A Method for Synthesizing Sequential Circuits

By GEORGE H. MEALY

(Manuscript received May 6, 1955)

The theoretical basis of sequential circuit synthesis is developed, with particular reference to the work of D. A. Huffman and E. F. Moore. A new method of synthesis is developed which emphasizes formal procedures rather than the more familiar intuitive ones. Familiarity is assumed with the use of switching algebra in the synthesis of combinational circuits.

CONTENTS

1. INTRODUCTION

1.1 *Foreword*

The designer of a sequential switching circuit — a circuit with storage or "memory" — faces a far more difficult problem than is faced by the

designer of, say, a simple translating circuit. In the latter case, comparatively simple and straightforward methods of synthesis are known.[1] In the former case, the designer frequently does not even know how to begin to solve the problem. Only recently did D. A. Huffman develop a method which, at an early point in the design, gives rather explicit procedures for carrying the design through to completion.[2] The method relies for its success on a tabular method of presenting the circuit requirements. This table, called a *flow chart*, may be subjected to simple manipulations which remove redundancies in the verbal statement of the circuit requirements. When supplemented by somewhat more complicated procedures, the flow chart is reduced to a form which leads directly to a circuit having a minimal number of storage elements. This process will be called *reduction* in this paper, and direct manipulation of the flow chart will be called *merging*.

Independently, E. F. Moore investigated the abstract properties of sequential circuits.[3] In particular, Moore asked what can be said about a circuit when one knows nothing about it except what may be inferred by performing experiments involving only the input and output terminals of the circuit. A by-product of Moore's theory was a general method for reducing (if necessary) a circuit whose description is completely known.* This method is essentially the same as Huffman's methods, *sans* flow chart manipulation.

The situation, then, is the following: Once a flow chart, or some equivalent statement of circuit requirements, has been obtained, one may use Moore's procedure for reducing the circuit. Once the circuit has been completely reduced, the remainder of the synthesis procedure is fairly uncomplicated. On the other hand, one may use the merging process of Huffman on the flow table. Very often this will result in complete reduction; less often it will be necessary to use additional procedures equivalent to the Moore process. Merging, when it is possible, is easier to use than is the Moore procedure, hence one would like to find a method which is as simple as merging and at the same time results in complete reduction more often than does merging.

Huffman's method was originally developed in connection with relay circuits, although it is applicable in other instances. It does not, however, always work in its unmodified form when applied to switching circuitry of the type that is commonly used in the design of digital computers.[4, 5] One then asks, how can Huffman's method be extended to cover such instances?

* We shall use the word "circuit" to refer both to physical circuits and to abstract representations of circuit requirements (such as flow charts). The latter of course, may correspond to many physical circuits.

This paper offers one possible solution to both questions. After describing an abstract model for sequential circuits, we develop Moore's method for reduction, as it applies to our model. We then develop a new method applicable to synchronous circuitry, which is commonly used in computer design. Finally, the method is extended to relay circuitry as an example of asynchronous circuitry. The relationship between our method and Huffman's method, as they are applied to this class of circuits, is then explained.

1.2 *Introductory Remarks*

It is very tempting at the outset to make the flat statement: There is no such thing as a synchronous circuit. This would be strictly true if we defined a synchronous circuit as one with the properties:

(*S1*) *Any lead or device within the circuit may assume, at any instant of time, only one of two conditions, such as high or low voltage, pulse or no pulse.*

(*S2*). *The behavior of the circuit may be completely described by the consideration of conditions in the circuit at equally-spaced instants in time.** Because it is quite clear that no *physical* circuit satisfies (S1) and (S2), such a blanket statement would be a quibble, for the engineer does recognize a certain class of circuits which he calls synchronous. The unfortunate fact is that the distinction between a synchronous and an asynchronous circuit is very hazy in many cases of actual engineering interest. Roughly, we may say that the more nearly a circuit satisfies (S1) and (S2), the more likely will an engineer be to identify it as a synchronous circuit.

As intuitive guides to the usual properties of a synchronous circuit, these characteristics are offered:

(1). There is a so-called clock which supplies timing pulses to the circuit.

(2). Inputs and outputs are in the form of voltage or current pulses which occur synchronously with pulses from the clock.

(3). The repetition rate of the clock pulses may be varied, within limits, without affecting the correct operation of the circuit, so long as input pulses remain synchronized with the clock.

Another assumption that is commonly made, although it does not bear on the distinction between synchronous and asynchronous circuits, should nevertheless be mentioned. If this assumption is made, then we may distinguish between combinational and sequential circuits.

* Actually, these need not be equally-spaced. However, the instants considered must not depend on any property of any sequence of inputs presented to the circuit, such as the duration of a pulse.

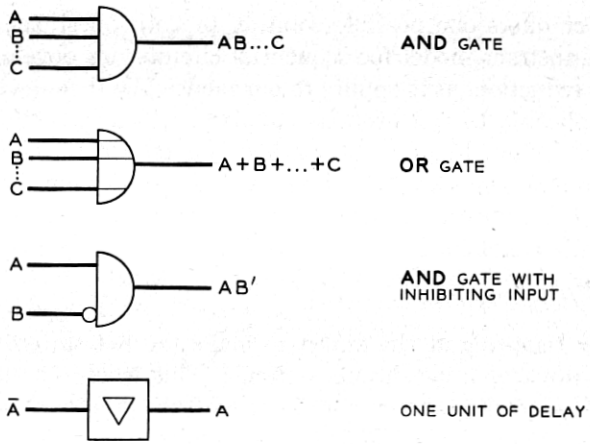

Fig. 1

(*D*). *Certain circuits contain no time delay — their input combinations in every case completely determine their output combinations.*

We will be concerned mainly with a technology in which these assumptions are nearly satisfied — that of the type employed in Leiner *et al.*[4, 5] In this technology, one uses AND gates (with or without inhibiting inputs), OR gates, delay lines, and amplifiers. For our purposes, we may ignore the need for amplifiers. The other basic circuits are as shown in Fig. 1. The properties of these circuit blocks are defined by the algebraic expressions in the illustration.*

The familiar switching (or Boolean) algebra is used, where 0 stands for no pulse, 1 for pulse, + for OR, · for AND and ( )′ for NOT. It is assumed that the reader is familiar with switching algebra and its use in practical design problems. We recall from switching algebra:

(1) A *switching function* is any (finite) expression in switching algebra.

(2) A *minimal polynomial* of n variables is any product of the form:

$$x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$$

where

$$x_i^{a_i} = \begin{cases} x_i' & a_i = 0 \\ x_i & a_i = 1 \end{cases}$$

(3) We define

$$P_j = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$$

* The unit of delay is the interval between the start of two successive clock pulses. The notation "$\bar{A}$", used in Fig. 1, will be explained in Section 2.1.

where j is the decimal form of $a_1a_2 \cdots a_n$, considered as a binary number. For example, if $n = 3$, $P_0 = x_1'x_2'x_3'$, $P_1 = x_1'x_2'x_3$, etc.

(4) Every switching function of n variables may be brought into a unique *canonical form*:

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{2^n-1} f_i P_i$$

where

$$f_j = f(a_1, a_2, \cdots, a_n)$$

(5) Corresponding to each function is a *truth-table* which displays the value of the function for each set of arguments. For $n = 2$, the truth-table corresponding to the canonical form is found in Table I. The correspondence between the truth-table and canonical form is one-to-one.

For further information about switching algebra see, for instance, Reference 9.

As an example, consider the function

$$f(x, y) = x' + y'$$

Its truth-table is Table II, and, therefore, $f_0 = f_1 = f_2 = 1$ and $f_3 = 0$. The canonical form is

$$f(x, y) = x'y' + x'y + xy'$$

2. A MODEL FOR SEQUENTIAL CIRCUITS

2.1 *The Model*

We begin by giving an abstract definition of a *switching circuit*:

A switching circuit is a circuit with a finite number of inputs, outputs.

TABLE I

x_1	x_2	$f(x_1, x_2)$
0	0	f_0
0	1	f_1
1	0	f_2
1	1	f_3

TABLE II

x	y	$f(x, y)$
0	0	1
0	1	1
1	0	1
1	1	0

and (internal) states. Its present output combination and next state are determined uniquely by the present input combination and the present state. If the circuit has one internal state, we call it a combinational circuit; *otherwise, we call it a* sequential circuit.

We have now to explain what we mean by this definition when we apply it to the technology introduced in Section 1. First, we assume a circuit has n binary-valued *input variables*, $x_1, x_2, \cdots, x_n$; m binary-valued *output variables*, $y_1, y_2, \cdots, y_m$; s binary-valued *excitation variables*, $\bar{q}_1, \bar{q}_2, \cdots, \bar{q}_s$; and s binary-valued *state variables*, $q_1, q_2, \cdots, q_s$, corresponding one-to-one with the excitation variables. In order to facilitate discussion, we note that a set of minimal polynomials may be associated with each set of variables. Specifically, corresponding to the input variables, we have the *input combinations*, X_j; associated with the output variables are the *output combinations*, Y_l; corresponding to the excitation variables are the *next states*, $\bar{Q}_k$; and with the state variables, we associate the *present states*, Q_i. For example, if $n = m = s = 3$, we have:

$$X_4 = x_1x_2'x_3'$$

$$Y_2 = y_1'y_2y_3'$$

$$\bar{Q}_1 = \bar{q}_1'\bar{q}_2'\bar{q}_3$$

$$Q_7 = q_1q_2q_3$$

We will use this notation and terminology for convenience. Rather than stating that, at some time, $x_1 = 1$, $x_2 = 0$, and $x_3 = 0$, we will say that input combination X_4 (or its equivalent — input combination 100) is present. That is, $X_4 = 1$ and thus the inputs are, respectively, 1, 0, and 0.

Now, according to the definition given above, to each circuit we must be able to assign some set of equations relating the $\bar{q}_i$ and y_i to the x_i and q_i. These equations will have the general form:

$$\bar{Q}_k = \bar{Q}(Q_i, X_j)$$

$$Y_l = Y(Q_i, X_j)$$

That is, k and l must be uniquely determined by i and j. Each circuit is associated with a truth-table with its columns headed (in order):

$$q_1, \cdots, q_s, \qquad x_1, \cdots, x_n; \qquad \bar{q}_1, \cdots, \bar{q}_s, \qquad y_1, \cdots, y_m.$$

The number of circuits having n input, m output, and s internal vari-

ables is equal to

$$2^{(m+s)2^{(n+s)}}$$

since the truth table has $2^{(n+s)}$ rows and $m + s$ columns which must be filled in with 0's and 1's.

The interpretation of this model is now fairly straightforward. We have assumed (S1), (S2), and (D) and know that, physically, the delay unit provides storage. We assign the $\bar{q}_i$, the excitation variables, to the inputs of delay lines, and we assign the q_j, the state variables, to delay line outputs. The present state of the circuit is the combination of conditions on the delay line outputs. The next state is the combination of conditions on the delay line inputs, since one time unit later this combination will be present on the outputs.

To make the discussion concrete, consider Fig. 2. The circuit equations are:

$$\bar{q}_1 = q_1'q_2' + x'q_2'$$
$$\bar{q}_2 = q_1q_2' + xq_1$$
$$y = q_1'q_2'$$

From these equations, we write Table III.

2.2 *State Diagrams*

It is usually not clear from an examination of the circuit diagram or circuit equations just what a sequential circuit does. The truth-table

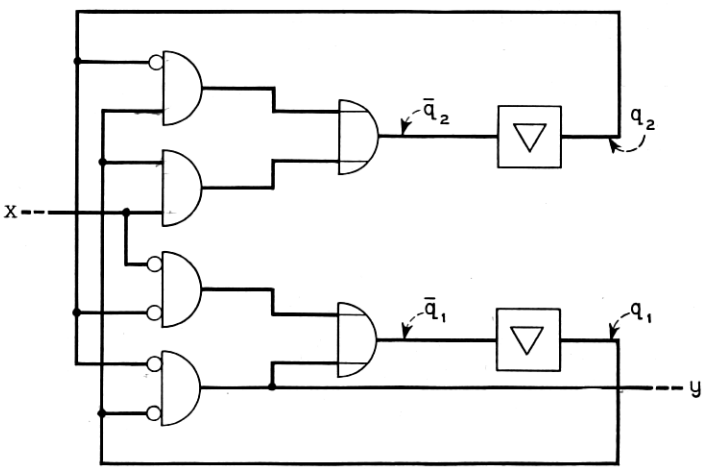

Fig. 2

TABLE III

q_1	q_2	x	$\bar{q}_1$	$\bar{q}_2$	y
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	0

is more helpful and tells the whole story if we put it in a different form, called a *state diagram*. In this diagram, circles will represent states. Each line of the truth-table will be represented by an arrow going from the present to the next state. A label on the arrow will give the corresponding input and output combination. The state diagram for the circuit discussed in Section 2.1 is given in Fig. 3.

The arrows in the state diagram correspond to changes of state of the associated circuit, and both the arrows and the changes of state are called *transitions*. A transition begins at a present state and ends at the next state. The transition is labeled X/Y. X is an input combination and Y is the corresponding output combination.

As an example, consider Table IV, which gives the sequences of states and outputs which correspond to each initial state of the circuit and the input sequence 100. Depending upon what state the circuit is started in, the input sequence 100 produces three different output sequences. It is

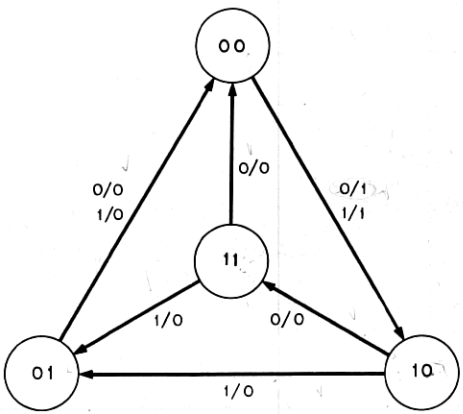

Fig. 3

TABLE IV

x	0 0 1	1 0 0	1 0 0	1 0 0
q_1	0 0 1	1 0 0	1 0 0	0 1 1
q_2	1 0 0	1 1 0	0 1 0	0 0 1
$\bar{q}_1$	0 1 1	0 0 1	0 0 1	1 1 0
$\bar{q}_2$	0 0 1	1 0 0	1 0 0	0 1 0
y	0 1 0	0 0 1	0 0 1	1 0 0

difficult and probably of little value to put into words exactly what this particular circuit does. However, given any initial state and any sequence of inputs, we can immediately tell what happens from the state diagram. (The truth table may be used for the same purpose, but less easily. It is far more difficult to determine circuit behavior by chasing signals around the circuit diagram.) The problem of circuit analysis is now completely solved. Given any circuit, we may immediately write its circuit equations. A truth-table is easily obtained from the equations. Given the truth-table or given the associated state diagram, we may determine exactly how the circuit behaves for any initial state and input sequence.

Conversely, once a state diagram or truth-table is found for a proposed circuit, the above steps may be traced backwards in order to arrive at a circuit diagram. The only problem here is designing combinational circuits economically. The really significant problem in sequential circuit synthesis is finding a suitable state diagram or truth-table. This problem, in turn, may be subdivided into two problems:

(1) finding *any* state diagram or its equivalent which fulfills the circuit requirements and

(2) reducing this to the state diagram which is to be used for the final part of the design process.

The next section of this paper develops Moore's method of reduction and is basic in justifying the methods developed in the succeeding sections.

3. CIRCUIT EQUIVALENCE

3.1 *Moore's Theory*

The key to the synthesis of sequential circuits is the concept of circuit equivalence which was discovered independently by Huffman[2] and Moore.[3] We are concerned mainly with the portions of Moore's theory which have direct application to synthesis; certain differences in treatment are necessary since Moore's model for sequential machines is differ-

ent from ours. All of Moore's arguments carry over with only slight changes.

Roughly speaking, we call two circuits equivalent if we cannot tell them apart by performing experiments involving only their inputs and outputs. Once we have solved the first problem of synthesis by finding any state diagram which fulfills the circuit requirements it will usually be found that the state diagram has more states than are necessary to perform the assigned task. In such a case, we usually wish to simplify the circuit by removing redundant states in such a way that the final circuit is equivalent to the original one.

We must now make the concept of equivalence more precise. We define:

(1) *Two* states, Q_i *in circuit* S *and* Q_j *in circuit* T, *are called* equivalent *if, given* S *initially in state* Q_i *and* T *initially in state* Q_j, *there is no sequence of input combinations which, when presented to both* S *and* T, *will cause* S *and* T *to produce different sequences of output combinations.*

(2) *Two* circuits, S *and* T, *are called* equivalent *if, corresponding to each state* Q_i *of* S, *there is at least one state* Q_j *of* T *such that* Q_i *is equivalent to* Q_j; *and corresponding to each state* Q_j *of* T *there is at least one state* Q_k *of* S *such that* Q_j *is equivalent to* Q_k.

In (1), it should be noted that T may be a copy of S. Hence (1) is also a definition for equivalence between states in the same machine. Moore has shown that even if no two states in a given machine are equivalent, it is not always possible to find out what state the machine started in by some experiment. That is, there is not always a sequence of input combinations which will result in a different sequence of output combinations for each possible initial state of the circuit. The state diagram of Fig. 3 is the example used by Moore to prove this; state 11 may not be distinguished from state 10 by any experiment which begins with a 1, and state 01 may not be distinguished from state 11 by any experiment which begins with a 0.

If there are two states in a circuit which are equivalent, it should be possible to eliminate one of them. This will result in a circuit equivalent to the original circuit. This is indeed possible, and the process of reduction may be carried out in an essentially unique manner, as is stated by

Theorem 1 (Moore). Corresponding to each circuit, S, *is a circuit* T *which has the properties: (1)* T *is equivalent to* S, *(2)* T *has a minimal number of states, (3) no two states in* T *are equivalent, and (4)* T *is unique, except for circuits that result from* T *by relabeling its states.* T *is called the* reduced form *of* S.

We shall state the procedure to be followed in deriving T from S,

referring the reader to Reference 3 for a complete proof of Theorem 1. First, divide the states of S into sets such that (1) all states in a given set are equivalent, (2) if a state is in a given set then all states equivalent to that state are also in the same set, and (3) no state is in two different sets. These sets are called *equivalence sets* or *classes.* Now, assign a state of T to each equivalence set of states. If there is a transition, bearing the symbol X/Y, from a state in one equivalence set of S to a state in a different equivalence set of S, insert a transition bearing the same symbol X/Y between the corresponding states in T. If there is a transition between two states in the same equivalence set of S, insert a transition in T which begins and ends at the corresponding state of T. Do this for all transitions in S.

We have not given as yet an effective procedure for determining the equivalence sets. This procedure will be provided by the method of proof of the next theorem. Before stating the theorem, we state a precise definition of what we mean by "experiment." By an *experiment of length* k, we mean the process of presenting a circuit which is in some specified initial state with a sequence of k successive input combinations. By the *result* of an experiment, we mean the sequence of output combinations produced by the experiment. We say that two states are *indistinguishable by any experiment of length* k if for all experiments of length k the result does not depend on which was the initial state. We may now state

*Theorem 2 (Moore). Given a circuit S whose reduced form has a total of p states, then for any two states, Q_i and Q_j, in S, Q_i is equivalent to Q_j if and only if Q_i is not distinguishable from Q_j by any experiment of length $(p - 1)$.**

Proof: Consider all experiments of length k. All states may be divided into equivalence sets by the rule: put two states in the same equivalence set if and only if they are indistinguishable by any experiment of length k. For each k, there is now defined a set of equivalence sets which we will call P_k.

Consider two states, a and b, that are not equivalent but are indistinguishable by any experiment of length k. Since a and b are not equivalent, there is an experiment of some minimum length, say n, that will distinguish a from b. Consider the two states, $\bar{a}$ and $\bar{b}$, that a and b are taken into by the first $(n - k - 1)$ input combinations of the experiment. $\bar{a}$ and $\bar{b}$ are then distinguishable by an experiment of length $(k + 1)$ but by no shorter experiment.

We have now proved that P_k is not already the set of equivalence sets

* This theorem is a trivial extension of Moore's result.

TABLE V

q_1	q_2	x	$\bar{q}_1$	$\bar{q}_2$	y
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0			
1	1	1			

As a more complete example, including the construction of a reduced machine, consider Fig. 4(a). Applying Rule I, we get:

$$\bar{P}_1 : (0)\ (1, 2, 3, 4)$$
$$\bar{P}_2 : (0)\ (2, 4)\ (1, 3)$$

1

To construct the reduced circuit, assign state A in the new circuit to (0), B to (1, 3), and C to (2, 4). The resulting circuit is shown as Fig. 4(b).

In order to develop a physical circuit, it is necessary to assign a binary code to the states. The assignment is more or less arbitrary for synchronous circuits, but will in general affect the number of circuit elements used. In this instance, we choose to make the assignment:

$$\text{A} \rightarrow 01$$
$$\text{B} \rightarrow 00$$
$$\text{C} \rightarrow 10$$

Rewriting the state diagram as a truth-table, we get Table V. Two rows in the right half of the truth-table are blank, since state 11 does not appear in the state diagram. It is legitimate to fill these rows in in any way, and it is preferable to fill them in in a manner that results in simplification of the final circuit. Taking advantage of this fact, we may set:

$$\bar{q}_1 = q_1' q_2' x$$
$$\bar{q}_2 = q_1$$
$$y = q_2$$

The final circuit is shown in Fig. 5. Fig. 6 shows the state diagram for the completed circuit. As it happens, state 11 is not equivalent to any other state.

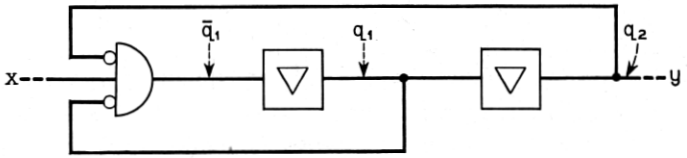

Fig. 5

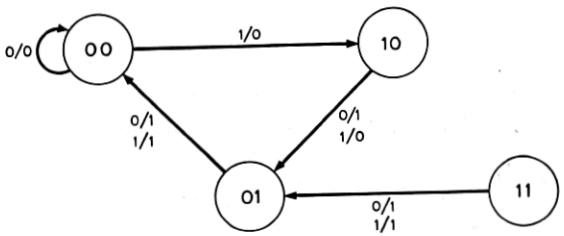

Fig. 6

This concludes the material on circuit equivalence. In the following section, we develop the method for synchronous circuits. As will be seen, an essential feature of the method is the use of truth-tables rather than state diagrams (which become unmanageable for circuits with more than a few variables) and a very much simplified form of Rule I which may be applied directly to truth-tables. Our program will be (1) to describe the kind of argument used in going from verbal circuit requirements to a truth-table; (2) to restate Rule I in a form (Rule II) which is adapted to synthesis and applies to truth-tables; (3) to develop Rule III, a generalized form of Huffman's merging process; (4) to discuss "don't care" situations, familiar to the reader from the study of combinational circuits; and (5) to give a summary of the method. A complete design example will be given in Section 5.5, following application of the method to asynchronous circuits.

4. DEVELOPMENT OF THE METHOD FOR SYNCHRONOUS CIRCUITS

4.1 *Introductory Remarks*

As seen in the last section, the first problem in synthesis is finding some state diagram that will behave according to the circuit requirements. The state diagram need not be very efficient in the sense that it may have far more states than are actually needed, for the procedures developed in the last section give a straightforward procedure for removing redundant states. Unfortunately, the initial step in the process relies heavily on

the designer's ingenuity. However, we can outline procedures that are of some assistance in finding an initial state diagram.

The simplest case, and indeed the only wholly straightforward case, is that in which the circuit must always return to its initial state after it has received some fixed number of input combinations. Essentially, this case is simple because we may consider all possible input sequences. We assign a new state any time anything happens, up to the last input. The last input then takes us back to the initial state. For instance, suppose that we want a circuit which receives sets of three binary digits in serial form and puts a pulse out on one of eight leads during the third digit to indicate the number that was received. The state diagram may immediately be written down, as shown in Fig. 7. Rather than write sets of 8 binary digits for the output symbols, we have designated the lead that should be energized, if any, and otherwise have written "0".

It is immediately clear that this is even a reduced machine — no two states are equivalent. This is an extreme case; usually there will be certain sequences of inputs which will never occur and/or certain sequences of inputs for which (in Huffman's words) we do not care to specify the circuit action. More often, however, there will be patterns of successive input combinations that will produce the same circuit action. For instance, suppose that in a sequence of 4 inputs we wish to have a final output only if the input sequence is 1010 or 0101. Then we can draw a state diagram showing all sequences which is shown as Fig. 8(a). How-

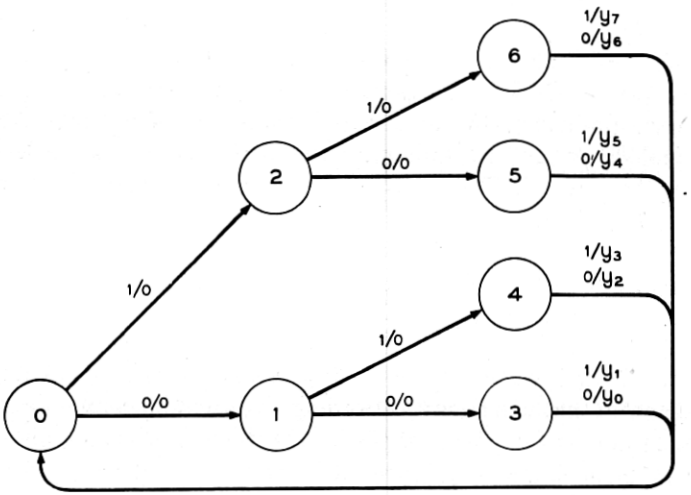

Fig. 7

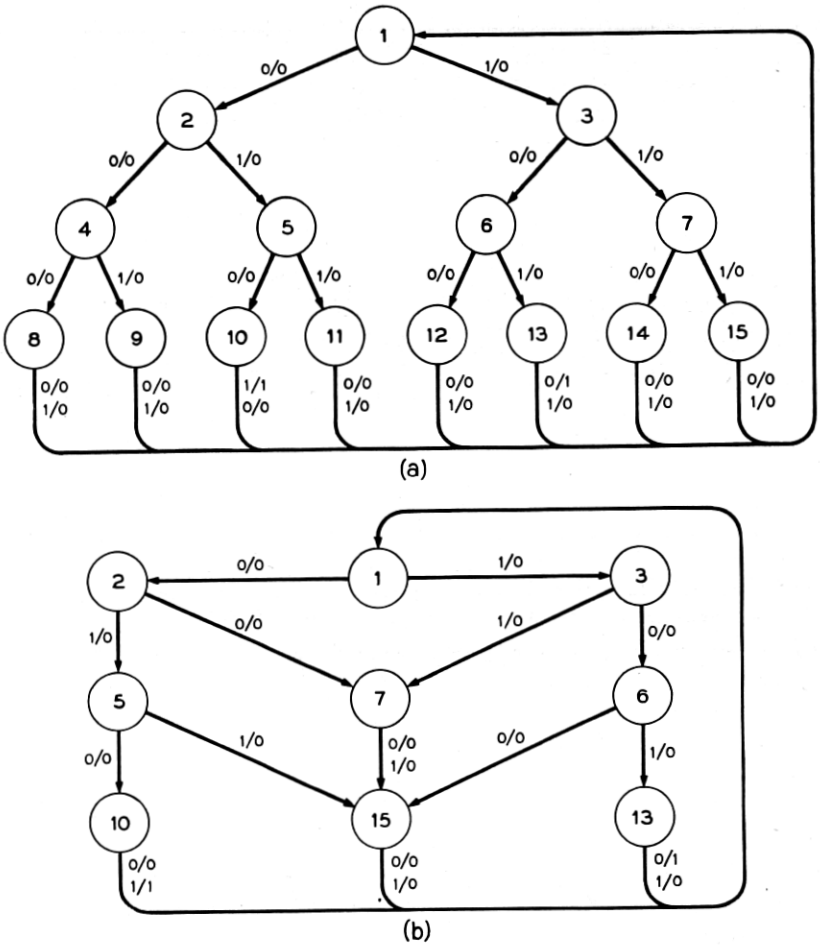

Fig. 8

ever, with a modest amount of ingenuity, we might have drawn Fig. 8(b) as our first attempt. In fact, it is clear that Fig. 8(b) shows the reduced form of the diagram in Fig. 8(a).

On the other hand, if there is no state which is entered cyclically, as above, no really explicit directions may be given for drawing an initial state diagram. In practice, one starts to draw a branching diagram such as the above. To terminate each branch, it is necessary to recognize that each transition from the state at the end of the branch may terminate in

TABLE VI

Input combination....	0	1	0	1
Present State	Next State		Output Combination	
1	2	3	0	0
2	4	5	0	0
3	6	7	0	0
4	8	9	0	0
5	10	11	0	0
6	12	13	0	0
7	14	15	0	0
8	1	1	0	0
9	1	1	0	0
10	1	1	0	1
11	1	1	0	0
12	1	1	0	0
13	1	1	1	0
14	1	1	0	0
15	1	1	0	0

some state which is already in the diagram. To the author's knowledge, no more specific directions for this are possible.

In practice, large state diagrams become very messy to draw. Where this is the case, it is better to revert to the truth-table, recast in a matrix form with states corresponding to rows and input combinations corresponding to columns. One of the most valuable features of this mode of presentation is that the truth table may be used directly to perform a large part of the reduction process. To illustrate the truth-table in a simple case consider Table VI which is the truth-table corresponding to the state diagram of Fig. 8(a). Of the two portions of the table, the left hand one represents the next states and the right hand one gives the output combinations.

4.2 *Modification of First Reduction Process*

At this point, we give an extension of Rule I which applies to truth-tables. It was noted above that Moore's theory assumes that each machine is completely specified, although the specification is not known to the experimenter. In our restatement of Rule I, we must allow for the possibility of blank entries in the truth-table. This provision amounts to calling two circuits equivalent if there is no evidence for believing that they are not equivalent.*

* This procedure is essentially that stated in Reference 2, pp. 183–185.

Rule II: Separate the rows of the truth-table into sets such that two rows are in the same set if and only if no corresponding entries in the right-hand portion of the rows are contradictory. (A blank entry is not considered to contradict any entry.) Call these sets "$\overline{P}_1$." Given the set of sets $\overline{P}_k$, find if possible two rows in the same set of $\overline{P}_k$ such that for some input combination the two rows have row designations (next states) which are not blank and correspond to rows in different sets of $\overline{P}_k$. Put one of these rows into a new set in $\overline{P}_{k+1}$ together with all rows in the original set of $\overline{P}_k$ which go into the same set in $\overline{P}_k$ for the given row and input combination. Leave the other sets in $\overline{P}_k$ fixed in $\overline{P}_{k+1}$. If this is not possible, the process terminates. Now apply the truth-table analog of the process described following Theorem 1.

Except for the stipulations concerning blank entries, Rule II is merely a reworded form of Rule I.

4.3 *Second Reduction Process*

Rule II, given above, seems rather complicated. Although this complication is more apparent than real, one still wishes to find a reduction rule that has both the effect and the appearance of simplicity. Presumably, one must pay for this in one way or another — the surprising thing is that one is not required to pay too heavily. In point of fact the reduction rule given below, when applied to asynchronous circuits, is somewhat more powerful than Huffman's rule for merging.

We ask, then, what are the simplest circumstances in which a state may be eliminated by using Rule II? Is it possible to consider only pairs of states instead of considering larger sets of states? To answer these questions, consider any pair of rows that are in the same set of $\overline{P}_1$. That is, no corresponding entries in the right-hand portion of the rows may be contradictory. Now if in addition no corresponding entries in the left-hand portion of the rows are contradictory, then the two states have the same output combination for a given input combination and the next state is the same, or may be made to be the same by filling in a "don't care" entry, for any given input combination.* Therefore, the two states are equivalent. This means that we may eliminate one and keep the other. If we eliminate state A in favor of state B, then any appearance in the table of "A" must be changed to read "B".

We restate the above more formally as Rule III. This process is called *merging*, after Huffman, since we will see that it is a general form of his merging process.

* Or the present state is also the next state in both cases.

TABLE VII

Input combination....	0	1	0	1
Present State	Next State		Output Combination	
1	2	3	0	0
2	7	5	0	0
3	6	7	0	0
5	10	15	0	0
6	15	13	0	0
7	15	15	0	0
10	1	1	0	1
13	1	1	1	0
15	1	1	0	0

TABLE VIII

Input combination....	0	1	0	1
Present State	Next State		Output Combination	
1	1	2	0	0
2	3	1	0	1
3	2	4	0	1
4	4	3	0	0

Rule III: To merge state A with stage B, change all appearances of "A" in the table to read "B" and copy the entries of row A into row B. Eliminate row A.

Rule III may be used whenever, after the "A's" have been changed to "B's", to each entry in row A corresponds either the same entry in row B or a blank in row B.

As an example, consider Table VI. We see that states 8, 9, 11, 12, and 14 may be merged with state 15. Then state 4 may be merged with state 7. The resulting table, Table VII, now corresponds to the state diagram of Fig. 8(b).*

Note that Rule III may not always give complete reduction. An example is Table VIII, to which Rule III may not be applied. However, Rule II leads to the conclusion that states 2 and 3 are equivalent, as are states 1 and 4.

* The reader is urged to write out the intermediate truth-tables derived by carrying out the mergers step by step.

4.4 *Blank Entries; Uniqueness of Reduction*

The provision for blank entries in Rules II and III corresponds to "don't care" situations, which usually result from restrictions on the input sequences. The result of merging rows in different orders is not always unique. The reason for this is simple — when truth-tables have blanks, they may usually be filled in in different ways so as to result in circuits which are not equivalent. Since merging usually results in filling in blanks, different orders of merging may result in blanks being filled in differently. This situation is not in contradiction with Moore's theory; there it is assumed that the state diagram is completely specified at the outset.

As an example, consider Table IX(a). Here, there are four output leads. The designation of which lead is to be energized is given in the right portion of the table — a dash indicates that no lead is to be energized. Clearly, we may merge 8 and 9 with 7; 4 and 5 with 3; and 6 with 1. The result is shown in Table IX(b). A final merging of 7 with 3 and 2 with 1 leaves the table of Table IX(c). On the other hand, if we merge 2 with 1; 4 with 3; 6 with 5; and 8 and 9 with 7 we get Table IX(d) instead, and Rule II tells us that this is a completely reduced circuit.

We have, incidentally, demonstrated that reduction is not necessarily unique even if only Rule II is used, since Rule III is a restricted form of Rule II. Therefore, Theorem 1 is not necessarily valid unless the initial truth-table has no blank entries. Again, this does not mean that the theorem as originally stated is false — it means only that we are applying it under conditions which are somewhat more general than those obtaining in Moore's theory. Actually, we are really considering sets of circuits in synthesis. Each circuit is described only partly by the initial truth table and the truth table is, in a mathematical sense, a kind of domain of definition for the circuits in the set considered. Within this domain all circuits in the set are identical while outside this domain the circuits are specified only by "don't cares" and therefore may differ. Moore's theory applies to each individual circuit. We, on the other hand, are applying it to sets of circuits and must therefore be prepared to find some differences in detail.

4.5 *Summary of Method*

In general we start synthesis by writing either (1) a state diagram or (2) a truth-table, as outlined in Section 4.1. Following this step, we use Rule I supplemented by stipulations concerning "don't cares" or Rule III followed by Rule II to achieve reduction. In case (1), the state dia-

TABLE IX(a)

Input Combination....	00	01	11	10	00	01	11	10
Present State	Next State				Output Combination			
1	1	6		2	—	AA		AB
2			3	2			—	AB
3		4	3	5		DB	—	DA
4	1	4			—	DB		
5	1			5	—			DA
6		6	7			AA	—	
7		9	7	8		DB	—	DA
8	1			8	—			DA
9	1	9			—	DB		

TABLE IX(b)

Input Combination....	00	01	11	10	00	01	11	10
Present State	Next State				Output Combination			
1	1	1	7	2	—	AA	—	AB
2			3	2			—	AB
3	1	3	3	3	—	DB	—	DA
7	1	7	7	7	—	DB	—	DA

TABLE IX(c)

Input Combination....	00	01	11	10	00	01	11	10
Present State	Next State				Output Combination			
1	1	1	3	1	—	AA	—	AB
3	1	3	3	3	—	DB	—	DA

TABLE IX(d)

Input Combination....	00	01	11	10	00	01	11	10
Present State	Next State				Output Combination			
1	1	5	3	1	—	AA	—	AB
3	1	3	3	5	—	DB	—	DA
5	1	5	7	5	—	AA	—	DA
7	1	7	7	7	—	DB	—	DA

gram must now be translated into a truth-table. At this point in the process binary coding must be assigned to the states in order to complete synthesis with two-valued storage elements. Two remarks are in order here:

(1) The simplicity of the final circuit will be affected by the exact coding assigned as well as by the truth-table finally chosen, if reduction is not unique.

(2) Using a minimum number of storage elements is not always wise. In practical situations, the choice of components dictates one's criterion for minimality, and this criterion must ultimately be based on considerations of economy and reliability. For instance, the present writer has seen an example in which it was much more economical to use seven, rather than three, storage elements in order to achieve eight states. In fact one has doubts that complete reduction, itself, is always desirable.

5. THE METHOD APPLIED TO ASYNCHRONOUS CIRCUITS

5.1 *Introductory Remarks*

In this section we carry out the transition from synchronous to asynchronous circuitry. A more exhaustive treatment of the subject of asynchronous circuitry is contained in Huffman.[2]

We agree (1) that no clock will be used and (2) that "1" in switching algebra will correspond to a high voltage or current, an energized relay coil, or operated relay contacts. We must now pay careful attention to circuit conditions at *every* instant of time. One very real difficulty arises since time delays inherent in the "combinational" circuit elements may frequently be of the same order of magnitude as the time required to change the state of a storage element. This may mean that spurious inputs to flip-flops may be produced by changes of input combination solely because of nonuniform delays in portions of the "combinational" circuitry. These difficulties will not be considered further since little can be said about them over and above noting their existence. Another problem — that of *race conditions* (a definition of this term will be given below) — can be resolved by logical methods; we shall treat this problem in moderate detail.

5.2 *Interpretation of the Model*

For the purpose of illustrating the pertinent facts and methods which relate to asynchronous circuits, we use relay circuitry as being typical of asynchronous circuitry. Fig. 9 illustrates our conventions and notations.

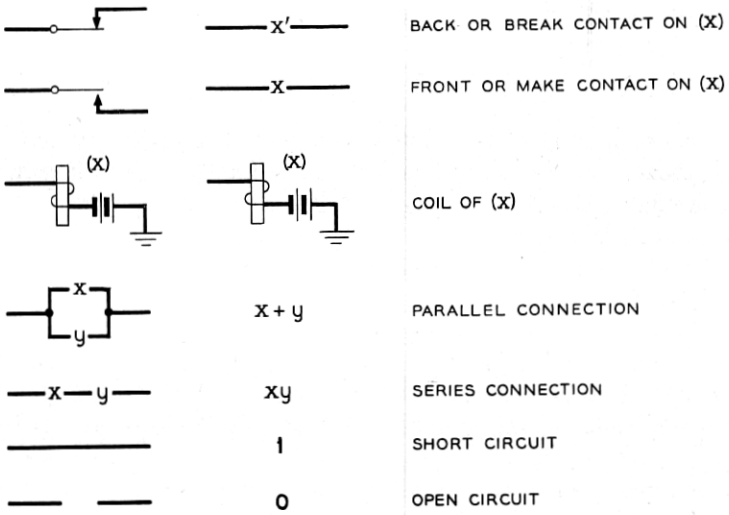

Fig. 9

Our interpretation of the abstract model for sequential circuits given in Section 2 must be changed somewhat. To be concrete, consider the circuit of Fig. 10. We think of this circuit as having two types of relays — to *primary* relays correspond input variables and to *secondary* relays correspond excitation and state variables. The general situation is shown in Fig. 11. The primary relays are controlled directly by the inputs; we shall use "x_i" to denote both the i^{th} input and the contacts on relay (x_i). The secondary relays are controlled by contacts on any or all relays; they furnish the storage in the circuit. Considering relay (q_i), we shall say that $\bar{q}_i = 1$ whenever the coil of (q_i) is energized and that $q_i = 1$

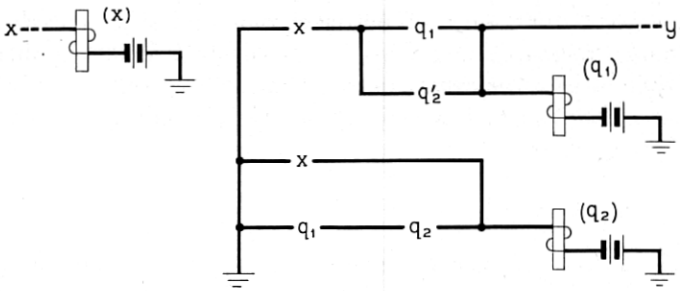

Fig. 10

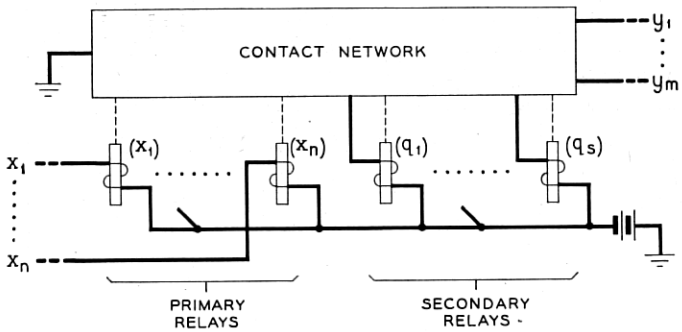

Fig. 11

whenever (q_i) is fully operated. Note carefully the distinction between these two statements!

5.3 *Race Conditions; Coding of States*

The meaning of "present state" is clear enough — it is determined by which secondary relays are *operated*. We shall say that the "next" state is determined by which secondary relays are *energized*. However, the "next" state may never be realized as a *present* state! We shall now reconsider the circuit of Fig. 10. On the basis of our previous agreement, we may draw a truth-table and state diagram. The truth-table is that given by Table X, and the state diagram is shown in Fig. 12.

In order to study the action of asynchronous circuits, it is often convenient to make use of *sequence diagrams*.[6] These are essentially pictures of what happens in a circuit as a function of time;* a line opposite a relay or lead designation represents an operated relay or a grounded lead. For instance, assume that both relays in Fig. 10 are released, a ground is applied and then released later on, and moreover that (q_1) is faster in operating than (q_2). The corresponding sequence diagram is shown in Fig. 13(a). Clearly, in this case, the circuit does almost what one would expect from consideration of the state diagram, except that the circuit goes from state 00 to state 11 by way of state 10! The situation is quite different if (q_2) is faster in operating than (q_1), as shown by Fig. 13(b). In this case, although state 11 is the "next state," it is never reached, since (q_2) in operating breaks the operating path of (q_1). A situation such as this is called a race condition. Whether it is harmful or not depends on the circuit requirements.

* The time scale is usually distorted, sequence of events being more important than their duration.

TABLE X

q_1	q_2	x	$\bar{q}_1$	$\bar{q}_2$	y
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	1	1	1

We say that a *race condition* exists in a circuit for input combination X_i and present state Q_j if the next state Q_k is such that the binary forms of j and k disagree by more than one binary digit. For, if they do, more than one relay is attempting to change its state of operation, and differences in operate and/or release times may lead to differences in circuit behavior.

In order to avoid races, it is necessary and sufficient that any distinct states directly connected by a transition disagree in exactly one binary digit. We can always avoid races if we add enough extra states. On the other hand, if a race condition is not harmful, removing the race condition generally decreases circuit operating speed.

One further remark must be made: it is often very helpful to assume that only one input variable may change its value at any given instant and to arrange connecting circuits in a system so that this condition is satisfied. To appreciate why this might be the case, consider a system containing two interconnecting circuits. These circuits may be viewed together as a single, larger circuit. If the above condition on the interconnecting leads is not fulfilled, then race conditions may be present in the over-all circuit even though they are not present in either circuit considered by itself.

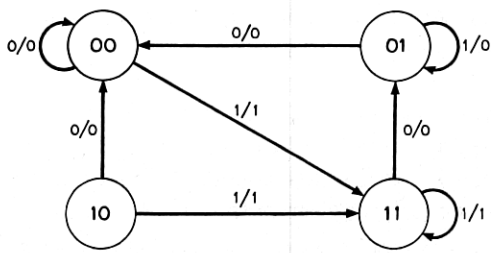

Fig. 12

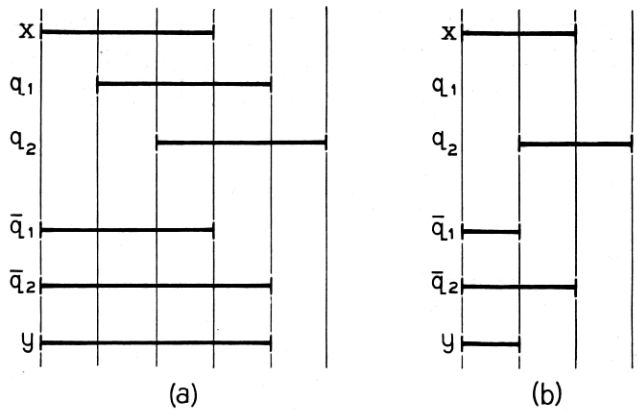

Fig. 13

The usual state of affairs in an asynchronous circuit is this: upon a change of input combination, if the "next" state of the circuit is different from its present state, the states of the individual storage elements will change until a final state of the circuit is reached in which no further change of state is possible. Two remarks are in order here. First, we have already seen that in the presence of race conditions the final state, if any, may depend on operate and release times as well as on the truth-table for the circuit. Second, there may be no final state — this is the case for certain pulse-generating circuits.* Usually however, if the new input combination is maintained for a sufficiently long interval, a final state will be reached. Since in most cases of practical interest the time required to reach the final state is much less than the interval during which any given input combination is held, design effort is fixed on the final states, rather than any possible intermediate states.

For the above reasons, the formal part of synthesis — that part of synthesis which ends with writing out circuit equations — is both different and more difficult in the case of asynchronous circuitry. Although it is true that we need not consider the possibility of race conditions until that point in synthesis in which we assign binary coding to the states, it is not true that the same truth table may always be used for both a synchronous and an asynchronous realization of a given circuit. (That this is possible for the circuit of Table IX is only accidental). The reason for this is tied in very closely with the fact that we speak of presence or absence of pulses in synchronous circuits but of

* See Reference 6, Chapter 18, for examples.

quasi-steady-state conditions on leads in asynchronous circuits. A pulse on lead x_2, for instance, might be represented by X_2 in a synchronous circuit but as X_0 followed by X_2 followed by X_0 in an asynchronous circuit.

5.4 *Huffman's Method*

The purpose of this section is not to outline Huffman's method of synthesis[2] but, rather, to support our claim made above that Rule III represents a slight generalization of Huffman's merging process. We shall assume familiarity with the contents of Reference 2.

The justification for this claim is immediate, if not already self-evident to the reader. Namely, suppose that an initial flow table is written down. By going immediately to the associated truth table, Huffman's rule for merging becomes the same as Rule III, except that Rule III allows somewhat more latitude for merging in that it is permissible to change the symbols corresponding to certain next states. In Huffman's method, it would be necessary to resort to equivalence arguments in such instances. We are considering here that the use of equivalence arguments is separate from the purely mechanical merging process, although there is evidence in Reference 2 that Huffman considers the use of such arguments to be a part of merging. Our point is that such arguments may be avoided in many cases if we work directly with the truth table and Rule III.

5.5 *Summary of Method*

We have now disposed of the basic principles of our method as applied to asynchronous circuits. The synthesis steps are:

(1) Write a truth-table which satisfies the circuit requirements.

(2) Use Rules II and III in reverse order, as applicable, to obtain a reduced truth-table.

(3) Code the states in a binary code. If possible, assign the code so that no harmful race conditions are present. Otherwise, add states in such a way as to make eliminate harmful races.[2, 6]

(4) Write the circuit equations.

(5) Synthesize the combinational networks.

As our final example, we consider the following problem, taken from Reference 6 (Problem 8–9):

A rotating shaft carries a single grounded brush which makes contact with three stationary commutator segments arranged symmetrically around the shaft. A relay circuit is required which will indicate the direction of shaft rotation by lighting a lamp when the shaft is rotating in the

clockwise direction. The shaft may reverse its direction at any time. Assume that the shaft is driven so that a brush contact closure is 0.25 second and that the open time between the brush leaving one segment and reaching another is 0.25 second. When the shaft changes direction, the output indication must change as quickly as possible, at most within 2 seconds.

Let the brush be grounded and the three segments be labelled "x_1," "x_2" and "x_3" respectively. For the output indication, let $y = 1$ when the shaft is rotating in the clockwise sense. Now, in order to write the initial truth-table, we may first consider what the circuit must do to keep track of the brush while it is rotating in only one direction. This situation is clearly taken care of by Table XI(a). All that remains is to enlarge the table to enable (say) the circuit to go from states 1–6 to states 7–12 when the direction is changed from clockwise to counterclockwise. A usable strategy is this: as one segment, say x_1, is passed the circuit expects x_2 to come up next. If x_3 comes up before x_2, we can cause the circuit to go to the counterclockwise state in which x_3 has occurred and x_2 is expected next. This has been done in Table XI(b).

With regard to the output note that it is sufficient to assign $y = 0$ to states $7 - 12$ and $y = 1$ to states $1 - 6$, regardless of input combination.

The possibilities for merging, (using Rule III), are obvious: merge 2 with 3, 4 with 5, 6 with 1, 8 with 9, 10 with 11, and 12 with 7. The result is Table XI(c). Now use Rule II to determine whether reduction is complete. Actually, literal use of Rule II is a waste of time, for we may use this argument:

$$\overline{P}_1 : (1, 3, 5)\ (7, 9, 11)$$

By examining input combination 100, we split off both 5 and 9 from the sets above, arriving at:

$$\overline{P}_2 : (1, 3)\ (5)\ (7, 11)\ (9)$$

By examining input combination 010, we see that 1 and 3 (7 and 11) are distinguishable. Hence, the circuit is completely reduced.

We now have to code the states. To assist in this process, we draw the state diagram shown in Fig. 14(a). Since there are two triangles in the diagram, we cannot assign codes to avoid races, and therefore extra states must be added. One way to do this is to insert new states between 5 and 1 and between 11 and 7 in such a way that the circuit will treat the new states as transient states. This has been done and coding has been assigned in Fig. 14(b). The corresponding truth-table is shown as Table XII.

TABLE XI(a)

Input Combination....	000	100	010	001	(All)
Present State	Next State				Output Combination
1	1	2			1
2	3	2			1
3	3		4		1
4	5		4		1
5	5			6	1
6	1			6	1
7	7			8	0
8	9			8	0
9	9		10		0
10	11		10		0
11	11	12			0
12	7	12			0

TABLE XI(b)

Input Combination....	000	100	010	001	(All)
Present State	Next State				Output Combination
1	1	2	10		1
2	3	2			1
3	3		4	8	1
4	5		4		1
5	5	12		6	1
6	1			6	1
7	7		4	8	0
8	9			8	0
9	9	2	10		0
10	11		10		0
11	11	12		6	0
12	7	12			0

TABLE XI(c)

Input Combination....	000	100	010	001	(All)
Present State	Next State				Output Combination
1	1	3	11	1	1
3	3	3	5	9	1
5	5	7	5	1	1
7	7	7	5	9	0
9	9	3	11	9	0
11	11	7	11	1	0

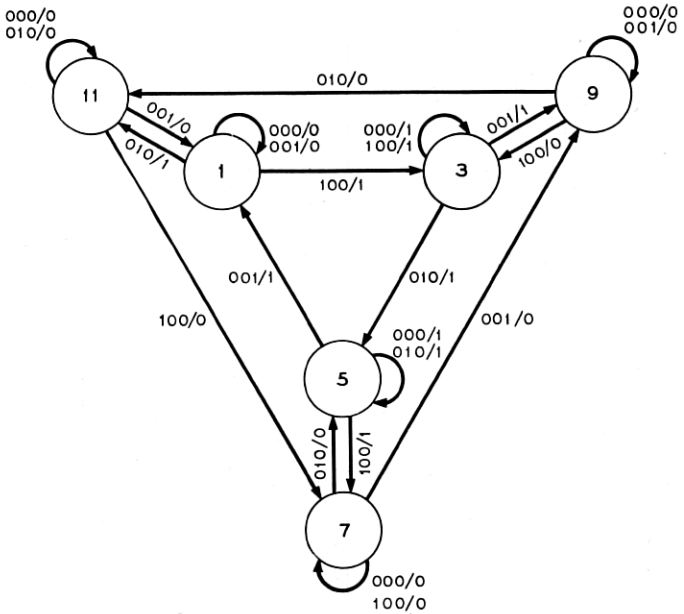

Fig. 14(a)

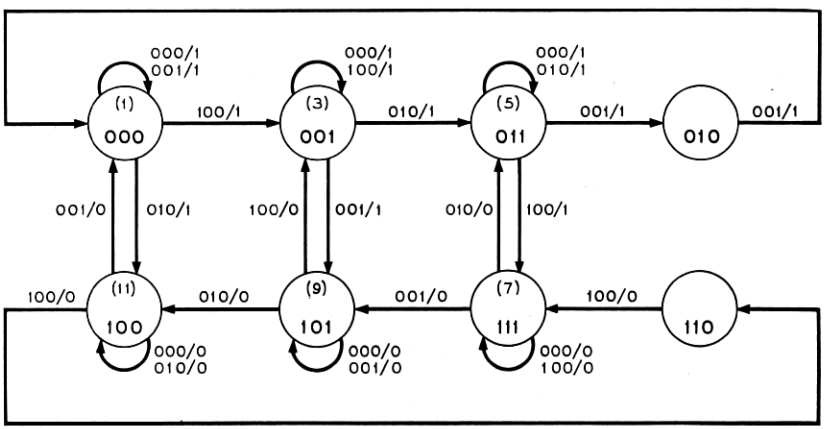

Fig. 14(b)

The circuit equations may be written as:

$$\bar{q}_1 = (q_2'q_3'x_2 + q_2q_3x_1 + q_2'q_3x_3 + q_1)\cdot(q_2'q_3'x_3 + q_2'q_3x_1 + q_2q_3x_2)'$$

$$\bar{q}_2 = (q_1'q_3x_2 + q_1q_3'x_1 + q_2)\cdot(q_3'x_3 + q_1'x_3)'$$

$$\bar{q}_3 = (q_1'q_2'x_1 + q_1q_2x_1 + q_3)\cdot(q_1'q_2x_3 + q_1q_2'x_2)'$$

In this particular case, if shunt-down operation[6] is not objectionable, it is even possible to dispense with primary relays.* A circuit that satisfies the stated conditions is given in Fig. 15. (The author does not guarantee that the circuit is minimal!)

6. DISCUSSION

Like any "systematic" method for synthesizing certain classes of switching circuits, our method leaves much to be desired. First, the problem of synthesizing really large circuits has not been touched — one wonders whether it is really possible to do this with any method that

TABLE XII

Input Combination....	000	100	010	001	(All)
Present State	Next State				Output Combination
000	000	001	100	000	1
001	001	001	011	101	1
011	011	111	011	010	1
010	d	d	d	000	1
100	100	110	100	000	0
101	101	001	100	101	0
111	111	111	011	101	0
110	d	111	d	d	0

relies on the use of a truth-table without making use of automatic design aids inasmuch as large truth-tables become unmanageable. Second, the first step of the process, as described in Section 4, has in no sense been eliminated — this is probably the step that asks the most of the designer's ingenuity and skill. Third, the process of coding the states may have a great effect on the final cost of the circuit — despite this, there are at present no rules for carrying out the coding in an optimal manner.

To compare our method with that of Huffman,[2] several pertinent comments may be made. First, our method applies equally well to synchron-

* This was pointed out to the writer by A. H. Budlong.

ous and asynchronous circuit synthesis whereas Huffman's method was formulated specifically for asynchronous circuit synthesis. We hasten to add, however, that the basic concepts of Huffman's paper are valid in both cases. Such changes in detail as are required to adapt his method to synchronous circuit synthesis would almost certainly result in a method identical with the method of this paper. Second, for asynchronous circuits, the initial truth table we write down is different only in appearance from the initial flow table that we might have written — neither method offers any advantage in this respect. Third, the ease of using Huffman's merging rule as opposed to the use of Rule III must be weighed against the necessity of translating the final flow table into a truth table in order to develop circuit equations. Finally, the present

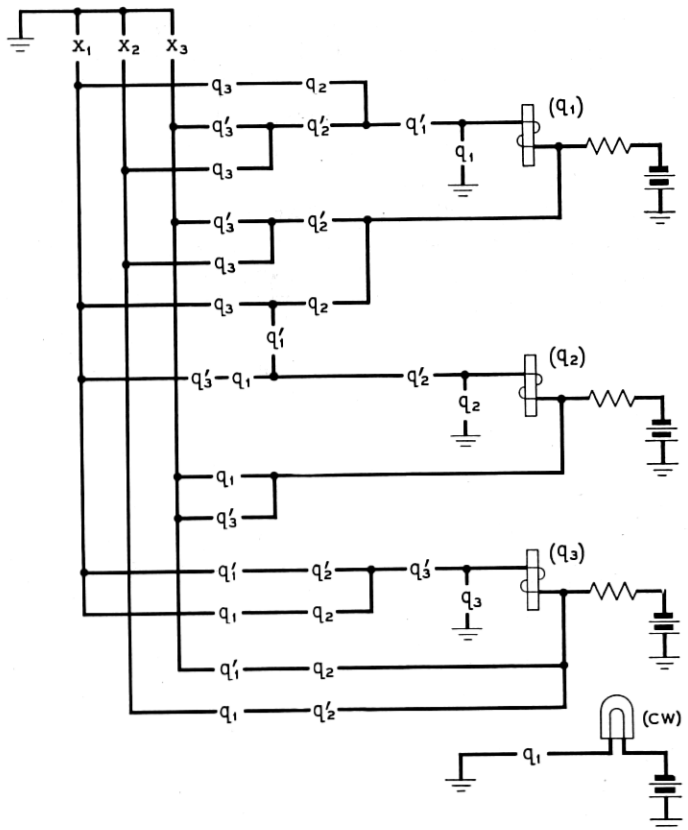

Fig. 15

method is more often successful (in principle, at least) in achieving complete reduction without the use of auxiliary equivalence arguments. Nevertheless, it is always advisable to use Rule II in order to test for complete reduction.

Finally, it should be pointed out that there are many cases where other, more intuitive, methods are more useful. Such methods for asynchronous circuit design are given in Reference 6.

In fact, the place of formal methods, such as that outlined in this paper, in the every day practice of synthesis is much smaller than might appear at first glance. It is probably fair to say that the theory furnishes, at present at least, only generalized methods of attack on synthesis together with a small handful of particularized tools for design. It is the author's belief that these methods are genuinely useful insofar as they aid in understanding the nature of sequential circuits and furnish a unified way of thinking about circuits during their design. It would be a mistake, however, to believe that they provide detailed design methods in the same sense in which such methods are available for electrical network synthesis. The engineer must make a judicious selection of his design tools and, most likely, must invent methods and diagrammatic devices which fit the particular problem at hand.

A few words should be said about the comparative originality of the author's treatment of this subject. The model proposed in Section 2 was suggested to the writer by the content of E. F. Moore, Reference 7, and, in the case of synchronous circuits, is almost identical with the discrete transducer of information theory.[8] Independently, S. H. Washburn proposed essentially the same model in an unpublished memorandum.

Our interpretation of the model for asynchronous circuits and consequences of that interpretation with relation to race conditions were independently treated by Huffman.[2] Our use of Rule III in the method owes much to Huffman's work.

7. ACKNOWLEDGEMENTS

The author gratefully acknowledges the constructive criticisms of many of his colleagues at Bell Telephone Laboratories, Inc. during the course of the work reported in this paper. He owes particular thanks to W. J. Cadden, E. F. Moore, and S. H. Washburn of Bell Telephone Laboratories, Inc. and D. A. Huffman of the Massachusetts Institute of Technology for their participation in many discussions, philosophical and otherwise, which have greatly aided the writer in clarifying his thoughts on this subject and which have resulted, it is hoped, in a far better presentation than would otherwise have been possible.

8. SELECTED BIBLIOGRAPHY

1. Karnaugh, M., The Map Method for Synthesis of Combinational Logic Circuits, Comm. and Electronics, No. 9, 1953.
2. Huffman, D. A., The Synthesis of Sequential Switching Circuits, J. Franklin Inst., **257**, pp. 161–190, 275–303, March and April, 1954.
3. Moore, E. F., Gedanken - Experiments on Sequential Machines, to be published in Automata Studies, Princeton University Press.
4. Leiner, A. L., Notz, W. A., Smith, J. L., and Weinberger, A., System Design of the SEAC and DYSEAC, Trans. I.R.E. Professional Group on Electronic Computers, Vol. EC-3, No. 2, June, 1954, pp. 8–22.
5. Felker, J. H., Typical Block Diagrams for a Transistor Digital Computer, Trans. A.I.E.E., **17-I**, pp. 175–182, 1952.
6. Keister, W., Ritchie, A. E., and Washburn, S. H., The Design of Switching Circuits, D. Van Nostrand, 1951.
7. Moore, E. F., A Simplified Universal Turing Machine, Proc. Assoc. for Computing Machinery, (Toronto meeting), 1952.
8. Shannon, C. E., A Mathematical Theory of Communication, B.S.T.J., **27**, July and Oct., pp. 279–423, 623–656, 1948.
9. Nelson, E. C., An Algebraic Theory for Use in Digital Computer Design, Trans. I.R.E. Professional Group on Electronic Computers, Vol. EC-3, No. 3, pp. 12–21, Sept., 1954.
10. Burks, A. W. and Wright, J. B., Theory of Logical Nets, Proc. I.R.E., **41**, pp. 1357–1365, Oct., 1953.
11. Murray, F. J., Mechanisms and Robots, J. Assoc. for Computing Machinery, **2**, pp. 61–82, Apr., 1955.