TRI-STATE GATES IN LOGICAL NETWORKS

by

Robert B. Aglietti

Thesis submitted to the Graduate Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

R. A. Thompson, Chairman

T. E. Bechert

A. W. Bennett

F. G. Gray

October, 1975

Blacksburg, Virginia

ACKNOWLEDGEMENTS

I would like to thank for generosity with his time and intuition, and National Science Foundation for which provided funding for much of the research which went into this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

I. INTRODUCTION

1.1 Motivation

Since its introduction in industry, tri-state logic has found its way into many digital systems; specifically bus oriented systems like micro and mini-computers. The logic is extremely useful and well suited to the task of acting as a high-impedance "switch" to allow several peripherals to be connected to a bus without the usual interface problems associated with systems of this type.

Because tri-state logic was so well suited to the bus interface problem, little thought was given to the application of this type of logic in other digital networks where the bussing problem is not a factor. Intuitively, a third state is worth considering since it should offer increased computational capability or simplification of realizations. It is with this thought in mind that this thesis is presented.

1.2 Output Structure of a Tri-State Gate

Before beginning a study of tri-state logic and its use in logic circuits, it is necessary to understand what separates tri-state from other presently available logic families. [1] Figure 1 illustrates a standard "totem-pole" output structure in a TTL gate. [2]

When operating properly, Q1 acts as a phase splitter which when a low output is indicated, is ON, and causes Q3

to be ON, putting the output Z at $V_{ce(sat)}$ above ground. For an output of logic ONE, Q1 is OFF and Q2 is driven into saturation causing the output to go high. In either case, one but not both of the output transistors is OFF. It is here that the similarity between standard logic families and those equipped with tri-state ends. When the tri-state option is enabled in a gate, both output transistors are cut off. This has the effect of placing the output in a high-impedance state, as the only current which the gate can sink is the leakage current in Q3 which is in the order of 10's of micro-amps. This property is what has led to the wide spread use of tri-state gates in bus oriented digital systems, since disabled gates (those in the high-impedance state) are in effect out of the circuit and not connected to the system bus.

At this point, some clarification of terminology is indicated. When referring to tri-state gates, the gate in question will be enabled when its output is not tri-state, that is, when its output is a conventional logic ZERO or ONE. The gate will be disabled when it is in tri-state and its output is in the high impedance state.

1.3 Special Considerations

There are several related factors which must be considered when using tri-state logic in a network of TTL or other current sinking logic families. These factors arise

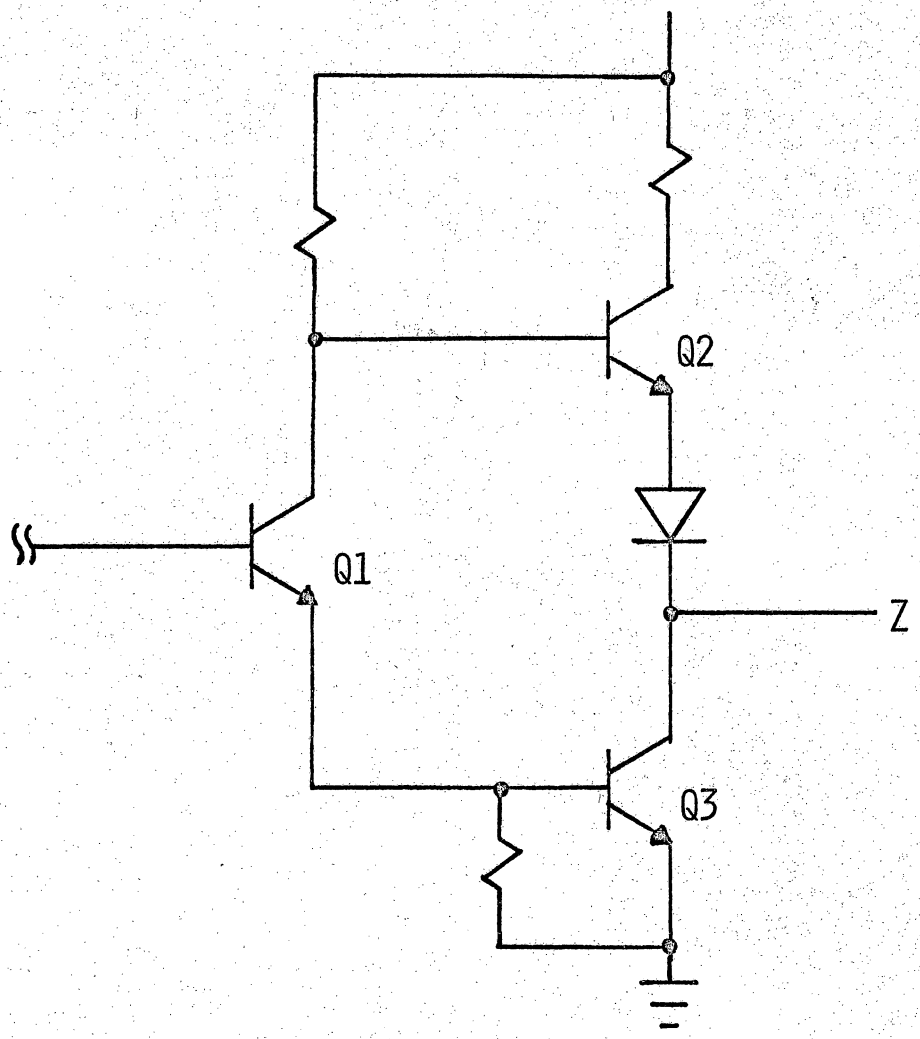


Figure 1 Totem pole output structure in a TTL gate

from the difference in the output structure of the gates. The first consideration is how the driven gate interprets the high impedance state when it is present on a gate input. In two level NAND sum-of-products or two level OR-AND product-of-sums forms the high impedance state in the output gate is equivalent to the implicant the logic represents being removed from the function. The implicant removal arises from the fact that the gate being driven interprets the high impedance as a negative assertion logical ZERO in the first case and a positive assertion logical ONE in the second case. In realizations of the forms mentioned, this signal into the output gate has no effect on the truth value of the function and the implicant is "removed". Thus, tri-state logic does not have a third logical state as may be implied by the name, but incorporates a state analogous to a don't care. The implicant's removal is effected by making the logic output of the network which realizes it a don't care.

The second consideration results from one of the combinational design procedures. It is the nature of TTL logic gates that an input which is left floating will usually be interpreted as a logic ONE. In essence, by floating it is meant that there is little or no current flow into or out from a certain point. Thus it is obvious that the high impedance state when present on a gate input will be interpreted in most cases as a logic ONE. It is the nature of one of the combinational design procedures that there is the

possibility of this very situation arising. It has been found, however, that this situation is often undesirable. For this reason, in some cases, a function will be realized as the function complement driving an inverter. It will be shown that this eliminates the difficulties and indeed in some cases results in a cheaper realization than would otherwise be expected. An illustrative example will be given in Chapter 3.

The use of tri-state gates also allows a true "wired-OR" connection as opposed to the wired-AND which is currently in use with open-collector gates. The junction of several open-collector gates whose outputs are tied common will be a logic ONE if and only if all outputs are high. If any one gate output is low it will draw the common point low. Thus, the function realized is positive assertion AND. In the case of tri-state gates see Figure 2 for a general configuration for a common output.

In Figure 2, each logic block has the same assertion enable and so both cannot be enabled simultaneously. It is possible to provide the output of one or the other logic block. It should be noticed that much the same function can be accomplished with open-collector gates. However, this method requires pull up resistors which cause an increase in both board space and power consumption. In certain cases, pull up resistors may be required for the tri-state realizations if there is the possibility that at some point there will be only disabled gates at some common

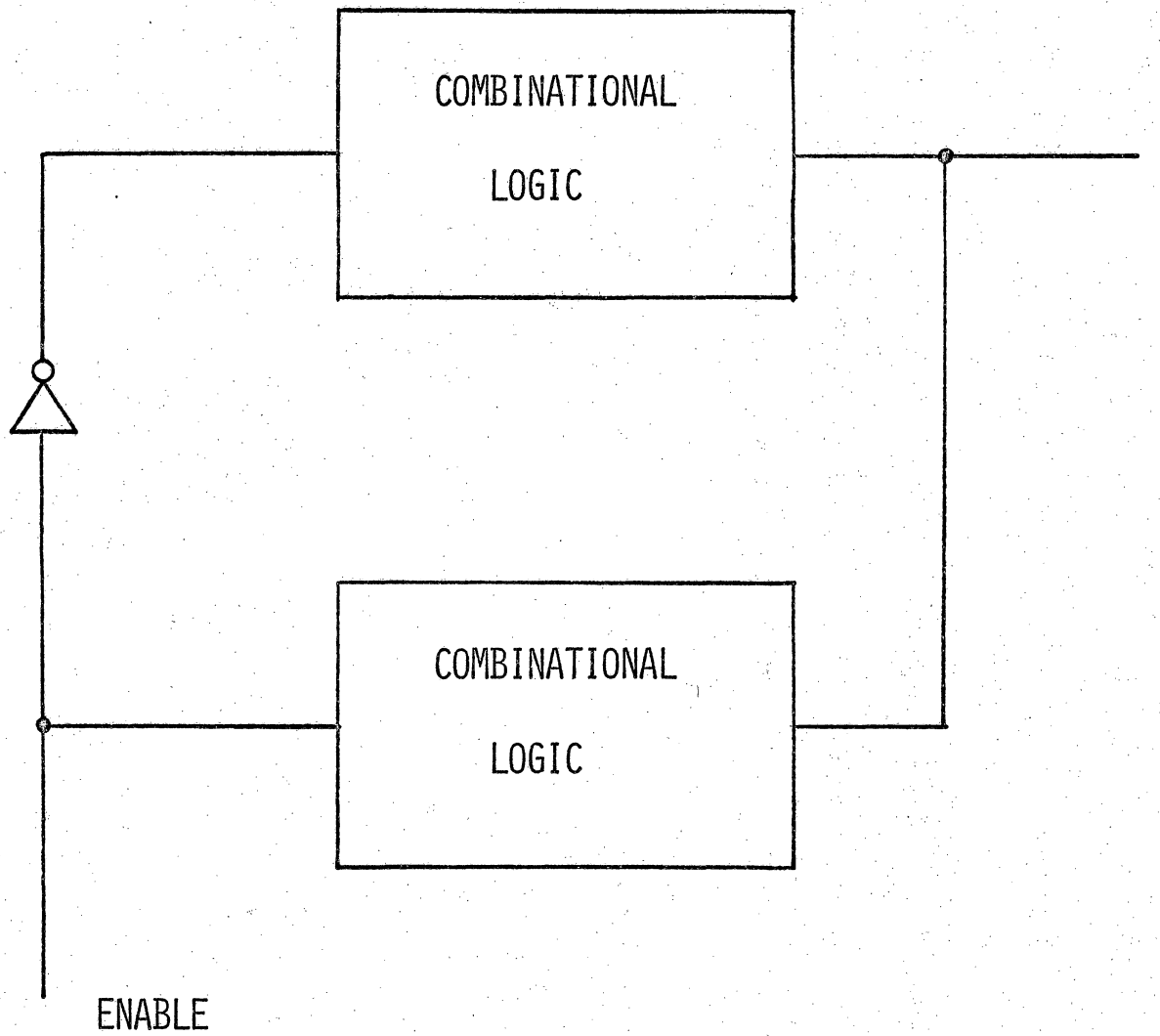


Figure 2 Combinational logic tri-state common output

point. Clearly, this situation is excluded for realizations of the form of Figure 2.

1.4 Capabilities of Tri-State Logic

In the first section, in the discussion of the differences between tri-state and conventional logic it was stated that tri-state gates, when disabled, present a high impedance source to the driven gates and are in effect out of the circuit. Here there are two cases to be considered; the first is where several gates act as inputs to an output AND/NAND gate as shown in Figure 3a. In this case, disabled gates present a logic ONE to the output AND/NAND gate and do not affect the truth value of the function. The second case is shown in Figure 3b. Here, the disabled gate's output, since it is high impedance, causes the logical function realized by the associated logic to be a don't care as its value is never seen; the logic block is removed from the circuit. It is this property which led to the present investigation of high impedance logic. If gates can be logically removed from the network the capability exists to reconfigure the network either by external control or by dynamic control by the network itself. Intuitively, this leads to an increase in the computational capabilities and/or a decrease in the structural complexity of the network. It would appear that the reconfiguration capability brings with it the ability to simplify logic as follows.

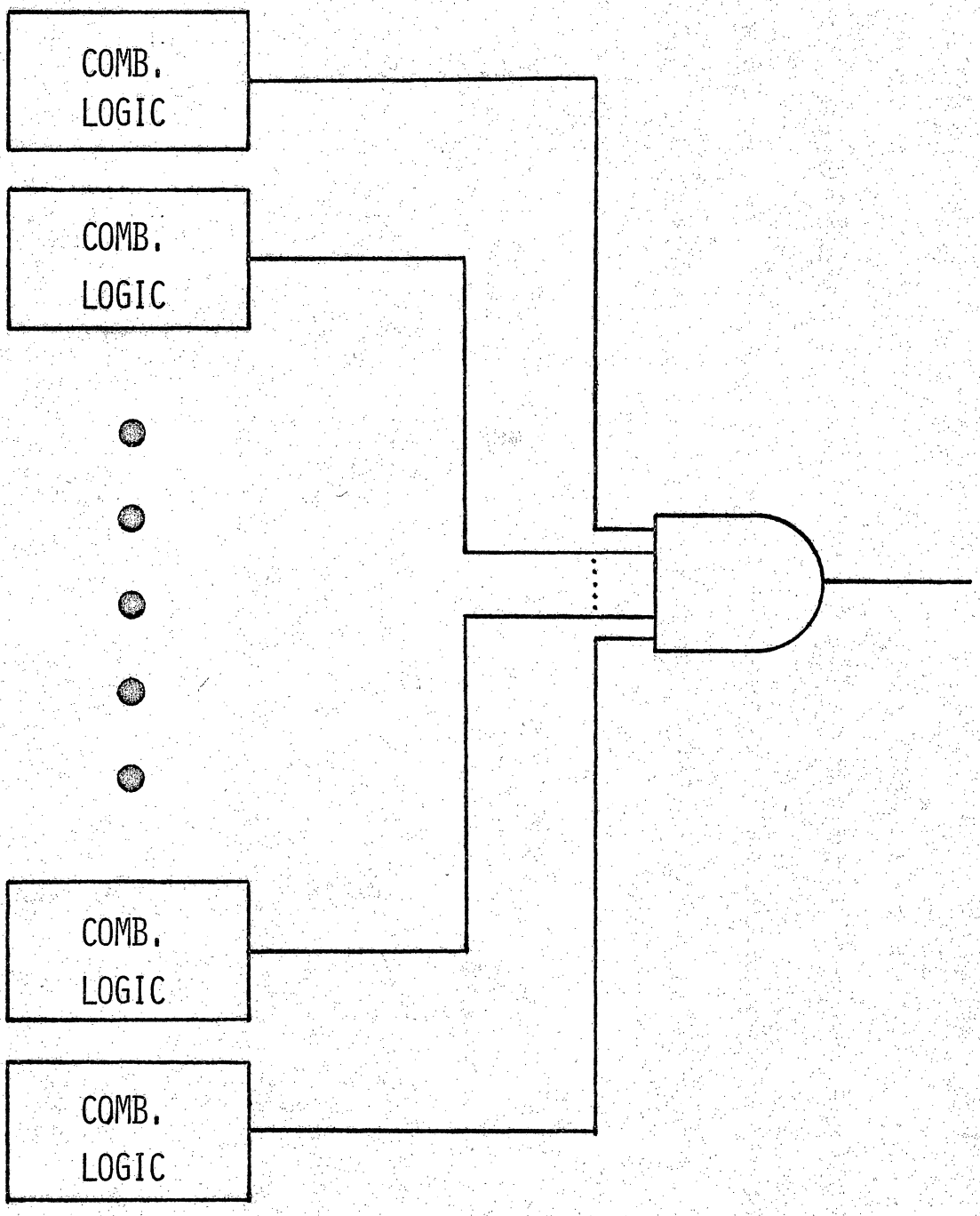


Figure 3a Removal of logic using tri-state gates

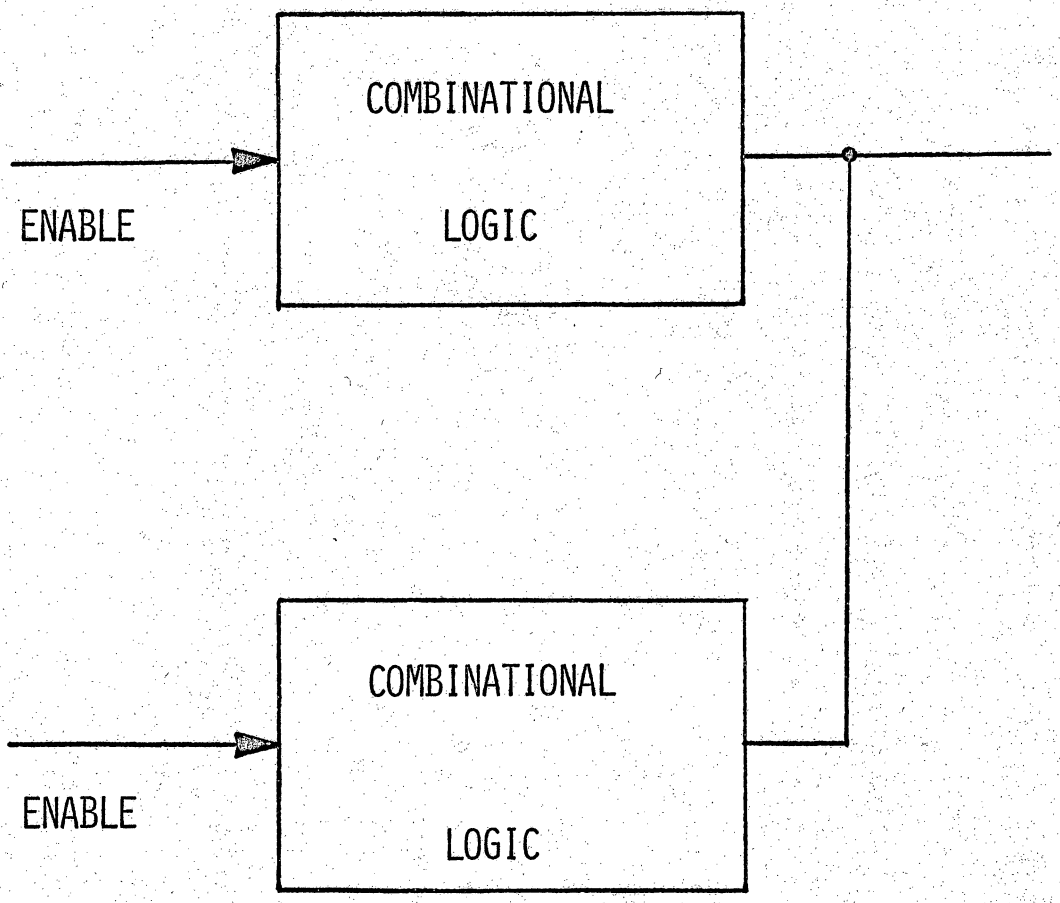


Figure 3b Removal of logic using tri-state gates

Consider a general network of tri-state gates. It is shown to be possible to devise such a network for some function (or functions) for which the logic needed to realize the general network is cheaper than the logic required to realize the function (or functions) by standard methods. This is the purpose of Chapter 2. In Chapter 5 a multi-functional cell will be designed which realizes a specific function at a specific time rather than all functions at all times as is now the practice in some cases, specifically in the ALU's of many computers. The network's configuration under some arbitrary application of gate disable signals would realize for example a flip-flop and under another combination the cell might be an adder or some other basic building block.

It is hoped that this discussion has served to show some of the motivation for the study of tri-state logic and to give the reader an intuitive feel for tri-state logic and its capabilities.

II. COMBINATIONAL DESIGN

The case of general combinational design using tri-state gates is now considered. A special design procedure which is useful only in specific cases is presented in Chapter 5 along with comments and suggestions which will help in determining its applicability.

Definition 1 A partition variable is an input variable of a combinational function which is used to separate the function into subfunctions each of which is realized by a separate network and enabled by the variable.

Definition 2 A scoring table is a tabular representation of the expected decrease in cost and complexity of a network which will result from the use of some variable of the function as the partition variable.

Conceptually, the partition variable separates the function f into two subfunctions each of which has half the Karnaugh map as a don't care, since it is a function of the opposite value of the partition variable. In the scoring table, there is a column for each variable of the function and a row for each prime implicant. Enter n in the $(i,j)^{th}$ position if the j^{th} prime implicant has p-n literals, one of which is i; p=# of literals in the function f. The variable with the largest column sum is the one

that the function depends on most, in a qualitative sense. Note that this procedure is not completely general as it only suggests a choice of a good variable to use; this is not necessarily the best choice. This procedure's informality can be justified in some sense by analogy to Armstrong's state assignment scoring table which yields good results but is by no means formal in that it too only suggests a good but not necessarily the best assignment. Illustration of the general combinational design procedure can best be accomplished by the following example.

EXAMPLE 1

Function f(A,B,C,D) is shown in Figures 4a,b as a minterm list and as the one-points of a Karnaugh map. The table in Figure 5 is the scoring table used to determine a partition variable for the function f. The purpose of this partition variable is to enable one or the other of the subfunctions f_1 or f_2. The subfunctions are mapped in Figure 6 for partition variable B.

The general design procedure is as follows. After generating the scoring table, determine the best choice for a partition variable by choosing the variable with the highest score. Map the two subfunctions defined by the variable. Note that on the subfunction maps, half the map is a don't care, specifically the half which is a function

$$f(A,B,C,D)=\sum m(0,1,4,7,8,9,10,11,13)$$

Figure 4a Minterm list for function f

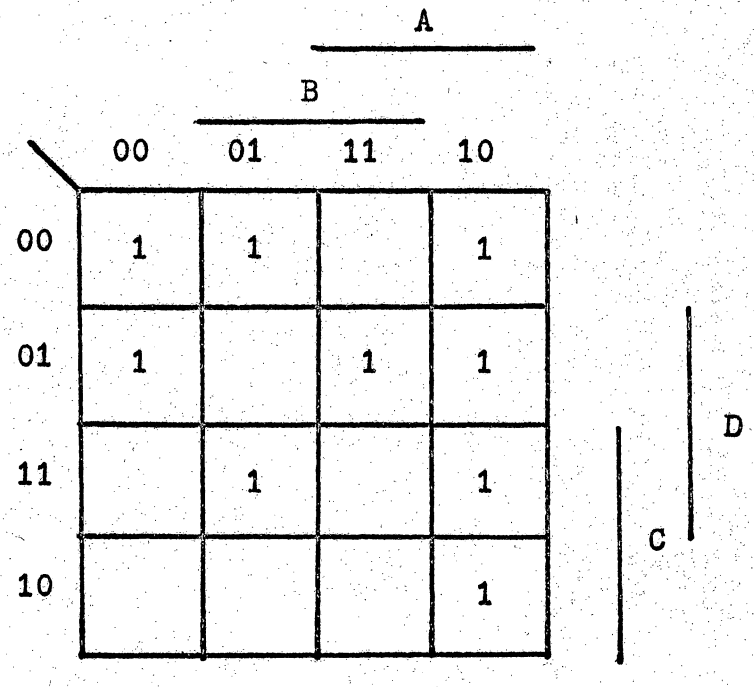


Figure 4b Map of function f

IMPLICANT	A	B	C	D
$A\bar{B}$	2	2		
$\bar{B}\bar{C}$		2	2	
$\bar{A}\bar{C}\bar{D}$	1		1	1
$A\bar{C}D$	1		1	1
$\bar{A}BCD$	0	0	0	0
	4	4	4	2

Figure 5 Scoring table for function f

CD \ AB	00	01	11	10
00	1	-	-	1
01	1	-	-	1
11		-	-	1
10		-	-	1

- Represents a don't care

Figure 6a Map of subfunction f_1 corresponding to partition variable B. Value of the variable is $\bar{B}$.

CD \ AB	00	01	11	10
00	-	1		-
01	-		1	-
11	-	1		-
10	-			-

- Represents a don't care

Figure 6b Map of subfunction f_2 corresponding to partition variable B. Value of the variable is B.

of the opposite value of the partition variable. This allows the one points of the subfunctions to be covered in larger than expected implicants. Thus this design procedure allows simplification by decreasing the size of the gates required to realize the implicants of the whole function. This method of realization, breaking the function up into two subfunctions, suggests a standard form for the resulting networks. This form is illustrated in Figure 7. In general, f_1 and f_2 are functions of all the variables of the function f except the partition variable x_p which is included by virtue of the fact that it enables the subfunction logic. Once the subfunctions have been mapped in the usual way, their individual logic is realized by standard procedures. Only the output gate in each logic block need be tri-state since these are the only ones which will have their outputs in common. Figure 8 shows the standard form NAND realization of the example function. The fractional chip cost is also shown.

Figure 9 shows realizations of the example function using the general tri-state procedure for each choice of a partition variable. Note that using variables A,B and C do indeed yield networks of about the same cost as suggested by the scoring table.

This procedure can be applied to any function which would be realized by classical methods and should yield a cheaper realization. Multiple output functions can be

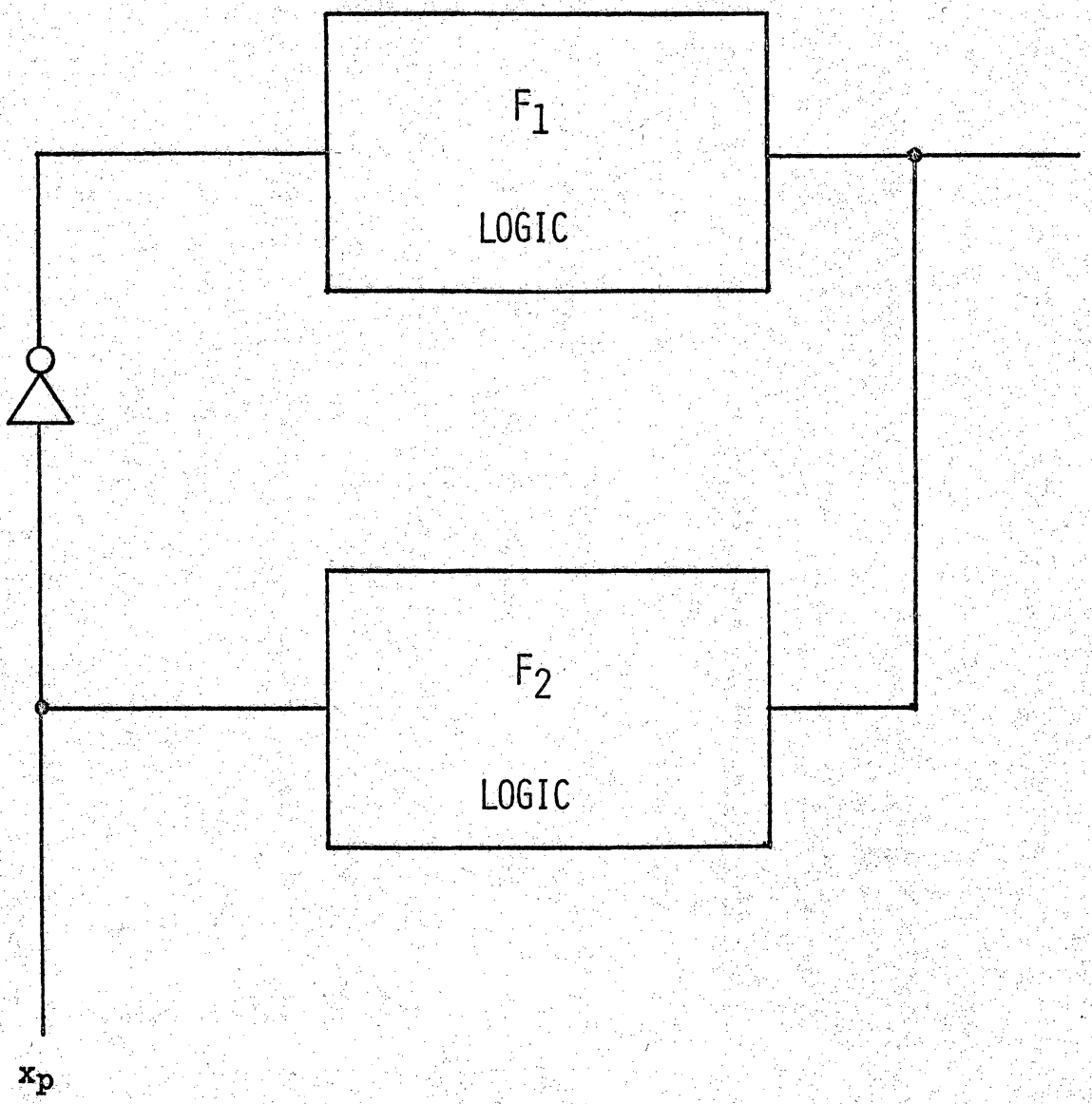
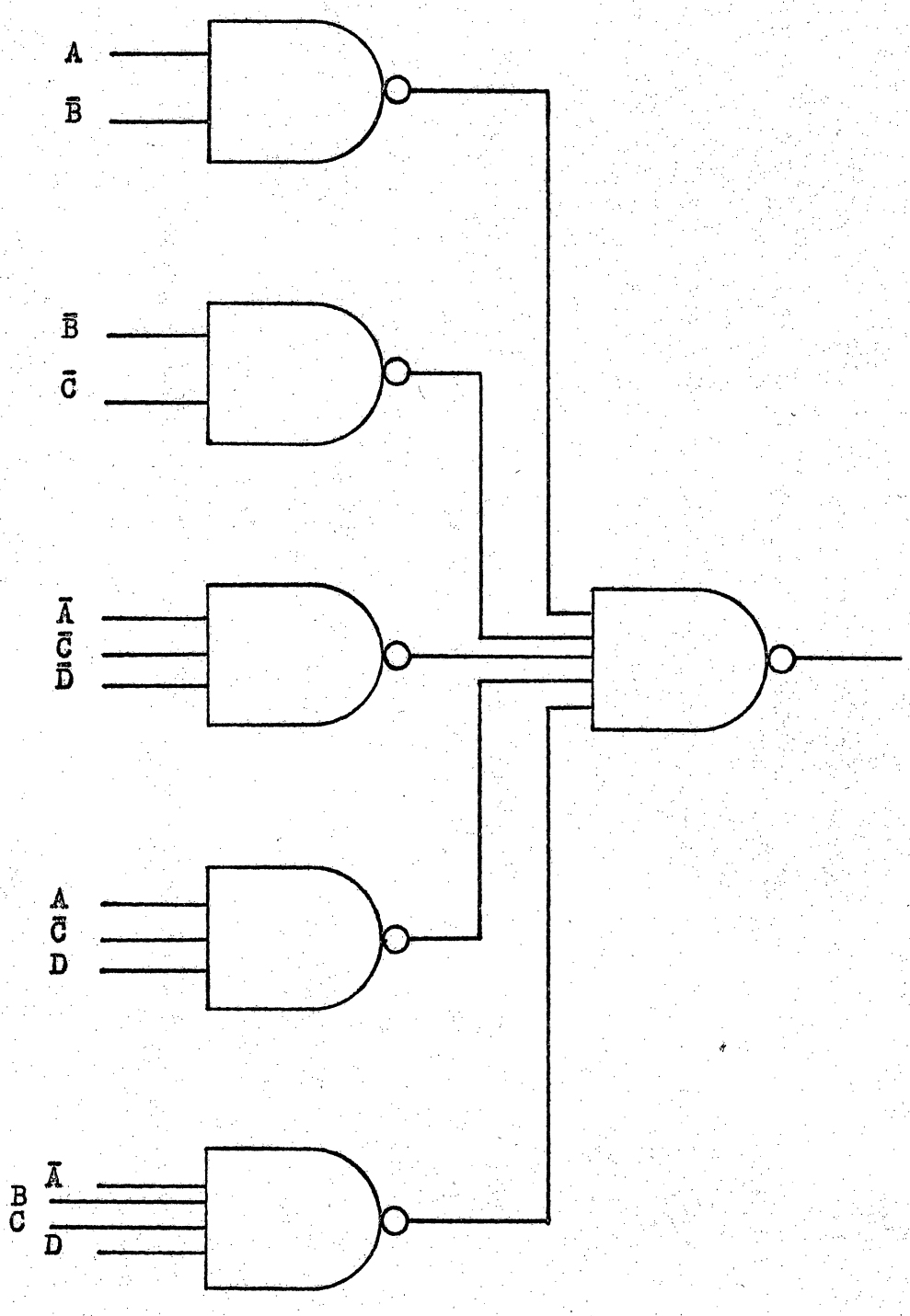
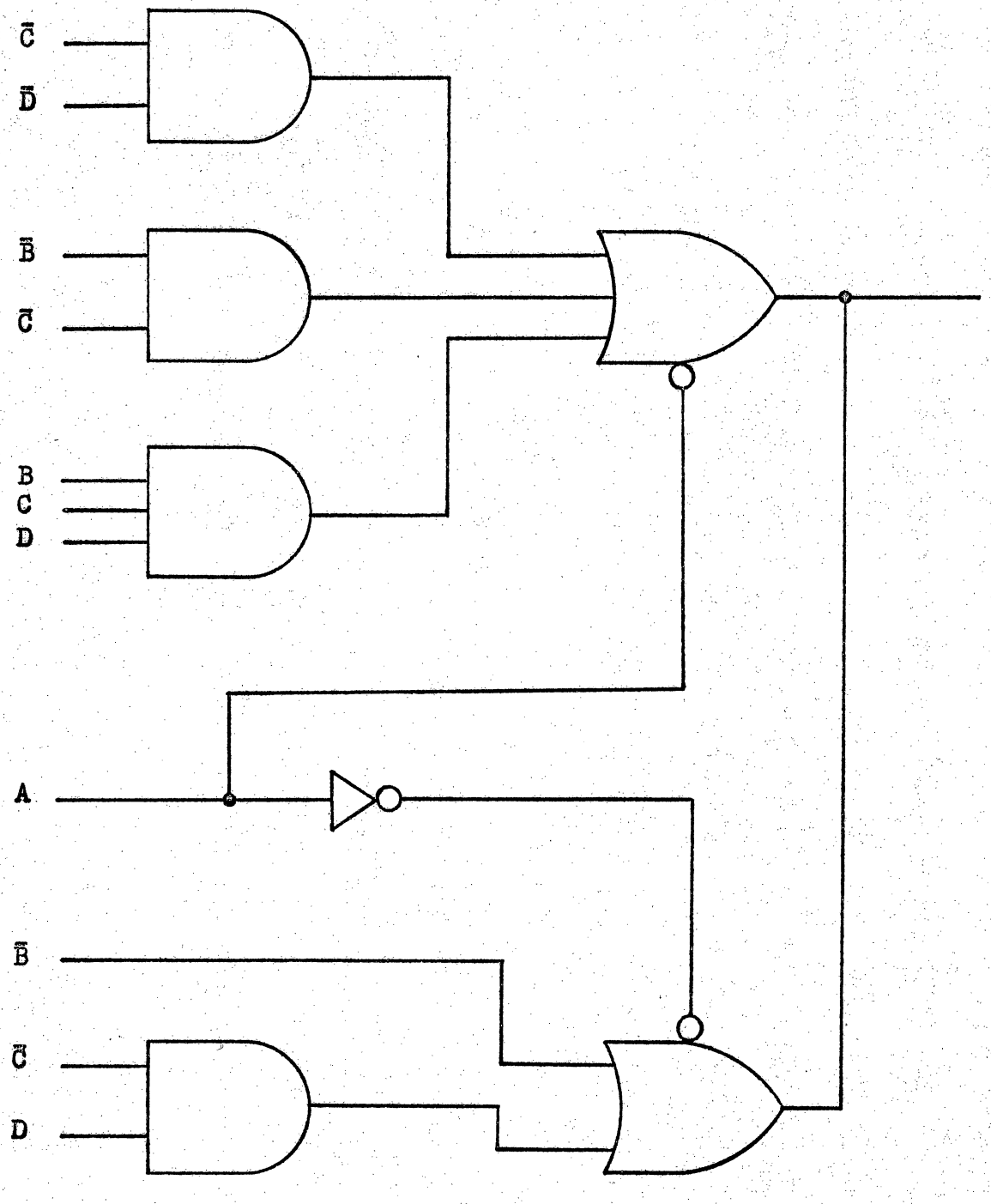


Figure 7 Standard form of a tri-state combinational network



Cost=2 2/3

Figure 8 Standard form realization of example function



Cost=2 1/12

Figure 9 Realization of example function using partition variable A.

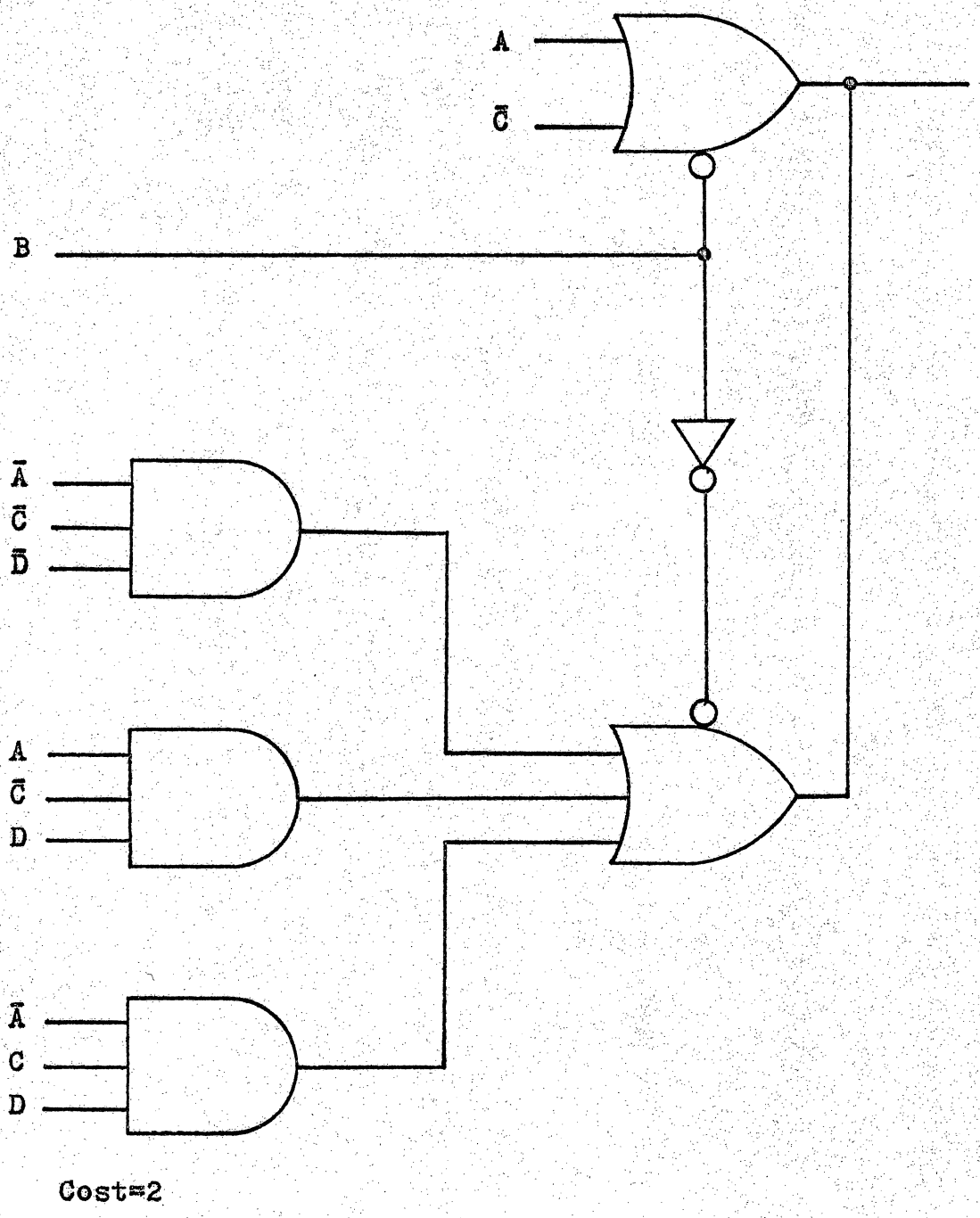
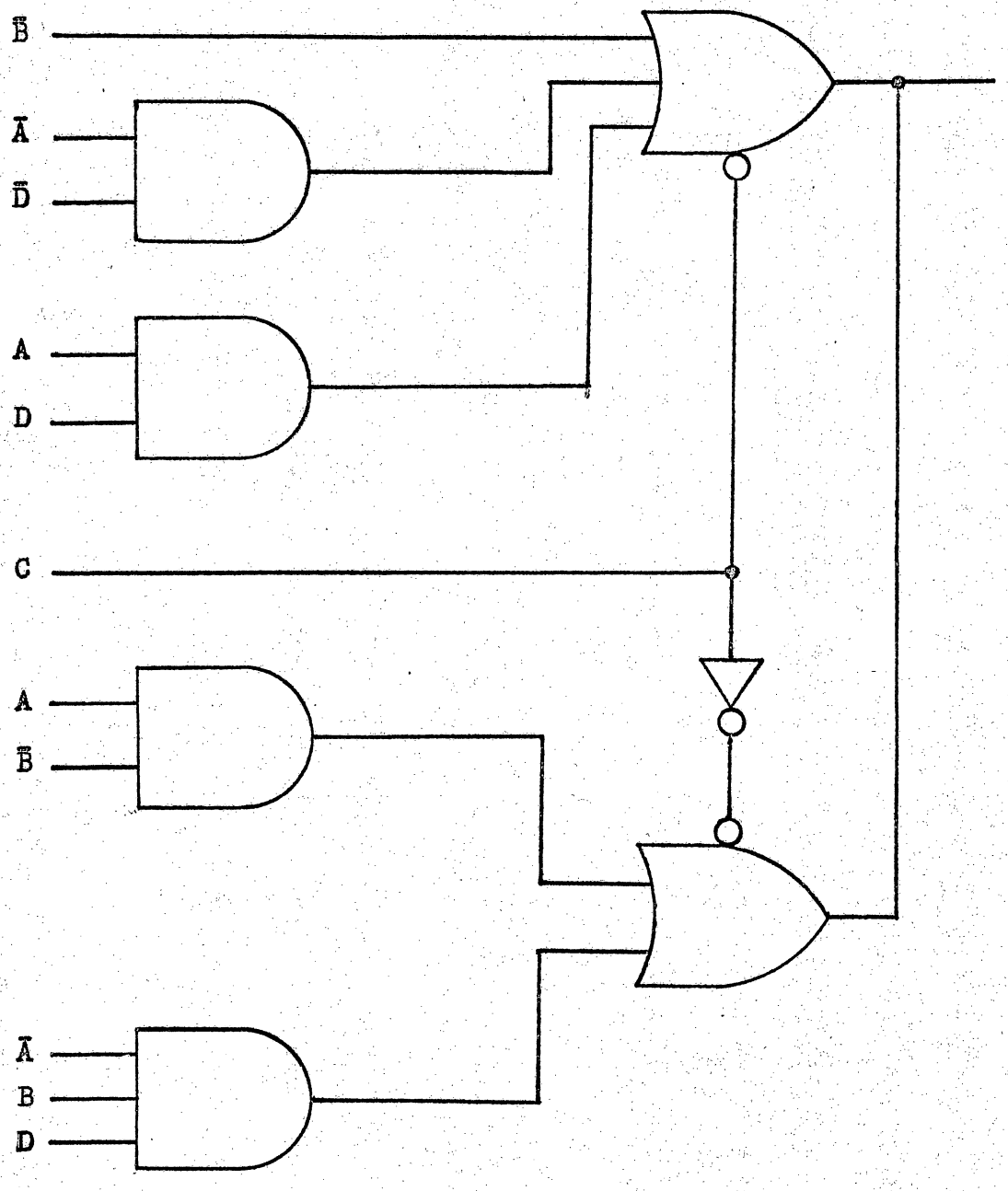
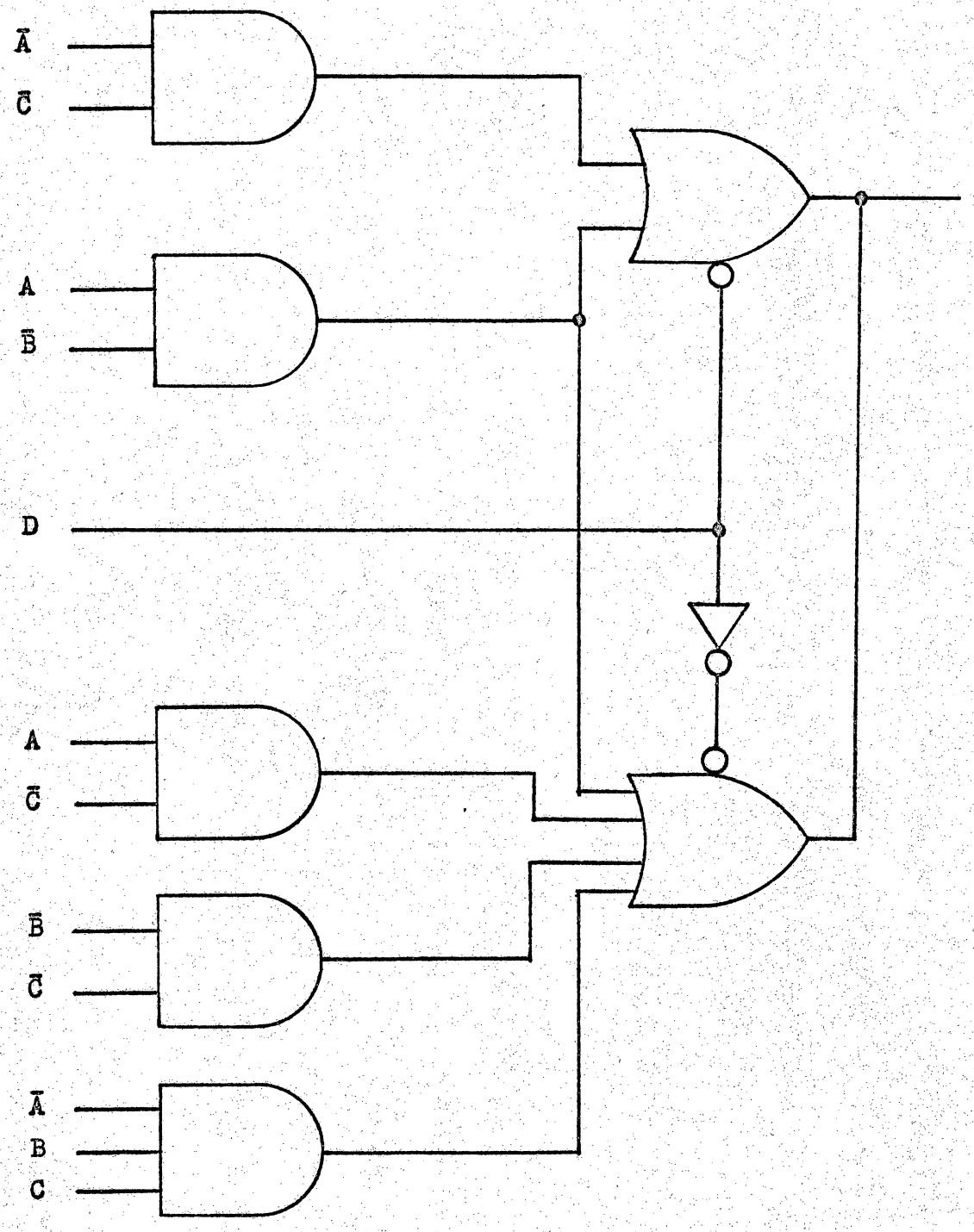


Figure 9 (continued) Realization of example function using partition variable B



Cost=2 1/12

Figure 9 (continued) Realization of example function using partition variable C



Cost=3 1/12

Figure 9 (continued) Realization of example function using partition variable D

accomodated by the same methods, except that a different partition variable may be indicated for each function.

III. SEQUENTIAL DESIGN

3.1 General Sequential Networks

As an introduction to the study of general sequential networks, consider Figure 10 which shows the first of three proposed canonic forms of tri-state logic sequential machines. Since the states themselves can define a specific function (single or multiple valued) which must be realized to generate the next state information, the state variables are a likely choice for use in enabling subfunctions. The next state logic could be thought of as consisting of several small, simple functions which are required singly and exclusively of the others, each of which generates the next state information for some current state. The counter - decoder combination would cycle through all such functions and supply the appropriate outputs which are needed to modify the S - C latches used to store the state of the machine.

It should be noted that in order to avoid transient states appearing at the state lines, it would be necessary to use a master - slave type organization and to gate the finished output to the slave section at the appropriate time.

Examination of realizations of this form reveals 1) that the individual subfunction logic required for each current state is a significant and excessively expensive

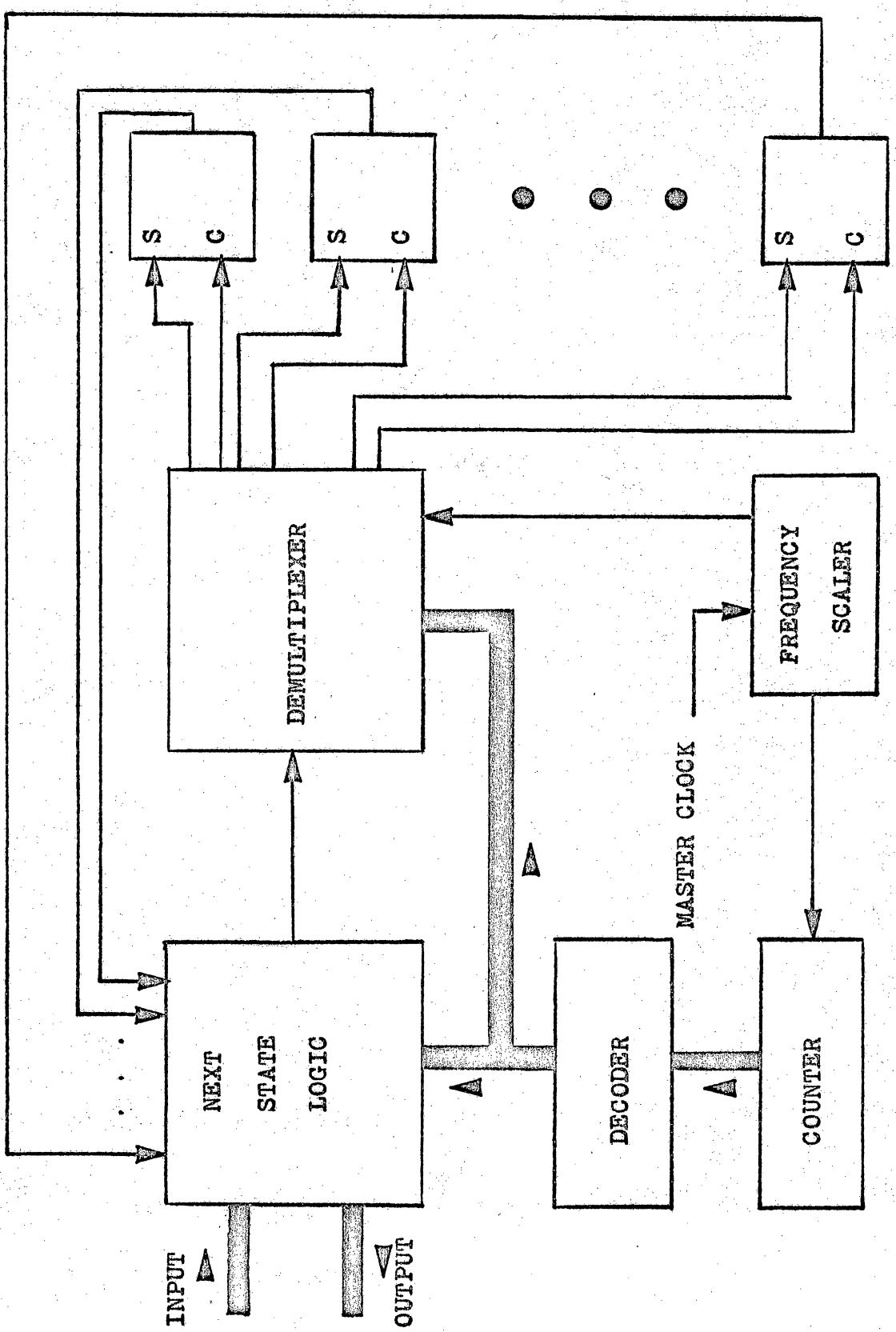


Figure 10 Proposed canonic form

portion of the final cost, 2) that the counter - decoder combination adds enough additional cost to the network to make it unjustifiable from the chip cost stand point, and 3) that there is extra circuitry necessary to adjust the system clock for machines of this form, again increasing cost. For example, consider two machines M_1 and M_2. M_1 has x states and M_2 has y. M_1 requires a clock which runs 2x times faster than the system clock, in order to cycle through all flip - flop inputs in the period of the master clock. M_2 by the same token needs a clock running at 2y times the system clock rate. Thus, the cost of this extra circuitry cannot be distributed in a system, since the scale factor is different in every case, and in terms of both chip cost and complexity this structure is discarded as a general canonic form although it may prove practical in certain special cases.

One characteristic of this circuit which makes it quite specialized is that the state transition is not synchronous as previously mentioned. As the machine steps through the S and C flip - flop inputs and changes the individual state code bits, there are transient states present on the flip - flop outputs. These states could of course affect the system's performance as there now exists the possibility of races. This flaw is corrected as shown in the second proposed canonic form presented below.

Figure 11 shows a structure which embodies several

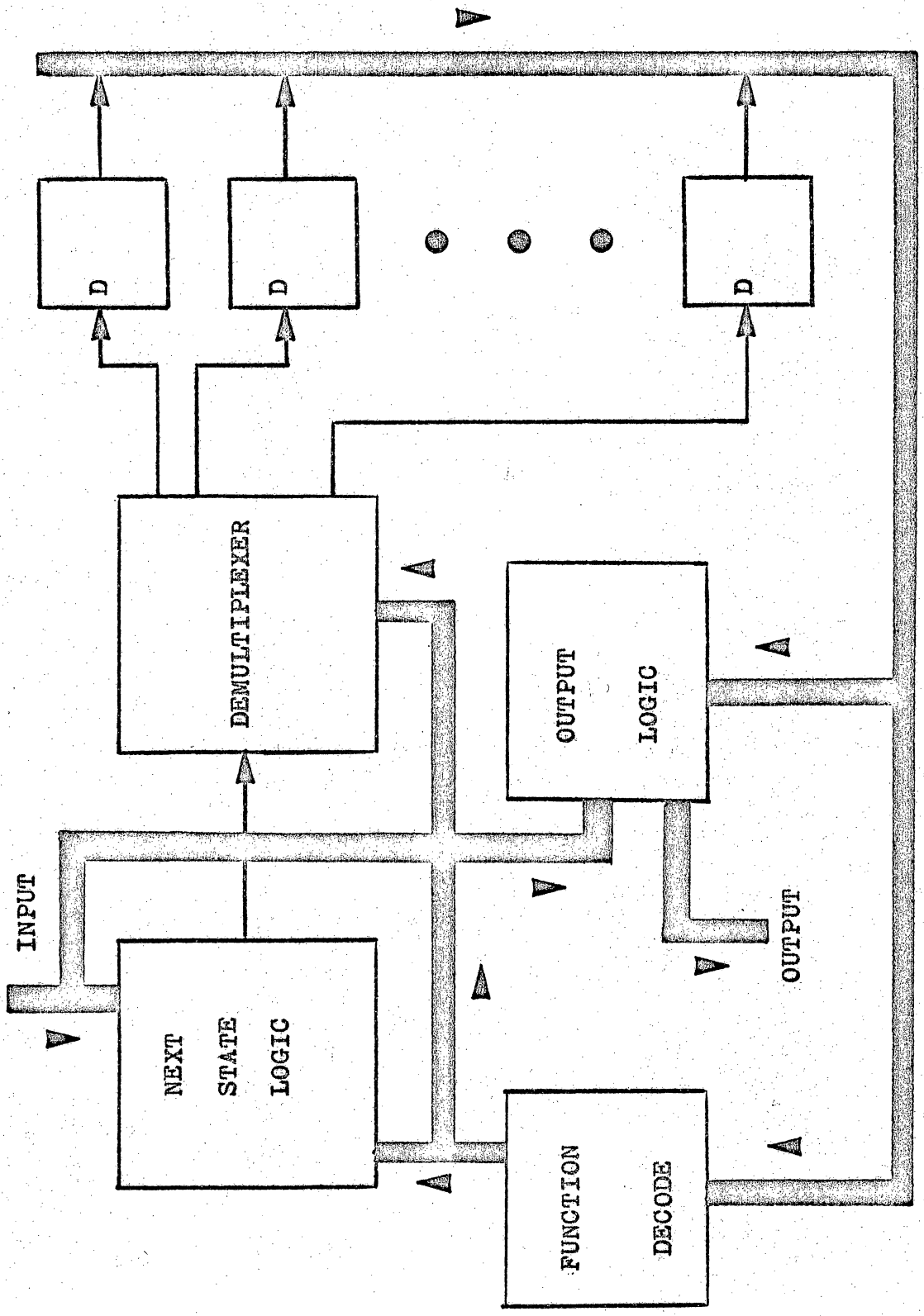


Figure 11 Proposed canonic form

modifications to the one discussed previously. Here, it was thought that a "one-hot" state assignment would eliminate the cost of frequency modifying the system clock, and also eliminate the possibility of races as mentioned above. This would at the same time reduce the cost by eliminating the counter - decoder block. In this case, the function logic serves to enable the demultiplexer to route the output of the next state logic to the appropriate lead of the appropriate flip - flop. It was thought, since each state in general only makes transitions to some subset of the total state set Q of the machine, that the function logic could be simplified to where this form would be cost competative.

This model was found faulty for three reasons: 1) the function logic is not, in general, as cheap as expected, 2) the cost of the extra flip - flops required for the one-hot state assignment could not be absorbed into the network and 3) the demultiplexer again adds cost without benefit, however, again it may prove useful in certain cases. These factors led to the proposal of the third and most useful of the forms investigated and one which will be discussed in some detail. It will be shown that the following form is indeed the most general of the three forms and that machines in this form yield cheaper realizations than machines realized in standard forms using conventional gates and flip - flops.

3.2 The Canonic Form

In the previous sections consideration has been given to proposed canonic forms which were found not generally useful in part due to the high cost of decoding which function is to be enabled in some logical network. For this reason the possibility of decoding some subset of, but not necessarily all the state variables for use in enabling the subfunctions was considered. It is found that this approach does indeed offer much simplification both over the previously proposed forms and also standard canonic form realizations. Figure 12 shows the final proposed canonic form for a tri-state sequential machine.

This structure is clearly a slight modification of the conventional form of a sequential machine. At this point, the general procedure for the design of a tri-state sequential machine consists of determining which state or input variables to decode, and then designing the combinational logic by the previously mentioned methods. The following example will serve to illustrate the sequential design procedure.

EXAMPLE 2

The flow table of machine M is shown in Figure 13. The first step in the sequential design procedure is to

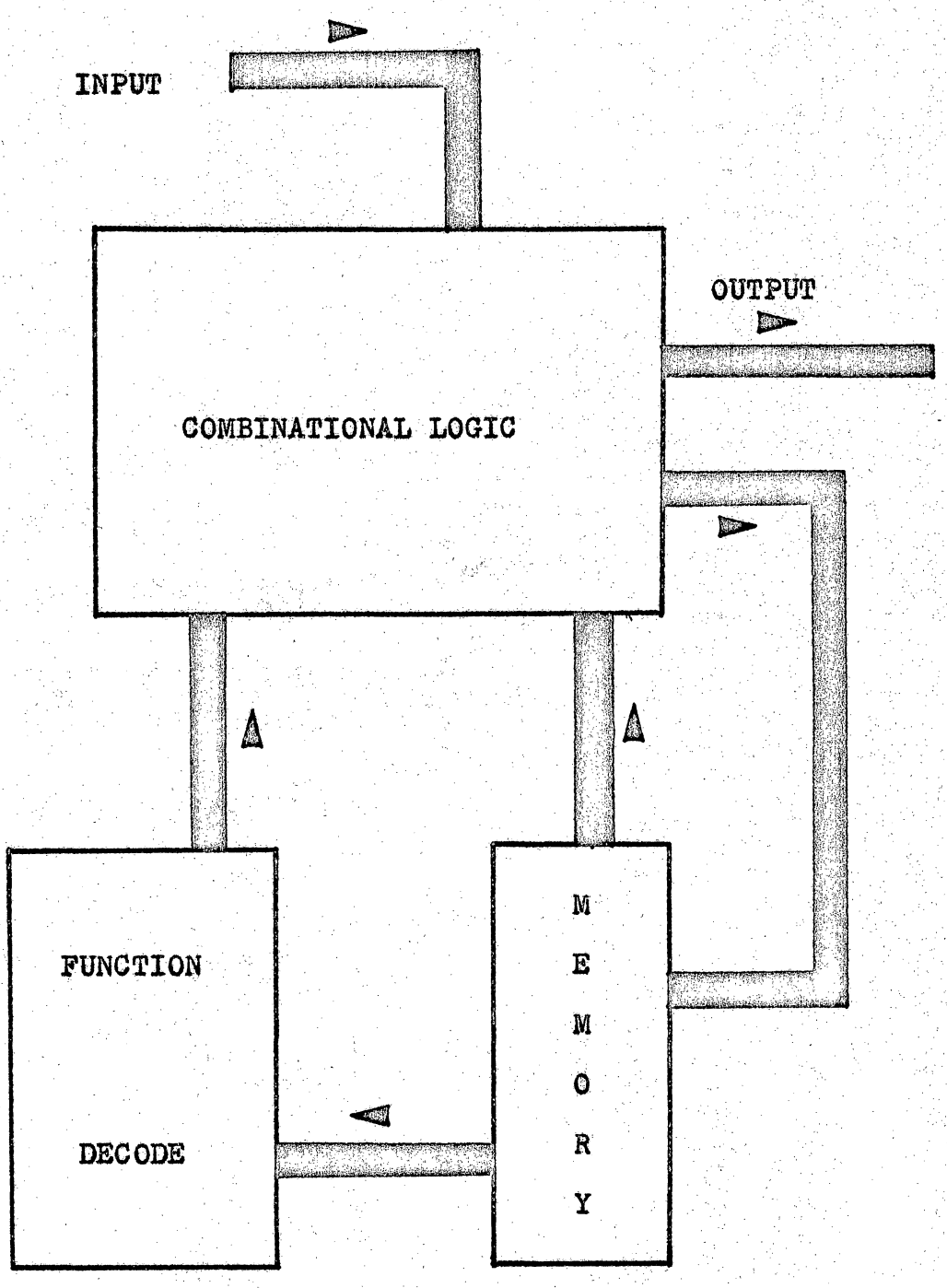


Figure 12 Canonic form for tri-state sequential machines

State \ Input	a	b
1	3,1	2,0
2	4,0	3,0
3	5,0	2,0
4	1,0	3,0
5	3,0	4,1

Figure 13 Flow table of machine M

generate the Mm lattices of State-State, State-Input, Output-State and Input-Output pairs. For this reason, the inputs are listed as lower case in the flow table in Figure 13 to signify that the exact variable assignment is to be determined. The State-State partitions are shown in Figure 14. The Input-State and Input-Output lattices are trivial; the State-Output lattice has one partition pair which is not of interest and is not included.

Having generated the lattices, the procedure becomes rather intuitive as is the case in classical design. [3] If a certain variable contains a large amount of information about M, then decoding it should yield simple functions as a result. Indeed, it will be shown that this is the case. Therefore, partitions suggested by classical techniques define a good choice for partition variables.

It has been found that in almost every case, the cheapest realizations arise from the feedback - decoding of a single state or input variable because there is none of the extra circuitry required for other decoding if only one variable is used to enable the logic. There has been no relationship observed between the cost of the realization and the number of variables used for feedback, when the number of feedback variables is not one.

Consider the State-State partitions shown in Figure 14. A good assignment for the state variables is shown in Figure 15; the assignment obtained by classical methods.

(125,3,4) (1,234,5)
(13,2,4,5) (1,2,35,4)
(14,2,3,5) (123,4,5)
(15,2,3,4) (1,24,3,5)
(1,23,4,5) (1,23,45)
(1,24,3,5) (14,2,3,5)
(1,25,3,4) (1,2,34,5)
(1,2,34,5) (15,23,4)
(135,2,4) (1,24,35)
(1,245,3) (134,2,5)
(1235,4) (1,2345)
(1245,3) (1234,5)
(125,34) (15,234)
(134,2,5) (1235,4)
(13,24,5) (14,2,35)
(13,25,4) (1,2,345)
(1,245,3) (1345,2)
(14,23,5) (123,45)
(15,24,3) (124,3,5)
(1,234,5) (145,23)
(135,24) (124,35)

Figure 14 State-State partitions for machine M

Y_1 - (125,34) (15,234)

Y_2 - (135,24) (124,35)

Y_3 - (14,23,5) (123,45)

Figure 15a Chosen partitions for state assignment

	Y_1	Y_2	Y_3
1	0	0	0
2	1	0	0
3	1	1	0
4	1	0	1
5	0	1	1

Figure 15b State assignment for machine M

In this example, no partition is suggested more strongly than the others and so no partition is indicated as a good choice for defining the partition variable. Clearly, this does not mean that no variable will define simple functions. Indeed, the scoring table of Figure 16 shows a very definative suggestion, the input variable x. Since the combinational logic is a function of the input as well as the state variables and cannot distinguish between them in any real sense, the extension to using the scoring table, which may indicate using an input variable and not a state variable is straight forward.

The procedure as illustrated can be used for Mealy or Moore circuits. Also, for this example, the Input-State lattice was trivial and as such did not suggest any partition. Clearly, this will not be the case in general and input variables may be suggested as partition variables. In any event generation of the scoring table should not be discarded for sequential designs just because it is an informal tool where the Mm lattice is rigorous and formal.

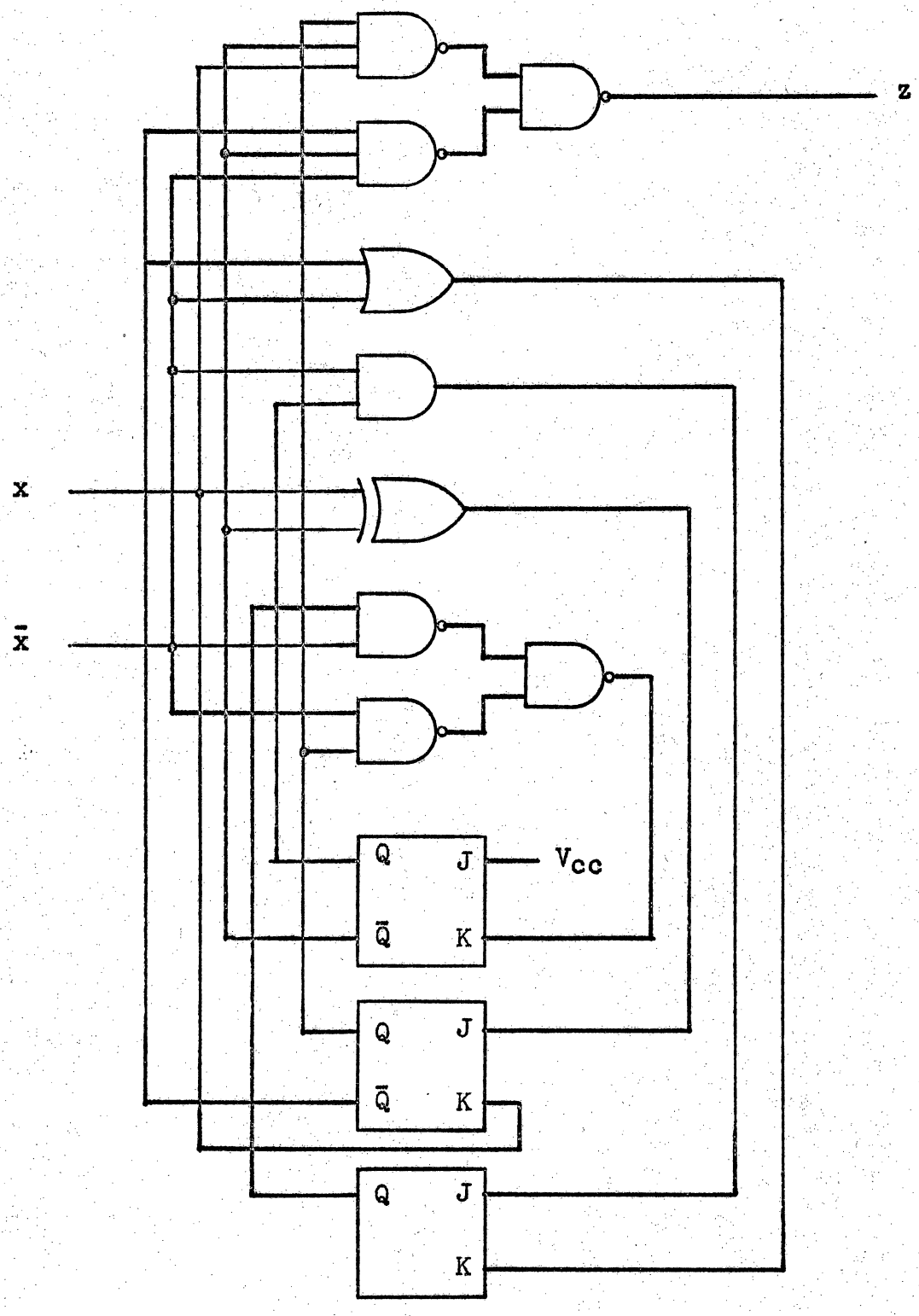
Figure 17 shows the canonic J-K and tri-state D type realizations of the example machine. It should be noted that this comparison is between the cheapest classical and the cheapest tri-state realization.

3.3 Comments

Recall section 1.3 where special considerations of

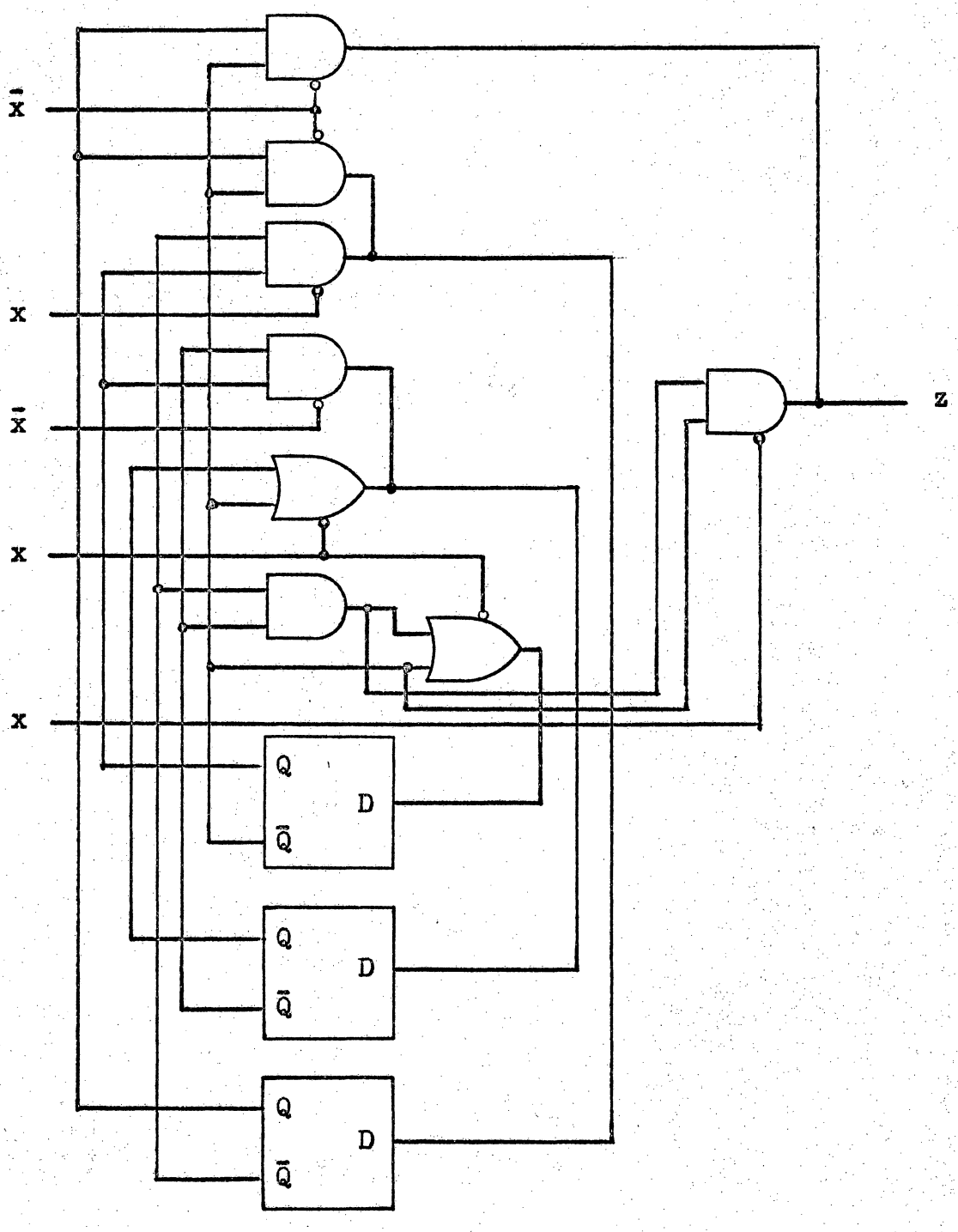
IMPLICANT	y_1	y_2	y_3	x
$\bar{y}_1$	3			
x				3
$\bar{x}\bar{y}_2\bar{y}_3$		1	1	1
$\bar{x}\bar{y}_1$	2			2
$\bar{x}y_2$		2		2
xy_1y_3	1		1	1
$xy_1\bar{y}_2$	1	1		1
$\bar{x}y_1y_2$	1	1		1
$\bar{x}y_1\bar{y}_3$	1		1	1
$x\bar{y}_1y_2$	1	1		1
	10	6	3	13

Figure 16 Scoring table for machine M



Cost=3 11/12

Figure 17 Conventional logic realization of machine M



Cost=3 1/4

Figure 17 (continued) Tri-state realization of machine M

tri-state logic implementations were considered. For this example, there are more constant ONEs to be realized in the f circuit. As a result, fewer buffers are required to realize constant ZEROs and the network is simplified. Had it been the case that in the f logic there were more constant ZEROs to be realized, a complement realization would have been indicated. When converted to $\bar{f}$ form, the f equations would have more ONEs and would be a simpler network. This is something the designer must consider when deciding on an implementation. In general, it has been found to be the case that the f to $\bar{f}$ transformation will be required in some but not all designs. It should be noted that all the functions do not have to be the same assertion; that is, in the example given here, it might have been desirable to invert the D_1 excitation function but not the functions driving D_2 or D_3.

A major criticism of the sequential design procedure is the necessity of generating the lattices of Mm pairs for the machine. The Mm lattice is both tedious and hard to construct and as a result is a drawback to the generality and applicability of the procedure. The generation algorithm is, however, well suited to computer implementation. Indeed, most of the examples worked in preparation of this thesis were run on a program written by Robert Hallissy of this department. Thus, the time required to generate the Mm lattices can be drastically reduced to the point

where the procedure is not so complex as to be prohibitive and the design procedure presented here is applicable.

Finally, the reader should be aware that the example presented here was meant to illustrate the design method being suggested and to give an idea of what kind of savings are possible using tri-state gates and methods. It should not be inferred that all networks will be simplified if the algorithms are employed. Indeed, in some cases there will be no simplification of the realization possible over the classical J-K design. The point is that simplification is possible and the methods described here will often realize whatever reduction is possible. It is up to the designer to use his experience to guide him in the application of tri-state methods to logic design in the general case.

IV. ITERATIVE NETWORKS

There has been considerable work done recently in the area of iterative networks. They offer simplification of the logic realization and also lend themselves to applications which are not easily accomplished by standard two-level logic design. Consider the parallel adder. The iterative implementation avoids the problem of designing a 2n input n+1 output logic network. Also there is the benefit of operation on words of arbitrary lengths, merely by the addition of deletion of the proper number of cells. In addition, the computational capability of iterative networks is demonstrated by Moore's firing squad problem. The following example will serve to illustrate the application of tri-state design procedures to iterative networks.

EXAMPLE 3

As an example of the tri-state design procedure in iterative networks consider the following. It is desired to build an n-bit magnitude comparitor which, when given two n-bit words, will determine the magnitude relationship between them. Clearly, the requirement of an n-bit circuit indicates that an iterative approach is in order. A general cell is shown in Figure 18 with maps of G_i and L_i.

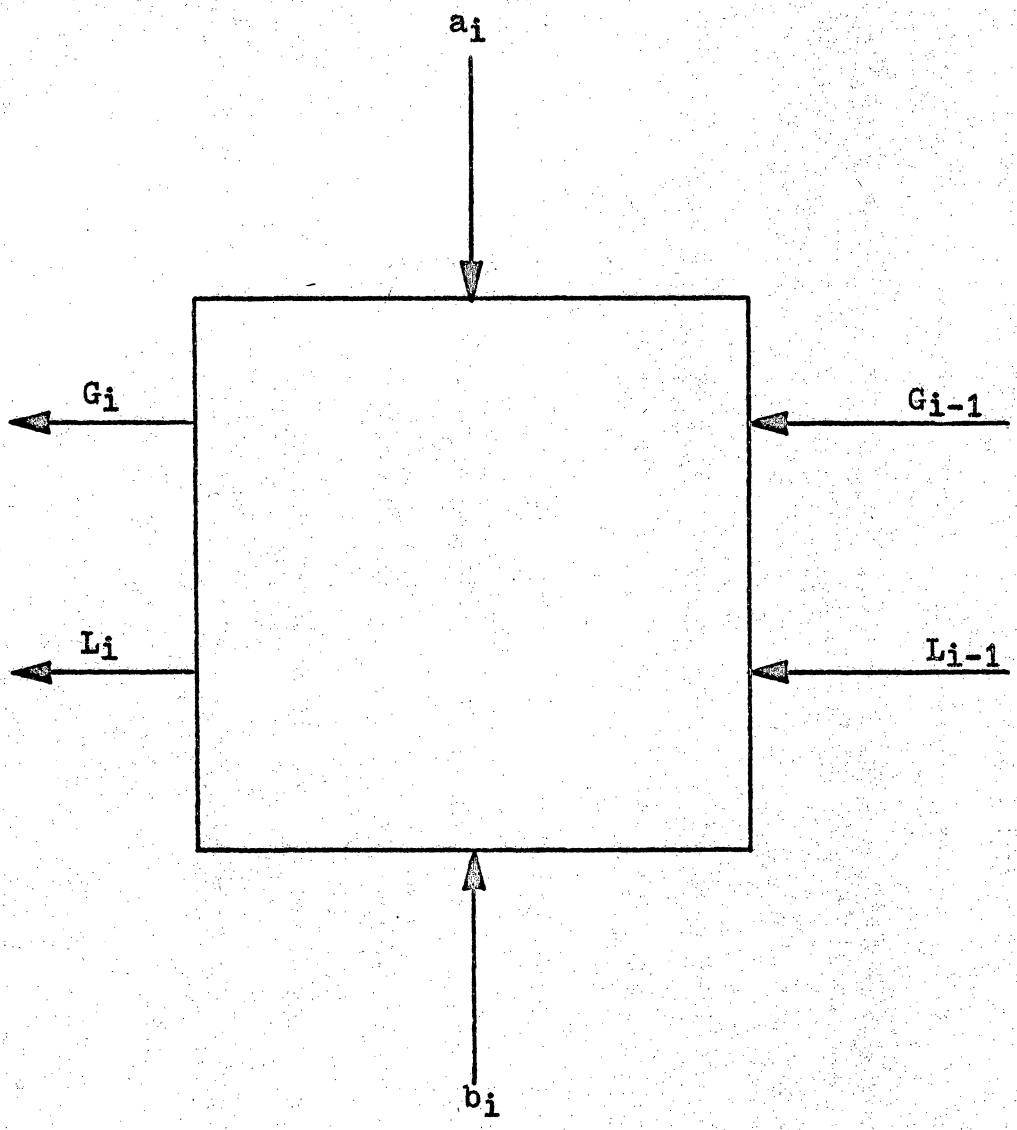
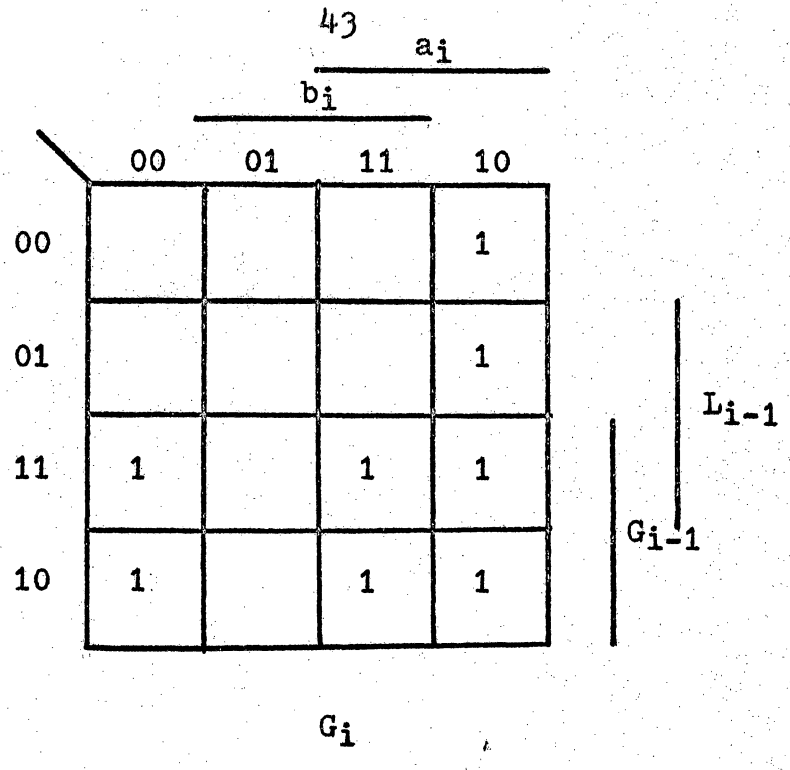


Figure 18 Typical cell in magnitude comparitor for example 3



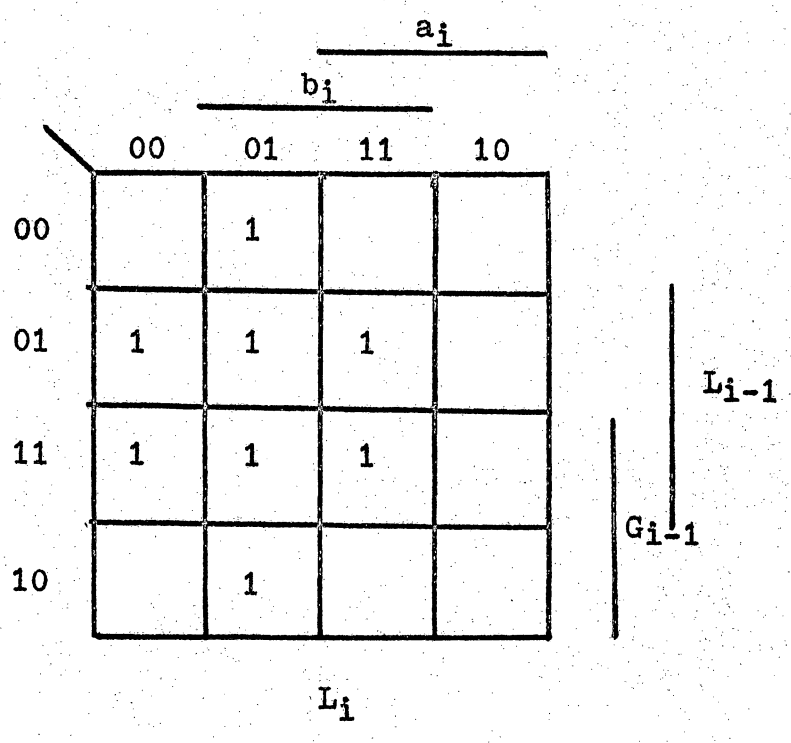


Figure 18 (continued) Karnaugh maps of G_i and L_i for example 3

Scoring tables are shown in Figure 19. In Figure 19 variable L_{i-1} has the same score as a_i and b_i for function L_i. It should be noted, however, that function L_i is symmetric with respect to variable L_{i-1} and thus the simplification which should occur by placing the one-points of the function in larger implicants than usual does not occur. L_{i-1} therefore should not be considered for a partition variable. A_i and b_i score the same; arbitrarily choose a_i. For function G_i choose a_i also. The logic is shown in Figure 20. Clearly, the general case of an iterative network can be extended to sequential iterative networks by applying the general design procedure from Chapter 3.

IMPLICANT	a_i	b_i	G_{i-1}	L_{i-1}
$a_i\bar{b}_i$	2	2		
a_iG_{i-1}	2		2	
$\bar{b}_iG_{i-1}$		2	2	
	4	4	4	0

G_i

IMPLICANT	a_i	b_i	G_{i-1}	L_{i-1}
$\bar{a}_ib_i$	2	2		
$\bar{a}_iL_{i-1}$	2			2
b_iL_{i-1}		2		2
	4	4	0	4

L_i

Figure 19 Scoring tables for G_i and L_i from Example 3

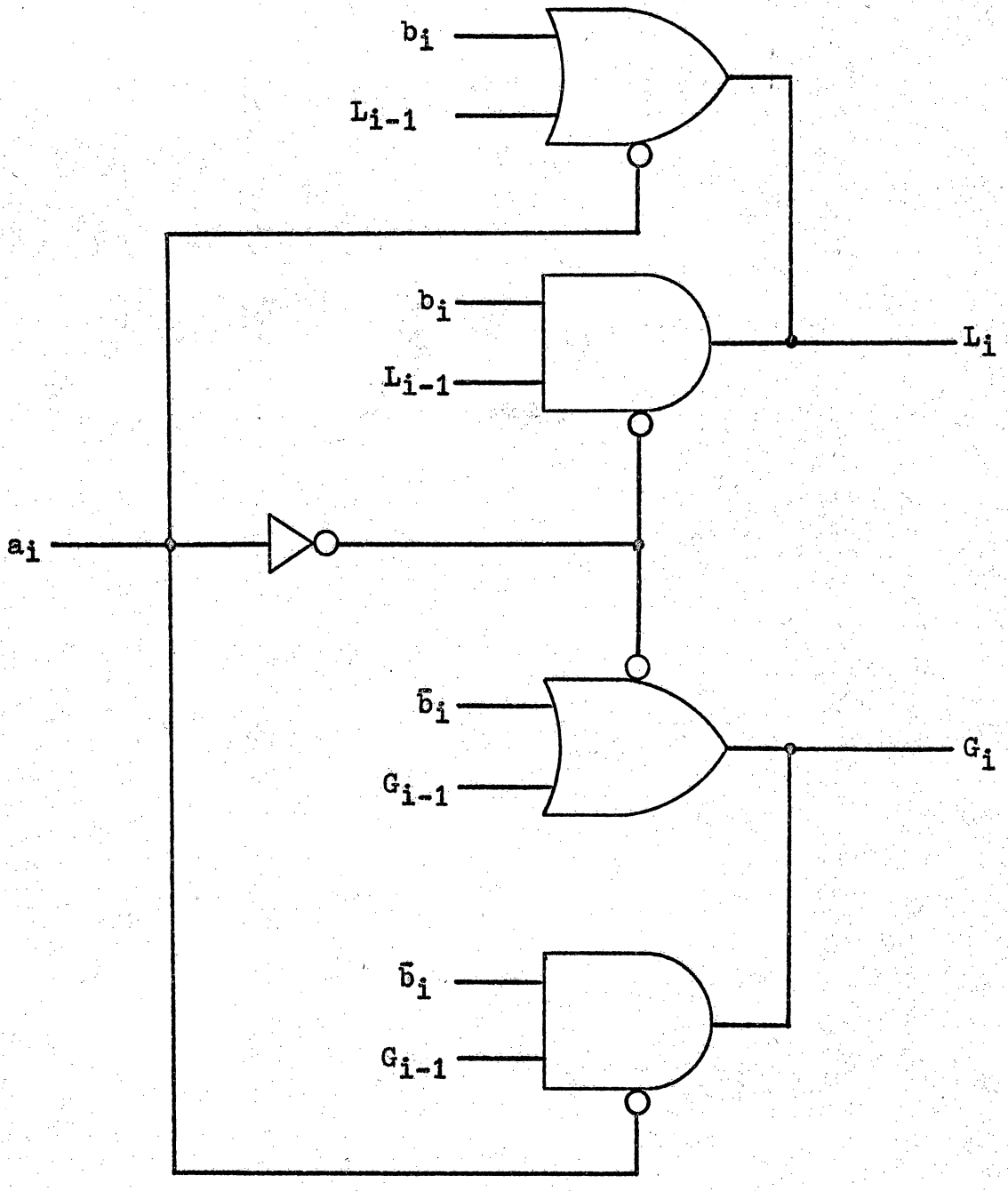


Figure 20 Tri-state realization of iterative network example.

V. MULTI-FUNCTIONAL CELLS

The case of multi-functional cell design will now be considered. However, before an example is presented, some general properties and considerations of multi-functional cells will be discussed.

The general property of logic networks which affords the reduction in cost and complexity of realizations by the algorithms previously discussed is that the partition variable or state variables separate the large network into several simpler networks whose total cost is less than the large network. In general, the simpler a function is, the cheaper is the resulting realization. Thus, a function which is of itself made up of easily discernable simple functions lends itself to the tri-state design procedure; a slight modification to the general procedure is all that is necessary. It should not be necessary to separate the function into subfunctions by the choice of some input or state variable as a partition variable if the function is composed of separate functions already. This observation is what marks the difference between the general combinational design procedure in Chapter 2 and the specialized procedure which is illustrated here.

Determination of the applicability of one design procedure over another can best be determined when the design goals are specified in the problem's statement.

The reason for this is that the word description of the problem will by nature specify the different functions and the decision of which algorithm to use can be made based on the amount of separation evident. Keeping the distinction in mind between the general and specific algorithms, the following example will illustrate multi-functional cell design by the use of the specialized procedure.

EXAMPLE 4

It is desired to design the ALU of a hypothetical computer using multi-functional cells. The general configuration of each cell is shown in Figure 21a. Figure 21b lists the functions which the ALU is required to realize.

After some consideration it becomes abvious that each of the functions to be realized is a very simple function of the function specification bits and perhaps one or two of the input variables. Clearly, this design should be implemented by the specialized procedure. The gate realization is shown in Figure 22. The design in this case amounted for the most part to translating the word description of the functions into gates directly. Specifically, consider functions 0,1,4,5,6 and 7. In order to generate the S_i output all that is necessary is to provide one gate per function, each of which is enabled at the appropriate

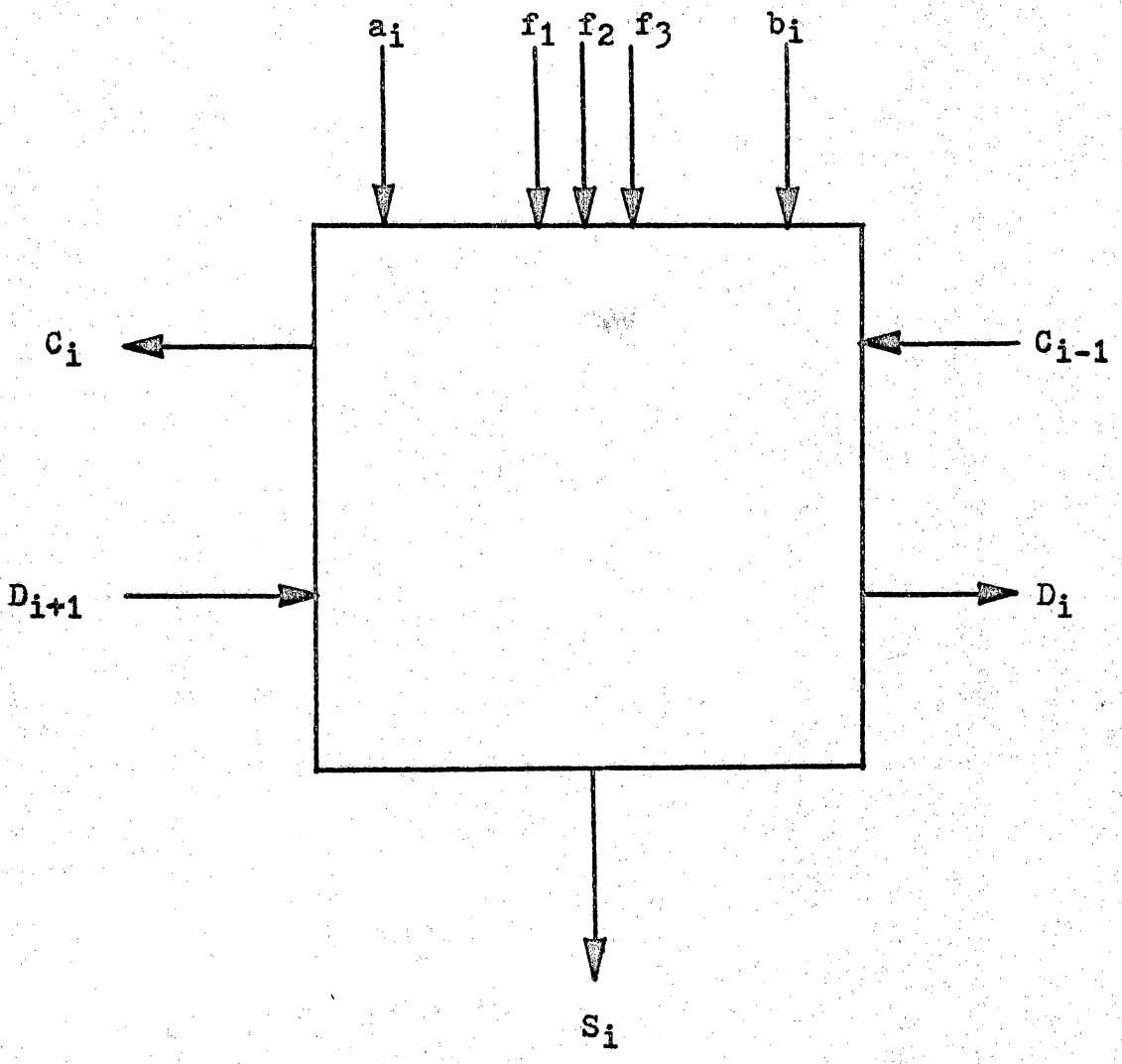


Figure 21a Configuration of general cell for example 4

FUNCTION SPECIFICATION BITS	FUNCTION
0 0 0	Shift a_i left
0 0 1	Shift a_i right
0 1 0	Sum
0 1 1	Difference $(a_i - b_i)$
1 0 0	Logic AND of a_i and b_i
1 0 1	Logic OR of a_i and b_i
1 1 0	Shift b_i left
1 1 1	Shift b_i right

Figure 21b List of functions to be realized by ALU of example 4

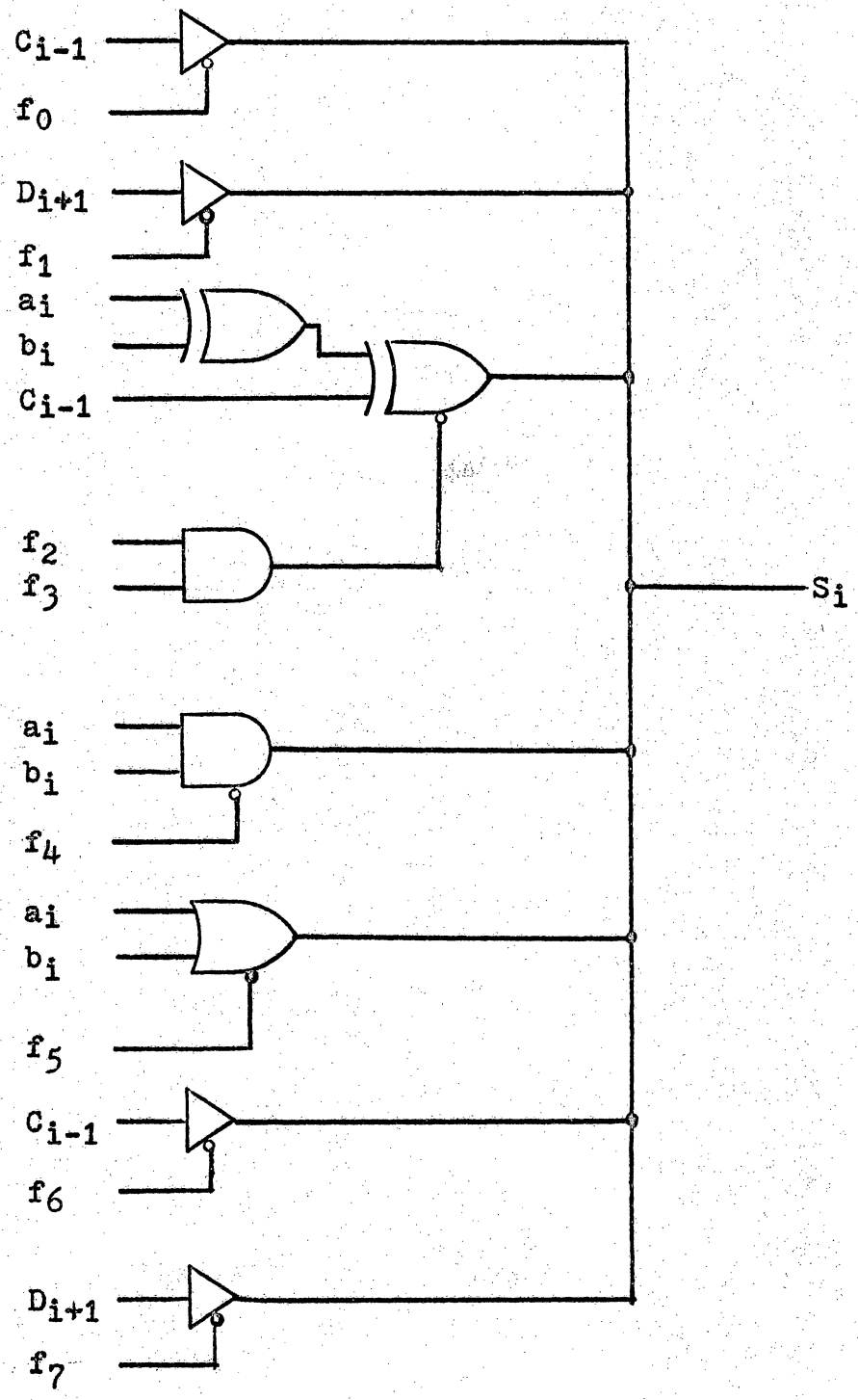


Figure 22 Gate realization of example function

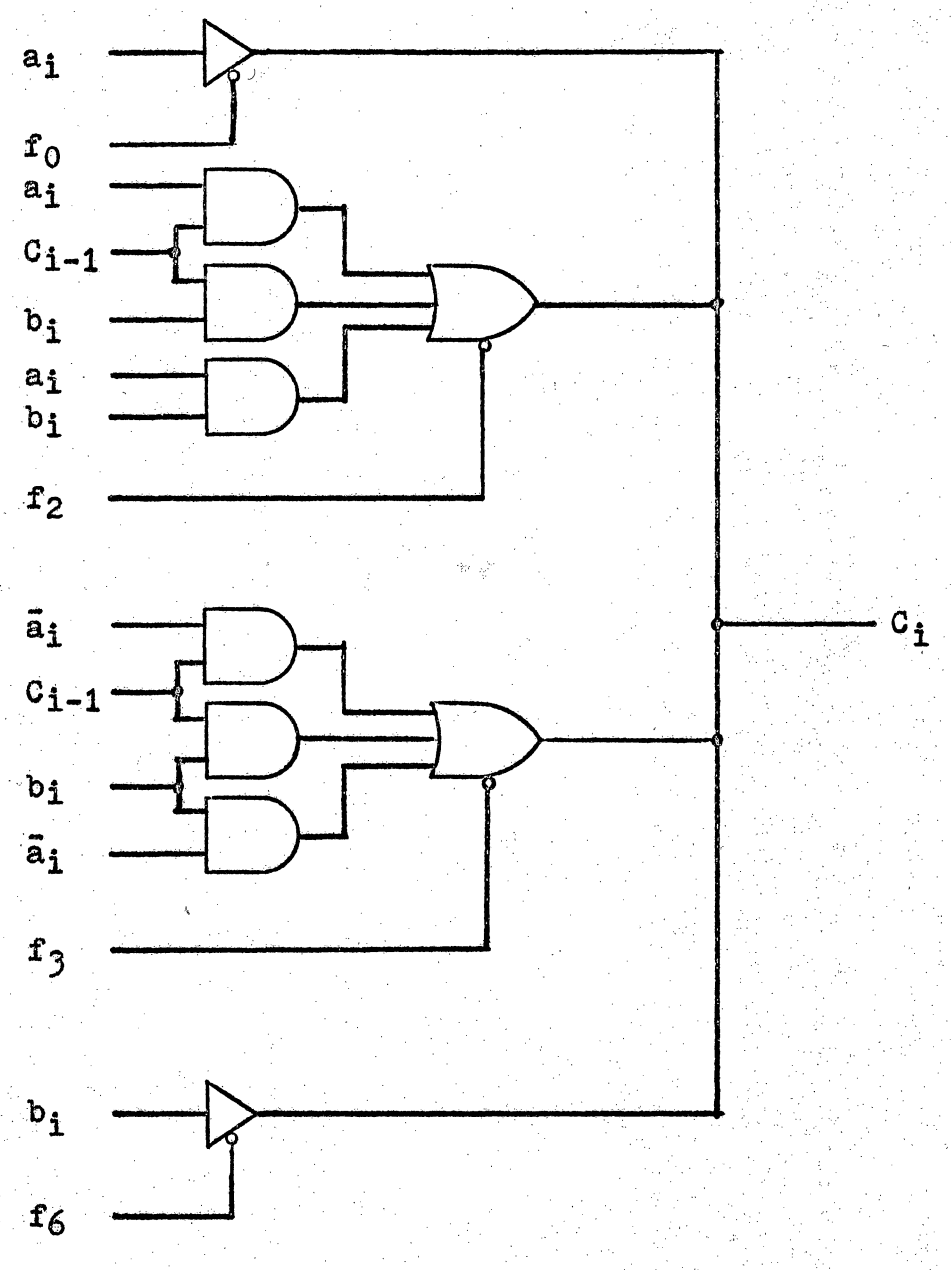


Figure 22 (continued) Gate realization of example function

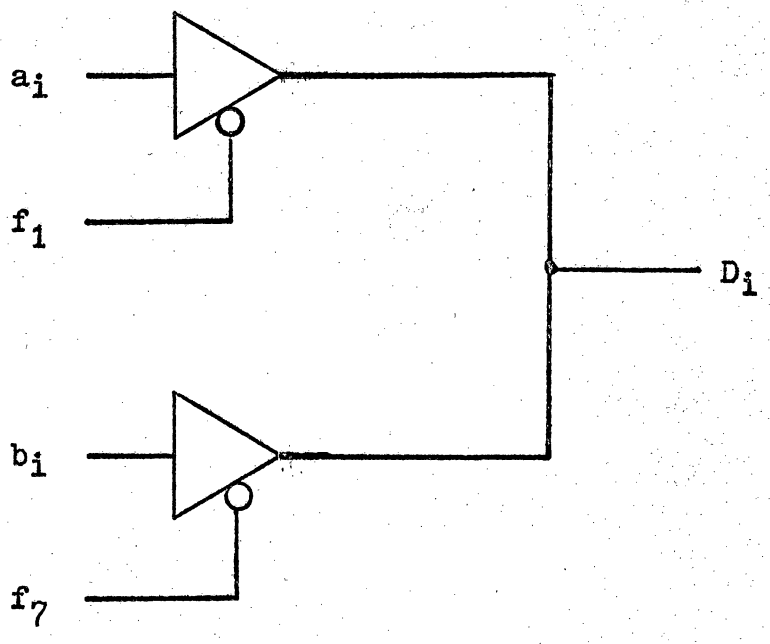


Figure 22 (continued) Gate realization of example function

time by decoding the function specification bits. Indeed, study of Figure 22 reveals that functions 2 and 3 are the only ones which require more than one gate to realize the appropriate truth value at any of the three cell outputs, D_i, C_i or S_i.

A one of eight decoder will serve to enable the logic corresponding to the function realized. Figure 23 shows the design equations and cost for the same network by the standard Karnaugh map combinational design. The Karnaugh maps used to determine the implicants for this realization are not shown due to their complexity. Each map is a function of seven input variables; three of which are the function specification bits. It might seem that a more efficient approach to this problem even by standard methods would be to somehow decode the function bits and thereby simplify the logic, however, standard production ALU's currently available (e.g. 74181) being LSI designs, consider the function bits as inputs to the combinational network. This convention was followed here. [4] Figure 24 lists the scoring tables and design equations for the generalized design procedure with one partition variable and the resulting cost.

Note that the general procedure is more expensive than the specialized procedure although it too is cheaper than the standard Karnaugh map approach. Thus, the general

$$S_i = \bar{f}_1 f_2 C_{i-1} a_i b_i + \bar{f}_1 f_2 C_{i-1} \bar{a}_i \bar{b}_i + \bar{f}_1 f_2 \bar{C}_{i-1} a_i \bar{b}_i + \bar{f}_1 f_2 \bar{C}_{i-1} \bar{a}_i b_i + \bar{f}_1 \bar{f}_2 f_3 D_{i+1} + f_1 f_2 f_3 D_{i+1} + \bar{f}_1 \bar{f}_2 \bar{f}_3 C_{i-1} + f_1 \bar{f}_2 f_3 a_i + f_1 f_2 \bar{f}_3 C_{i-1} + f_1 \bar{f}_2 a_i b_i + f_1 \bar{f}_2 f_3 b_i$$

$$C_i = a_i \bar{f}_2 + f_1 f_2 b_i + f_2 C_{i-1} b_i + \bar{a}_i f_2 f_3 C_{i-1} + \bar{a}_i b_i f_3 + a_i \bar{f}_1 f_2 \bar{f}_3 C_{i-1}$$

$$D_i = a_i \bar{f}_2 + f_1 b_i$$

Cost=11 5/6

Figure 23 Design equations and cost of general algorithm method

IMPLICANT	f_1	f_2	f_3	a_i	b_i	C_{i-1}	D_{i+1}
$\bar{f}_1 f_2 C_{i-1} a_i b_i$	2	2		2	2	2	
$\bar{f}_1 f_2 C_{i-1} \bar{a}_i \bar{b}_i$	2	2		2	2	2	
$\bar{f}_1 f_2 \bar{C}_{i-1} a_i \bar{b}_i$	2	2		2	2	2	
$\bar{f}_1 f_2 \bar{C}_{i-1} \bar{a}_i b_i$	2	2		2	2	2	
$\bar{f}_1 \bar{f}_2 f_3 D_{i+1}$	3	3	3				3
$f_1 f_2 f_3 D_{i+1}$	3	3	3				3
$\bar{f}_1 \bar{f}_2 \bar{f}_3 C_{i-1}$	3	3	3			3	
$f_1 \bar{f}_2 f_3 a_i$	3	3	3	3			
$f_1 f_2 \bar{f}_3 C_{i-1}$	3	3	3			3	
$f_1 \bar{f}_2 a_i b_i$	3	3		3	3		
$f_1 \bar{f}_2 f_3 b_i$	3	3	3		3		
	29	29	18	14	14	14	6

Figure 24 Scoring table for cell output S_i

IMPLICANT	f_1	f_2	f_3	a_i	b_i	C_{i-1}	D_{i+1}
$\bar{f}_2 a_i$		5		5			
$f_1 f_2 b_i$	4	4			4		
$f_2 b_i C_{i-1}$		4			4	4	
$f_2 f_3 \bar{a}_i C_{i-1}$		3	3	3		3	
$\bar{f}_2 f_3 \bar{a}_i b_i$		3	3	3	3		
$\bar{f}_1 f_2 \bar{f}_3 a_i C_{i-1}$	2	2	2	2		2	
$f_2 f_3 \bar{a}_i b_i$		3	3	3	3		
	6	24	11	16	14	9	0

Figure 24 (continued) Scoring table for cell output C_i

IMPLICANT	f_1	f_2	f_3	a_i	b_i	C_{i-1}	D_{i+1}
$\bar{f}_1\bar{f}_2a_i$	4	4		4			
f_1b_i	5				5		
$f_1\bar{f}_2a_i$	4	4		4			
	13	8	0	8	5	0	0

Figure 24 (continued) Scoring table for cell output D_i

$\bar{f}_1$–

$$S_i = f_2 C_{i-1} a_i b_i + f_2 C_{i-1} \bar{a}_i \bar{b}_i + f_2 \bar{C}_{i-1} a_i \bar{b}_i + \bar{f}_2 f_2 D_{i+1} + f_2 \bar{C}_{i-1} a_i b_i + \bar{f}_2 \bar{f}_3 C_{i-1}$$

f_1–

$$S_i = f_2 f_3 D_{i+1} + \bar{f}_2 f_3 a_i + f_2 \bar{f}_3 C_{i-1} + \bar{f}_2 a_i b_i + \bar{f}_2 f_3 b_i$$

$\bar{f}_2$–

$$C_i = a_i + \bar{a}_i b_i f_3$$

f_2–

$$C_i = f_1 b_i + C_{i-1} b_i + \bar{a}_i f_3 + a_i \bar{f}_1 \bar{f}_3 C_{i-1} + \bar{a}_i b_i f_3$$

Figure 24 (continued) Design equations for cell outputs S_i and C_i

$\underline{\bar{f}_1}$

$D_i = a_i \bar{f}_2$

$\underline{f_1}$

$D_i = a_i \bar{f}_2 + b_i$

Cost per cell = 10 1/3

Figure 24 (continued) Design equations for cell output D_i and cost of total network.

procedure has not failed to realize some savings over standard methods but it is not as effective as the specific procedure in this case. It should also be noted that functions C_i and D_i have cases where none of the common gates are enabled. This is due to the fact that for certain functions a constant ONE is to be outputted and as discussed previously, this can be accomplished by not enabling any gate; the following logic will interpret this as a logic ONE. It may, however, be necessary to use pull up resistors if it is found that the "floating" input to the driven logic is causing erroneous operation. This may or may not be necessary, and the decision to use external resistors must be made for specific cases.

By their very nature, multi-functional cells lend themselves to this specialized design procedure. There are of course cases which are composed of functions which are relatively complex. In this case, the general design procedure should be investigated if the cost of a network is very important, as in the case of a network to be produced in volume.

The extension of this one-dimensional example to multi-dimensional arrays is abvious. All that is required is the defining of the logical functions which must be realized by each cell. The implementation of a cell can be designed using the general procedures. It should also be noted that reconfigurable networks are also realizable as they too are comprised of combinational or sequential cells.

Clearly, the generalized procedures presented here are applicable to the design, given that the functions of the cell are well defined. Indeed, at this time, these procedures are being considered as a method of implementation of reconfigurable arrays under an N.S.F. grant currently in effect.

The advantage of reconfigurability allows the network to tolerate multiple unrestricted faults within the body of the array, thus increasing reliability greatly.

VI. REMARKS

The procedures shown here were derived for the most part by observation of several example problems worked in an attempt to generalize the capabilities of tri-state gates in a logic network. Reconfigurability is clearly possible with tri-state logic and further study in this area is needed. Additional research directed toward predicting the simplification to be expected in a given network would aid in the application of tri-state methods, since at this point, the tri-state method does not always yield simplification.

The following are open questions:

1. What does reconfigurability offer in terms of network capability?

2. Is there a way to predict the savings possible with a tri-state realization?

3. Is there formal basis for the improvements observed in tri-state realizations over classical methods?

4. Is there a better variable choice than using one of the input or state variables for the partition variable? Perhaps computing an extra output variable whose sole function is to enable the logic, would realize even more savings.

BIBLIOGRAPHY

1. "Tri-State" is a registered trade-mark of National Semiconductor Corporation.

2. Fairchild Semiconductor, Fairchild TTL Data Book, Mountainview, California; June 1972.

3. Gray, F. G.,Thompson, R. A., Theory of Finite State Machines, Unpublished Notes, VPI&SU, 1975.

4. Signetics Corporation, Signetics Digital, Linear, Mos Data Book,Menlo Park, California, 1974.

The vita has been removed from the scanned document

TRI-STATE GATES IN LOGICAL NETWORKS

by

Robert B. Aglietti

(ABSTRACT)

Several methods were presented to facilitate the use of tri-state gates in the design of logic networks. Special considerations for the use of tri-state gates are discussed in order that the reader might get a feel for the capability of tri-state logic and so that he will be aware of problems which may appear in the course of the design and development of a tri-state logic network. Several problems are worked, both combinational and sequential examples, and the advantages of tri-state over classical logic are discussed. Two additional examples, one of an iterative network and one of a multi-functional cell illustrate the applicability of tri-state gates to many digital designs.