Chapter 4

ASYNCHRONOUS SEQUENTIAL CIRCUITS

4.1 INTRODUCTION

Sequential circuits may be classified as synchronous or asynchronous. In the former, circuit inputs are allowed to change only during periods when the clock pulses essentially disable the circuit and prevent it from changing states. These clock pulses also mask the effects of delays associated with gates and lines. Hence gates and lines can be assumed to have zero delay and the outputs and states of the circuit are of interest only at fixed instants of time.

The aforementioned restrictions simplify the analysis and synthesis of synchronous circuits considerably. However, under certain conditions, the restrictions required for synchronous operation are not satisfied. For example, consider two synchronous circuits $A$ and $B$ operating under the control of independent clocks. Suppose we wish to design the circuits so that an interrupt signal generated by the circuit $A$ will cause $B$ to stop operation. Upon receiving this interrupt, $B$ should not lose any information about its internal state prior to the occurrence of the interrupt signal. Clearly, we cannot assume any correlation between the arrival of the interrupt signal and the clock pulses of $B$. The circuit that processes the interrupt signal should be designed without taking advantage of any clock pulses.

Circuits which are designed to operate without synchronizing clock pulses are called asynchronous circuits. Such circuits are used in applications in which inputs occur at random times. They are also used as interfaces to control interactions between two synchronous circuits operating at different speeds such as a digital computer and a peripheral unit. In addition it is sometimes possible to design asynchronous circuits
which are faster than synchronous circuits that perform the same function.

Analysis and synthesis techniques used for asynchronous circuits may sometimes be applicable to synchronous circuits. For synchronous circuits with clock pulse widths comparable to gate delays, it may be necessary to analyze them as asynchronous circuits in order to resolve some timing problems. In fact, synchronous circuits may be considered to be a special case of asynchronous circuits with certain constraints on input changes, which lead to simpler analysis and synthesis procedures. Methods of analyzing asynchronous circuits are also useful in analyzing some faults in synchronous circuits since a fault in a synchronous circuit may have the same effect as the clock assuming a constant value of one or zero, thereby causing it to operate as an asynchronous circuit.

We shall refer to the mathematical model of an asynchronous sequential circuit as an asynchronous sequential machine. A convenient means of representing an asynchronous sequential machine is a flow table. As in the state table of a synchronous machine, the flow table has a row corresponding to each internal state, and a column corresponding to each input combination (also called input state). A combination of an input state and an internal state is called a total state. For every total state, the flow table specifies the next internal state and the output.

We denote the next state and the output for the total state \((q_i, I_f)\) by \(N(q_i, I_f)\) and \(Z(q_i, I_f)\) respectively. If \(N(q_i, I_f) = q_j, q_i\) is said to be a stable state under the input \(I_f\) and is circled in the flow table, as shown in Figure 4.1. All other states are unstable.

Unlike synchronous circuits which are generally assumed to operate with pulse inputs, asynchronous sequential circuits are usually assumed to operate with level inputs (although this assumption is not necessary). Since there are no clock pulses, state transitions are caused by input changes. If a transition from stable configuration \((q_i, I_m)\) caused by changing \(I_m\) to \(I_f\) results in state \(q_j\) and \(Z(q_i, I_m)\) is not stable, the state changes again to \(N(q_j, I_f)\) and continues to change within column \(I_f\) until a stable configuration is reached. The interval between successive input changes is assumed to be sufficient for the circuit to complete its response to the previous input change and reach a stable state. This mode of operation is called fundamental mode [15].

Another common assumption is that only one input variable is allowed to change at a time, since inputs which change together cannot be presumed to change exactly simultaneously, due to delays in circuit gates. We shall show later in this chapter how this assumption can be relaxed.

In the machine specified by the flow table of Figure 4.1, transitions are only allowed between columns differing in a single input variable, under the single-input change restriction. If the circuit is initially stable in state \(1\) with inputs \(00\), a change to input \(01\) will cause a transition to state \(3\), as shown by the solid directed lines in the flow table. If this input is followed by input \(11\), the machine will reach the stable state \(1\), as shown by the dotted lines. The next state and the output are not specified for the total state \((2,11)\), because it cannot be reached from any state under the single-input change restriction. The machine will be stable in state \(2\) only if the input state is \(00\), and the only input states that may follow \(00\) are \(01\) and \(10\). Furthermore, the \(11\) column of the flow table does not contain state \(2\) as a next state entry for any state.

Asynchronous flow tables are commonly classified according to the maximum number of output changes produced by a single input change. An important class of flow tables is characterized by the property that any input change produces at most one output change. Such tables are called single output change (SOC) flow tables. Figure 4.2 shows

![Figure 4.1 An asynchronous flow table](image)

![Figure 4.2 (a) An SOC flow table (b) a normal mode flow table](image)
two SOC tables that differ in only one entry (indicated by an asterisk). In the flow table $M$, an input change $0 \rightarrow 1$ with the circuit initially in state 1 causes a transition to state 2 and then to state 3. The output changes from 00 to 01 when the machine reaches state 2. In the flow table $M'$, the transition from state 1 due to the input change $0 \rightarrow 1$ is directly to state 3. The output changes to 01 as soon as the input changes. The only difference between the external behavior of the two tables is in the duration of certain outputs. Flow tables $M$ and $M'$ may be considered to be equivalent if the duration of outputs is of no consequence, since they produce the same output sequence for all input sequences.

An SOC table in which every transition leads directly to a stable state (as in $M'$ of Figure 4.2(b)) is called a normal mode flow table. Any SOC flow table can be converted to a normal mode flow table if the durations of the outputs are of no interest.

Flow tables in which a single input change may produce a sequence of output changes are called multiple output change (MOC) flow tables. The following are examples of MOC flow tables:

\[
\begin{array}{c|c|c|c|c}
| x | & 00 & 01 & 11 & 10 \\ 
|---|---|---|---|---|
| 0 | I_0 & I_1 & I_3 & I_2 \\
| 1 | 2.10 & 1.11 & 1.00 & 2.10 & 1.11 & - \\
| 2 | 3.00 & 1.11 & 2.10 & 3.00 & - \\
| 3 | 3.01 & 2.01 & 4.01 & 4.01 & - \\
| 4 | 1.00 & 4.01 & 2.10 & - \\
\end{array}
\]

**Figure 4.3** (a) An MOC flow table (b) a non-fundamental mode flow table

In the flow table $M_1$, an input change $1 \rightarrow 0$ when the machine is stable in state 1 will produce the output sequence $11 \rightarrow 10 \rightarrow 00 \rightarrow 01$. Similarly, the input transition $0 \rightarrow 1$ when the machine is in stable state 3 will produce the output sequence $01 \rightarrow 10 \rightarrow 11$. The machine represented by the flow table $M_2$ will produce single-output changes for input transitions between $I_0$ and $I_1$. However, if the input $I_1$ is applied when the machine is stable in the total state $(2, I_1)$ or $(4, I_1)$, the machine will not reach a stable state but will cycle through states 2, 3, and 4. The number of output changes produced and the final state reached will depend on the duration of the input $I_1$. Unlike the MOC table $M_1$, fundamental mode operation is impossible in this case. Flow tables such as $M_2$ where fundamental mode operation is impossible are called nonfundamental mode flow tables. They are characterized by the presence of state cycles in one or more columns. All normal mode tables are fundamental mode but not all fundamental mode tables are normal mode. SOC tables may be normal or non-normal and fundamental mode or non-fundamental mode. In a non-fundamental mode SOC table, the output associated with all total states through which it may be cycle must be the same.

The inputs to asynchronous sequential circuits need not be restricted to the level type. Asynchronous circuits can be designed to operate with input pulses whose widths are within specified bounds. The interval between pulses may vary, but should be sufficient to allow the circuit to reach a stable state. Such circuits are called pulse input asynchronous circuits.

Two types of pulse input asynchronous circuits may be defined, depending on the restrictions on the input pulse width. If the widths of input pulses are such that the circuit can change state only once, then the behavior of the circuit will be similar to a synchronous circuit in which the clock pulses do not occur at equal intervals. These circuits can be described by state tables as in Chapter 3 and may be analyzed using similar techniques.

If the width of the input pulse is not restricted so that the circuit can change state only once during each input pulse, the circuit can be analyzed using methods applicable to level input circuits. Figure 4.4 shows the flow table of the same machine as Figure 4.1, but with pulse inputs. The columns corresponding to the four columns of Figure 4.4. A pulse mode flow table
4.1 have been labeled \( I_i \) through \( I_k \) and a column \( I_0 \) has been added. Each column \( I_j, j \neq 0 \), represents a pulse input, and \( I_0 \) represents the absence of any pulse. Every internal state in the column \( I_i \) is stable. The behavior of the circuit when the input changes from a pulse \( I_j \) to a pulse \( I_k \) can be analyzed by considering the transitions from the column \( I_i \) to \( I_0 \) and then from \( I_0 \) to \( I_k \). Transitions corresponding to the two transitions shown in Figure 4.1 are shown in Figure 4.4.

A pulse input sequential circuit may be designed to produce pulse or level outputs. In Figure 4.4, the outputs in column \( I_0 \) are required to be unchanged from their previous values, each such output being indicated in the table by the symbol \( S \). The outputs can be maintained at their previous values by taking them from flip-flops which are neither set nor reset when the input is \( I_0 \). Thus, the machine of Figure 4.4 will produce level outputs in response to pulse inputs.

### 4.2 FLOW TABLE SPECIFICATION

The first step in the design of an asynchronous sequential circuit is the construction of a flow table describing the circuit behavior from a word description of the function to be realized. In doing this, it is often relatively easy to first construct a primitive flow table which has exactly one stable state in each row.

**Example 4.1** Consider an asynchronous counter with two binary inputs \( x_1, x_2 \). The input \( x_1 = x_2 = 0 \) is a reset input which clears the counter to an initial state producing the output 00. Only a single input variable is allowed to change in a transition. The count is to be incremented by one when the input \( x_1, x_2 = 01 \) is received and incremented by two when the input: \( x_1, x_2 = 10 \) is received, the count being modulo 4.

The input \( x_1, x_2 = 11 \) does not produce any change in the count. The outputs \( z_1, z_2 \) represent the current count.

Figure 4.5 is a primitive flow table for the asynchronous counter described above. For each row \( q_i \) of the table, next state entries are specified for input states that may follow the input state containing the stable state in row \( q_i \). The reset state, 1, is stable in column 00. This input may be followed by 01 which results in a next state entry of 2 and a count of 1 (\( z_1, z_2 = 01 \)) or by 10, resulting in a next state entry 3 and a count of 2 (\( z_1, z_2 = 10 \)). The entry in row 1, column 11 is unspecified. The stable state 2 in column 01 may be followed by the inputs 00 or 11, resulting in a reset to state 1 or a transition to state 4 with no change in count respectively. The remaining entries are similarly specified. Outputs are specified only for stable states, assuming that the outputs may change at any time during a transition. □

A flow table generated from a word description of a sequential function, such as the primitive flow table of Example 4.1, usually contains more internal states than necessary to realize the specified function. For more economical realizations, it may be desirable to reduce the number of states in a flow table. A primitive flow table is particularly useful in deriving an equivalent table with the minimum number of states.

### 4.3 FLOW TABLE REDUCTION

The methods of state table reduction discussed in Chapter 3 are also applicable to asynchronous flow tables with only minor modifications.
The single-input change restriction often makes it possible to reduce some flow tables that are not otherwise reducible. This restriction can be easily modelled by unspecified entries in a primitive flow table. However, if a flow table has two or more stable states in a row, the single input change constraint cannot be accurately modelled by unspecified entries, since a particular input may be permitted from one of the columns with a stable state within the row but not from another. Therefore, the reduced version of a primitive table may require fewer states than that of the corresponding non-primitive table if only single input changes are permitted. Consider the flow table of Figure 4.6(a) and the corresponding primitive table of Figure 4.6(b), assuming single input variable changes. The table of (a) is not reducible while the primitive table of (b) can be reduced to the 3-state table of Figure 4.6(c). Of course any normal mode SOC table can be expanded to a primitive flow table as an initial step in the state minimization procedure.

With fundamental mode operation, input changes occur only when the circuit is in a stable state. Furthermore, the exact times at which output changes occur and the duration of output signals are usually not of interest, provided they are within certain bounds. Instead, we are interested only in the output sequences generated by input sequences. These facts sometimes enable additional types of state mergers for non-normal mode tables which are not possible in the synchronous case.

In an SOC flow table, the output associated with an unstable state must be the same as the output associated with the stable state at the beginning or the end of the transition. If we assign the same output to these unstable states as to that of the final stable state, the only effect of this change will be in the timing of the output changes. If we are not concerned with this effect, the flow table so obtained may be considered to be equivalent to the original table. Also, if a state is unstable in a particular column, its next state and output can be made the same as the stable state in that column that will be ultimately reached from the unstable state. That is, if \( q_i \) is unstable in column \( L_i \), and \( q_j \) is the stable state that will be reached from \( q_i \), we make \( N(q_i,L_i) = q_j \) and \( Z(q_i,L_i) = Z(q_j,L_i) \). This procedure, which we shall refer to as normalization converts an SOC table into a normal mode flow table.

**Lemma 4.1** If a pair of states \( q_i \) and \( q_j \) of an SOC flow table \( M \) are compatible, then \( q_i \) and \( q_j \) are compatible in the normal mode flow table \( M' \) obtained by normalizing \( M \).

![Flow Table Reduction](image)

Figure 4.6 Flow table reduction
Proof: Let the stable states reached from \( q_i \) and \( q_j \) under input \( I_k \) be \( q_m \) and \( q_n \) respectively. Since \( q_i \) and \( q_j \) are compatible in \( M \), \( q_m \) and \( q_n \) must be compatible. In \( M' \), \( N(q_m, I_k) = q_m \) and \( N(q_n, I_k) = q_n \). Therefore, \( q_i \) and \( q_j \) will be compatible in \( M' \) if \( q_i \) and \( q_j \) are output compatible. By our construction of \( M' \), \( Z(q_i, I_k) = Z(q_m, I_k) \) and \( Z(q_j, I_k) = Z(q_n, I_k) \). But \( Z(q_m, I_k) = Z(q_n, I_k) \) since \( q_m \) and \( q_n \) are compatible. Therefore \( Z(q_i, I_k) = Z(q_j, I_k) \) and \( q_i \) and \( q_j \) are compatible in \( M' \). □

Normalizing an SOC table does not destroy any compatible pairs but may make some new pairs compatible. The state table reduction method discussed in Chapter 3 can be applied to the normal mode flow table so obtained. It follows from Lemma 4.1 that the flow table obtained by minimizing the number of states in the normalized flow table will not exceed the number of states obtained by reducing the original table.

Example 4.2 The flow table shown in Figure 4.7(a) has no compatible pair of states. Since this table has only single-output changes, it can be converted to the normal mode flow table of Figure 4.7(b) by changing the entries indicated by asterisks. Now (1,3) and (2,4) are compatible pairs which form a closed set and may be merged to obtain the reduced table of Figure 4.7(c). This reduction would have been impossible had the original table represented a synchronous machine. □

The method discussed above has to be modified somewhat for MOC flow tables. These tables may be interpreted in two ways. The duration of each output state may be assumed to be proportional to the number of successive states in the transition during which the particular output state is generated. This implies that the output is time-dependent. Alternatively, we may be interested only in the order of output changes but not the duration of the outputs.

For fundamental mode MOC sequential machines, we assume that the machine always starts in a stable state and that input changes occur only when the machine is in a stable state. As a result, for any unstable total state, only the next state and output entries in one input column need be considered during any transition. This may cause pairs of states that are not compatible under the synchronous assumption to become compatible. For example, consider the flow table segment shown in Figure 4.8. If the flow table is interpreted as that of a synchronous machine, states 1 and 5 are not compatible since the states 2 and 4 implied by them are not output compatible. However, assuming

\[
\begin{array}{c|c|c|c}
00 & 01 & 11 & 10 \\
\hline
1 & 1,0 & *2,0 & *2,0 & 1,0 \\
2 & 4,1 & 3,0 & 2,1 & 2,1 \\
3 & 1,0 & 3,0 & 4,1 & - \\
4 & 4,1 & *3,1 & 4,1 & 4,1 \\
\end{array}
\]

\[\text{(a)}\]

\[
\begin{array}{c|c|c|c|c}
00 & 01 & 11 & 10 \\
\hline
1 & 1,0 & 3,0 & 2,1 & 1,0 \\
2 & 4,1 & 3,0 & 2,1 & 2,1 \\
3 & 1,0 & 3,0 & 4,1 & - \\
4 & 4,1 & 3,0 & 4,1 & 4,1 \\
\end{array}
\]

\[\text{(b)}\]

\[
\begin{array}{c|c|c|c|c}
00 & 01 & 11 & 10 \\
\hline
1 & 1,0 & 1,0 & 2,1 & 1,0 \\
2 & 2,1 & 1,0 & 2,1 & 2,1 \\
\end{array}
\]

\[\text{(c)}\]

\[
\text{Figure 4.7 Reduction of a normalized flow table}
\]

asynchronous fundamental mode operation, states 1 and 5 are compatible, since they are output compatible and both states lead to the stable state 3 with the output sequence 101, if the input \( I_2 \) is applied.
Asynchronous Sequential Circuits

\[
\begin{array}{|c|c||c|}
\hline
 & I_1 & I_2 \\
\hline
1 & 1,0 & 2,1 \\
2 & 2,1 & 3,0 \\
3 & 3,0 & 3,1 \\
4 & 4,0 & 3,0 \\
5 & 5,0 & 4,1 \\
\hline
\end{array}
\]

Figure 4.8 A flow table segment

The effect of the input restriction implied by fundamental mode operation can be represented in the flow table by augmenting it as follows. For each unstable total state \((q_i, I_j)\), we add a row to the flow table, labeled \(q_k\). Wherever \(q_i\) appears as a next state entry in column \(I_j\), we replace it with \(q_k\). The next state and output of \((q_k, I_j)\) is specified to be the same as that of \((q_i, I_j)\) in the original flow table, i.e., \(N(q_k, I_j) = N(q_i, I_j)\) and \(Z(q_k, I_j) = Z(q_i, I_j)\). The next state and output entries are unspecified in all other columns of the row \(q_k\).

Example 4.3 Consider the flow table of Figure 4.9(a), which is to be reduced under the time-dependent assumption. The given flow table does not contain any compatible pair of states. The table is expanded to that in (b) by adding the states 6, 7 and 8 corresponding to total states \((2,01)\), \((2,11)\) and \((2,00)\) respectively. The expanded table can be reduced to the flow table (c), which is equivalent to (a) under the time-dependent assumption. Assuming that the duration of a particular output state is proportional to the number of consecutive internal states during which the output is produced, it can be verified that when multiple output changes are produced, the relative magnitudes of the duration of the changes are also the same for both of these flow tables. Note that the transient state 2 of table (a) is replaced by states 6, 7 and 8, and state 2 itself can be eliminated and hence is not covered by any state in the reduced table. □

If the time-independent interpretation is used, the definition of compatibility has to be modified so that the duration and instants of occurrence of outputs are ignored. For example, the sequences 010, 00110, and 000100 should be considered to be essentially equivalent. Two states, \(q_i\) and \(q_j\), are said to be B-compatible if for every input \(I_k\), the output sequence generated by the machine in state \(q_i\) is essentially equivalent to the output sequence generated by the machine in state \(q_j\) or can be made essentially equivalent by suitably specifying the unspecified

\[
\begin{array}{|c|c|c|c|c|}
\hline
 & 00 & 01 & 11 & 10 \\
\hline
1 & 1,0 & 2,1 & 2,0 & 1,0 \\
2 & 1,0 & 3,1 & 4,1 & - \\
3 & 2,0 & 3,0 & 3,0 & - \\
4 & 1,0 & 5,0 & 4,1 & 4,1 \\
5 & 2,0 & 5,0 & 4,1 & - \\
\hline
\end{array}
\]

(a)

\[
\begin{array}{|c|c|c|c|c|}
\hline
 & 00 & 01 & 11 & 10 \\
\hline
1 & 1,0 & 6,1 & 7,0 & 1,0 \\
2 & - & - & - & - \\
3 & 8,0 & 3,0 & 3,0 & - \\
4 & 1,0 & 5,0 & 4,1 & 4,1 \\
5 & 8,0 & 5,0 & 4,1 & - \\
\hline
(2,01) & 6 & - & 3,1 & - \\
(2,11) & 7 & - & 4,1 & - \\
(2,00) & 8 & 1,0 & - & - \\
\hline
\end{array}
\]

(b)

Figure 4.9 Time-dependent flow table reduction
outputs), and the stable states reached in both cases are also B-compatible.
The procedure for reducing an MOC flow table under the time-independent assumption consists of expanding the table as in the time-dependent case and using B-compatibility in reducing the expanded table. However, the procedure may not always lead to the minimal solution with the time-independent interpretation. (See Problem 4.13.)

Example 4.4 In the expanded flow table of Figure 4.9(b), states 1 and 6 are B-compatible, in addition to the states that were found to be compatible in Example 4.3. Although they have different next state entries in the column 01, they both reach the same stable state, namely 3, producing the output sequence 10. Since the final state is the same for both states there are no implied sets of states. Therefore the flow table of Figure 4.9(a) can be reduced to the 3-state table of Figure 4.10 under the time-independent assumption.

4.4 STATE ASSIGNMENT

Figure 4.11 shows a general model of asynchronous sequential circuits [10] which we shall use for most of our discussions in this chapter. This is the same model as for synchronous circuits except for the absence of a clock. In synchronous circuits clocked D-FF’s can be modelled as delay elements of identical magnitude. In asynchronous circuits delay elements may be used in feedback loops but they cannot be assumed to be of identical magnitude. The internal state of the circuit is represented by the state variables $y_i$ and the next state values of the state variables are denoted by $Y_i$, $i = 1, 2, ..., n$. The variables $x_1, ..., x_m$ and $z_1, ..., z_k$ represent the inputs and outputs respectively of the circuit.

![Figure 4.11 Asynchronous sequential circuit model](image)

The first step in designing an asynchronous sequential circuit that realizes a given flow table is to represent the internal states by combinations of values of binary state variables. This is called the asynchronous state assignment problem and differs from its synchronous counterpart in several respects.

For a synchronous machine, $[\log_2 n] = S_0$, state variables are necessary and sufficient for representing $n$ states, and the assignment can be made arbitrarily without affecting the proper operation, as long as no coding is assigned to more than one state, although the structural complexity of the realization will usually depend on the assignment. For an asynchronous flow table additional constraints must be satisfied and hence $S_0$ state variables may not be sufficient.

Consider the flow table,* state assignment $A$, and realization shown

\[\begin{array}{cccc}
00 & 01 & 11 & 10 \\
(1,6) & 1 & 1.0 & 2.1 & 3.0 & 1.0 \\
(3,8) & 2 & 1.0 & 2.0 & 3.0 & - \\
(4,5,7) & 3 & 1.0 & 3.0 & 3.1 & 3.1 \\
\end{array}\]

Figure 4.10 Time-independent reduced flow table

*Note that output entries have been omitted from the flow table. In the rest of this chapter, output entries will be omitted from flow tables whenever we are interested only in their state behavior. We shall assume that such flow tables are reduced.
in Figure 4.12. Let the circuit be stable in state 2 (represented by \( y_1 = 0, y_2 = 1 \)) and input \( x = 1 \). For the transition when \( x \) changes value from 1 to 0, \( G_2 \) will begin to change to 1 and both \( Y_1 \) and \( Y_2 \) will begin to change value. If \( y_2 \) changes to 0 before \( y_1 \) changes to 1, \( G_2 \) will change to 0 and the variable \( y_1 \) may oscillate, or even stabilize at \( y_1 = 0 \) instead of \( y_1 = 1 \) if the delay associated with \( G_4 \) is sufficiently large. Similar problems occur if \( y_1 \) changes value first. In effect the circuit may pass through state 1, represented by \( y_1 = y_2 = 0 \), if \( y_2 \) changes first, or state 4, represented by \( y_1 = y_2 = 1 \), if \( y_1 \) changes first. However, both of these states lead to state 1 in column 0, of the flow table rather than the correct next state 3, which will be reached if \( y_1 \) and \( y_2 \) change simultaneously. Thus, the final state reached will depend on the relative magnitudes of stray delays in the circuit. In practice, the stable state may be indeterminate or the circuit may oscillate due to similar problems. This difficulty does not occur in synchronous circuits because the clock prevents the state variable values from being used to generate excitations until they become stable, thus effectively ensuring that they appear to change simultaneously.

When two or more variables change during a transition, a race is said to occur among the changing variables. In the flow table of Figure 4.12, the state variables \( y_1 \) and \( y_2 \) of assignment \( A \) are involved in a race in the transition from state 2 to state 3 and also the transition from state 4 to state 1. If the final state reached may depend on the order of changes, as in this case, the race is said to be critical. Otherwise it is called a noncritical race.

In assignment \( B \) for the same flow table, all transitions are between states that differ in exactly one state variable (adjacent states). This assignment is race-free and is therefore suitable for the asynchronous flow table. A race-free assignment is sufficient but not necessary for proper operation, as we shall see later.

The combination of state variable values used to represent a state is called the coding of the state. The assignments considered so far used only one coding per state and are called unicode assignments. It is not always possible to derive a unicode state assignment so that all transitions occur between adjacent codings.

Consider the flow table of Figure 4.13(a). The undirected graph of Figure 4.13(b) has nodes corresponding to the states of the flow table. It has an edge between two nodes if and only if there is a direct transition between the two states. This graph, which we shall call the adjacency graph of the flow table, shows states whose codings should be adjacent. From the adjacency graph, we see that the codings of states 2, 4, and 5 must be adjacent to one another as must the codings of states 4, 5, and 6, in order to be able to make all state transitions by changing only one state variable. However, it is impossible to have three binary codings such that each pair of these codings differs in exactly 1 bit. It follows that there exists no unicode state assignment for the flow table of Figure 4.13(a) such that all transitions are between adjacent states.

Different methods could be used to resolve the difficulty encountered in the above example. One method is to permit transitions between
nonadjacent states. When two or more variables are required to change during a transition, these variables may change simultaneously if there are no critical races, or they may change in some prescribed order. Another method involves the use of more than one coding per state. In this case, it is possible to derive state assignments such that state transitions occur only between adjacent states. More efficient multicode assignments (using more than one coding per state) can be obtained if noncritical races are permitted.

4.4.1 Connected Row Set Assignments

One method of obtaining an assignment such that any transition can be made by a sequence of single variable changes is to assign a set of codings \( R_i \) to each row (state) \( q_i \) of the flow table. The set of codings (points) assigned to a row of the flow table is called a row set. If any member in a row set can be reached from any other member in the same set by a sequence of single variable changes without passing through any point that is not in the set, the set of codings is called a connected row set. Two connected row sets \( R_i \) and \( R_j \) are said to be intermeshed if there is some coding in \( R_i \) that is adjacent to some coding in \( R_j \). In order to use a connected row set assignment for a flow table, the row sets assigned to states between which there are transitions must be intermeshed. Let \( R_k \) and \( R_l \) be intermeshed connected row sets assigned to states \( q_k \) and \( q_l \) respectively. A transition from \( q_k \) to \( q_l \) is made by first making a sequence of transitions within \( R_k \) until a point adjacent to some point in \( R_l \) is reached, and then making the transition to the point in \( R_l \). The intermeshed nature of \( R_k \) and \( R_l \) guarantees that this can always be done.

Example 4.5 A connected row set assignment for the flow table of Figure 4.13 is shown in the Karnaugh map and table of Figure 4.14. The arrows indicate how two transitions are made. If the circuit is stable in state 6, represented by the coding 1010 in Figure 4.14(a), and the input changes to 11, the first step of the transition will be to 1011 (another coding for state 6) by changing \( y_4 \). Then \( y_3 \) will change to 0 to reach state 4 represented by 1001. Now if the input changes to 01, state 5 represented by 1111 is reached via 1101, which is in the row set assigned to state 4. This transition is shown by the dotted arrows in Figure 4.14.

With the assignment shown in Figure 4.14, any state transition in the flow table of Figure 4.13 can be made in at most two steps. The maximum number of steps required for any state transition using a particular state assignment is referred to as the transition time of the assignment. Thus, the assignment of Example 4.5 has a transition time of 2. The transition time of the state assignment used affects the minimum period between input changes and the speed of operation of the sequential circuit (assuming fundamental mode operation).
The derivation of a connected row set assignment with the minimum number of variables involves essentially an exhaustive search. The following example shows how a state assignment can be obtained for a given flow table.

**Example 4.6** Figure 4.15(a),(b) show a flow table and the adjacencies to be satisfied by the different row sets. Since there are six states, at least three variables are necessary in any state assignment. First, we assign an arbitrary coding, say 000, to state 2 which has the largest number of adjacencies. Any point in a k-cube has only k points adjacent to it. Since four adjacencies are needed for state 2, a single 3-variable coding is not sufficient for state 2. If we assign a second coding adjacent to the first coding, then the two codings together will have four adjacent points which can be assigned to states 1, 3, 4 and 6. One possible assignment for state 2 and its four adjacent states is shown in Figure 4.15(c). Now, state 5 has to be adjacent to states 1, 3, and 4, requiring the assignment of the coding 110 to this state and the coding 111 to 3. Thus a connected row set has been formed for state 3. Examining the flow table, we note that all required adjacencies have been satisfied. The complete state assignment is shown in Figure 4.15(d). Since the assignment required only three state variables, it is an assignment with the minimum number of variables. The search process does not always function so smoothly. Figure 4.15(e) shows another assignment for state 2 and its four adjacent states. Since state 5 has to be adjacent to states 1, 3 and 4, both the unused codings have to be assigned to state 5. However, the requirement that state 3 be adjacent to states 2 and 6 cannot be satisfied, and we are unable to obtain a connected row set assignment with three variables. Another valid connected row set assignment is shown in Figure 4.15(f).

It is sometimes possible to modify the flow table so as to reduce the number of adjacency constraints to be satisfied and possibly the number of state variables required in a connected row set assignment. Consider an SOC flow table in which $N(q_s I_j) = N(q, I_j) = q_m$ where $q_m$ is not equal to $q_s$ or $q_k$. If we modify the table by making $N(q_s I_j) = q_s$, leaving $N(q, I_j) = q_m$, there will be no change in the behavior of the machine, except that the transition $q_i \rightarrow q_m$ in the column $I_j$ is replaced by $q_i \rightarrow q_s \rightarrow q_m$. Alternatively, we could make $N(q_s I_j) = q_s$ and $N(q, I_j) = q_m$. This modification is the opposite of normalization discussed in Section 4.1 and may result in a state assignment requiring fewer variables.

**Example 4.7** In the flow table of Figure 4.16(a), the column 00 has transitions 4 \(\rightarrow\) 1 and 3 \(\rightarrow\) 1. The 2-variable state assignment shown in the Karnaugh map of Figure 4.16(b) is valid for this table, since
the transition $4 \rightarrow 1$ can be changed to $4 \rightarrow 3 \rightarrow 1$, as discussed above. Effectively, we conclude from column 00 that $(1,3)$ and $(1,4)$ or $(1,3)$ and $(3,4)$ or $(1,4)$ and $(3,4)$ must be adjacent pairs instead of only $(1,3)$ and $(1,4)$.

For any given number $n$ of states, a state assignment that is valid for any $n$-state flow table is called a universal state assignment. Such assignments can be constructed using systematic procedures, but frequently require more state variables than an assignment for a specific flow table with the same number of states. However, they provide useful upper bounds on the number of variables required for a flow table with a given number of states.

We shall first consider a class of universal assignments which require $2S_0 - 1$ state variables for an $n$-state table $[11]$, where $S_0 = \lceil \log_2 n \rceil$. The universal assignment for four states is shown in Figure 4.17. In order to show that this is indeed a universal assignment, we have to verify that the codings for each state are connected and that the row sets assigned to any pair of states are intermeshed. This is readily verified from Figure 4.17 since the two codings for each state differ only in one variable and for any pair $i,j$ of states, there is a coding for state $i$ that differs from a coding of state $j$ in only one variable.

The above universal assignment can be generalized to obtain assignments for larger numbers of states. Consider the Karnaugh map of Figure 4.18. The variables $y_1$, $y_2$ and $y_3$ divide the map into octants as shown. The map is constructed so that variables with odd-numbered

![Figure 4.17](image1.png)

- **Figure 4.17** Universal connected row set state assignment for 4 states

![Figure 4.18](image2.png)

- **Figure 4.18** Universal 8-state connected row set assignment

subscripts, $y_1$, $y_2$, $y_3$, define columns and those with even-numbered subscripts, $y_2$, $y_3$, define rows. The codings assigned to any state consist of a column in one octant and a row in an adjacent octant (defined by changing the value of $y_2$ for half of the states and the value of $y_3$ for the other half). Therefore, the codings of every state form a connected set. Furthermore, for every pair of states $q_i$ and $q_j$, there exist codings of the two states in adjacent octants, and since rows in an octant are connected to all columns in every adjacent octant (and vice versa), there exist codings for states $q_i$ and $q_j$ which differ in exactly one variable. Therefore, the assignment of Figure 4.18 is a universal 8-state connected row set assignment.

The pattern of Figure 4.18 can be used for generating universal assignments for any number of states equal to an integral power of 2. The variables $y_1$, $y_2$ and $y_3$ define octants in a Karnaugh map.
The row sets assigned to any state will correspond to rows in one octant and columns in an adjacent octant. Let us use rows in octants corresponding to an even number of ones in the variables \( y_1, y_2, y_3 \), and columns in octants corresponding to an odd number of ones in the same three variables. Note that a row is obtained by assigning a set of fixed values to \( y \)-variables whose subscripts are even and assigning all possible combinations of values to the variables with odd subscripts, \( y_1, y_2, \) and \( y_3 \) being constant. Similarly, the columns can be obtained by keeping the variables with odd-numbered subscripts constant and varying those with even-numbered subscripts. A 16-state universal assignment is shown in Figure 4.19.

![Figure 4.19 Universal 16-state assignment](image)

We have seen that three variables are used in a 4-state assignment. In order to double the number of states, we have to double the number of rows and columns. This can be accomplished by adding two variables. The number of variables required for a universal \( n \)-state assignment using this construction is therefore \( 2S_n - 1 \).

If noncritical races are permitted and utilized, at most four transition times are required for making any state transition when a \((2S_n - 1)\) universal assignment is used for any \( n \)-state table, independent of \( n \). In the worst case, the transitions involved will be (1) a transition within any column (row) in an octant (possibly involving a noncritical race), (2) changing an octant variable \((y_1, y_2, \) or \(y_3)\) to reach an adjacent octant, (3) a transition within a row (column) in the new octant (possibly involving a noncritical race again), and finally (4) a transition to a coding of the destination state by changing an octant variable. An example of a transition between the points \( a \) and \( e \) requiring four steps is shown in Figure 4.19. The transitions from \( a \) to \( b \) and from \( c \) to \( d \) involve noncritical races. The other two transitions require only single variable changes. Another universal state assignment having \( 2S_n \) state variables but requiring only two transition times has also been derived by Huffman [11]. (See Problem 4.16).

### 4.4.2. Shared Row Assignments

In connected row set assignments, transitions between different pairs of states are made using disjoint paths (intermediate states). However, if two different state transitions occur under different inputs, these transitions need not be made disjoint and may share one or more intermediate states. Such assignments, which are called **shared row assignments**, frequently require fewer state variables than connected row set assignments.

A **connected path** between two points \( a \) and \( b \) is an ordered sequence of points beginning at \( a \) and ending at \( b \), such that all consecutive pairs of points are adjacent. A valid shared row state assignment for a normal mode flow table must satisfy the following two conditions: (1) the point (or points) representing every state must be distinct from those representing all other states, and (2) there must be a connected path associated with every transition, disjoint from the paths associated with all other transitions in the same column of the flow table, except when they have the same final state.

**Example 4.8** Figure 4.20 shows a shared row assignment for the flow table of Figure 4.13. The transitions are shown by the lines connecting adjacent points, for the columns (00, 01, 11) in the flow table. The

![Figure 4.20 Transitions in each column for flow table of Figure 4.13](image)
point 111 is shared by the transitions 5 → 6 in column 00, 4 → 5 in column 01, and 6 → 4 in column 11.

As in the case of a connected row set assignment, the procedure for deriving a shared row set assignment is exhaustive in nature. The adjacency constraints used for the former are also useful for the latter, as the following example demonstrates.

**Example 4.9** Figure 4.21 shows a 7-state flow table and its adjacency constraints. We start with \( S_0 = \lfloor \log_2 7 \rfloor = 3 \) state variables. Since state 1 has the largest number of adjacency constraints, we assign a coding (000) to it first. Its four adjacencies make it necessary to assign a transition point (shared row) adjacent to it, indicated by - in Figure 4.21(c). For the same reason state 2 must be adjacent to the shared row. The other 3 points adjacent to 1 and the shared row must be filled with the required adjacencies to state 1, (3, 4, and 6). Since 3 and 4 must be adjacent to both 1 and 2 while 6 need only be adjacent to 1, these entries are placed as shown. The adjacency requirements of states 7 and 5 determine the codings assigned as shown. The transitions that use the shared row are 4 → 1 in the 00 column, 1 → 2 in the 01 column, and 2 → 4 in the 10 column. Since all the transitions that use the shared row are in different columns, the assignment is a valid shared row assignment. It is easily shown that a connected row set assignment for this flow table requires four state variables.

---

**Figure 4.21** Derivation of shared row state assignment—Example 4.9

---

**Figure 4.22** Universal shared row state assignments
Although there are no known universal shared row assignments for tables of arbitrary size, Saucier [19] has obtained such assignments with one coding per state for 8- and 12-state normal mode flow tables. These assignments requiring four and five variables respectively are shown in Figure 4.22. Both assignments were obtained and proved to be minimal by an exhaustive search algorithm using a computer.

Kashef and McGhee [12] have defined a class of codes called augmented parity check codes. These codes, which are generalizations of Saucier's assignments, require approximately $S_0 + \lceil \log_2 S_0 \rceil$ state variables for $n$ states where $S_0 = \lceil \log_2 n \rceil$. They have been conjectured to be universal state assignments, but this has not been proven.

### 4.4.3 Single Transition Time Assignments

The state assignments discussed in the preceding section generally require a sequence of state variable changes for a state transition. This may not be desirable from the point of view of the speed of operation of the circuit because input changes cannot be permitted to occur until the entire sequence of state variable changes is completed. Assignments in which all variables that must change during a transition are allowed to change simultaneously are called single transition time (STT) assignments.

One method of completing all transitions in a single transition time is to assign adjacent points to states between which transitions are to be made. As was pointed out earlier, this is impossible in general using a binary assignment (although it may be possible for some specific flow tables). However, it is possible to assign a set of codings to each state of any flow table, such that for any two states $q_i$ and $q_j$, every coding assigned to $q_i$ is adjacent to some coding assigned to $q_j$. Unlike the connected row set assignment discussed earlier, the points assigned to any state need not form a connected set since transitions are not made between points in the same row set. Such assignments are sometimes called one-shot assignments.

The Hamming code assignment is a universal STT assignment with the above properties. It is related to the Hamming single error correcting code [9]. If $n$, the number of states in the flow table, is equal to $2^k$, the universal $n$-state Hamming code assignment has $m = 2^k - 1$ variables. Since transitions should be permitted between all pairs of states and only one variable is allowed to change during any transition, such an assignment will require at least $n - 1$ variables. Thus, if $n = 2^k$, the Hamming code assignment is a universal assignment with the minimum number of variables, in which all transitions are made with single variable changes.

Let us label the $n$ states of the flow table with the integers $0, 1, \ldots, n - 1$. For each integer $i, 0 \leq i \leq n - 1$, we can represent $i$ as a $k$-bit binary number $b_{k-1}(i)b_{k-2}(i)\ldots b_0(i)$, called the binary representation of $i$. For example, if $k = 3$, then $0 \leq i \leq 7$, and the binary representation of 6 is $110$: $b_2(6) = 1, b_1(6) = 1$, and $b_0(6) = 0$.

We will assign to each state $i, 0 \leq i \leq n - 1$, a set of $(n - 1)$-bit vectors (i.e., a set of codings of $n - 1$ variables). Each of the $2^{n-1}$ vectors will be assigned to a unique state. The assignment will be made as follows: Let $P_j, j \leq k$, be the set of integers $i \leq n - 1$ whose binary representations have 1's in position $j$, where the rightmost position is position 1. For $k = 3, P_1 = \{1,3,5,7\}, P_2 = \{2,3,6,7\}$ and $P_3 = \{4,5,6,7\}$. For each vector $y$, we define $k$ parameters:

$$C_j(y) = \sum_{i \in P_j} y_i$$

where $\Sigma$ represents the modulo 2 sum and $j$ is an integer $\leq k$.

Thus, if $k = 3$,

$$C_1(y) = y_1 \oplus y_2 \oplus y_3 \oplus y_4$$
$$C_2(y) = y_2 \oplus y_3 \oplus y_4 \oplus y_5$$
$$C_3(y) = y_4 \oplus y_5 \oplus y_6 \oplus y_7$$

These $k$ parameters always take the value $0$ or $1$, and taken as a group $C_kC_{k-1}\ldots C_1$ define the binary representation of an integer $i, 0 \leq i \leq n - 1$. If the parameters $C_j(y), 1 \leq j \leq k$ define the binary representation of $I$, then the vector $y$ is assigned to state $I$. For every state $i$, the set of vectors assigned to $i$ is denoted by $R_i$ and defined as:

$$R_i = \{y | C_k(y)C_{k-1}(y)\ldots C_1(y) \text{ is the binary representation of } I\}$$

**Example 4.10** For a 4-state Hamming code assignment, $k = 2$. The number of state variables required is $4 - 1 = 3$. The set of codings (vectors) assigned to state 0 is $R_0 = \{y | C_2(y)C_1(y) = 00\}$. The vectors in $R_0$ should satisfy the conditions:

$$C_1(y) = y_1 \oplus y_2 = 0$$
$$C_2(y) = y_2 \oplus y_3 = 0$$
Thus, \( R_0 = \{000,111\} \). Similarly, \( R_1 = \{011,100\} \) satisfying
\[
C_1(y) = y_1 \oplus y_3 = 1 \\
C_2(y) = y_2 \oplus y_3 = 0;
\]
\( R_2 = \{010,101\} \), satisfying \( y_1 \oplus y_3 = 0 \), \( y_2 \oplus y_3 = 1 \); \( R_3 = \{001,110\} \), satisfying \( y_1 \oplus y_3 = 1 \), \( y_2 \oplus y_3 = 1 \). These four sets are shown in the Karnaugh map of Figure 4.23. From the map, it is clear that if each \[
\begin{array}{c|cccc}
\ & y_1 & y_2 & y_3 & y_4 \\
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 \\
\end{array}
\]
Figure 4.23 Universal 4 state one-shot assignment (Hamming code)
state of any 4-state flow table is assigned the two members of a set \( R_j \), then all transitions can be made by changing exactly one variable. Thus, it is a universal single transition time state assignment for any 4-state table. \( \square \)

The above method can be used to obtain an assignment for an \( n \)-state table, where \( n = 2^5 \). We shall now prove that this assignment is a universal (one-shot) STT assignment.

**Theorem 4.1** The Hamming code assignment is a universal \( n \) state (one-shot) STT assignment if \( n = 2^5 \).

**Proof:** From the definition of \( R_j \), it follows that each vector \( y \) is contained in exactly one \( R_j \). Since there are \( n \) such sets, the assignments of different states are distinct.

In order to show that it is a universal STT assignment, we show that for every vector \( y \in R_j \) and any \( J \), there exists a \( y' \in R_j \) such that \( y \) and \( y' \) differ in exactly one bit position. (Thus the transition from \( y \) to some vector assigned to any other state can be made by changing a single state variable.) For any two integers \( I \) and \( J \) whose binary representations are \( b_k(I) b_{k-1}(I) \ldots b_1(I) \) and \( b_k(J) b_{k-1}(J) \ldots b_1(J) \) respectively, we define
\[
I \oplus J = \sum_{k=1}^{5} (b_k(I) \oplus b_k(J)) \cdot 2^{k-1}
\]
That is, \( I \oplus J \) is the value obtained by treating the bit-wise modulo-2 sum of \( I \) and \( J \) as a binary number. For example, \( 9 \oplus 5 = 1001 \oplus 0101 = 1100 = 12 \).

Let \( y \in R_j \) and \( Q = I \oplus J \). Let \( y' \) be the vector obtained by changing the \( Q \)-th bit (from the right) of \( y \). Then the theorem can be proved by showing that \( y' \in R_j \). (Exercise.) \( \square \)

The number of variables required for an STT assignment can be reduced considerably by allowing more than one variable to change simultaneously, but without critical races. We shall now consider STT assignments with a single coding per state, usually referred to as *unicode* assignments. Such assignments were first proposed by Liu [13] and further developed by Tracey [21]. For an STT assignment, if several variables change during a transition the precise order of change cannot be prespecified. It is useful to define the transition *subcube*, \( T(q_1,q_1) \), associated with states \( q_i \) and \( q_j \) as the set of all points that may be passed through during a transition between \( q_i \) and \( q_j \) (including \( q_i \) and \( q_j \)), assuming that the variables that change during a transition may do so in any arbitrary order. The transition subcube \( T(q_1,q_1) \) consists of all points which are identical to \( q_i \) and \( q_j \) in the variables in which \( q_i \) and \( q_j \) agree. For example, if \( q_i \) and \( q_j \) are assigned \( y_1 y_2 y_3 y_4 = 0110 \) and \( 1100 \) respectively, the circuit may pass through \( 0100 \) or \( 1110 \). The transition subcube \( T(S_1,S_1) \) contains the four points \( 0110 \), \( 0101 \), \( 1100 \) and \( 1110 \), and is denoted by \( T(q_1,q_1) = y_1 y_2 y_3 y_4 \) indicating that \( y_1 \) and \( y_2 \) may assume any value during the transition.

The intersection of \( T(q_1,q_1) \) and \( T(q_m,q_m) \), denoted by \( T(q_1,q_1) \cap T(q_m,q_m) \), consists of those points contained in both subcubes. If in a column \( L_k \) of a flow table there are transitions from \( q_i \) to \( q_j \) and from \( q_m \) to \( q_n \), where \( q_i \neq q_j \), there is a critical race (assuming all variables change simultaneously) unless \( T(q_1,q_1) \cap T(q_m,q_m) = \emptyset \) (i.e. there are no points contained in both subcubes). (This requirement is comparable to the disjoint path requirement for multi-step shared row state assignments.) In order that \( T(q_1,q_1) \cap T(q_m,q_m) = \emptyset \), there must be some variable \( y_p \) which takes the value 0 for both \( q_i \) and \( q_j \) and the value 1 for \( q_m \) and \( q_n \) or vice versa. (Exercise.)

Liu [13] has developed a simple procedure for obtaining a state assignment with this property. The method consists of assigning a set of state variables associated with each column of the flow table so
as to distinguish the stable states in that column from one another, each such stable state having a unique coding in the associated variables. Unstable states in the column are assigned the same values as the stable states reached from them. This procedure is only applicable to normal mode flow tables.

\textbf{Procedure 4.1} (Liu's unicode STT \texttt{ALGEBRAIC}). Let the normal mode flow table have \( m \) columns, \( I_1, I_2, \ldots, I_m \) and let the column \( I_j \) have \( \alpha_j \) stable states, \( 1 \leq j \leq m \). For every column \( I_j \), define \( \log \alpha_j \) variables \( y_{j1}, y_{j2}, \ldots, y_{jm} \) so that
(a) each stable state in column \( I_j \) has a unique coding in these variables;
(b) if \( N(q_i, I_j) = q_k \neq q_i \), then the states \( q_i \) and \( q_k \) have the same coding in these variables. \( \square \)

\textbf{Example 4.11} Consider the flow table shown in Figure 4.24. The column 00 has two stable states, 2 and 5 which are distinguished by \( y_1 \). States

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
\( x_1, x_2 \) & 00 & 01 & 11 & 10 \\
\hline
1 & 2 & & & \textcircled{1} \\
2 & & 6 & 4 & \textcircled{2} \\
3 & 2 & 3 & 4 & \textcircled{3} \\
4 & 5 & 4 & & \textcircled{4} \\
5 & & 3 & 1 & \textcircled{5} \\
6 & 2 & 6 & 3 & \textcircled{6} \\
\hline
\end{tabular}
\end{center}

1, 3, and 6 all of which lead to state 2 in the column 00 and state 2 itself are assigned \( y_1 = 0 \). States 4 and 5 are assigned \( y_1 = 1 \), since state 4 leads to the stable state 5 in the column 00. Similarly, the variables \( y_2, y_3 \) and \( y_4 \) are used to distinguish between the four stable states in column 01. The unstable states 2 and 5 are assigned the same values for the variables \( y_2 \) and \( y_3 \) as the states 6 and 3 respectively. Column 11 has three states and requires two state variables. However, it is possible to distinguish between them even if one variable is partially unspecified. Since \( y_2 = 0 \) for states 1 and 3, \( y_2 = 1 \) will distinguish state 4 from them and \( y_3 \) may be unspecified for state 4. We will shortly demonstrate how such unspecified entries may be useful in reducing the number of state variables required. Column 10 necessitates a variable to distinguish stable states 2 and 5. The previously defined variable \( y_3 \) or \( y_2 \) can be used for this purpose. \( \square \)

\textbf{Theorem 4.2} Procedure 4.1 generates an STT state assignment without critical races.

\textbf{Proof:} Consider a transition from \( q_i \) to \( q_j \) in column \( I_k \). By the construction of the state assignment, all the variables \( y_{kj} \) defined for column \( I_k \) remain constant during this transition, and the transition subcube \( T(q_i, q_j) \) is specified in these variables. For any other transition from \( q_m \) to \( q_n \), \( q_m \neq q_j \), this set of variables distinguishes \( q_m \) and \( q_n \) from \( q_i \) and \( q_j \). Therefore, \( T(q_i, q_j) \cap T(q_m, q_n) = \emptyset \), and no point outside \( T(q_i, q_j) \) can be reached during the transition from \( q_i \) to \( q_j \). Hence, no state \( q_n \neq q_j \) can be reached during the transition, and there is no critical race in the transition. Two states will have the same coding only if the flow table can be reduced. \( \square \)

It may be possible to reduce the number of state variables in the state assignment obtained by Procedure 4.1 without introducing any critical races. For this purpose, it is convenient to consider an assignment table whose rows consist of the coding assigned to the states of the flow table. Thus, each column will represent the values assumed by a state variable in the different states. A column \( y_i \) includes a column \( y_j \) denoted by \( y_i \supseteq y_j \) if \( y_i \) agrees with \( y_j \) wherever \( y_i \) is specified. A column \( y_i \) covers a column \( y_j \) if \( y_i \supseteq y_j \) or \( y_j \supseteq y_i \), where \( y_i \) is obtained by complementing the 0's and 1's in \( y_j \). The intersection of two columns \( y_i \) and \( y_j \) exists if and only if the two columns agree wherever both are specified. The intersection agrees with both columns when both are specified and is equal to one of them when the other is unspecified.

\textbf{Theorem 4.3} The number of state variables in the assignment table can be reduced by (a) deleting columns covered by other columns, and (b) replacing columns \( y_i \) and \( y_j \) by their intersection or by the intersection of \( y_i \) and \( y_j \), if either exists, without introducing critical races.

\textbf{Proof:} Exercise. \( \square \)
Example 4.12 Consider the flow table of Figure 4.25(a). The assignment table of Figure 4.25(b) is obtained by using Procedure 4.1. State variables

\[
\begin{array}{cccccccc}
00 & 01 & 11 & 10 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \\
1 & 4 & (1) & 2 & - & 0 & 1 & 0 & 0 & 0 \\
2 & 3 & 2 & 2 & - & 0 & 0 & 0 & 1 & 0 \\
3 & 3 & 3 & 5 & - & 0 & 0 & 1 & 0 & 0 \\
4 & 4 & 4 & 6 & - & 0 & 1 & 1 & 1 & 1 \\
5 & 5 & 3 & 3 & - & 1 & 0 & 1 & 0 & 0 \\
6 & 6 & 1 & 6 & - & 1 & 1 & 0 & 0 & 1 \\
\end{array}
\]

(a) \hspace{1cm} (b)

Figure 4.25 Reduction of state variables in Liu-type ALGORITHM—Example 4.12

Instead of distinguishing between all stable states in a column, Tracey’s method considers all pairs of transitions in each column. Let us define a partial dichotomy (or simply dichotomy) of the states of a machine as a disjoint 2-block partition of a subset of the set of states. We can then associate a dichotomy with any pair of transitions in a column of a flow table. The dichotomy associated with the transitions \( q_i \rightarrow q_j \) and \( q_k \rightarrow q_m \) is \( (q_i, q_j; q_k, q_m) \). A state variable is said to cover a dichotomy if that variable is 0 for all states in one block and 1 for all states in the other block. For example, a variable \( y_p \) covers a dichotomy \( (q_i, q_j; q_k, q_m) \) if \( y_p(q_i) = y_p(q_j) = 0 \) and \( y_p(q_k) = y_p(q_m) = 1 \), or vice versa, where \( y_p(q_i) \) denotes the value of the state variable \( y_p \) associated with state \( q_i \).

If a column has a stable state \( q_i \) and a transition \( q_k \rightarrow q_m \), we can associate a degenerate dichotomy \( (q_i; q_k, q_m) \) with the stable state and the transition. Similarly, a dichotomy \( (q_i; q_m) \) can be associated with a pair of stable states in a column. The following theorem due to Tracey [21] gives necessary and sufficient conditions for a unicolored single transition time assignment for any normal flow table.

**Theorem 4.4** A state assignment for a normal mode flow table is a unicolored STT assignment if and only if for every pair of transitions \( q_i \rightarrow q_j \) and \( q_k \rightarrow q_m \), \( q_i \neq q_m \), appearing in the same column of the table, the dichotomy \( (q_i, q_j; q_k, q_m) \) is covered by some \( y \)-variable. The dichotomies should also include the degenerate (2-state and 3-state) dichotomies that may be required in each column.

**Proof: Sufficiency:** From the definition of the covering of a dichotomy by a state variable, it follows that the variable covering the dichotomy \( (q_i, q_j; q_k, q_m) \) will be 0 during the transition \( q_i \rightarrow q_j \) and 1 during the transition \( q_k \rightarrow q_m \), or vice versa. The transition subcubes \( T(q_i, q_j) \) and \( T(q_k, q_m) \) are therefore disjoint and there can be no critical races involving these two transitions. If for every stable state \( q_i \), and every transition \( q_k \rightarrow q_m \) in the same column, the degenerate dichotomy \( (q_i; q_k, q_m) \) is covered by some \( y \)-variable, then \( q_i \neq T(q_k, q_m) \) and the state \( q_i \) cannot be reached during the \( q_k \rightarrow q_m \) transition, independent of the order of change of the variables. Covering of the dichotomies of the type \( (q_i; q_m) \) guarantees that stable states in any column are always distinguished.

**Necessity:** If a column contains transitions \( q_i \rightarrow q_j \) and \( q_k \rightarrow q_m \) and the dichotomy \( (q_i, q_j; q_k, q_m) \) is not covered by any variable, the transition subcubes \( T(q_i, q_j) \) and \( T(q_k, q_m) \) will not be disjoint. Since there will
methods similar to those used for the sequential machine state reduction problem. The concept of maximal compatibles used for the latter is also useful for deriving minimal variable STT assignment.

Two dichotomies may be defined to be compatible if they can be covered by a single state variable. Thus, the dichotomies \((12, 34)\) and \((12, 56)\) are compatible. Although the dichotomy \((34, 56)\) is compatible with both of the above dichotomies, it is impossible to cover all three dichotomies by a single variable. This difficulty can be overcome by defining a pair of ordered dichotomies for each dichotomy to be covered in the state assignment. Each dichotomy \((A, B)\), where \(A\) and \(B\) are sets of states, is replaced by the ordered dichotomies \((A, B)\) and \((B, A)\). Two ordered dichotomies are compatible if no state that appears in the first block of one dichotomy appears in the second block of the other dichotomy. A set of ordered dichotomies is compatible if and only if the ordered dichotomies in the set are pair-wise compatible. A compatible set of dichotomies that is not covered by any other compatible set is called a maximal compatible set. A minimal variable STT assignment can be obtained by finding a minimal set of maximal compatibles that cover all (unordered) dichotomies of the flow table. Note, that it is sufficient to choose either one of a pair of ordered dichotomies to cover the corresponding unordered dichotomy. A minimal variable Tracey type assignment can be obtained by the following procedure.

Procedure 4.2 (Tracey's unicode STT assignment).
1. List the (unordered) dichotomies to be covered.
2. Delete the dichotomies covered by other dichotomies.
3. Replace each remaining dichotomy by a pair of ordered dichotomies and determine the maximal compatible sets of dichotomies. Procedure 3.2 may be used for this.
4. Determine a minimal set of maximal compatibles, so that every unordered dichotomy is covered by some maximal compatibles. Any of the methods used for solving the prime implicant covering problem (Chapter 1) may be used.
5. Define a state variable to cover each maximal compatible set of Step 4.

Example 4.14 For the flow table of Figure 4.27(a) the dichotomies to

*We shall group together the states in a block of a dichotomy and separate blocks by a comma, whenever this notation is not ambiguous.
be covered are (12, 3), (12, 45), and (3, 45) in column 00, (13, 2), (13, 4), (2, 35), (2, 4), and (35, 4) in column 01, (12, 35), (12, 4), and (35, 4) in column 11, and (13, 25) and (13, 45) in column 10. Deleting dichotomies covered by others, we find that the state assignment should cover the following dichotomies: (12, 45), (12, 25), (35, 4), (13, 25), (13, 45). Using a pair of ordered dichotomies for each of these, the pair chart of Figure 4.27(b) is obtained. The compatible pairs of dichotomies are indicated by \( \checkmark \). The maximal compatible dichotomies are (12, 345), (123, 45), (124, 35), and (13, 245). The covering table is shown in Figure 4.27(c). The dichotomies (35, 4) and (13, 25) are only covered by (124, 35) and (13, 245) respectively. The remaining dichotomies are covered by (123, 45). These maximal compatible dichotomies give the 3-variable state assignment shown to the right of the flow table.

Procedure 4.2 can be simplified in the following manner. For any state \( q_i \), the ordered dichotomies which have \( q_i \) in the first block will only be compatible with other similar dichotomies or dichotomies not containing \( q_i \). By symmetry arguments, corresponding to every maximal compatible with \( q_i \) in block 2 there will be an equivalent maximal compatible with \( q_i \) in block 1. Since the dichotomies to be covered are unordered we can delete all dichotomies in which \( q_i \) appears in block 2. The minimal covers for this reduced table will be equivalent to those of the original table. This process can be performed on any state \( q_i \); so we should select that state which appears in the greatest number of dichotomies. Tracey has also proposed some methods to obtain near-minimal state assignments for larger flow tables.
It is possible to define universal unidecode STT assignments. Such assignments cover all possible dichotomies on a set of \( n \) states and hence are valid STT assignments for any normal mode \( n \) state sequential machine. The following theorem gives an upper bound on the number of state variables required for a universal unidecode STT assignment for an \( n \)-state normal mode flow table [6].

**Theorem 4.4** There exists a universal \( n \)-state unidecode STT assignment with \( m \) variables, where

\[
m \geq \frac{\log_3 \left( \begin{array}{c} n \\ 4 \end{array} \right)}{\log_2 \left( \frac{16}{14} \right)} \geq 20 \log_2 \left( \frac{n}{4} \right) = 20 S_0
\]

**Proof:** Consider the class of \( n \times m \) binary arrays, where the rows represent states, the columns state variables, and the entries define a unidecode state assignment. There are \( 2^m \) arrays in this class. Such an array will cover a dichotomy \( (q_1,q_2,q_3,q_m) \) if and only if \( q_1 = q_2 = 0 \), \( q_3 = q_m = 0 \). There are \( 2^m - 2 = 14 \) other combinations of values for these four states that do not cover this dichotomy. Thus the total number of arrays that do not cover this dichotomy is \( 14^m \cdot 2^{n-4m} = 2^m \left( \frac{14}{16} \right)^m \). The total number of dichotomies to be covered by a universal \( n \)-state assignment is \( 3 \cdot \left( \begin{array}{c} n \\ 4 \end{array} \right) \). Thus, there are at most \( 3 \left( \begin{array}{c} n \\ 4 \end{array} \right) \cdot 2^m \left( \frac{14}{16} \right)^m \) arrays that do not cover some dichotomy and hence there are at least \( 2^n - 3 \left( \begin{array}{c} n \\ 4 \end{array} \right) \cdot 2^m \left( \frac{14}{16} \right)^m \) arrays that cover all dichotomies. We wish to find the smallest value of \( m \) such that this is a positive integer. That is,

\[
2^m > 3 \left( \begin{array}{c} n \\ 4 \end{array} \right) \cdot \left( \frac{14}{16} \right)^m
\]

\[
1 > 3 \left( \begin{array}{c} n \\ 4 \end{array} \right) \cdot \left( \frac{14}{16} \right)^m
\]

Thus, \( 20 S_0 \) variables are sufficient for a universal \( n \)-state unidecode STT assignment. However, this bound is nonconstructive. A constructive bound of \( (S_0^3 + 5 S_0^6) / 6 \) applicable to all tables, and another bound proportional to \( S_0^{1.59} \) with some restrictions on the number of states have been presented by Friedman, et al. [6].

If more than one coding is assigned to some or all states of a flow table, it may be possible to obtain STT assignments with fewer variables than a unidecode STT assignment. For example consider the state assignment shown below:

\[
\begin{array}{cccccc}
\begin{array}{c}
   y_1 \\
   y_2 \\
   y_3 \\
   y_4 \\
   y_5 \\
   y_6
\end{array}
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 0 & 0 \\
1' & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 1 & 1 & 0 & 0 \\
2' & 1 & 0 & 0 & 1 & 1 \\
3 & 1 & 1 & 0 & 0 & 1 \\
3' & 0 & 0 & 1 & 1 & 0 \\
4 & 0 & 0 & 0 & 1 & 1 \\
4' & 1 & 1 & 1 & 0 & 0 \\
5 & 1 & 0 & 1 & 1 & 0 \\
5' & 0 & 1 & 0 | 0 & 1 \\
6 & 0 & 1 & 1 & 1 & 1 \\
6' & 1 & 0 & 0 & 0 & 0
\end{array}
\]

**Figure 4.28** Universal 6-state multicode STT assignment

If the codings \( i \) and \( i' \) are assigned to state \( i \), Figure 4.28 is a universal assignment for a 6-state table with two codings per state. The codings assigned to each state are complements of each other. The transitions in the flow table are executed as follows: If the circuit is in state \( q_i \),
and the next state is \( q_i \), the transition will be made to the coding of the state \( q_i \) that differs from the coding of \( q_i \) from which the transition is made in the fewest number of variables. For example, consider a transition from state 1 to state 3. If the circuit is in state 00000, representing state 1, the next state will be 00110. If, on the other hand, state 1 is currently represented by 11111, the next state will be 11001. By examining the transition subcubes of all transitions, it can be shown that the assignment of Figure 4.27 is a universal STT assignment for a 6-state table, with two codings per state [6]. A universal unicode STT assignment for a 6-state table can be shown to require 7 variables, compared to the 5 variables required with two codings per state.

Although it seems likely that multicode STT assignments may require fewer variables than unicode assignments, general methods for constructing them and bounds on the number of state variables required are not known at present. By using a trial and error method of splitting states, multicode STT assignments requiring fewer variables than unicode assignments can be obtained for some normal mode flow tables [23].

A universal STT assignment satisfies all dichotomies of the form \( (q_i, q_j; q_k, q_m) \) and has been called a \((2, 2)\) separating system. Such an assignment can be used to realize any normal mode table if delays are used in all state variables. A state assignment which satisfies all dichotomies of the form \( (q_i, q_j; q_k) \) is called a \((2, 1)\) separating system. A state assignment which assigns a unique coding to each state satisfies all dichotomies of the form \( (q_i; q_j) \) and is called a \((1,1)\) separating system. Magó [14] has developed canonical realizations of normal mode flow tables using state assignments corresponding to \((2,1)\) and \((1,1)\) separating systems. If a delay element is placed in each state variable and both the input \( Y \) and output \( Y \) are inputs to the state variable excitation logic, then a state assignment corresponding to a \((2,1)\) separating system can be used to realize any normal mode table. If a delay element is placed in each input variable and each state variable and both the input and output of each delay element is an input to the excitation logic, state assignments based on \((1,1)\) separating systems can be used to realize any normal mode table.

4.4.4 State Assignments for MOC Flow Tables

The state assignments discussed in the preceding sections are applicable to MOC tables with little or no modifications. If the time-independent assumption is used, connected row set, shared row set or Hamming code assignments may be used. With the time-dependent assumption the duration of the outputs is proportional to the transition time. It is therefore necessary to ensure that all transitions require approximately the same length of time. The Hamming Code assignment satisfies this condition.

If unique STT assignments of the type discussed in the preceding section are to be used for MOC flow tables, it is necessary to ensure that transitions do not reach the final stable state without passing through all specified intermediate transient states. For example, if a column of an MOC table contains a transition \( q_i \rightarrow q_j \rightarrow q_k \), the transition should pass through the state \( q_j \). This can be accomplished by requiring that the assignment cover the dichotomies \( (q_i, q_j; q_k) \) and \( (q_i, q_j; q_k) \) in addition to the dichotomies required for eliminating critical races. If a column \( I_n \) contains transitions \( q_i \rightarrow q_j \) and \( q_k \rightarrow q_j \) and \( Z(q_i, I_n) \neq Z(q_k, I_n) \) then the dichotomy \( (q_i, q_j; q_j) \) and \( (q_k, q_j; q_j) \) must be covered, so that the circuit does not produce an incorrect output [23].

4.5 SPECIFICATION OF THE Y-MATRIX

As with synchronous circuits, once the state assignment has been specified, the logic circuit can be realized with the use of Y-matrices and/or flip-flop excitation matrices. However, there are several important differences between the asynchronous and synchronous cases. In the synchronous circuit Y-matrix specification, those entries corresponding to codes which were not assigned to any state could be left unspecified, since the actual circuit could never reach such a configuration. For asynchronous circuits using shared row state assignments or STT assignments, unassigned codes may be entered during transitions. The entries of the Y-matrix in this case must be specified so that the transition is properly completed. The entries which must be specified and how they must be specified depend on the type of assignment used. All possible intermediate codes for any transition must be specified. In addition, the entries corresponding to a code for a state \( q_i \) in column \( I_k \) will not always be equal to \( N(q_i, I_k) \). If this is part of a multistep transition, the entry may correspond to an intermediate state in the transition. The following examples demonstrate the construction of Y-matrices for shared row and unicode STT assignments.

Example 4.15 Consider the flow table of Figure 4.29(a) and the shared row assignment, originally specified in Example 4.8. The Y-matrix of Figure 4.29(b) is obtained, if the transitions are made as shown in Figure 4.20. The transition from state 5 to state 6 in column 00 as specified in Figure 4.20 is a multistep transition \( 011 \rightarrow 111 \rightarrow 101 \). Therefore, the next state entry for 011 in column 00 is specified as 111, and the
Example 4.16 Consider the flow table of Figure 4.30(a) and the 3-variable unicode STT assignment derived in Example 4.13 and shown to the right of the flow table. The associated $Y$-matrix is shown in Figure 4.30(b). If a coding is assigned to state $q_i$, and $N(q_i, I_k) = q_j$, the next state entry of this coding in column $I_k$ is the coding assigned to state $q_j$. All variables which must change value are permitted to be excited simultaneously. Since these variables may change in any order, all points in the transition subcube $T(q_i, q_j)$ in column $I_k$ must be specified with the coding for state $q_j$. Thus since the transition subcube associated

![Figure 4.29](image)  

![Figure 4.30](image)

next state entry of the unassigned code $111$ is specified as $101$ in column 00. Similarly, the transition from state 4 to state 5 in column 01 is specified as $110 \rightarrow 111 \rightarrow 011$, and the state 6 to state 4 transition in column 11 as $101 \rightarrow 111 \rightarrow 110$. All other transitions are between adjacent states. In general, in shared row as well as connected row set assignments, each transition is accomplished by a sequence of single variable changes, and the $Y$-matrix is defined accordingly. □
with the transition from state 3 to state 1 in column 00 of the table of Figure 4.30(a) is -0, the four codings 000, 100, 010, and 110 in the Y-matrix, all have entries 000 corresponding to state 1. 

The outputs of the sequential circuit were ignored in the previous examples. In a normal mode flow table, if \( N(q_i, I_k) = q_i \), then \( Z(q_i, I_k) = Z(q_j, I_k) \). In order to keep the output constant during the \( q_i \to q_j \) transition, all points that may be entered during the transition must be assigned the output \( Z(q_i, I_k) \). If a multistep transition is used as in connected row set or shared row assignments, all codings in the transition are assigned this output in column \( I_k \). In a unique STT assignment, all points in the transition subcube \( T(q_l, q_k) \) are assigned this output in column \( I_k \).

### 4.6 DELAYS AND HAZARDS

So far we have only concerned ourselves with the problem of ensuring that the inputs to the combinational logic which generates the memory excitation functions (in the model of Figure 4.11) are restricted in such a way that the next state and output can be unambiguously determined in any transition, and hence with idealized combinational excitation logic, the circuit will function properly. We also assumed that only one input variable changes at any time, and that the state assignment be restricted so that any transition is accomplished by the change of a single state variable, a sequence of single state variable changes, or, in the case of an STT assignment, the state variables which do not change value in the transition and the input variables unambiguously determine the desired next state and output. However the combinational excitation logic is not ideal, and care must be exercised in its design to ensure proper operation.

Any physical switching circuit has delays associated with gates and interconnecting lines. While these delays were not important in synchronous circuits because of the clock pulse, their effects on the operation of asynchronous circuits are significant. The delays associated with gates and lines are usually called stray delays to distinguish them from delays that may be inserted in the circuit to ensure proper operation. The stray delay in a line may represent the time taken for a signal to propagate along the line. A gate may operate only when its inputs are above some threshold value. Since a signal cannot change instantaneously, the effect of the rate of change of signal value and the threshold for switching can be most conveniently represented by a suitable delay at the gate input. Similarly, the switching time associated with the gate may be represented by a delay at the gate output. Although the exact values of the stray delays in a circuit are usually unknown because of the variation in the properties of the components used, the range of values is known. We shall assume that the values of stray delays range from 0 to some known upper bound much less than the magnitude of delay elements which can be inserted in feedback loops and used as memory elements. Stray delays are assumed to be lumped at the inputs and outputs of gates and in connecting lines.

Two types of delays are of interest. A pure delay of magnitude \( D \) produces an output \( f(t-D) \) in response to an input \( f(t) \). That is, the output is merely delayed by a fixed amount \( D \), and the output and input waveforms are identical. An inertial delay of magnitude \( D \) delays an input change by \( D \) if it persists for at least time \( D \). Input changes of duration less than \( D \) are not transmitted through the inertial delay. Thus, inertial delays can be used to represent the property of some physical devices that require the inputs to persist for a certain length of time before the device responds to it. For example, a pulse of short duration at the input of a flip-flop may have no effect on its state. Figure 4.31 illustrates the difference between pure and inertial delays. Stray delays may be assumed to have both pure and inertial components.

![Figure 4.31 Behavior of pure and inertial delays](image)

A circuit is said to contain a hazard if there exists some possible combination of values of stray delays which will produce a spurious pulse or cause the circuit to enter an incorrect stable state, for some input change. Note that a hazard represents only a possibility of malfunction. A specific circuit may not malfunction even though a hazard
exists, because the relative magnitudes of actual stray delays may ensure proper operation, and hence the hazard represents the worst case.

4.6.1 Combinational Hazards

The possibility of the occurrence of spurious pulses on the outputs of a purely combinational circuit for certain input changes is called a combinational hazard. Combinational hazards are usually divided into two classes—static and dynamic. A static hazard is said to be present when the output of a circuit is required to remain constant during a transition, but for some distribution of stray delays, the output may contain one or more pulses (i.e., the output changes an even number of times). Static hazards are classified as 0- and 1-hazards, depending on whether the output is specified to be 0 or 1 during the transition. A dynamic hazard may produce a sequence of three (or a greater odd number) output changes when a single change is required.

The following sequence of lemmas presents necessary and sufficient conditions for the presence of hazards in 2-level combinational circuits, for single input variable changes.

**Lemma 4.2** Let \( f \) be a combinational function such that \( f(I_1) = f(I_2) = 0 \), where \( I_1 \) and \( I_2 \) are input states which differ in only one variable \( x_k \). A 2-level sum-of-products realization of \( f \) has a static 0-hazard for a transition between \( I_1 \) and \( I_2 \) if and only if it has a term realized by an AND gate having both \( x_k \) and \( \overline{x_k} \) as inputs, and the remaining inputs to this gate are 1 for both of the input states \( I_1 \) and \( I_2 \).

**Proof: Sufficiency:** Consider the transition \( I_1 \rightarrow I_2 \), and without loss of generality, let \( x_k = 0 \) in \( I_1 \). When the input state is \( I_1 \), the outputs of all AND gates will be 0. Consider the gate satisfying the conditions of the lemma. Let the stray delays be such that the \( 1 \rightarrow 0 \) change in \( x_k \) reaches the AND gate before the \( 1 \rightarrow 0 \) change in \( \overline{x_k} \). Thus, all the inputs to the AND gate may be 1 for a long enough period to produce a spurious 1 output, resulting in the sequence \( 0 \rightarrow 1 \rightarrow 0 \) at the output of the circuit.

**Necessity:** Since the OR gate itself cannot produce a 1 output if all its inputs are 0, one of the AND gates must produce the \( 0 \rightarrow 1 \rightarrow 0 \) transient if a static hazard exists. An AND gate can produce such an output sequence only if at least one input undergoes a \( 0 \rightarrow 1 \) change and at least one input undergoes a \( 1 \rightarrow 0 \) change, while the remaining inputs stay fixed at 1. Since \( x_k \) is the only input variable that changes during the input transition, the inputs that change can only be \( x_k \) and \( \overline{x_k} \) and the remaining inputs to that gate must be 1.

It follows from Lemma 4.2 that a 2-level sum-of-products realization will contain static hazards for transitions during which the output is required to remain at 0 only if it contains terms with both a variable and its complement. Such terms are clearly unnecessary and may be discarded without changing the function realized, and at the same time eliminating this class of static hazards.

**Lemma 4.3** For a combinational function \( f \), let \( I_1 \) and \( I_2 \) be two input states differing in only one input variable \( x_k \), and let \( f(I_1) = f(I_2) = 1 \). A 2-level sum-of-products realization of \( f \) will have a static 1-hazard for a transition between \( I_1 \) and \( I_2 \) if and only if the realization contains no product term that is 1 for both \( I_1 \) and \( I_2 \).

**Proof: Sufficiency:** If the condition of the lemma is satisfied, the set of terms that are 1 for the input state \( I_1 \) are disjoint from those that are 1 for the input state \( I_2 \). Thus, during the \( I_1 \rightarrow I_2 \) transition, the outputs of one set of AND gates change from 0 to 1 and the other set from 1 to 0. Since no input to the OR gate remains fixed at 1, all inputs to the OR gate may be 0 long enough to produce a \( 1 \rightarrow 0 \rightarrow 1 \) transient at the output.

**Necessity:** Let there be a term that is 1 for both \( I_1 \) and \( I_2 \). Clearly, this term will be independent of \( x_k \), the variable that changes during the transition. This term will have the fixed value of 1 throughout the transition and prevent the occurrence of a \( 1 \rightarrow 0 \rightarrow 1 \) transient at the output.

**Example 4.17** Consider the circuit of Figure 4.32(a) and the transition from the input state 111 to 011. When the input state is 111, \( a = 1 \) and \( b = 0 \). When the input \( x_1 \) changes to 0, \( a \) changes from 1 to 0, while \( b \) changes from 0 to 1. If the \( 0 \rightarrow 1 \) change on \( b \) reaches the OR gate after the \( 1 \rightarrow 0 \) change on \( a \) has reached it (due to a larger stray delay in the path to \( b \) than in the path to \( a \)), then both the inputs to the OR gate may be 0 long enough to produce a 0 output. This hazard can be removed by adding an AND gate realizing \( x_2 x_3 \) to the realization. The term \( x_2 x_3 \) will be 1 for both the input states 111 and 011 (Figure 4.32(b)).

Another method of eliminating the transient during the 111 → 011
transition is to delay the $1 \rightarrow 0$ change on $a$ (possibly by inserting a delay element) so that the $0 \rightarrow 1$ change on $b$ reaches the OR gate first. However, this will produce a transient during the transition from 011 to 111.

Lemma 4.4  For a combinational function $f$, let $I_1$ and $I_2$ be two input states which differ in only one variable, $x_k$, and let $f(I_1) \neq f(I_2)$. A 2-level sum-of-products realization of $f$ has a dynamic hazard for the transition between $I_1$ and $I_2$ if and only if it has an AND gate with both $x_k$ and $\bar{x}_k$ as inputs and the remaining inputs to this gate are 1 for both $I_1$ and $I_2$.

Proof: Sufficiency: From the proof of Lemma 4.2, it follows that the AND gate satisfying the conditions of the lemma may produce a $0 \rightarrow 1 \rightarrow 0$ transient during transitions between $I_1$ and $I_2$. If $f(I_1) = 0$ and $f(I_2) = 1$, some other AND gate output must undergo a $0 \rightarrow 1$ change during the $I_1 \rightarrow I_2$ transition. If this change is delayed with respect to the $0 \rightarrow 1 \rightarrow 0$ transient, the output of the circuit will undergo a $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$ transition. Similarly, for the $I_2 \rightarrow I_1$ transition, if all $1 \rightarrow 0$ changes in AND gate outputs occur before the $0 \rightarrow 1 \rightarrow 0$ transient, the output will undergo a $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ transient.

Necessity: Since the OR gate itself cannot produce a transient, it can occur only if an AND gate produces a $0 \rightarrow 1 \rightarrow 0$ or $1 \rightarrow 0 \rightarrow 1$ transient. The latter cannot be produced by an AND gate and the former can be produced only if the conditions of the lemma are satisfied (see the proof of Lemma 4.2).

It follows from Lemma 4.4 that the elimination of static hazards is also sufficient for the removal of dynamic hazards for single-input changes in 2-level sum-of-products realizations of combinational functions. The principle of duality can be used to derive similar results for 2-level product-of-sums circuits.

Combination hazards due to multiple-input changes differ from single-input change hazards in that the former cannot always be eliminated by proper design. (It is for this reason that only single-input changes are frequently allowed in asynchronous circuit design.) For example, consider the Karnaugh map of Figure 4.33. If the input to a circuit realizing this function changes from $a = 0101$ to $b = 1111$, the circuit may pass through 0111 or 1101 due to stray delays. The input state 0111 is required to produce a 0 output and therefore this function cannot be realized without any hazard in the transition from $a$ to $b$. On the other hand, the transition from $a$ to $c$ may pass through 0001 or 1101, both of which are 1-points of the function. Hazards during this transition can be eliminated by including in the realization, an implicant covering both $a$ and $c$.

Now, let us consider the transition from $d = 0000$ to $e = 1101$, requiring a $0 \rightarrow 1$ output change. Depending upon the relative magnitudes of stray delays, the circuit may respond to the following sequence of input states: $0000 \rightarrow 0100 \rightarrow 1100 \rightarrow 1101$. Since the circuit is required to produce a 1-output for the input state 0100 and a 0-output for the input state 1100, any circuit realizing this function may produce the output sequence $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$ instead of $0 \rightarrow 1$. This hazard is inherent in the function and cannot be removed if the input variables are allowed to change in any arbitrary order, under the delay assumptions we have made.

Hazards which are inherent in the function are called function hazards as opposed to logic hazards which can be eliminated by "proper" design.
These hazards can be further classified as static and dynamic hazards, as in the single-input change case. A function \( f \) is said to have a static function hazard for the input change \( I_1 \rightarrow I_2 \), if \( f(I_1) = f(I_2) \) and there exists some input \( I_3 \) contained in the transition subcube \( T(I_1, I_2) \) such that \( f(I_3) \neq f(I_1) \). If no such input \( I_3 \) exists, but a particular realization can produce a spurious pulse during the input change, for some distribution of stray delays, the realization is said to contain a static logic hazard. If \( f(I_1) \neq f(I_2) \) and there exist inputs \( I_3 \in T(I_1, I_2) \) and \( I_4 \in T(I_1, I_2) \) and \( f(I_3) \neq f(I_4) = f(I_1) \), then the function has a dynamic function hazard for the transitions between \( I_1 \) and \( I_2 \). The circuit may pass through \( I_3 \) and then \( I_4 \) during the \( I_1 \rightarrow I_2 \) transition. If \( f(I_1) = f(I_4) \) and \( f(I_2) = f(I_3) \), the output will change three times instead of once. The hazard cannot be removed by proper design. If a function has no dynamic function hazards, but the output of a realization can change more than once for some distribution of stray delays, the realization contains a dynamic logic hazard. Thus, function hazards are properties of the function being realized, whereas logic hazards are realization dependent.

From lemmas 4.2 and 4.3 it follows that a 2-level sum-of-products realization of a function \( f \) will have no static 0-hazards and will only have static 1-hazards for single input variable changes if there exist two adjacent 1-points of \( f \) which are not covered by a common product term. Such a hazard can be eliminated by adding a product term which covers both of these 1-points to the realization. It can be shown that all static logic hazards in a sum-of-products realization of \( f \) can be eliminated by including every prime implicant of the function in the realization and excluding terms containing a variable and its complement [4]. For single input variable changes the inclusion of all prime implicants is sufficient but not necessary to have each pair of adjacent 1-points covered by a common product term and hence eliminate all static hazards (See Problem 4.9).

Static function hazards cannot be removed by adding product terms. Some of these hazards can be eliminated by inserting delays so that the circuit does not pass through any input state requiring a change in functional value. For example, if the transition from \( a \) to \( b \) in Figure 4.33 is guaranteed to be made through \( x \) by delaying the \( x \)-change, the circuit will not pass through the input state \( f \) requiring a 0-output. However, an output transient will be produced during the \( b \rightarrow a \) transition, since it will now pass through the 0-point \( f \).

If a transition in an asynchronous circuit is initiated by a single input variable change, using proper circuit design hazard free excitations can be generated which may cause one or more state variables to change value. Delays are usually inserted in feedback loops to ensure that the excitation logic completes its response to the input change before any \( y \)-change occurs. (As we shall see in the following section delays may sometimes not be required). The state variables either change one at a time, in which case proper design ensures hazard free responses, or in non-critical races. In this case the entire transition subcube defined by the changing state variables, generates the same excitations and can and must be covered by a single product term (assuming the excitation is 1) to ensure hazard free operation.

The elimination of dynamic logic hazards is more difficult and there is no systematic method for doing so. Although dynamic hazards may be unavoidable, it is possible to design sequential circuits whose state transitions are unaffected by them by the use of inertial delays or certain types of flip-flops, that are insensitive to such transients. The use of such elements also eliminates the necessity of designing static hazard free combinational excitation logic.

We have restricted our discussions so far to combinational hazards in 2-level sum-of-products realizations. The following theorem provides a method of determining whether any combinational circuit contains static hazards, by transforming it into a 2-level sum-of-products form. It can be proved by showing that each type of transformation used does not add or remove static hazards [23].

**Theorem 4.6** If the Boolean expression representing a combinational circuit is transformed into a sum-of-products expression using only the associative and distributive laws and DeMorgan's law (which can always be done), the resulting expression will have the same static hazards as the original circuit. \( \Box \)

From Theorem 4.6 and Lemmas 4.2 and 4.3 it follows that a combinational circuit has static hazards if and only if the corresponding sum-of-products expression \( E \) derived by use of the associative and distributive laws and DeMorgan's law, has a product term \( P \) containing both \( x \) and \( \bar{x} \) for one variable only, and if all other literals in \( P \) are set to 1, \( E = xx \) or if for some possible transition between 1-points \( I_1 \) and \( I_2 \), no product term of \( E \) covers both \( I_1 \) and \( I_2 \). However Theorem 4.6 is not valid for dynamic hazards. (See Problem 4.15.) Let \( E \) be a Boolean expression corresponding to a multiple level combinational circuit. From Lemma 4.4 it can be shown that if by assigning constants to some set of variables the resulting expression becomes \( x + xx \) or \( x(x + \bar{x}) \),
then the original circuit has a dynamic hazard for the transition caused by changing \( x \) while the other variables remain constant at the appropriate values which cause \( E \) to assume the appropriate form. Thus if

\[
E(x_1, x_2, x_3, x_4) = \bar{x}_1(x_2 + \bar{x}_1) + (x_1 + \bar{x}_2)(x_2 + x_3x_4)
\]

then

\[
E(0, x_2, 1, 0) = \bar{x}_2 + \bar{x}_2x_2
\]

and consequently there is a dynamic hazard for the transition caused by changing \( x_2 \) if \( x_1 = 0, x_3 = 1 \) and \( x_4 = 0 \). If by assigning each variable to be a constant or identifying it with \( x \) or \( \bar{x} \) the resulting expression becomes \( x + x\bar{x} \) or \( x(x + \bar{x}) \) the circuit has a dynamic hazard for the multiple variable change involving all variables identified with \( x \) or \( \bar{x} \).

4.6.2 Sequential Hazards

The model of asynchronous sequential circuits which we have been considering has a delay element in each feedback loop. The presence of these inserted delays ensures that the combinational excitation logic will have completed its response to an input change before it sees any state variable changes produced by that input change. This along with the other precautions we have taken regarding the state assignment, design of hazard free combinational logic, and the single input variable change assumption are sufficient to ensure proper operation. In analyzing such circuits, it is usually assumed that there are no restrictions on the relative magnitudes of stray delays in the circuit, but that these delays are non-negative and the upper bound \( D' \) is known. The lower bound on the magnitudes of the inserted delays \( D \) is assumed to be known and it is usually assumed that \( D >> D' \). The minimum time between successive input changes for fundamental mode operation is determined by the state assignment and the bounds on the magnitudes of stray and inserted delays. In some cases the circuit may function properly without some or all of these inserted delay elements. However, in general their removal may cause the circuit to malfunction in which case the circuit is said to have a sequential hazard. We shall restrict our attention to normal mode flow tables. Similar techniques are applicable to non-normal tables.

There are two types of hazards that may be present in asynchronous sequential circuits. A circuit is said to contain a steady state hazard if there exists some distribution of stray delays such that the circuit may reach an incorrect state for some input transition. An output hazard (sometimes called a transient hazard) is said to be present if a spurious output pulse may be produced during some transition for some distribution of stray delays.

Figure 4.34(b) shows a realization of the flow table of Figure 4.34(a) using the state assignment shown to its right. Let us consider the transition from 3 to 4. When \( x \) changes from 0 to 1, the output of AND gate \( a \) changes from 0 to 1 and the output of gate \( b \) changes from 1 to 0. As a result, \( Y_1 \) may undergo a \( 1 \rightarrow 0 \rightarrow 1 \) transient. If the \( 0 \rightarrow 1 \) change on \( a \) is delayed sufficiently with respect to the \( 1 \rightarrow 0 \) change on \( b \), the circuit may pass through state 2 represented by \( y_1 = 0 \), \( y_2 = 1 \). Since this is a stable state in the \( x = 1 \) column, the circuit may stabilize in state 2 instead of reaching state 4 as specified in the flow table. This is a steady state hazard caused by the static hazard in the realization of \( Y_1 \). Similarly, a steady state hazard exists during the transition from

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y_1 )</th>
<th>( y_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,0</td>
<td>2,1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3,0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4,0</td>
</tr>
</tbody>
</table>

(a)

Figure 4.34 (a) A flow table and state assignment (b) its realization (c) Karnaugh maps of \( y_1 \) and \( y_2 \)
state 2 to state 3. The circuit may reach state 1 instead of state 3. From the Karnaugh maps of Figure 4.34(c), we see that the static hazards and consequently the steady state hazard can be removed by adding the term \( y_1 y_2 \) and \( \bar{y}_1 y_2 \) to the realizations of \( Y_1 \) and \( Y_2 \) respectively.

The above circuit also contains an output hazard for the transition from state 3 to state 4. The \( 1 \rightarrow 0 \rightarrow 1 \) transient resulting from the static hazard in \( Y_1 \) may produce a \( 0 \rightarrow 1 \rightarrow 0 \) transient in the output \( z \) or an incorrect \( 0 \rightarrow 1 \) change in the output. In this circuit, removal of the steady state hazard will also eliminate the output hazard.

The delays used in the feedback loops and/or as memory elements should be such that the effects of input changes reach all parts of the circuit before any change produced by state variable changes. If the delay element in some feedback path is eliminated or does not satisfy this constraint, malfunctioning may occur even if all prime implicants of the next state functions are used in the realization.

Consider the circuit realizing the flow table of Figure 4.34(a) and assume that it is initially stable in state 1, with input \( x = 0 \). Now let us analyze the circuit behavior when the input changes to 1, if the delay memory elements do not satisfy the aforementioned constraints. The circuit first changes to state 2 \( (y_2 \text{ changes to 1}) \). Due to large stray delays in the circuit, the effects of the \( y_2 \) change may reach some part of the circuit before the \( 0 \rightarrow 1 \) input change. The next state for state 2, input 0 is 3 and therefore the circuit undergoes the \( 2 \rightarrow 3 \) state transition \( (y_2 \text{ changes to 1}) \). Finally, when the effects of the \( 0 \rightarrow 1 \) change in \( x \) reaches all parts of the circuit, the circuit may reach the stable state 4 \( (y_2 \text{ changes back to 0}) \), instead of state 2.

This type of hazard is a property of the flow table. Let \( I_1 \) and \( I_2 \) be two adjacent input states and \( q_i \) an internal state of a flow table, such that \( N(q_i, I_1) = q_i \) and \( N(q_i, I_2) = q_i \). The flow table contains an essential hazard for the \( I_1 \rightarrow I_2 \) transition from state \( q_i \) if the input sequence \( I_2 I_1 I_2 \) applied to the total state \( (q_i, I_1) \) results in a final state \( q_k \neq q_i \).

Let the circuit realizing the flow table of Figure 4.34(a) be initially stable in state 1, with input \( x = 0 \). If the input changes to 1, the correct next state is 2, but the state reached by applying the input sequence 101 is 4. Thus the flow table has an essential hazard. In the flow table of Figure 4.34(a), all transitions involve essential hazards, although this is not generally true for arbitrary tables.

A realization of a flow table which contains an essential hazard may malfunction without inserted delay elements in some state variables due to possible critical races between the changing input variable and state variable(s) caused by stray delays. The problem can be solved by inserting delay elements in the feedback paths thus ensuring that the circuit completes its response to the input change before any changes in the state variable occur. The necessity of inserted delay elements in realizations of flow table containing essential hazards is stated in the following theorem due to Unger and proven in [22, 23].

**Theorem 4.7** Inserted delay elements are required for eliminating steady state hazards in a sequential circuit if and only if its flow table contains at least one essential hazard.

It is only necessary to delay those variables which change during the hazardous transitions. Unger has also shown constructively that one inserted delay element is sufficient for realizing any normal flow table, assuming that only one input variable changes during any transition. (See Problem 4.12) Later in this chapter, we shall present a method of realization requiring only one delay element, but also permitting multiple-input changes, provided that all input changes associated with any input transition are completed within a fixed interval of time.

Flow tables without essential hazards can be realized without inserted delays, but special state assignments may be necessary in some cases. Consider the flow table segment shown in Figure 4.35.

<table>
<thead>
<tr>
<th>( I_1 )</th>
<th>( I_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 4.35** A d-trio

The transition from the total state \((1, I_1)\) to \((2, I_2)\) does not involve any essential hazard. If an STT assignment is used, the circuit may reach any point in the transition subcube \( T(1,2) \) during the transition from state 1 to state 2, before the input change reaches some gate in the circuit. Since \( N(2, I_1) = 3 \), the circuit may then begin the transition from state 2 to state 3. Let the set of state variables which change in the transitions from 1 to 2 and 2 to 3 be denoted by \( A \) and \( B \) respectively. If no inserted delay elements are used, the set of variables \( C = \)
Asynchronous Sequential Circuits

A \cup B must be assumed to change in any order and the circuit may pass through any coding which can be reached from state 2 in this manner. To ensure proper operation the state assignment must be such that for all such codings, the next state entry in column 1 of the Y-matrix can be specified with the coding for state 2. Unger [24] has shown that it is possible to define an STT assignment which covers some additional dichotomies, which has this property. A state assignment in which only a single variable changes during any transition (such as the Hamming code assignment) may also be used. The configuration shown in Figure 4.35 is called a d-trio [22].

In the manner explained above flow tables without essential hazards can be realized without inserted delay elements. Such realizations will not contain steady state hazards but may contain output hazards (which cannot be removed).

It may not be necessary to insert delay elements in all feedback loops for flow tables with essential hazards. Delay elements will be necessary only for state variables that change during transitions involving essential hazards.

Example 4.18 The flow table of Figure 4.36(a) has no essential hazards and hence can be realized without any inserted delay elements. Using the state assignment shown to the right of the table, to prevent malfunctioning in the transition (1,11) \rightarrow (2,01), associated with the only d-trio, we must specify the entry in column 01 and row 10 of the Y-matrix with the coding 01 associated with state 2. This results in the excitation functions

\[ Y_1 = \bar{x}_2 y_1 + x_1 y_1 + x_1 x_2 y_2 \]
\[ Y_2 = y_1 + \bar{x}_1 y_2 + x_2 y_2 + \bar{x}_1 x_2 \]

The flow table of Figure 4.36(b) has an essential hazard associated with the transition from the total state (1,00) to the total state (2,01). Since this is the only essential hazard and the only state variable that changes during this transition is \( y_2 \) (using the state assignment shown), a delay element is required only for this variable. The only d-trio is associated with the transition (3,10) \rightarrow (2,11). By specifying the entry in row 10 and column 11 with the coding for state 2, the resulting circuit will operate properly.

4.6.3 Detection of Hazards Using Ternary Algebra

Eichelberger [4] has proposed a method of hazard detection in combinational and sequential circuits using ternary algebra. The value 1/2 is used to represent signals that are changing value. For a transition \( I_1 \rightarrow I_2 \), the output of the circuit for input \( I_1 \), \( f(I_1) \), is first computed. Then all changing inputs are set to 1/2 and the output of the circuit is recomputed where the output of an AND is equal to the minimum value of any of its inputs, the output of an OR is the maximum value of any of its inputs, the output of a NAND (NOR) is \( 1 - \min \) (inputs) \((1 - \max \) (inputs)). This is referred to as the halves pass. Finally \( f(I_2) \) is computed. A static hazard exists for this transition if and only if \( f(I_1) = f(I_2) \) and the halves pass output is 1/2. If \( f(I_1) \neq f(I_2) \) the halves pass value will always be 1/2, whether or not there is a dynamic hazard. Hence this method is not capable of detecting dynamic hazards.
Example 4.19 For the circuit of Figure 4.37(a), consider the transition $I_1 \rightarrow I_2$ where $I_1 = (0,1,1)$ and $I_2 = (1,1,1)$. We first compute $z_1(I_1) = z_1(0,1,1) = 1$. For the halves pass we must compute $z_2(1/2,1,1)$. The output of $G_1$ is $\min(1/2,1) = 1/2$, the output of $G_2$ is $\min(1/2,1,1) = 1/2$, and the output of $G_3 = \max(1/2,1/2) = 1/2$. Since $z_1(I_2) = z_1(1,1,1) = 1$, there is a static 1-hazard on output $z_1$. For output $z_2$, $z_2(I_1) = 0$, $z_2(I_1) = 1$ and $z_2(1/2,1,1) = \min(1/2,1,1,1/2) = 1/2$. Thus the output sequence generated by the transition is $0 \rightarrow 1/2 \rightarrow 1$. For the circuit of Figure 4.37(b), $G_5 = 1$ for $I_1, I_2$, and $(1/2,1,1)$ so $z_4(I_1) = z_4(1/2,1,1) = 1$ and there is no static hazard on $z_4$. The output $z_5(I_1) = 0$, $z_5(I_2) = 1$, and $z_5(1/2,1,1) = \min(1/2,1,1,1/2) = 1/2$. This is the same output sequence generated by the circuit of Figure 4.37(a) although the transition has a dynamic hazard for that circuit but not for the circuit of Figure 4.37(b).

This ternary algebra may also be used for detecting steady state hazards in sequential circuits with or without inserted delay elements, and for single-input or multiple-input variable changes. If the circuit contains delay elements (pure or inertial), their magnitudes are assumed to be greater than the time taken by the circuit to respond to the $x$-input changes.

Procedure 4.3 (Hazard detection in sequential circuits).

1. Consider a transition from a stable total state $(q_i, I_j)$ resulting from the input change $I_k$. For every input variable $x_i$ that changes during the transition, set $x_i = 1/2$. Leave the other input variables and all state variables at previous known values.

2. Compute the values of all the gates in the circuit as specified previously, assuming that the outputs of inserted delay elements do not change yet. Repeat until all gate outputs stabilize.

3. Change all the input variables which were assigned the value 1/2 in step 1, to their new values corresponding to the input state $I_k$ and recompute gate outputs as in (2) until they stabilize, still not permitting delay element outputs to change.

4. Change the outputs of pure delay elements to values of their respective inputs at the end of step 2. For inertial delays, change their outputs to the values of their respective inputs at the end of step 3. Recompute all gate outputs, until they stabilize.

5. Change the outputs of pure delay elements to values of their respective inputs at end of step 3. Recompute all gate outputs until they stabilize.

6. If any delay element inputs computed in (4) or (5) are different from the delay outputs, change those outputs to the respective input values and recompute gate outputs. Repeat until no further changes occur. If some state variables remain at 1/2, the circuit contains a steady state hazard for the transition under consideration.

Example 4.20 Consider the circuit of Figure 4.38(a), which is intended to realize the flow table of Figure 4.38(b). Let us first consider the case when there are no inserted delay elements in the feedback paths. With the circuit initially stable with $x = y_1 = y_2 = 0$, the effect of changing $x$ to 1 can be determined by setting $x$ to 1/2. If $x = 1/2$, $Y_2$ becomes

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y_1$</th>
<th>$y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) An asynchronous circuit (b) and the flow table of the corresponding machine—Example 4.20
With no delay elements, $y_2$ becomes 1/2, which in turn causes $Y_1$, and then $y_1$ to become 1/2. Since no further change is possible, we now change $x$ to 0. $Y_1$ and $Y_2$ remain at 1/2, indicating the presence of a steady state hazard for this transition. This flow table contains an essential hazard for this transition as shown previously.

Now, let us consider the same circuit with a pure delay element of suitable magnitude inserted in each of the feedback paths. Analyzing the same transition, we note that $Y_2$ changes to 1/2 when $x$ is changed to 1/2. However, the delay element prevents this change from reaching $y_2$ immediately. Changing $x$ from 1/2 to 1 results in $Y_1 = 0$, $Y_2 = 1$. If we now change $y_2$ to 1/2, $Y_1$ and $Y_2$ remain at 0 and 1 respectively, indicating that the transition is hazard-free.

If the circuit is initially stable with $x = 1$, $y_1 = 0$, $y_2 = 1$, and $x$ changes to 0, we change $x$ to 1/2. This results in $Y_1 = Y_2 = 1/2$. Changing $x$ to 0, assuming that the delays are pure, produces $Y_1 = Y_2 = 1$, the correct state. However, when we set $y_1 = y_2 = 1/2$, as specified in Step 4 of Procedure 4.3, $Y_1$ and $Y_2$ both become 1/2, indicating a steady state hazard. This hazard is the result of the static hazard present in the combinational excitation logic. If the inserted delays in the circuit are assumed to be inertial, then it is not necessary to set $y_1 = y_2 = 1/2$ because the inertial delay will mask the transients in $Y_1$ and $Y_2$, and the transition is hazard-free.

Now, let us consider the case where a pure delay is inserted in the feedback path of $y_2$, but none in the other feedback path. Let the circuit be initially in the state $y_1 = 0$, $y_2 = 1$ and let the input change from 1 to 0. Letting $x = 1/2$ as before, we obtain $Y_1 = Y_2 = 1/2$. Since there is no delay in the $y_1$ path, we change $y_1$ to 1/2 and recompute $Y_1$ and $Y_2$, which are both still 1/2. Changing $x$ to 0, $Y_1$ and $Y_2$ are both still 1/2 indicating a steady state hazard.

An 8-valued algebra (similar to a 9-valued algebra proposed by Fantauzzi [5]) can be used for detecting static and dynamic hazards in combinational circuits. The 8 possible values of a signal are 0, 1, + indicating a 0 → 1 transition, indicating a 1 → 0 transition, S0 indicating a static 0-hazard, S1 for a static 1-hazard, D+ indicating a dynamic hazard in a 0 → 1 transition and D- for a dynamic hazard in a 1 → 0 transition. Only one computation is required in place of the three passes for the ternary algebra. For two input AND- and OR- gates and inverters, the outputs can be computed from the tables of Figure 4.39.

![Figure 4.39 Tables for 8-valued algebra](c)
Thus the output of an AND gate with inputs + and – is S0. For the circuit of Figure 4.37(a) and the transition \( I_1 \rightarrow I_2 \) where \( I_1 = (0,1,1) \) and \( I_2 = (1,1,1) \) the inputs to \( G_1 \) are + and 1 and its output is +. The inputs of \( G_2 \) are – and 1 and its output is –. The inputs to \( G_3 \) are + and – and its output is S1 indicating a static 1-hazard on \( z_1 \). The inputs to \( G_4 \) are + and S1 and its output is \( D^+ \) indicating a dynamic hazard on \( z_2 \). For the circuit of Figure 4.37(b) the output of \( G_1 \) is +, the output of \( G_2 \) is – and the output of \( G_4 \) is 1. Using the principles of associativity and commutativity which can be shown to be valid for this algebra the output of \( G_3 \) is determined to be 1 and the output of \( G_3 \) is +. Thus this transition is hazard free.

In addition to the 8 values used here, Fantauzzi [5] uses a ninth value to represent unknown signal values. By this extension, the method can be used for circuits containing flip-flops but no feedback (other than that within the flip-flops).

4.6.4 A Synthesis Example

So far we have discussed different problems related to synthesis. Frequently, several alternatives are available during the design. For example, static hazards can be eliminated by adding product terms to the combinational logic or by using inertial delays in the state variable feedback loops. The effects of d-trios can be eliminated by restricting the state assignment and logic realizations or by using inserted delay elements. The following example demonstrates the complete synthesis of an asynchronous sequential circuit from a flow table, including some of these alternative design options.

Example 4.21 The flow table of Figure 4.40 has transitions between every pair of states. Therefore, a connected row set or shared row state assignment will require more than two variables, since every state must be adjacent to three other states. However it is possible to obtain 3-variable STT assignments so we will only consider these. The Hamming assignment is such an assignment. A unicode STT assignment must cover the dichotomies (12, 4), (13, 4), (12, 34), (14, 23) and (13, 24). The 3-variable assignment of Figure 4.41(a) covers all of these dichotomies.

The flow table contains essential hazards for the transitions (1, 11) \( \rightarrow \) (2, 01); (2, 01) \( \rightarrow \) (3, 11); (3, 11) \( \rightarrow \) (4, 01); (4, 00) \( \rightarrow \) (2, 10); (4, 01) \( \rightarrow \) (1, 11). For both the Hamming and unicode state assignments each of the state variables changes during some of these transitions so delay elements are required in all feedback paths. We shall only consider the generation of the excitation logic for the unicode STT assignment. The \( y_2z \)-matrix of Figure 4.41(b) is obtained by assigning to every point in a transition sub-circuit the same next state and output as the final stable state.

![Flow table for Example 2.21](figure4.40)

<table>
<thead>
<tr>
<th>( y_1 )</th>
<th>( y_2 )</th>
<th>( y_3 )</th>
<th>( 01 )</th>
<th>( 01 )</th>
<th>( 11 )</th>
<th>( 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) A unicode STT state assignment. (b) \( y_2z \)-matrix

![Y2Z-matrix](figure4.41)

To ensure proper operation the circuit should contain no static hazards. If each \( Y_i \) is realized in the sum-of-products form, it is sufficient to verify that for any transition \( (q_i,l_i) \rightarrow (q_m,l_k) \) in which \( Y_i \) is to remain constant at 1 the points \( (q_i,l_i) \) and \( (q_m,l_k) \) are covered by a single term of \( Y_i \) and all points in the transition subcircuit \( T(q_i,q_m) \) in column \( l_k \) must be covered by a single term of \( Y_i \). This assumes that delay elements are inserted in all feedback loops.

Thus the transition (3, 11) \( \rightarrow \) (4, 01) requires the terms \( x_2y_1y_2 \) and \( x_1x_2y_1 \) in \( Y_1 \). From the \( Y \), \( z \)-matrix, we obtain the following hazard free expressions for the next state and output functions: