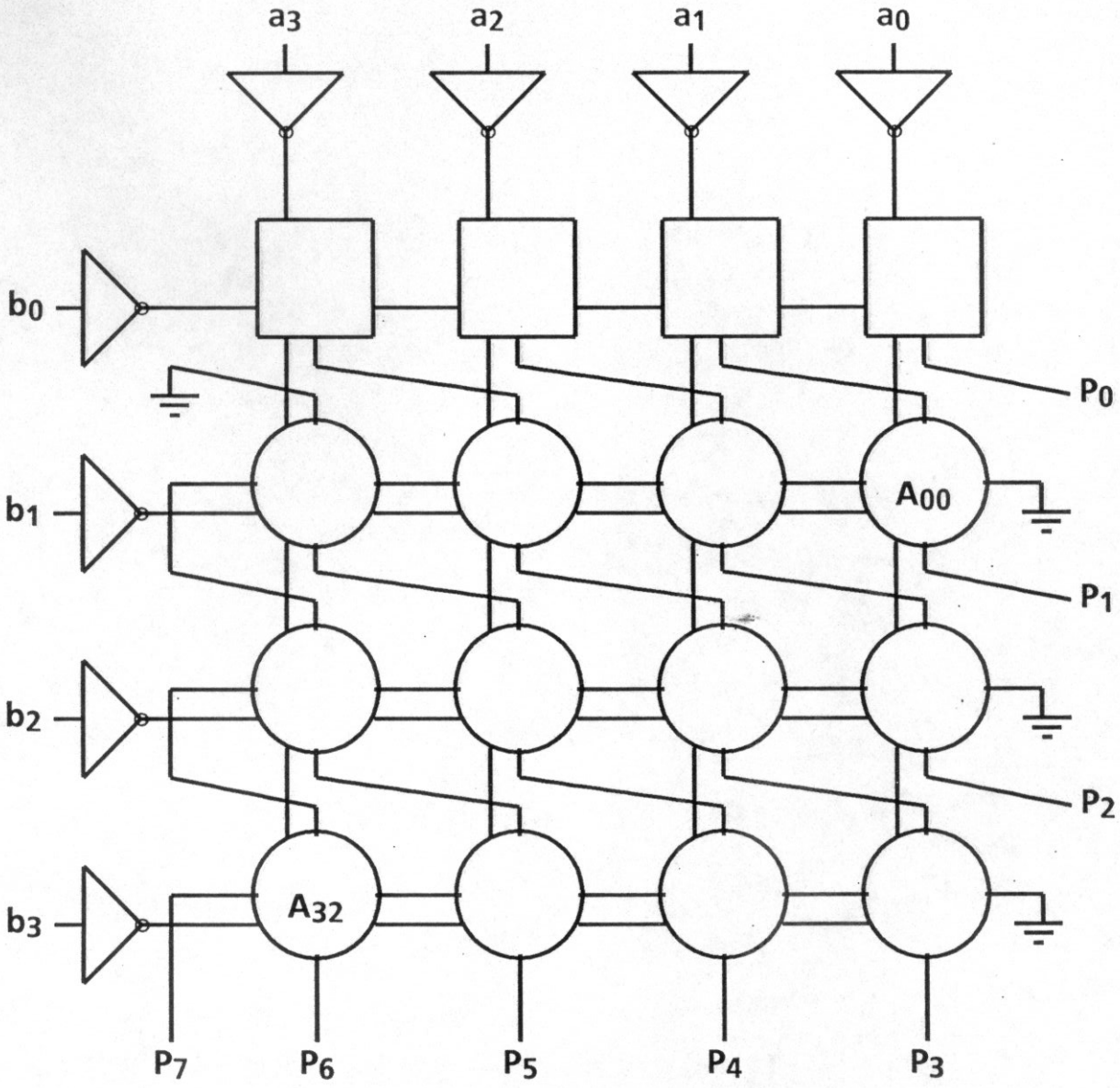

Figure 5a. 4 by 4 multiplier

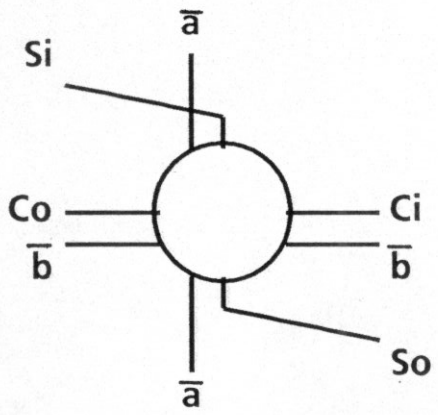

Figure 5b. One-bit full adder cell

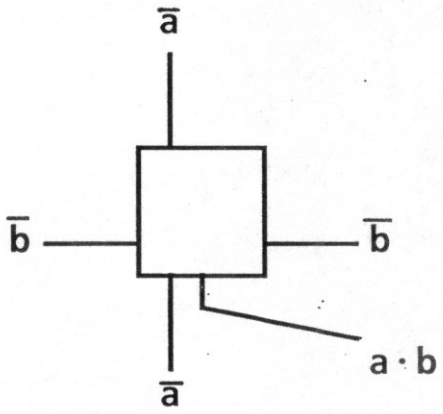

Figure 5c. AND cell

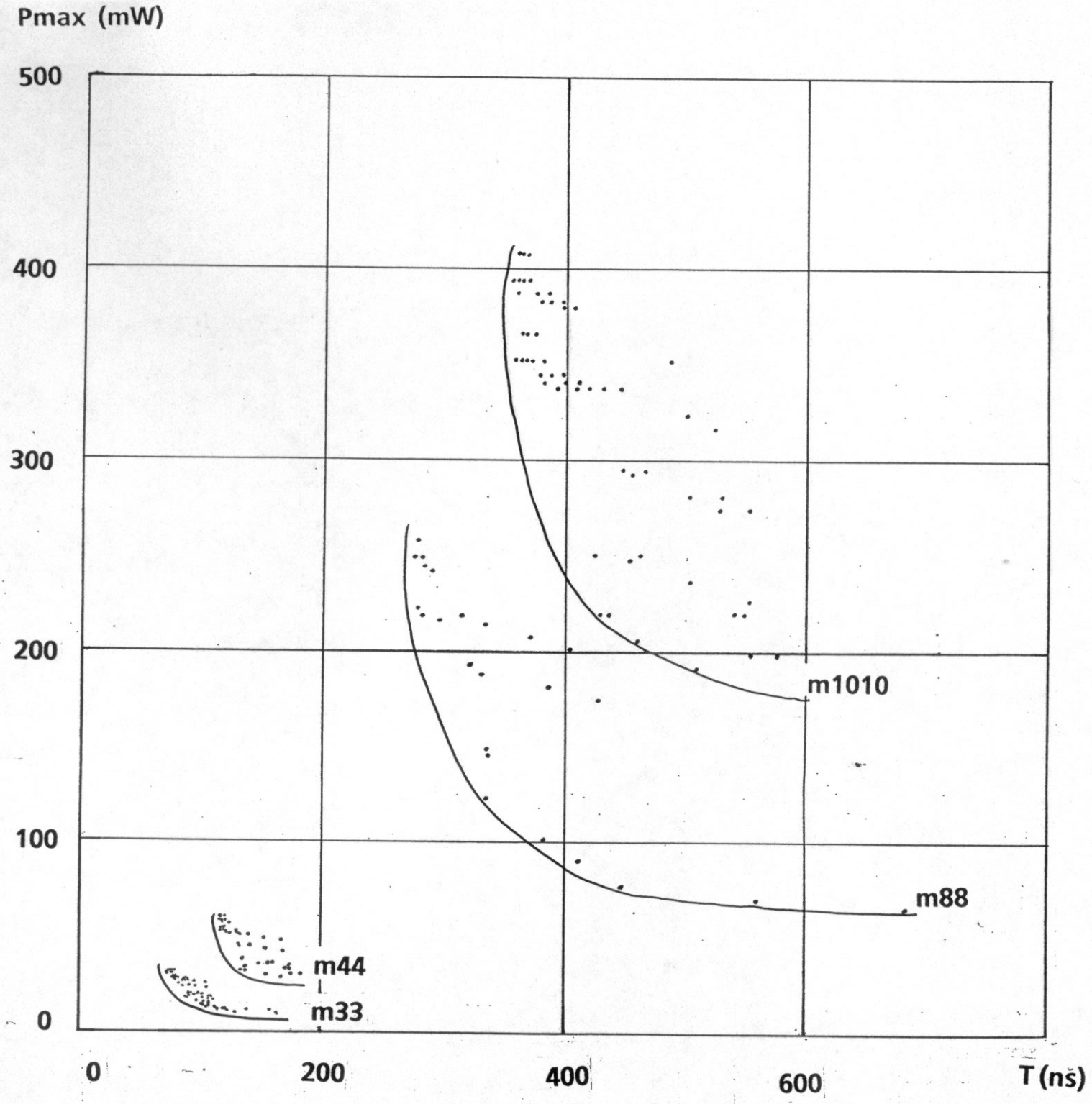

Figure 6.
The P-T trade off curves of multipliers

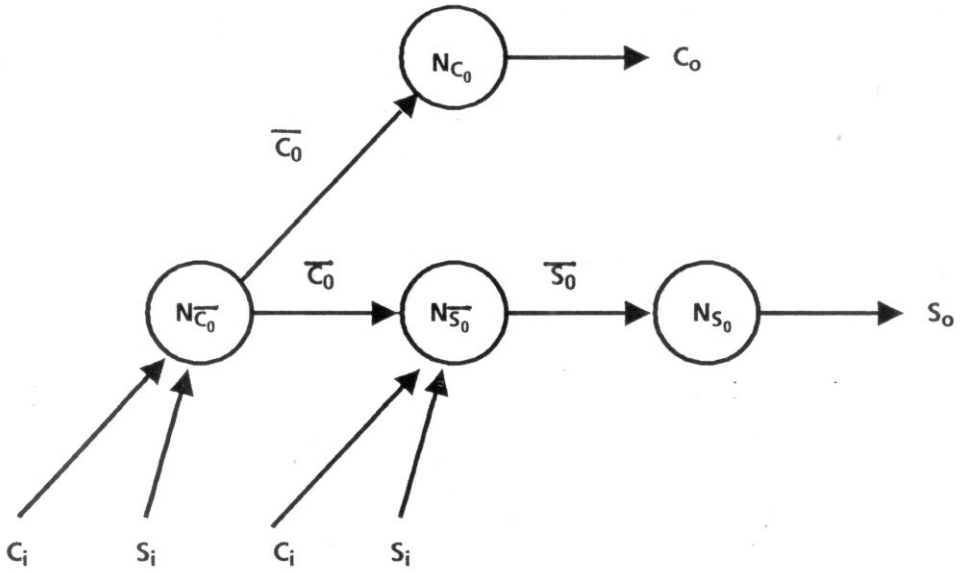

Figure 7.
The nodes in the one-bit full adder

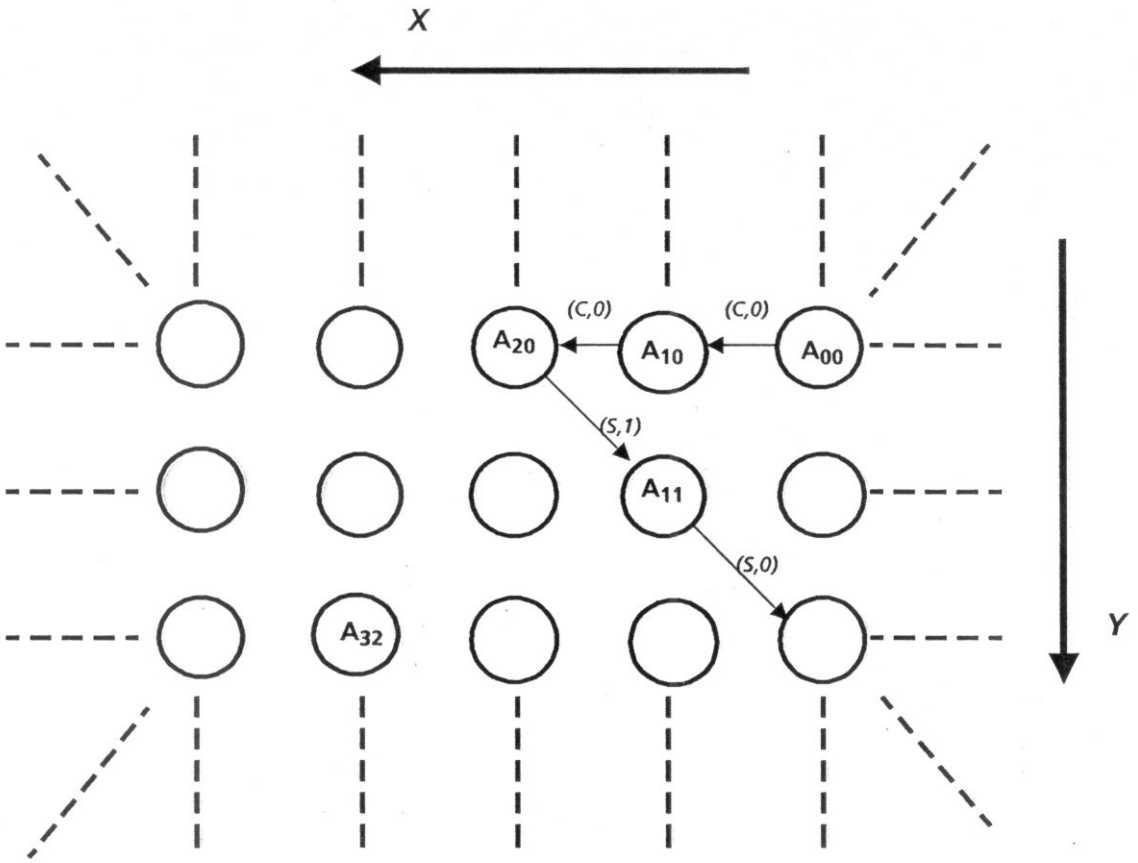

Figure 8.
Two-dimensional array of one-bit full adders

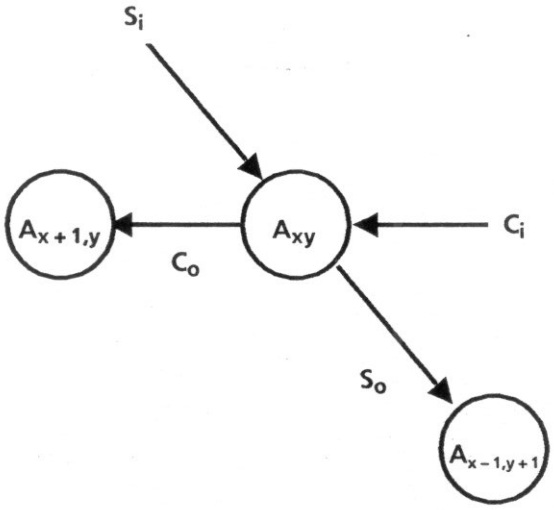

Figure 9.
The basic cells and the interface rule

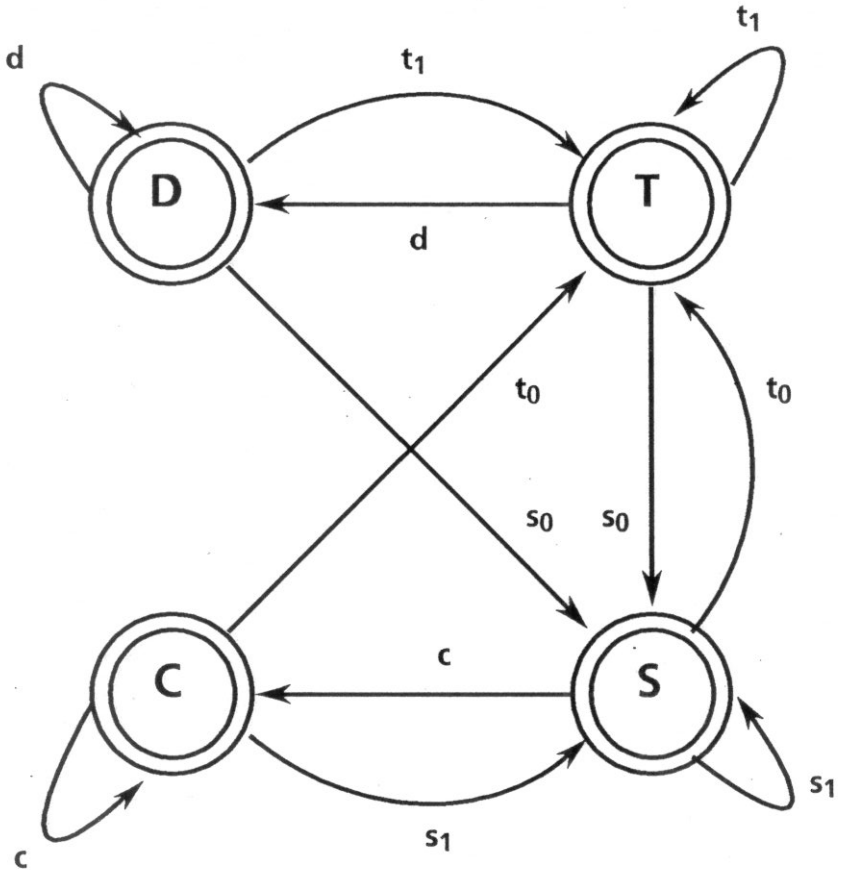

Figure 10a.
State transition diagram along a critical-path

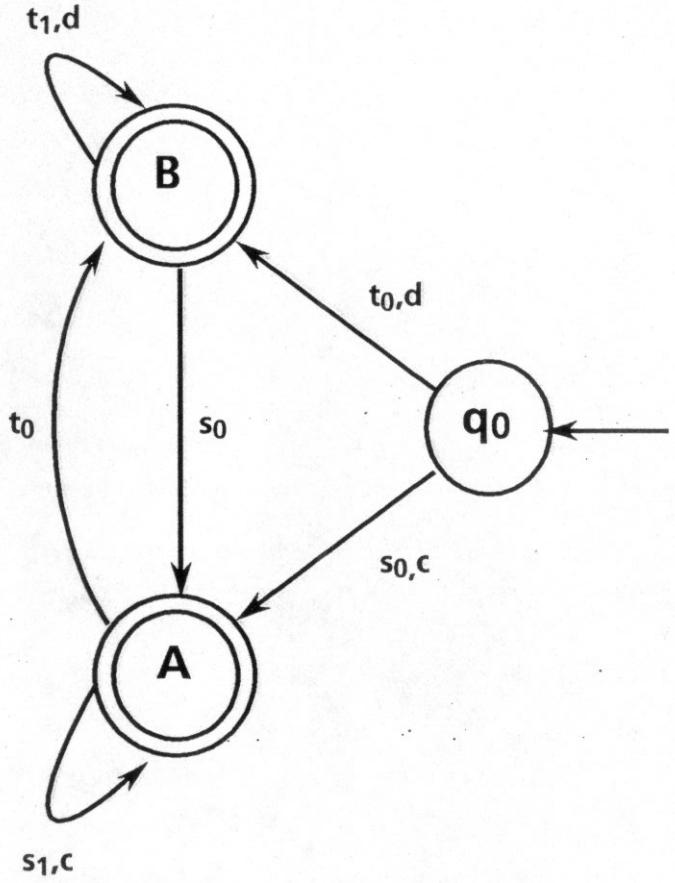

Figure 10b.
Minimized state transition diagram

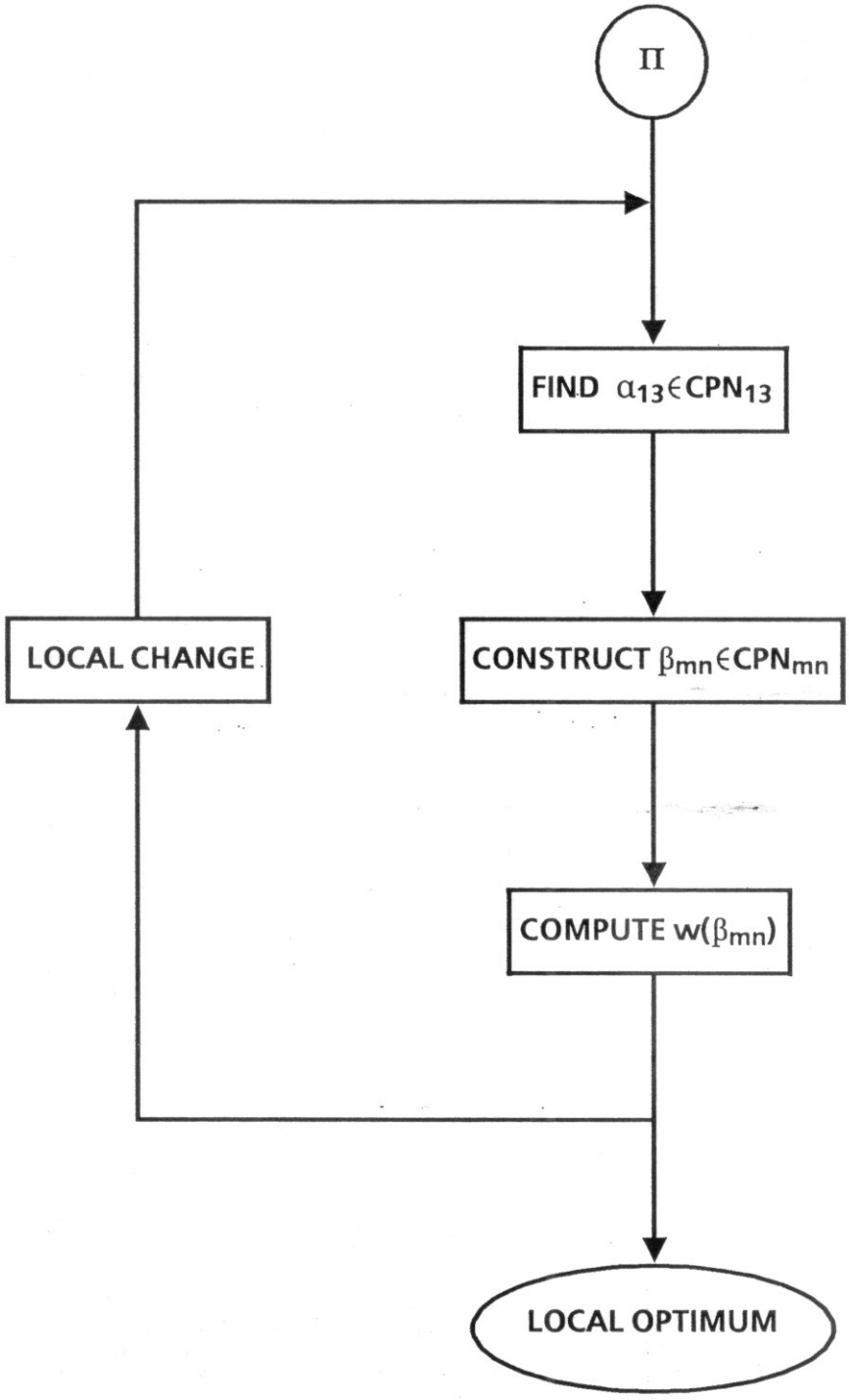

Figure 11.
New implementation of the critical-path optimization method

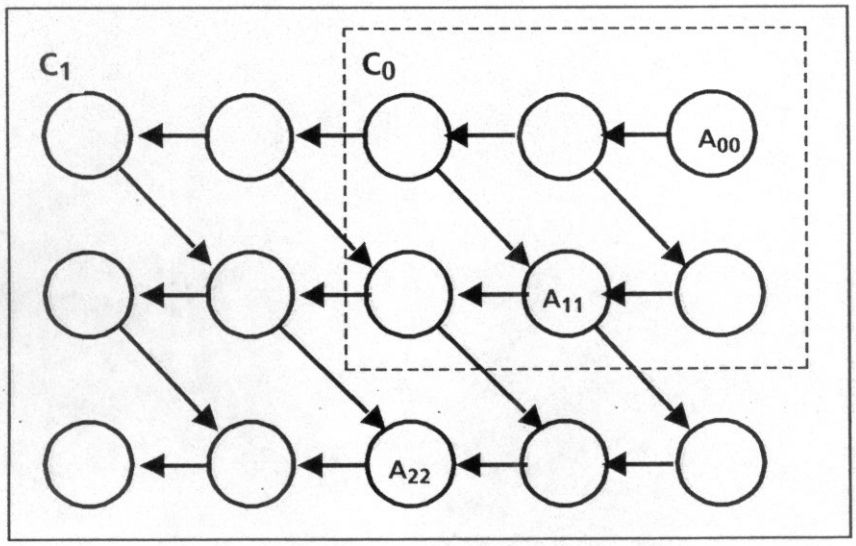

Figure 12. The circuits C_0 and C_1