Máquinas de Estados

1) Lógica Seqüencial e Lógica Combinacional

Os circuitos combinacionais são caracterizados por terem os estados das saídas determinados apenas pelas condições presentes nas entradas, diferentemente dos circuitos seqüenciais nos quais os estados presentes nas saídas dependem não somente das entradas atuais como também da seqüência anterior dos estados das entradas. Isto implica em que um circuito seqüencial tenha memória, e então possa armazenar informações derivadas das entradas passadas. As informações armazenadas constituem os estados internos do circuito. As máquinas de estados são circuitos seqüenciais que utilizam flip-flops com sinal de clock para que as informações armazenadas possam ser atualizadas sincronamente a intervalos regulares (a cada pulso de clock).

2) Tipos de Máquinas de Estados

De forma geral, podemos dizer que um circuito lógico digital é composto por um bloco combinacional (portas lógicas) e um bloco de armazenamento de informações (flip-flops). Este é conhecido como modelo de Huffman e está representado na figura 1.

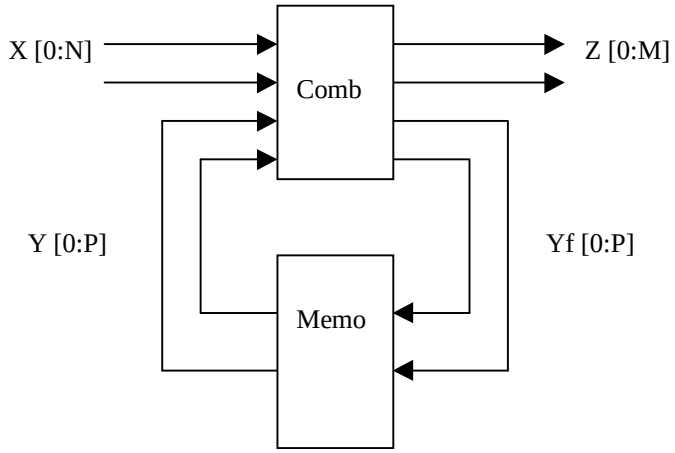


Figura 1 – Modelo de Huffman para um circuito digital generalizado

Neste modelo, se retirarmos os flip-flops, teremos um circuito combinacional simples, se retirarmos as entradas primárias (X), teremos um contador, outros tipos de circuitos, como as máquinas de estados, podem ser obtidos suprimindo outras partes do circuito. Para explicar o princípio de funcionamento das máquinas de estados, vamos dividir os circuitos combinacionais C_1 e C_2 em duas partes, conforme visto na figura 2.

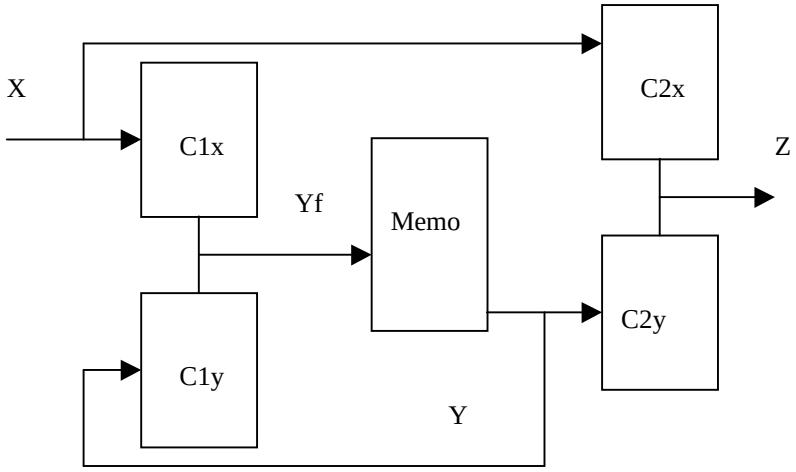


Figura 2 – Máquina de estados generalizada

O circuito C_{1x} relaciona as entradas do circuito de memória (flip-flops) às entradas primárias (X), o circuito C_{1y} relaciona as entradas da memória aos estados internos (Y), o circuito C_{2x} relaciona as saídas (Z) às entradas X e C_{2y} relaciona Z a Y. as entradas dos flip-flops são indicadas como Y_f porque, após o sinal de clock, as saídas Y assumirão os valores de Y_f.

Diferentemente dos contadores, as máquinas de estados possuem uma ou mais entradas que podem modificar a seqüência a ser seguida pelos valores armazenados (seqüência de estados ou seqüência de contagem). Desta forma, os estados futuros (Y_f) dependem de X e de Y, e as saídas (Z) serão função de X e de Y (estados atuais).

Dois tipos de máquinas de estados foram estudados e bem definidos na década de 1950, no Bell Laboratories, o primeiro é conhecido como Máquina de Mealy (por G.H.Mealy, em 1955) e o segundo é conhecido como Máquina de Moore (por E.F.Moore, em 1956). A figura 3 representa o modelo geral da Máquina de Mealy e a figura 4 representa o modelo geral da Máquina de Moore.

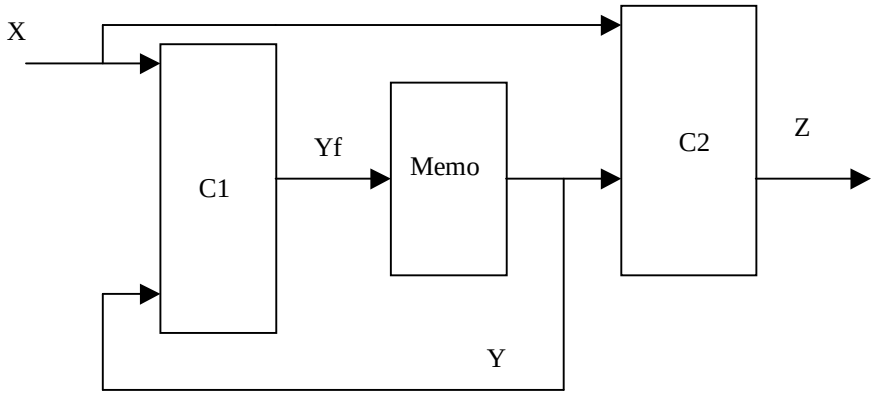


Figura 3 – Máquina de Mealy

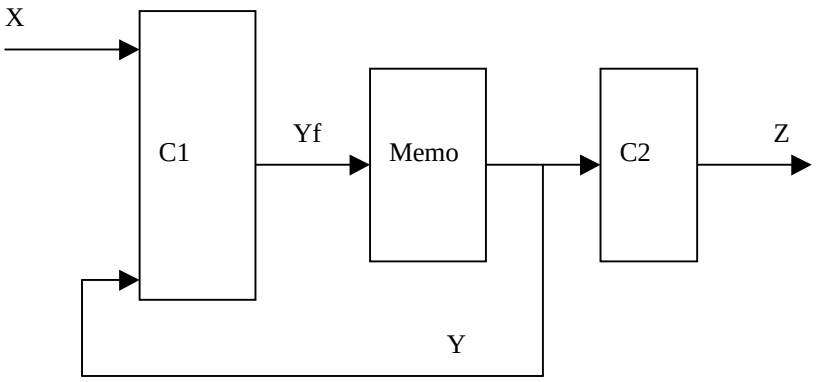


Figura 4 – Máquina de Moore

A diferença básica entre as duas é que na Máquina de Moore as saídas dependem apenas dos estados atuais, enquanto que na Máquina de Mealy as saídas (Z) são função dos estados atuais (Y) e das entradas (X). Portanto, em Mealy as saídas são modificadas assincrona e sincronamente, enquanto que em Moore as saídas são modificadas apenas quando ocorre um pulso de clock.

3) Primeiro exemplo de uma Máquina de Estados – Somador Serial

A figura 5 mostra uma máquina de estados simples, cuja função é fazer a soma de dois números binários que são enviados de forma serial, e enviar o resultado de forma também serial.

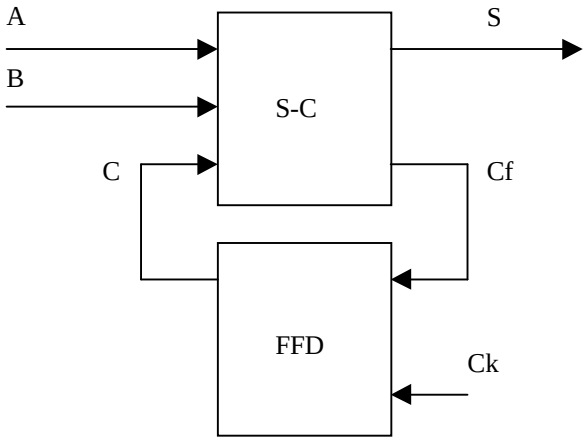


Figura 5 – Somador Serial

Antes de entender o funcionamento do somador serial, devemos pensar em como fazemos uma soma de números binários, por exemplo de 4 bits.

$a_3\ a_2\ a_1\ a_0\ +\ b_3\ b_2\ b_1\ b_0\ = s_4\ s_3\ s_2\ s_1\ s_0$

$c_4\ c_3\ c_2\ c_1$

$a_3\ a_2\ a_1\ a_0$

\+

$b_3\ b_2\ b_1\ b_0$

$s_4\ s_3\ s_2\ s_1\ s_0$

Primeiro, somamos a_0 com b_0 e obtemos $c_1 s_0$, sendo que s_0 é o resultado da soma e c_1 é o bit que vai participar da soma da próxima coluna, coluna 1.

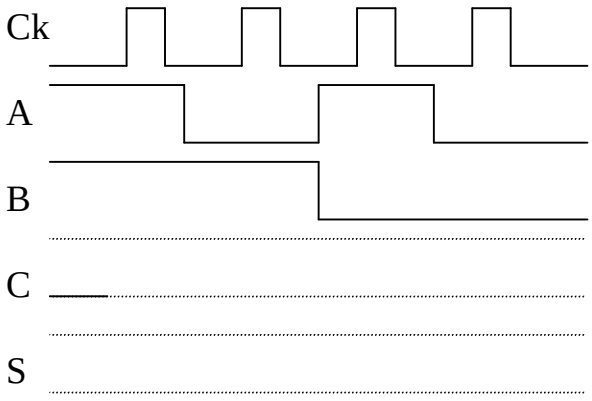
No próximo passo, somamos a_1 com b_1 e c_1, e obtemos $c_2 s_1$, sendo que s_1 é o resultado da soma e c_2 é o bit que vai participar da soma da próxima coluna, coluna 2.

No terceiro passo, somamos a_2 com b_2 e c_2, e obtemos $c_3 s_2$, sendo que s_2 é o resultado da soma e c_3 é o bit que vai participar da soma da próxima coluna, coluna 3.

Finalmente fazemos a quarta soma obtendo $c_4 s_3$. Poderia ser incluído ainda mais um passo para levar c_4 à saída s_4.

Na descrição acima, podemos observar que existe uma seqüência de passos repetidos a cada bit (coluna) da soma, o que indica que não é necessário se fazer a soma de modo paralelo, mas esta pode ser feita bit a bit, serialmente. O que deve ser feito é a combinação das entradas **a b c**, que resultarão na soma s e no próximo valor de **c**, e deve-se armazenar o valor de c a cada passo, para que este possa ser utilizado na soma do próximo bit. Desta forma percebemos que precisamos de um circuito combinacional para gerar a soma (**s**) atual e o carry (**c**) futuro, e de um dispositivo para armazenar o valor de **c** gerado.

Verifique o funcionamento do circuito para as seguintes entradas: A=0101 e B=0011, com 4 pulsos de clock.



4) Procedimento de análise de uma máquina de estados

O comportamento de um circuito seqüencial é definido pela relação entre seus sinais de entradas e de saída. No caso de circuitos combinacionais podemos listar todas as possibilidades dos sinais de entrada, juntamente com os sinais de saída que eles produzem, na forma de tabela verdade. A descrição correspondente do comportamento de um circuito seqüencial exigiria de nós uma lista de todas as possíveis seqüências de entradas e saídas que pudessem ser aplicadas ao circuito. Devido ao fato de existir uma infinidade de tais seqüências e porque estas seqüências podem ter tamanhos arbitrários, seria impraticável construir uma tabela verdade para a descrição deste tipo de circuito, mesmo para circuitos simples como o somador serial. (John P. Hayes, Introduction to Digital Logic Design, Addison-Wesley Publishing Company.

A seguir é apresentado um trecho do livro Digital Design de M. Morris Mano que explica este procedimento, e apresenta exemplos.

6.4 – ANALYSIS OF CLOCKED SEQUENTIAL C1RCUITS

The behavior of a sequential circuit is determined from the inputs, the outputs, and the state of its flip-flops. The outputs and the next state are both a function of the inputs and the present state. The analysis of a sequential circuit consists of obtaining a table or a diagram, for the time sequence of inputs, outputs, and internal states. It is also possible to write Boolean expressions that describe the behavior of the sequential circuit. However, these expressions must include the necessary time sequence, either directly or indirectly.

A logic diagram is recognized as a clocked sequential circuit if it includes flip-flops. The flip-flops may be of any type and the logic diagram may or may not include combinational circuit gates. In this section, we first introduce a specific example of a clocked sequential circuit with *D* flip-flops and use it to present the basic methods for describing the behavior of sequential circuits. Additional examples are used throughout the discussion to illustrate other procedures.

Sequential-Circuit Example

An example of a clocked sequential circuit is shown in Fig. 6-16. the circuit consists of two *D* flip-flops *A* and *B*, an input *x*, and an output *y*. Since the *D* inputs determine the flip-flops' next state, it is possible to write a set of next-state equations for the circuit:

$$A(t+1) = A(t)x(t) + B(t)x(t)$$

$$B(t+1) = A'(t)x(t)$$

A state equation is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation denotes the next state of the flip-flop and the right side of the equation is a Boolean expression that specifies the present slate and input conditions that make the next state equal to 1. Since all the variables in the Boolean expressions are a function of the present state, we can omit the designation (t) after each variable for convenience. The previous equations can be expressed in more compact form as follows:

$$A(t+1) = Ax + Bx$$
$$B(t+1) = A'x$$

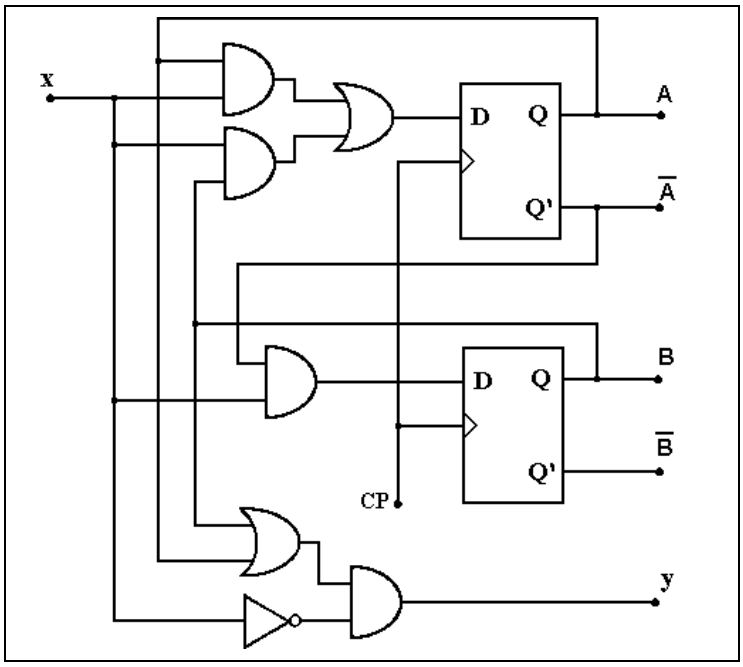


Figure 6-16 – Example of a Sequential Circuit

The Boolean expressions for the next slate can he derived directly from the gates that form the combinational-circuit part of the sequential circuit. The outputs of the combinational-circuit are applied to the *D* inputs of the flip-flops. The *D* input values determine the next state.

Similarly, the present-state value of the output can be expressed algebraically as follows:

$$y(t) = [A(t) + B(t)]\, x'\ (t)$$

Removing the symbol (t) for the present state, we obtain the output Boolean function:

$$y = (A + B)\, x'$$

State Table

The time sequence of inputs, outputs, and flip-flop states can be enumerated in a *state table*. The state table of the circuit of Fig. 6-16 is shown in Table 6-1. The table consists of four sections labeled *present state*, *input*, *next state,* and *output*. The present state section shows the states of flip-flops *A* and *B* at any given time *t*. The input section gives a value of *x* for each possible present state. The next-state section shows the states of the flip-flops one clock period later at time *t*+1. The output section gives the value of *y* for each present state.

The derivation of a state table consists of first listing all possible binary combinations of present State and inputs. In this case, we have eight binary combinations from 000 to 111. The next-state values are then determined from the logic diagram or from the State equations. The next state of flip-flop *A* must satisfy the state equation

$$A(t + 1) = Ax + Bv$$

The next-state section in the state table under column *A* has three 1' s where the present state and input value satisfy the conditions that the present state of *A* and input *x* are both equal to 1 or the

present state of *B* and input *x* are both equal to 1. Similarly, the next state of flip-flop 8 is derived from the state equation

$$B(t+1) = A'x$$

It is equal to 1 when the present state of *A* is 0 and input *x* is equal to 1. The output column is derived from the output equation

$$y = Ax' + Bx$$

TABLE 6-1 – State Table for the Circuit of Figure 6-16

Present State	Input	Next State	Output
A B	x	A B	y
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	0 0	1
0 1	1	1 1	0
1 0	0	0 0	1
1 0	1	1 0	0
1 1	0	0 0	1
1 1	1	1 0	0

TABLE 6-2 – Second Form of the State Table

	Next State		Output	
Present State	x=0	x=1	x=0	x=1
A B	A B	A B	y	y
0 0	0 0	0 1	0	0
0 1	0 0	1 1	1	0
1 0	0 0	1 0	1	0
1 1	0 0	1 0	1	0

The state table of any sequential circuit with *D*-type flip-flops is obtained by the same procedure outlined in the previous example. In general, a sequential circuit with *m* flip-flops and *n* inputs needs 2^{m+n} rows in the state table. the binary numbers from 0 through $2^{m+n} – 1$ are listed under the present-state and input columns. the next-state section has n columns, one for each flip-flop. The binary values for the next state are derived directly from the state equations. The output section has as many columns as there are output variables. Its binary value is derived from the circuit or from the Boolean function in the same manner as in a truth table. Note that the examples in this chapter use only one input and one output variable, but, in general, a sequential circuit may have two or more inputs or outputs.

It is sometimes convenient to express the state table in a slightly different form. In the other configuration the slate table has only three sections: present state, next state, and output. The input conditions are enumerated under the next-state and output sections, the state table of Table 6-1 is repeated in table 6-2 using the second form. For each present state, there are two possible next states and outputs, depending on the value of the input. We will use both forms of the state table. One form may be preferable over the other, depending on the application.

State Diagram

The information available in a state table can he represented graphically in a state diagram. In this type of diagram, a state is represented by a circle. and the transition between states is indicated by directed lines connecting the circles. The state diagram of the sequential circuit of Fig. 6-16 is shown in Fig. 6-17. The state diagram provides the same information as the state table and is obtained directly from Table 6-2, the binary number inside each circle identifies the state of the flip-flops. The directed lines are labeled with two binary numbers separated by a slash. The input value during the present state is labeled first and the number after the slash gives the output during the present state. For example, the directed line from state 00 to 01 is labeled 1/0, meaning that when the sequential circuit is in the present state 00 and the input is 1, the output is 0. After a clock transition, the circuit goes to the next slate, 01. The same clock transition may change the input value. If the input changes to 0, then the output becomes 1, but if the input remains at 1, the output stays at 0. This information is obtained from the state diagram along the two directed lines emanating from, the circle representing state 0l. A directed line connecting a circle with itself indicates that no change of slate occurs.

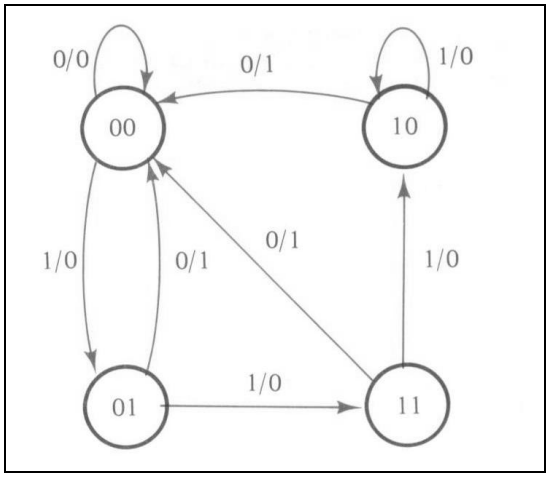


FIGURE 6-17 - State diagram of the circuit of Fig 6-16

There is no difference between a state table and a state diagram except in the manner of representation. The state table is easier to derive from a given logic diagram and the stale diagram follows directly from the state table. The state diagram gives a pictorial view of state transitions and is the form suitable for human interpretation of the circuit operation. For example. the state diagram of Fig. 6-17 clearly shows that. starting from, state 00, the output is 0 as long as the input stays at 1. The first 0 input after a string of 1‘s gives an output of 1 and transfers the circuit back to the initial stale 00.

Flip-Flop Input Functions

The logic diagram of a sequential circuit consists of flip-flops and gates. The interconnections among the gates from a combinational circuit may be specified algebraically with Boolean functions. Thus, knowledge of the type of flip-flops and a list of the Boolean functions of the combinational circuit provide all the information needed to draw the logic diagram of a sequential circuit. The part of the combinational circuit that generates external outputs is described algebraically by the *circuit output functions*.

The part of the circuit that generates the inputs to flip-flops are described algebraically by a set of Boolean functions called *flip-flop input functions*, or sometimes *input equations*.

We shall adopt the convention of using two letters to designate a flip-flop input func tion; the first to designate the name of the input and the second the name of the flip-flop. As an example, consider the following flip-flop input functions:

$$JA = BC'x + B'Cx'$$

$$KA = B + y$$

JA and *KA* designate two Boolean variables. The first letter in each denotes the *J* and *K* input, respectively, of a *JK* flip-flop. The second letter, *A*, is the symbol name of the flip-flop. The right side of each equation is a Boolean function for the corresponding flip-flop input variable. The implementation of the two input functions is shown in the logic diagram of Fig. 6-18. The *JK* flip-flop has an output symbol *A* and two inputs labeled *J* and *K*. The combinational circuit drawn in the diagram is the implementation of the algebraic expression given by the input functions. The outputs of the combinational circuit are denoted by *JA* and *KA* in the input functions and go to the *J* and *K* inputs, respectively, of flip-flop *A*.

From this example, we see that a flip-flop input function is an algebraic expression for a combinational circuit. The two-letter designation is a variable name for an *output* of the combinational circuit. This output is always connected to the input (designated by the first letter) of a flip-flop (designated by the second letter).

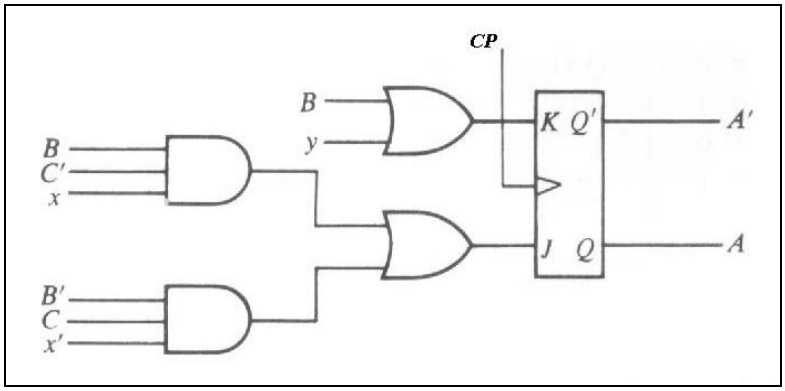


Figure 6-18 – Implementation of the flip-flop input functions $JA = BC'x + B'Cx'$ and $KA = B + y$

Analysis with JK and Other Flip-Flops

It was shown previously that the next-state values of a sequential circuit with *D* flip-flops can be derived directly from The next-state equations. When other types of flip-flops are used, it is necessary to refer to the characteristic table. The next-state values of a sequential circuit that uses any other type of flip-flop such as *JK*, *RS*, or *T* can be derived by following a two-step procedure:

1. Obtain the binary values of each flip-flop input function in terms of the present state and input variables.
2. Use the corresponding flip-flop characteristic table to determine the next state.

To illustrate this procedure, consider the sequential circuit with two *JK* flip-flops *A* and *B* and one input *x*, as shown in figure 6-19. The circuit has no outputs and, therefore, the state table does not

need an output column (The outputs of the flip-flops may he considered as the outputs in this case). The circuit can be specified by the following flip-flop input functions:

JA = B JB = x'
KA = Bx' KB = A'x + Ax' = A⊕X

The state table of the sequential circuit is shown n Table 6-4. First, we derive the binary values listed under the columns labeled flip-flop inputs. These columns are not part of the slate table, but they are needed for the purpose of evaluating the next state as specified in step 1 of the procedure. These binary values are obtained directly from the four input flip-flop functions in a manner similar to that for obtaining a truth table from an algebraic expression. The next state of each flip-flop is evaluated from the corresponding *J* and *K* inputs and the characteristic table of the *JK* flip-flop listed in Table 6-3. There are four cases to consider. When *J* =1 and *K* =0, the next state is 1. When *J* = 0 and *K* = 1, the next state is 0. When *J* = *K* = 0, there is no change of state and the next-state value is the same as the present state. When *J* = *K* = 1, the next-state bit is the complement of the present-state bit. Examples of the last two cases occur in the table when the present state *AB* is 10 and input *x* is 0. *JA* and *KA* are both equal to 0 and the present state of *A* is 1. Therefore, the next state of *A* remains the same and is equal to 1. If the same row of the table, *JB* and *KB* are both equal to 1. Since the present state of *B* is 0, the next state of *B* is complemented and changes to 1.

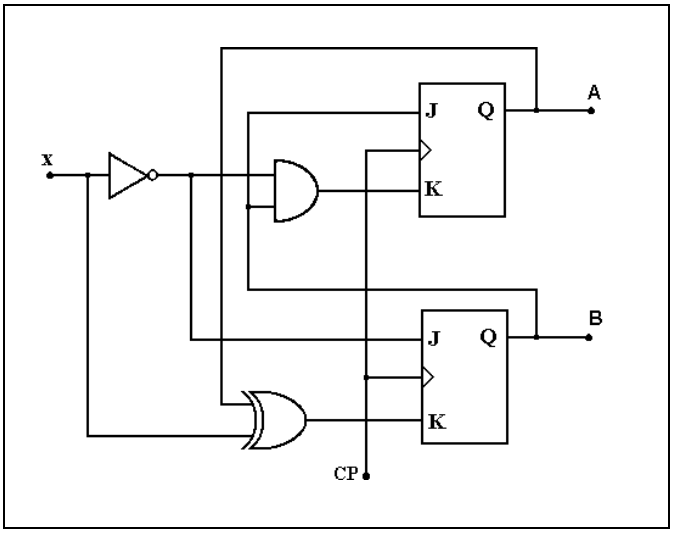


FIGURE 6–19 – Sequetial circuit with *JK* flip-flops

TABLE 6-4 – State Table for Sequential Circuit with JK flip-flops

Present State	Input	Next State	Flip-Flop inputs
A B	x	A B	JÁ KA JB KB
0 0	0	0 1	0 0 1 0
0 0	1	0 0	0 0 0 1
0 1	0	1 1	1 1 1 0
0 1	1	1 0	1 0 0 1
1 0	0	1 1	0 0 1 1
1 0	1	1 0	0 0 0 0
1 1	0	0 0	1 1 1 1
1 1	1	1 1	1 0 0 0

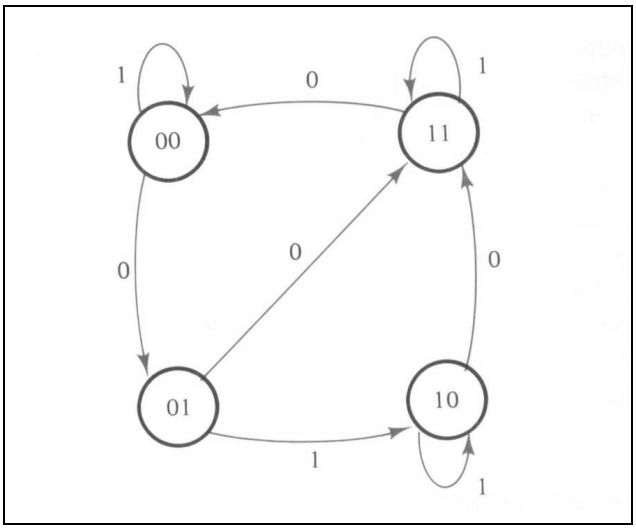


FIGURE 6-20 – State diagram of the circuit of Fig. 6-19

The state diagram of the sequential circuit is shown in Fig. 6-20. Note that since the circuit has no outputs, the directed lines out of the circles are marked with one binary number only to designate the value of input x.

Mealy and Moore Models

The most general model of a sequential circuit has inputs, outputs, and internal states. It is customary to distinguish between two models of sequential circuits: the Mealy model and the Moore model. In the Mealy model, the outputs are functions of both the present state and inputs. In the Moore model, the outputs are a function of the present state only. An example of a Mealy model is shown in Fig. 6-16. Output *y* is a function of both input *x* and the present state of *A* and *B*. The corresponding state diagram shown in Fig. 6-17 has both the input and output values included along the directed lines between the circles. An example of a Moore model is shown in Fig. 6-19. Here the outputs are taken from the flip-flops and are a function of the present state only. The corresponding state diagram in Fig. 6-20 has only the inputs marked along the directed lines. The outputs are the flip-flop states marked inside the circles. The outputs of a Moore model can be a combination of flip-flop variables such as $A \oplus B$. This output is a function of the present state only even though it requires an additional exclusive-OR gate to generate it.

The state table of a Mealy model sequential circuit must include an output section that is a function of both the present state and inputs. When the outputs are taken directly from the flip-flops, the state table can exclude the output section because the outputs are already listed in the present-state columns of the state table. In a general Moore model sequential circuit. there may be an output section, but it will be a function of the present state only.

In a Moore model, the outputs of the sequential circuit are synchronized with the clock because they depend on only flip-flop outputs that are synchronized with the clock. In a Mealy model, the outputs may change if the inputs change during the clock-pulse period. Moreover, the outputs may have momentary false values because of the delay encountered from the time that the inputs charge and the time that the flip-flop outputs change. In order to synchronize a Mealy type circuit, the inputs of the sequential circuit must be synchronized with the clock and the outputs must be sampled only during the clock-pulse transition.

6.5 – STATE REDUCTION AND ASSIGNMENT

The analysis of sequential circuits starts from a circuit diagram and culminates in a state table or diagram. The design of a sequential circuit starts from a set of specifications and culminates in a logic diagram. Design procedures are presented starting from section 6-7. This section discusses certain properties of sequential circuits that may be used to reduce the number of gates and flip-flops during the design.

State Reduction

Any design process must consider the problem of minimizing the cost of the final circuit. The two most obvious cost reductions are reductions in the number of flip-flops and the number of gates. Because these two items seem the most obvious, they have been extensively studied and investigated. In fact, a large portion of the subject of switching theory is concerned with finding algorithms to minimizing the number of flip-flops and gates in sequential circuits.

The reduction of the number of flip-flops in a sequential circuit is referred to as the state-reduction problem. State-reduction algorithms are concerned with procedures for reducing the number of states in a state table while keeping the external input-output requirements unchanged. Since m flip-flops produce 2^m states, a reduction in the number of states may (or may not) result in a reduction in the number of flip-flops. An unpredictable effect in reducing the number of flip-flops is that sometimes the equivalent circuit (with less flip-flops) may require more combinational gates.

We shall illustrate the reed for state reduction with an example. We start with a se quential circuit whose specification is given in the state diagram of Fig. 6-21. In this example, only the input-output sequences are important; the internal states are used merely to provide the required sequences. For this reason, the states marked inside the circles are denoted by letter symbols instead of by their binary values. This is in contrast to a binary counter, where the binary-value sequence of the states themselves are taker as the outputs.

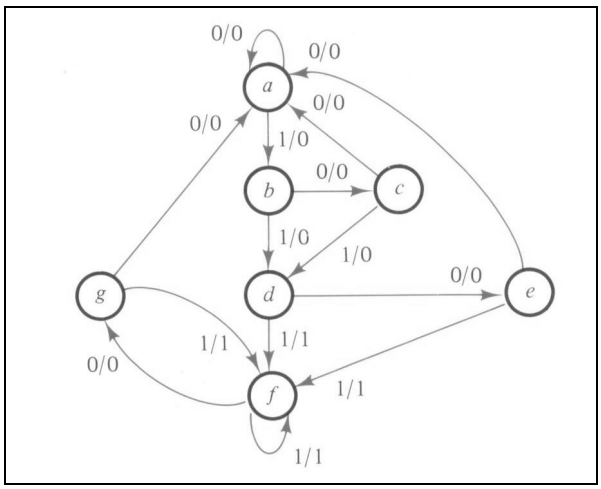


FIGURE 6-21 - State diagram

There are an infinite number of input sequences that may be applied to the circuit; each results in a unique output sequence. As an example, consider the input sequence 01010110100 starting from the initial state *a*. Each input of 0 or 1 produces an output of 0 or 1 and causes the circuit to go to the next state. From the state diagram. we obtain the output and state sequence for the given input sequence as follows: With the circuit in initial state *a*, an input of 0 produces an output of 0 and the circuit remains in state *a*. With present state *a* and input of 1, the output is 0 and the next state is *b*.

With present state *b* and input of 0, the output is 0 and next state is *e*. Continuing this process, we find the complete sequence to be as follows:

state	a	a	b	c	d	e	f	f	g	f	g	a
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

In each column, we have the present state, input value, and output value. The next state is written on top of the next column. It is important to realize that in this circuit, the states themselves are of secondary importance because we are interested only in output sequences caused by input sequences.

Now let us assume that we have found a sequential circuit whose state diagram has less than seven states and we wish to compare it with the circuit whose state diagram is given by Fig. 6-21. If identical input sequences are applied to the two circuits and identical outputs occur for all input sequences, then the two circuits are said to be equivalent (as far as the input-output is concerned) and one may be replaced by the other. The problem of state reduction is to find ways of reducing the number of states in a sequential circuit without altering the input-output relationships.

We shall now proceed to reduce the number of states for this example. First, we need the state table; it is more convenient to apply procedures for state reduction here than in state diagrams. The state table of the circuit is listed in Table 6-5 and is obtained directly from the state diagram of Fig. 6-21.

TABLE 6-5 – State Table

	Next State		Output	
Present State	x=0	x=1	x=0	x=1
a	a	b	0	y
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

An algorithm for the state reduction of a completely specified state table is given here without proof: "Two states are said to he equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state. When two states are equivalent, one of them can be removed without altering the input-output relationships."

We shall apply this algorithm to Table 6-5. Going through the state table, we look for two present states that go to the same next state and have the same output for both input combinations. States *g* and *e* are two such states: they both go to states *a* and *f* and have outputs of 0 and 1 for $x = 0$ and $x = 1$, respectively. Therefore, states *g* and *e* are equivalent; one can, be removed. The procedure of removing a state and replacing it by its equivalent is demonstrated in Table 6-6. The row with present state *g* is crossed out and state *g* is replaced by state *e* each time it occurs in the next-state columns.

TABLE 6-6 – Reducing the State Table

	Next State		Output	
Present State	x=0	x=1	x=0	x=1
a	a	b	0	y
b	c	d	0	0
c	a	d	0	0
d	e	~~f~~ d	0	1
e	a	~~f~~ d	0	1
~~f~~	~~g~~ e	f	0	1
~~g~~	a	f	0	1

Present state *f* now has next states *e* and *f* and outputs 0 and 1 for *x* = 0 and x = 1, respectively. The same next states and outputs appear in the row with present state *d*. Therefore, states *f* and *d* are equivalent; state *f* can be removed and replaced by *d*. The final reduced table is shown in Table 6-7. The state diagram for the reduced table consists of only five states and is shown in Fig. 6-22. This state diagram satisfies the original input-output specifications and will produce the required output sequence for any given input sequence. The following list derived from the state diagram of Fig. 6-22 is for the input sequence used previously. We note that the same output sequence results although the state sequence is different:

state	a	a	h	e	d	e	d	d	e	d	e	a
input	0	1	0	1	0	1	l	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

In fact, this sequence is exactly the same as that obtained for Fig. 6-21, if we replace *g* by *e* and *f* by *d*.

The checking of each pair of states for possible equivalence can be done systematically by means of a procedure that employs an implication table. The implication table consists of squares, one for every suspected pair of possible equivalent states. By judicious use of the table, it is possible to determine all pairs of equivalent states in a state table. The use of the implication table for reducing the number of states in a state table is demonstrated in Section 9-5.

TABLE 6-7 – Reduced State Table

	Next State		Output	
Present State	x=0	x=1	x=0	x=1
a	a	b	0	y
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

It is worth noting that the reduction in the number of states of a sequential circuit is possible if one is interested only in external input-output relationships. When external outputs are taken directly from flip-flops, the outputs must he independent of the number of states before state-reduction algorithms are applied.

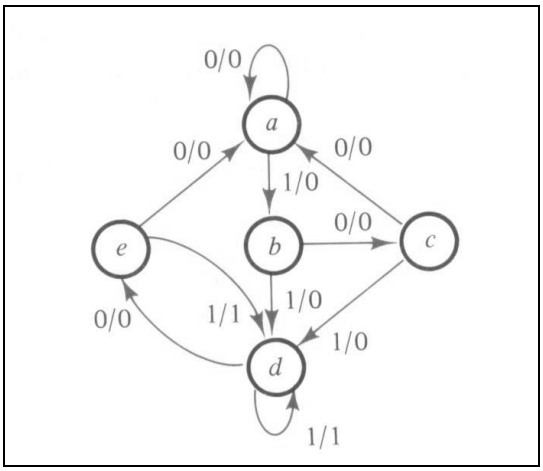


Figure 6-22 – Reduced State Diagram

The sequential circuit of this example was reduced from seven to five states, In either case, the representation of the states with physical components requires that we use three flip-flops, because *m* flip-flops can represent up to 2^m distinct states. With three flip-flops, we can formulate up to eight binary states denoted by binary numbers 000 through 111, with each bit designating the state of one flip-flop. If the state table of Table 6-5 is used, we must assign binary values to seven states; the remaining states is unused. If the state table of Table 6-7 is used, only five states need binary assignment, and we are left with three unused states. Unused states are treated as don't-care conditions during the design of the circuit. Since don't-care conditions usually help in obtaining a simpler Boolean function, it is more likely that the circuit with five states will require fewer combinational gates than the one with seven states. In any case, the reduction from seven to five states does not reduce the number of flip-flops. In general, reducing the number of states in a state table is likely to result in a circuit with less equipment. However, The fact that a state table has been reduced to fewer states does not guarantee a saving in the number of flip-flops or the number of gates.

State Assignment

The cost of the combinational-circuit part of a sequential circuit can be reduced by assign the known simplification methods for combinational circuits. However, there is another factor, known as the state assignment problem, that comes into play in minimizing the combinational gates. State-assignment procedures are concerned with, methods for assigning binary values to states in such a way as to reduce the cost of the combinational circuit that drives the flip-flops. This is particularly helpful when a sequential circuit is viewed from this external input-output terminals. Such a circuit may follow a sequence of internal states, but the binary values of the individual states may be of no consequence as long as the circuit produces the required sequence of outputs for any given sequence of inputs. This does not apply to circuits whose external outputs are taken directly from flip-flops with binary sequences fully specified.

TABLE 6-8 – Three Possible Binary State Assignments

State	Assignment 1	Assignment 2	Assignment 3
a	0 0 1	0 0 0	0 0 0
b	0 1 0	0 1 0	1 0 0
c	0 1 1	0 1 1	0 1 0
d	1 0 0	1 0 1	1 0 1
e	1 0 1	1 1 1	0 1 1

The binary state-assignment alternatives available cam be demonstrated in conjunction with the sequential circuit specified in Table 6-7. Remember that, in this example. the binary values of the states are immaterial as long as their sequence maintains the proper input-output relationships. For this reason, any binary number assignment is satisfactory as long as each state is assigned a unique number. Three examples of possible binary assignments are shown in Table 6-8 for the five states of the reduced table. Assignment 1 is a straight binary assignment for the sequence of states from *a* through *e*. The other two assignments are chosen arbitrarily. In fact, there are 140 different distinct. assignments for this circuit.

Table 6-9 is the reduced state table with binary assignment 1 substituted for the letter symbols of the five states. It is obvious that a different binary assignment will result in a state table with different binary values for the states, whereas the input-output relationships remains the same. The binary form of the state table is used to derive the combinational-circuit part of the sequential circuit. The complexity of the combinational circuit obtained depends on the binary state assignment chosen.

Various procedures have been suggested that lead to a particular binary assignment from the many available. The most common criterion is that the chosen assignment should result in a simple combinational circuit for the flip-flop inputs. However, to date, there are no state-assignment procedures that guarantee a minimal-cost combinational circuit. State assignment is one of the challenging problems of switching theory. The interested reader will find a rich and growing literature

on this topic. Techniques for dealing with the slate-assignment problem are beyond the scope of this book.

TABLE 6-9 – Reduced State Table with Binary Assignment 1

	Next State		Output	
Present State	x=0	x=1	x=0	x=1
0 0 1	0 0 1	0 1 0	0	y
0 1 0	0 1 1	1 0 0	0	0
0 1 1	0 0 1	1 0 0	0	0
1 0 0	1 0 1	1 0 0	0	1
1 0 1	0 0 1	1 0 0	0	1

5) Procedimento de projeto de uma máquina de estados

Para se projetar uma de uma máquina de estados, alguns passos devem ser seguidos:

a) Especificação do comportamento dos estados – é a descrição precisa do comportamento entre entrada e saída desejado para o circuito final. Normalmente vem acompanhada de um diagrama de estados.
b) Construção da tabela de estados, especificando cada estado por letras.
c) Redução do número de estados, quando possível.
d) Atribuição de estados – tendo definido o comportamento do circuito, e o número de estados que ele terá, devemos atribuir valores binários a cada estado.
e) Cálculo do número de flip-flops necessários e escolha do tipo de flip-flop a ser usado.
f) Especificação da função combinacional – construção de uma tabela de excitação de estados que descreve o comportamento do circuito de uma forma binária. Tabelas verdade para as funções das entradas dos flip-flops e para as saídas.
g) Projeto do circuito combinacional.

6.7 – DESIGN PROCEDURE

The design of a clocked sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which the logic diagram can be obtained. In contrast to a combinational circuit, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalent representation, such as a state diagram.

A synchronous sequential circuit is made up of flip-flops and combinational gates. the design of the circuit consists of choosing the flip-flops and then finding a combinational gate structure that, together with the flip-flops, produces a circuit that fulfills the stated specifications. The number of flip-flops is determined from the number of states needed in the circuit. The combinational circuit is derived from the state table by methods presented in this chapter. In fact, once the type and number of flip-flops are determined, the design process involves a transformation from the sequential-circuit problem into a combinational-circuit problem. In this way, the techniques of combinational-circuit design can be applied.

This section presents a procedure for the design of sequential circuits. Although intended to serve as a guide for the beginner, this procedure can be shortened with experience. The procedure is first summarized by a list of consecutive recommended steps:

1. The word description of the circuit behavior is stated. This may he accompanied by a state diagram a timing diagram, or other pertinent information.
2. From the given information about the circuit, obtain the sate table.
3. The number of states may be reduced by state-reduction methods if the sequential circuit can be characterized by input-output relationships independent of the number of states.
4. Assign binary values to each state if the state table obtained in step 2 or 3 contains letter symbols.
5. Determine the number of flip-flops needed and assign a letter symbol to each.
6. Choose the type of flip-flop to be used.
7. From the state table, derive the circuit excitation and output tables.
8. Using the map or any other simplification method, derive the circuit output functions and the flip-flop input functions.
9. Draw the logic diagram.

The word specification of the circuit behavior usually assumes that the reader is familiar with digital logic terminology. It is necessary that the designer use intuition and experience to arrive at the correct interpretation of the circuit specifications, because word descriptions may he incomplete and inexact. However, once such a specification has been set down and the state table obtained, it is possible to make use of the formal procedure to design the circuit.

The reduction of the number of states and the assignment of binary values to the states were discussed in Section 6-5. The examples that follow assume that the number of states and the binary assignment for the states are known. As a consequence steps 3 and 4 of the design will not be considered in subsequent discussions.

It has already been mentioned that *m* flip-flops can represent up to 2^m distinct states. A circuit may have unused binary states if the total number of states is less than 2^m. The unused states are taken as don't-care conditions during the design of the combinational circuit part of the circuit.

The type of flip-flop to be used may be included in the design specifications or may depend on what is available to the designer. Many digital systems are constructed entirely with *JK* flip-flops because they are the most versatile available. When many types of flip-flops are available, it is advisable to use the *D* flip-flop for applications requiring transfer of data (such as shift registers), the *T* type for applications involving complementation (such as binary counters), and the *JK* type for general applications.

The external output information is specified in the output section of the state table. From it we can derive the circuit output functions. The excitation table for the circuit is similar to that of the individual flip-flops, except that the input conditions are dictated by the information available in the present-state and next-state columns of the state table, the method of obtaining the excitation table and the simplified flip-fiop input functions is best illustrated by an example.

We wish to design the clocked sequential circuit whose state diagram is given in Fig. 6-23. the type of flip-flop to be used is *JK*.

The state diagram consists of four states with binary values already assigned. Since the directed lines are marked with a single binary digit without a slash, we conclude that there is one input variable and no output variables. (The state of the flip-flops may be considered the outputs of the circuit, the two flip-flops needed to represent the four states are designated *A* and *B*. The input variable is designated *x*.

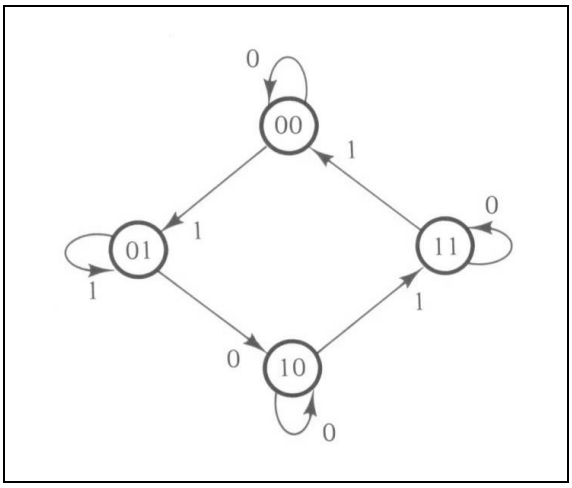


FIGURE 6-23 - State diagram for design example

The state table for this circuit, derived from the state diagram, is shown in Table 6-11. Note that there is no output section for this circuit. We shall now show the procedure for obtaining the excitation table and the combinational gate structure.

The derivation of the excitation table is facilitated if we arrange the state table in a different form. This form is shown in table 6-12, where the present state and input variables are arranged in the form

of a truth table. The next-state value for each present-state and input conditions is copied from Table 6-11 - The excitation table of a circuit is a list of flip-flop input conditions that will cause the required state transitions and is a function of the type of flip-flop used. Since this example specified *JK* flip-flops, we need columns for the *J* and *K* inputs of flip-flops *A* (denoted by *JA* and *KA*) and *B* (denoted by *JB* and *KB*).

TABLE 6-11 – State Table

	Next State	
Present State	x = 0	x = 1
A B	A B	A B
0 0	0 0	0 1
0 1	1 0	0 1
1 0	1 0	1 1
1 1	1 1	0 0

The excitation table for to *JK* flip-flop was derived in Table 6-10(b). This table is now used to derive the excitation table of the circuit. For example, in the first row of Table 6-12, we have a transition for flip-flop. A from 0 in the present state to 0 in the next state. In Table 6-10(b), we find that a transition of states from 0 to 0 requires that input *J* = 0 and input *K* = *X*. So 0 and *X* are copied in the first row under *JA* and *KA*, respectively. Since the first row also shows a transition for flip-flop *B* from 0 in the present state to 0 in the next state, 0 and *X* are copied in the first row under *JB* and *KB*. The second row of table 6-12 shows a transition for flip-flop B from 0 in the present state to 1 in the next state. From Table 6.10(b), we find that a transition from 0 to 1 requires that input *J* = 1 and input *K* = *X*. So 1 and *X* are copied in the second row under *JB* and *KB*, respectively. This process is continued for each row of the table and for each flip-flop, with the input conditions as specified in Table 6-10(b) being copied into the proper row of the particular flip-flop being considered.

Let us now pause and consider the information available in an excitation table such as Table 6-12. We know that a sequential circuit consists of a number of flip-flops and a combinational circuit. Figure 6-24 shows the two *JK* flip-flops needed for the circuit and a box to represent the combinational circuit. From the block diagram, it is clear that the outputs of the combinational circuit go to flip-flop inputs and external outputs (if specified). The inputs to the combinational circuit are the external inputs and the present state values of the flip-flops. Moreover, the Boolean functions that specify a combinational circuit are derived from a truth table that shows the input-output relations of the circuit. The truth table that describes the combinational circuit is available in the excitation table. The combinational circuit inputs are specified under the present state and input columns, and the combinational-circuit outputs are specified under the flip-flop input columns. Thus, an excitation table transforms a state diagram to the truth table needed for the design of the combinational-circuit part of the sequential circuit.

TABLE 6-12 – Excitation Table

Inputs of Combinational Circuit					Outputs of Combinational Circuit			
Present State		Input	Next State		Flip-Flops Inputs			
A	B	x	A	B	JÁ	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

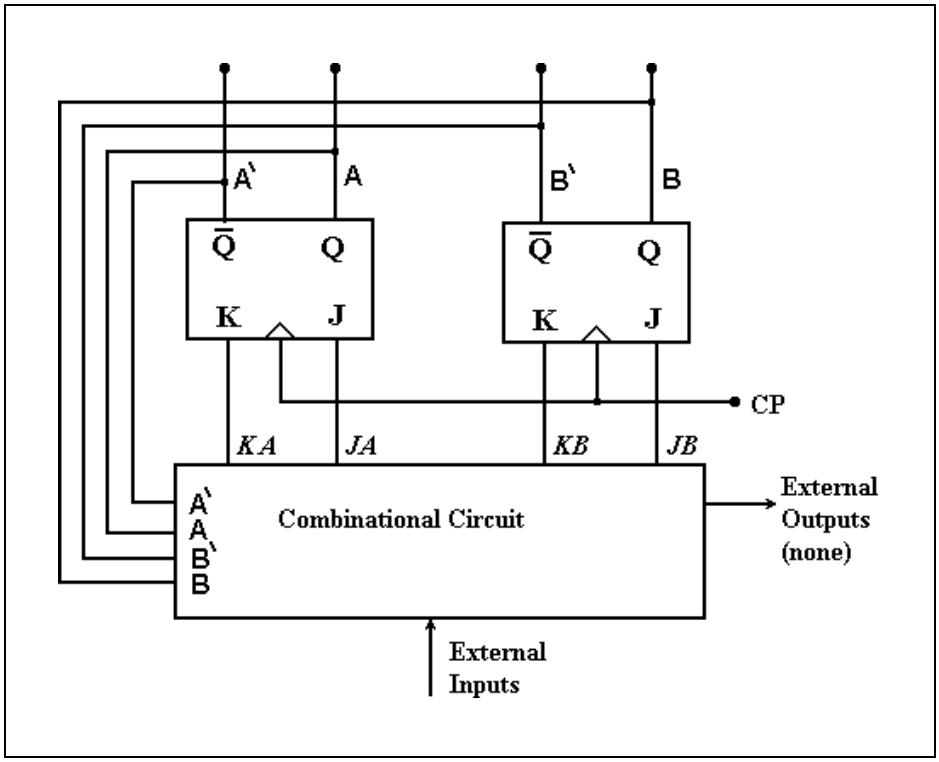


FIGURE 6-24 – Block diagram of sequential circuit

The simplified Boolean functions for the combinational circuit can now be derived. the inputs are the variables *A*, *B*, and *x*; the outputs are the variables *JA*, *KA*, *JB* and *KB*. The information from the truth table is transferred into the maps of Fig. 6-25, where the four simplified flip-flop input functions are derived:

JA = Bx' KA = Bx
JB = x KB = (A⊕x)'

JA	B' x'	B' x	Bx	Bx'
A'	0	0	0	1
A	X	X	X	X

KA	B' x'	B' x	Bx	Bx'
A'	X	X	X	X
A	0	0	1	0

JB	B' x'	B' x	Bx	Bx'
A'	0	1	X	X
A	0	1	X	X

KB	B' x'	B' x	Bx	Bx'
A'	X	X	0	1
A	X	X	1	0

FIGURE 6-25 – Maps for combinational circuit

The logic diagram is drawn in Fig. 6-26 and consists of two flip-flops, two AND gates, one exclusive-NOR gate, and one inverter.

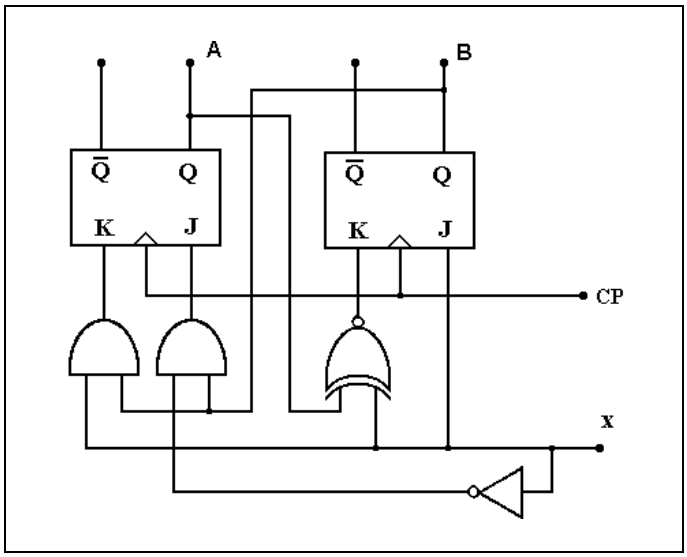


FIGURE 6-26 – Logic diagram of sequential circuit

The excitation table of a sequential circuit with *m* flip-flops, *k* inputs per flip-flop, and *n* external inputs consists of *m* + *n* columns for the present state and input variables and up to 2^{m+n} rows listed in some convenient binary count. The next-state section has *m* columns, one for each flip-flop. The flip-flop input values are listed in *mk* columns, one for each input of each flip-flop. If the circuit contains *j* outputs, the table must include *j* columns. The truth table of the combinational circuit is taken from the excitation table by considering the *m* + *n* present-state and input columns as inputs and the *mk* + *j* flip-flop input values and external outputs as *outputs*.

Design with *D* Flip-Flops

The time it takes to design a sequential circuit that uses 0 flip-flops can be shortened if we utilize the fact that the next state of the flip-flop is equal to its *D* input prior to the application of a clock pulse. This is shown in the excitation table of the *D* flip-flop listed in Table 6-10(c). The excitation table clearly shows that *D* = *Q*(t + 1), which means that the next-state values in the state table specify the *D* input conditions directly, so there is no need for an excitation table as required with other types of flip-flops.

The design procedure with D flip-flops will be demonstrated by means of an example. We wish to design a clocked sequential circuit that operates according to the state table shown in Table 6-13. This table is the same as the state table part of Table 6-12 except for an additional column that includes an output *y*. For this case, it is not necessary to include the excitation table for flip-flop inputs *DA* and *DR* since *DA* = *A*(t + 1) and *DB* = *B*(t + 1). The flip-flop input functions can be obtained directly from the next-state columns of *A* and *B* and expressed in sum of minterms as follows:

$DA(A, B, x) = \sum(2,4,5,6)$

$DB(A, B, x) = \sum(1, 3, 5, 6)$

$y(A, B, x) = \sum(1, 5)$

where *A* and *B* are the present-state values of flip-flops *A* and *B*, *x* is the input, and *DA* and *DB* are the input functions. The minterms for output *y* are obtained from the output column if the state table.

TABLE 6-13 – State Table for Design with *D* Flip-Flop

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

The Boolean functions are simplified by means of the maps plotted in Fig. 6-27. The simplified functions are

DA = AB' + Bx'
DB = A' x + B'x + Abx'
y = B' x

The logic diagram of the sequential circuit is shown in Fig. 6-28.

DA	B' x'	B' x	Bx	Bx'
A'	0	0	0	1
A	1	1	0	1

DB	B' x'	B' x	Bx	Bx'
A'	0	1	1	0
A	0	1	0	1

y	B' x'	B' x	Bx	Bx'
A'	0	1	0	0
A	0	1	0	0

FIGURE 6-27 – Maps for combinational circuit

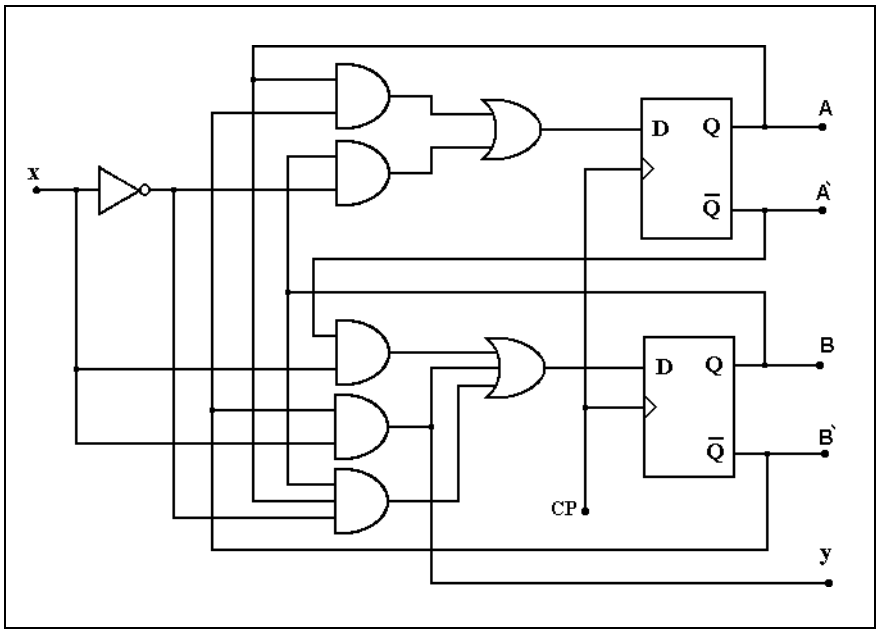


FIGURE 6-28 – Logic Diagram of a sequential circuit with *D* flip-flops

9.5 – REDUCTION OF STATE AND FLOW TABLES

The procedure for reducing the number of internal states in an asynchronous sequential circuit resembles the procedure that is used for synchronous circuits. An algorithm for state reduction of a completely specified state table is given in Section 6-5. We will review this algorithm and apply it to a state-reduction method that uses an implication table. The algorithm and the implication table will then be modified to cover the state reduction of incompletely specified state tables. This modified algorithm will be used to explain the procedure for reducing the flow table of asynchronous sequential circuits.

Implication Table

The state-reduction procedure for completely specified state tables is based on the algorithm that two states in a state table can be combined into one if they can be shown to be equivalent. Two states are equivalent if for each possible input, they give exactly the same output and go to the same next states or to equivalent next states. Table 6-6 shows an example of equivalent states that have the same next states and outputs for each combination of inputs. There are occasions when a pair of states do not have the same next states, but, nonetheless, go to equivalent next states. Consider, for example, the state table shown in Table 9-3. The present states *a* and *b* have the same output for the same input. Their next states are *c* and *d* for $x = 0$ and *b* and *a* for $x = 1$. If we can show that the pair of states (*c*, *d*) are equivalent, then the pair of states (*a*, *b*) will also be equivalent because they will have the same or equivalent next states. When this relationship exists, we say that (*a*, *b*) imply (*c*, *d*). Similarly, from the last two rows of Table 9-3, we find that the pair of states (*c*, *d*) imply the pair of states (*a*, *b*). The characteristic of equivalent states is that if (*a*, *b*) imply (*e*, *d*) and (*c*, *d*) imply (*a*, *b*), then both pairs of states are equivalent; that is, *a* and *b* are equivalent as well as *e* and *d*. As a consequence, the four rows of Table 9-3 can be reduced to two rows by combining *a* and *b* into one state and *e* and *d* into a second state.

TABLE 9-3 – State Table to Demonstrate Equivalent States

	Next State		Output	
Present State	x = 0	x = 1	x = 0	x = 1
a	c	b	0	1
b	d	a	0	1
c	a	d	1	0
d	b	d	1	0

The checking of each pair of states for possible equivalence in a table with a large number of states can be done systematically by means of an implication table. The implication table is a chart that consists of squares, one for every possible pair of states, that provide spaces for listing any possible implied states. By judicious use of the table, it is possible to determine all pairs of equivalent states. The state table of Table 9-4 will be used to illustrate this procedure. The implication table is shown in Fig. 9-22. On the left side along the vertical are listed all the states defined in the state table except the first, and across the bottom horizontally are listed all the states except the last. The re sult is a display of all possible combinations of two states with a square placed in the intersection of a row and a column where the two states can be tested for equivalence.

Two states that are not equivalent are marked with a cross (x) in the corresponding square, whereas their equivalence is recorded with a check mark (v). Some of the squares have entries of implied states that must be further investigated to determine whether they are equivalent or not. The step-by-step procedure of finding in the squares is as follows. First, we place a cross in any square corresponding to a pair of states whose outputs are not equal for every input. In this case, state *c* has a different output than any other state, so a cross is placed in the two squares of row *c* and the four squares of column *e*. There are nine other squares in this category in the implication table.

Next, we enter in the remaining squares the pairs of states that are implied by the pair of states representing the squares. We do that starting from the top square in the left column and going down and then proceeding with the next column to the right. From the state table, we see that pair (*a*, *b*)

imply (*d*, *e*), so (*d*, *e*) is recorded in the square defined by column *a* and row *b*. We proceed in this manner until the entire table is completed. Note that states (*d*, *e*) are equivalent because they go to the same next state and have the same output. Therefore, a cheek mark is recorded in the square defined by column *a* and row *e*, indicating that the two states are equivalent and independent of any implied pair.

TABLE 9-4 – State Table to be Reduced

	Next State		Output	
Present State	x = 0	x = 1	x = 0	x = 1
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0

The next step is to make successive passes through the table to determine whether any additional squares should be marked with a cross. A square in the table is crossed out if it contains at least one implied pair that is not equivalent. For example, the square defined by *a* and *f* is marked with a cross next to *c*, *d* because the pair (*c*, *d*) defines a square that contains a cross. This procedure is repeated until no additional squares can be crossed out. Finally, all the squares that have no crosses are recorded with check marks. These squares define pairs of equivalent states. In this example, the equivalent states are

(a, b) (d, e) (d, g) (e, g)

We now combine pairs of states into larger groups of equivalent states. The last three pairs can be combined into a set of three equivalent states (d, e, g) because each one of states in the group is equivalent to the other two. The final partition of the states consists of the equivalent states found from the implication table together with all the remaining states in the state table that are not equivalent to any other state.

(a, b) (c) (d, e, g) (f)

This means that table 9-4 can be reduced from seven states to four states, one for each member of the above partition. the reduced table is obtained by replacing state *b* by *a* and states *e* and *g* by *d*. The reduced state table is shown in Table 9-5.

b	d,e √					
c	X	X				
d	X	X	X			
e	X	X	X	√		
f	c,d X	c,e X a,b	X	X	X	
g	X	X	X	d,e √	d,e √	X
	a	b	c	d	e	f

FIGURE 9-22 – Implication table

TABLE 9-5 – Reduced State Table

	Next State		Output	
Present State	x = 0	x = 1	x = 0	x = 1
a	d	a	0	0
c	d	f	0	1
d	a	d	1	0
f	c	a	0	0

Merging of the Flow Table

There are occasions when the state table for a sequential circuit is incompletely specified. This happens when certain combinations of inputs or input sequences may never occur because of external or internal constraints. In such a case, the next states and outputs that should have occurred if all inputs were possible are never attained and regarded as don' t-care conditions. Although synchronous sequential circuits may sometimes be represented by incompletely specified state tables, our interest here is with asynchronous sequential circuits where the primitive flow table is always incompletely specified.

Incompletely specified states can be combined to reduce the number of states in the flow table. Such states cannot be called equivalent, because the formal definition of equivalence requires that all outputs and next states be specified for all inputs instead, two incompletely specified states that can be combined are said to be *compatible*. Two states are compatible if for each possible input they have the same output whenever specified and their next states are compatible whenever they are specified. All don't-care conditions marked with dashes have no effect when searching for compatible states as they represent unspecified conditions.

The process that must be applied in order to find a suitable group of compatibles for the purpose of merging a flow table can be divided into three procedural steps.

1. Determine all compatible pairs by using the implication table.
2. Find the maximal compatibles using a merger diagram.
3. Find a minimal collection of compatibles that covers all the states and is closed.

The minimal collection of compatibles is then used to merge the rows of the flow table. We will now proceed to show and explain the three procedural steps using the primitive flow table from the design example in the previous section.

Compatible Pairs

The procedure for finding compatible pairs is illustrated in Fig. 9-23. The primitive flow table in (a) is the same as Fig. 9-16. The entries in each square represent the next state and output. The dashes represent the unspecified states or outputs. The implication table is used to find compatible states just as it is used to find equivalent states in the completely specified case. The only difference is that when comparing rows, we are at liberty to adjust the dashes to fit any desired condition.

Two states are compatible if in every column of the corresponding rows in the flow table, there are identical or compatible states and if there is no conflict in the output values. For example, rows *a* and *b* in the flow table are found to be compatible, but rows *a* and *f* will be compatible only if *c* and *f* are compatible. However, rows *c* and *f* are not compatible because they have different outputs in the first column. This information is recorded in the implication table. A cheek mark designates a square whose pair of states are compatible. Those states that are not compatible are marked with a cross. The remaining squares are recorded with the implied pairs that need further investigation.

Once the initial implication table has been filled, it is scanned again to cross out the squares whose implied states are not compatible. The remaining squares that contain check marks define the compatible pairs. In the example of Fig. 9-23. the compatible pairs are

(a,b) (a,c) (a,d) (b,e) (b,f) (c,d) (e,f)

(a) Primitive flow table (b) Implication table

FIGURE 9-23 – Flow and implication tables

Maximal Compatibles

Having found all the compatible pairs, the next step is to find larger sets of states that are compatible. the *maximal compatible* is a group of compatibles that contains all the possible combinations of compatible states. the maximal compatible can he obtained from a merger diagram, as shown in Fig. 9-24. The merger diagram is a graph in which each state is represented by a dot placed along the circumference of a circle. Lines are drawn between any two corresponding dots that form a compatible pair. All possible compatibles can be obtained from the merger diagram by observing the geometrical patterns in which states are connected to each other. An isolated dot represents a state that is not compatible to any other state. A line represents a compatible pair. A triangle constitutes a compatible with three states. An *n*-state compatible is represented in the merger diagram by an *n*-sided polygon with all its diagonals connected.

The merger diagram of Fig. 9-24(a) is obtained from the list of compatible pairs derived from the implication table of Fig. 9-23. There are seven straight lines connecting the dots, one for each compatible pair. The lines form a geometrical pattern consisting of two triangles connecting (*a*, *c*, *d*) and (*b*, *e*, *f*) and a line (*a*, *b*). The maximal compatibles are

(a, b) (a, c, d) (b, e, f)

Figure 9-24(b) shows the merger diagram of an 8-state flow table. The geometrical patterns are a rectangle with its two diagonals connected to form the 4-state compatible (*a*, *b*, *e*, *f*), a triangle (*b*, *c*, *h*), a line (*c*, *d*), and a single state *g* that is not compatible to any other state. The maximal compatibles are

(a, b, e, f) (b, c, h) (c, d) (g)

The maximal compatible set can be used to merge the flow table by assigning one row in the reduced table to each member of the set. However, quite often the maximal compatibles do not necessarily constitute the set of compatibles that is minimal. In many cases, it is possible to find a smaller collection of compatibles that will satisfy the condition for row merging.

Closed Covering Condition

The condition that must be satisfied for row merging is that the set of chosen compatibles must *cover* all the states and must be *closed*. The set will cover all the states if it includes all the states of the original state table. the closure condition is satisfied if there are no implied states or if the implied states are included within the set. A closed set of compatibles that covers all the states is called a *closed covering*. The closed-covering condition will be explained by means of two examples.

Consider the maximal compatibles from Fig. 9-24(a). If we remove (*a*, *b*), we are left with a set of two compatibles:

(a, c, d) (b, e, f)

All six states from the flow table in Fig. 9-23 are included in this set. This satisfies the covering condition. There are no implied states for (a, c) , (a, d), (c, d), (b, e), (b, f), and (e, f), as seen from the implication table of Fig. 9-23(b). So the closure condition is also satisfied. Therefore, the primitive flow table can be merged into two rows, one for each of the compatibles. The detailed construction of the reduced table for this particular example was done in the previous section and is shown in Fig. 9-17(b).

the second example is from a primitive flow table (not shown) whose implication table is given in Fig. 9-25(a). The compatible pairs derived from the implication table are

(a, b) (a, d) (b, c) (c, d) (c, e) (d, e)

From the merger diagram of Fig. 9-25(b), we determine the maximal compatibles:

(a, b) (a, d) (b, c) (c, d, e)

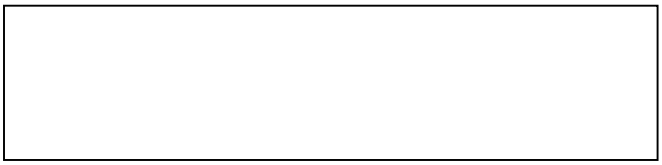


FIGURE 9-25 – Choosing a set of compatibles

If we choose the two compatibles

(a, b) (c, d, e)

the set will cover all five states of the original table. The closure condition can be checked by means of a closure table, as shown in Fig. 9-25(c). The implied pairs listed for each compatible are taken directly from the implication table. the implied states for (*a*, *b*) are (*b*, *c*). But (*b*, *c*) is not included in the chosen set of (*a*, *b*) (*c*, *d*, *e*), so this set of compatibles is not closed. A set of compatibles that will satisfy the closed covering condition is

(a, d) (b, c) (c, d, e)

the set is covered because it contains all five states. Note that the same state can be repeated more than once. The closure condition is satisfied because the implied states are (*b*, *e*) (*d*, *e*) and (*a*, *d*), which are included in the set. The original flow table (not shown here) can be reduced from five rows to three rows by merging rows *a* and *d*, *b* and *c*, and *c*, *d*, and *e*. Note that an alternative satisfactory choice of closed-covered compatibles would be (*a*, *b*) (*b*, c) (*d*, *e*). In general, there may be more than one possible way of merging rows when reducing a primitive flow table.