**NOTICE:** Do NOT distribute. This is copyrighted material. **Only** Kettering Students taking Digital Systems I may use it *only* during the Spring Academic Term of 2020. You may create one paper copy of it for your OWN personal use.

# Chapter 8

Spring 2010 Edition

# Design of Sequential Circuits

A *SUMMARY* of what you learned in Chapter 7:

The distinguishing characteristic of sequential circuits as opposed to combinational circuits

rs-latches

Clock: a global synchronizing signal

Clocked RS-latches

D-latches

Edge-triggered memory cells or flip-flops

D flip-flops: characteristic table, characteristic equation, excitation table

T flip-flops: characteristic table, characteristic equation, excitation table

JK flip-flops: characteristic table, characteristic equations, excitation table

State of a single memory cell

Finite state machines

State of a finite state machine

Input, output, state and excitation variables

The concept of analysis of finite state machines

An analysis methodology for finite state machines

- Excitation equations and output equations
- Excitation maps and output maps
- Partial transition tables, transition tables and state tables
- Transition graphs and state graphs.

## Introduction

The analysis of a finite-state machine (to eventually obtain a state diagram) was covered in Chapter 7. Remember that a state diagram specifies the next alphanumeric state of the corresponding state machine for each (current) alphanumeric state and every input combination. Starting from the lowest level of design (a circuit diagram) you learned how to reach a state diagram for the circuit under consideration. It may not be possible to reach a natural-language description (for the state machine in hand) from the corresponding state diagram, unless descriptive names are unambiguously assigned to the input and output variables of the state machine.

In this chapter a design methodology for finite-state machines is presented, which starts with a natural language description of the problem and ends with a digital circuit. The first step is to translate the problem description (in natural language) into a state diagram, which is normally considered the most challenging step of the design procedure. The remaining steps are a trivial part of what today's gigantic CAD tools are able to perform automatically. These steps are the same that we took in the analysis procedure in Chapter 7, but in reverse order and with two major exceptions, namely steps 3 and 4 on page 10.

Copyright © 2005-2020 N. Tabrizi

The first design step is elaborated on in Example 1 to Example 8, and then the following design steps are closely examined in Example 9 to Example 11. At the end of this chapter finite-state machines are classified into two different categories, namely Mealy and Moore machines.

**Example 1.** Develop a state diagram for the alarm control system of a chemical plant such that starting with the reset state (in which both the temperature and pressure are normal), when the pressure reaches its limit (P=1) *after* the temperature goes beyond a threshold (T=1), the alarm goes off (A=1) and continues to sound while both the temperature and pressure are high. The alarm should turn off if the pressure goes back to normal. However, it must sound again should the pressure rise again while the temperature remains high. When the temperature becomes normal, the alarm turns off and the above sequence is repeated.

Figure 1 shows a preliminary timing diagram for two different situations: in Figure 1*a* the temperature rises first, and then the pressure goes up as well, making the alarm sound. Figure 1*b* shows the opposite situation where the pressure goes up first. Therefore, no matter whether or not the temperature rises later, the alarm will not sound. Pay attention to two specific instants of time, $t_0$ and $t_1$, where identical inputs (P = T = 1) generate different outputs (A = 1 and A = 0, respectively). Therefore, a combinational circuit cannot control this alarm anymore, as identical inputs cannot produce different outputs in this category of digital circuits.

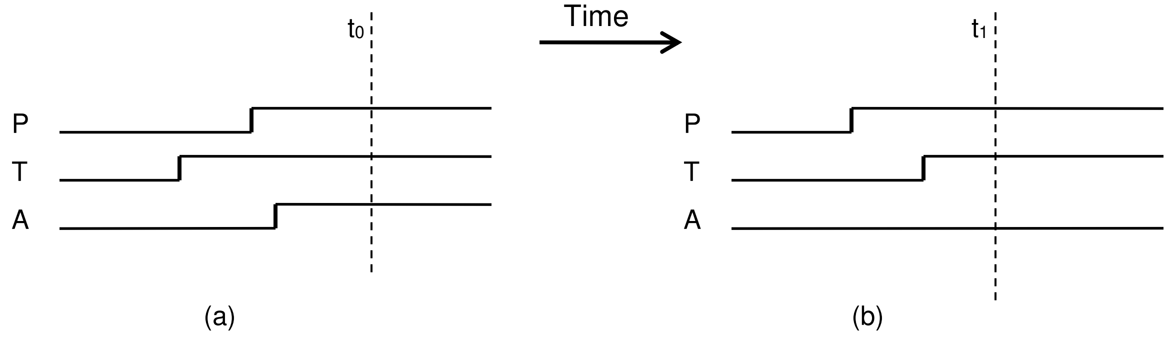


**Figure 1. Non-rigorous timing diagram for Example 1: (a) T goes up first, (b) P goes up first**

Let's develop a state diagram for this alarm control system. Assume that when the power is turned on, the system is first forced to a reset state, r, by asserting an external reset signal (the power-on reset) as shown in Figure 2*a*. This is a common assumption made in many real systems.

As shown in Figure 2*b* the state machine will stay in state r as long as both the temperature and pressure are normal (P = 0 and T = 0).

If the pressure goes up while the temperature is normal or if both go up together (PT = 1X), then the machine jumps to a wait state, w, as shown in Figure 2*c*, and will remain there until both the temperature and pressure become normal (PT = 00). This event will take the machine back to the reset state, r.

Now the question is "what is the difference between state r and state w"? Or "why do we need a second state in addition to r?" And the answer is "what is remembered in state r is different from what is remembered in state w". More specifically, state r remembers that the last input was PT = 00, but state w remembers that either input PT = 10 or input PT = 11 was applied to the most recent reset state and since then input PT = 00 has not been seen yet; therefore, the alarm will not go off while the machine is in state w. (So, we may call it a *passive* state.)

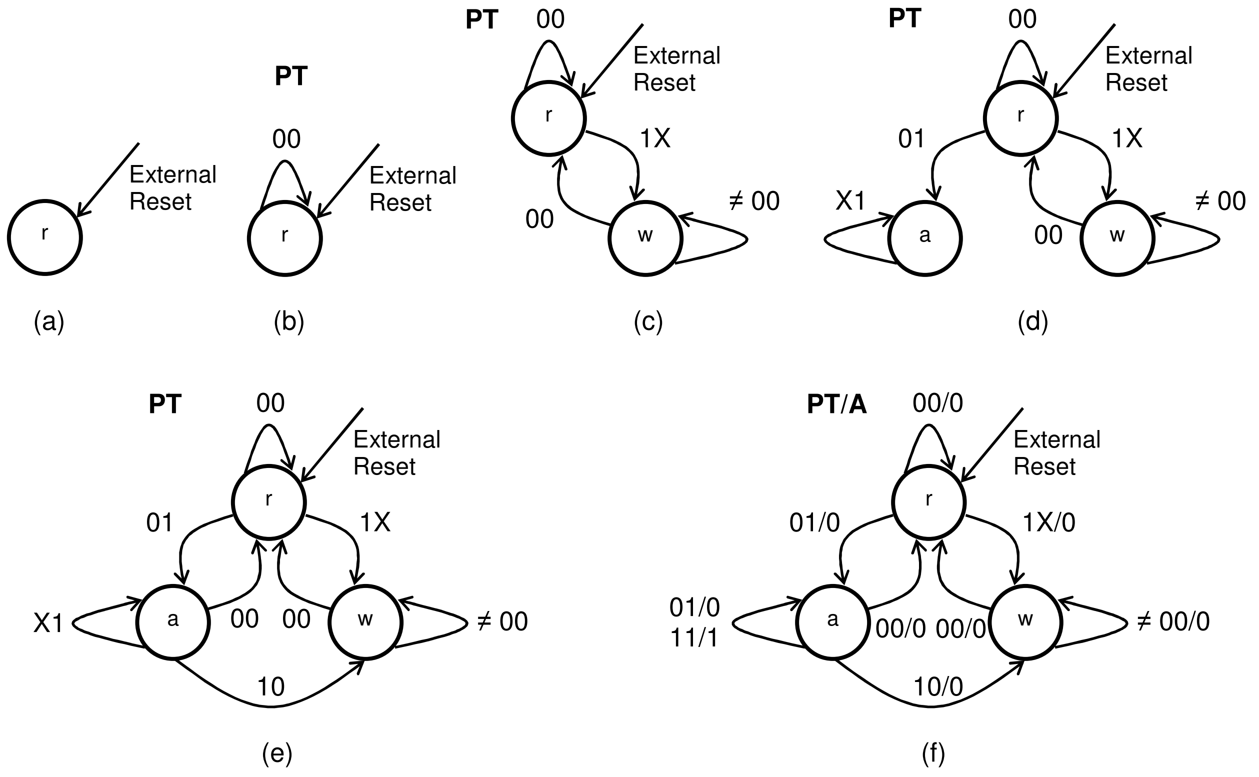


**Figure 2. Developing a state diagram given in Example 1**

Suppose that the machine is in state r and then the temperature goes up while the pressure is normal. This is an input combination that eventually might sound the alarm; therefore, the machine should remember this event by jumping to an *active* state, a (as opposed to passive state w), in which the machine will stay as long as the temperature is high, as shown in Figure 2*d*.

As soon as the temperature becomes normal, the machine must leave state a; but the destination state is determined by the pressure; if the temperature becomes normal while the pressure is high, the machine will jump to the wait state; otherwise, (if the pressure is normal as well), the destination state would be the reset state, as depicted in Figure 2*e*.

As the last step in developing a state diagram, output information is also added as illustrated in Figure 2*f* . We assume that the alarm sounds with a 1. According to the problem description the alarm must sound only if the temperature and pressure both are high while the machine is in state a. Notice that since in this state the two input combinations represented by PT = X1 (see Figure 2*e*) produce two different output values, we have to split PT = X1 into PT = 01 and PT = 11, as shown in Figure 2*f*.

Figure 3 illustrates a timing diagram for this alarm system. Pay close attention to when each input signal undergoes a transition. Each of these transitions (effect) happens right after a clock edge (cause) arrives, as highlighted in the upper-left corner of Figure 3; i.e., all signal transitions on T (as well as P) in this example are *synchronized* with the clock signal. Such signals are called *synchronous*, otherwise *asynchronous*. Asynchronous signals may change any time during a period of the clock signal, hence are independent of the clock signal. *Synchronizers* are special circuits to synchronize asynchronous signals. A well-known catastrophic phenomenon called *metastability* is a (rare but possible) side effect of utilizing asynchronous signals, hence synchronizers. This topic is out of the scope of this book. Interested readers may wish to consult text books on *advanced logic design*

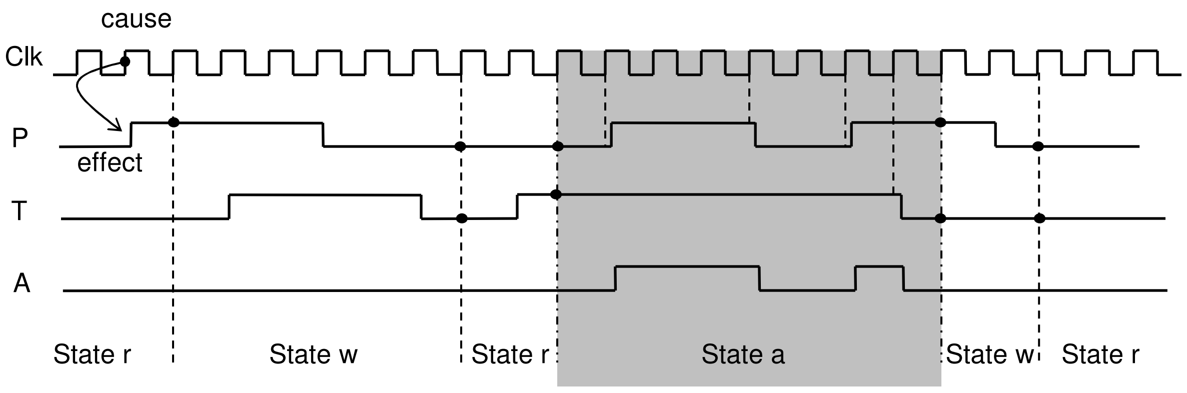


**Figure 3. Timing diagram for Example 1**

**Example 2.** In this example we make one change to Example 1: the input combination before PT = 01 is unimportant. In other words, now the alarm must sound if the pressure goes up after the temperature goes up, no matter what the inputs were before this specific input combination. The alarm should turn off if the pressure goes back to normal. However, it must sound again should the pressure rises again while the temperature remains high. When the temperature becomes normal, the system is reset, the alarm turns off and the above sequence is repeated.

Figure 4 illustrates a state diagram for this alarm control system. While PT ≠ 01, i.e., while the temperature is normal (T = 0) or if the temperature and pressure both go up together (PT = 11), the system stays in the reset state (because now these changes are not supposed to affect the circuit's reaction). In other words, there is a wait loop in this state that is active as long as PT = 00, PT = 10 or PT = 11.

If the temperature goes up while the pressure is normal (PT = 01), the system leaves the reset state and enters a new state, a, as shown in Figure 4. Now the question is "what is the difference between state r and state a?" or "why do we need a second state in addition to r?" And the answer is "in state a the system remembers something (from the past) that is not remembered in state r". More specifically, in state a the system remembers that the temperature went up while the pressure was normal; so that from now on if the pressure goes up as well, the alarm will sound. Additionally, the system will remain in state a as long as the temperature is high, no matter what the pressure is, as modeled in Figure 4.

According to the problem description, as soon as the temperature goes back to normal, no matter whether or not the pressure is high, the system is reset; in other words, the system's memory is cleared waiting for a new sequence of events.

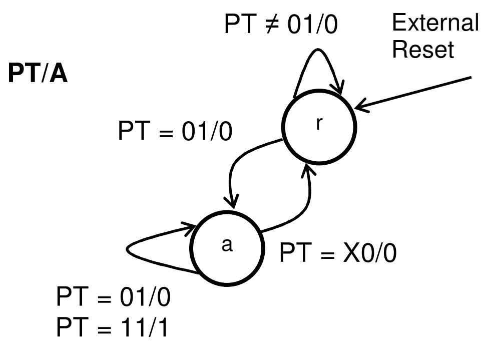


**Figure 4. State diagram in Example 2**

Output information is also shown in Figure 4. Again we assume that the alarm sounds with a 1. According to the problem description the alarm must sound only if the temperature and pressure both are high while the machine is in state a.

Let's take a second look at Figure 4 and see how it reads:

As a general rule in reading any state graph, current state will not change as long as no clock edge arrives.

If the state machine is in state r, and the input vector, PT, is 00, 10 or 11, then the output will be 0. Additionally, the machine will stay in this state even if a clock edge arrives.

If the state machine is in state r and the input vector, PT, is 01, then the output will be 0. Now, if a clock edge arrives, the state machine will go to state a.

If the state machine is in state a, and the input vector, PT, is 01, then the output will be 0. Additionally, the machine will stay in this state even if a clock edge arrives.

If the state machine is in state a, and the input vector, PT, is 11, then the output will be 1. Additionally, the system will stay in this state even if a clock edge arrives.

If the system is in state a, and the input vector, PT, is 00 or 10, then the output will be 0. Now, if a clock edge arrives, the system will return to state r.

**Example 3.** In this example we make two changes to Example 1: 1) the machine leaves the active state when the temperature and pressure both become normal; and 2) in the active mode now the alarm remains on if the temperature, the pressure or both are high.

Figure 5 illustrates a state diagram for this control system. As shown in this figure, the machine leaves state a only when the pressure and temperature both become normal. Now the destination is necessarily state r. Additionally, at least one high input will sound the alarm while the machine is in state a. The rest of this state graph has not undergone any changes. Figure 6 shows a timing diagram for this alarm control system.

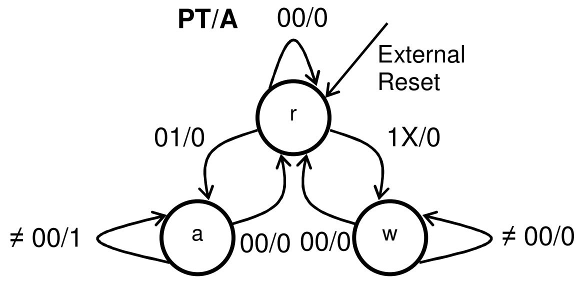


**Figure 5. State diagram for Example 3**

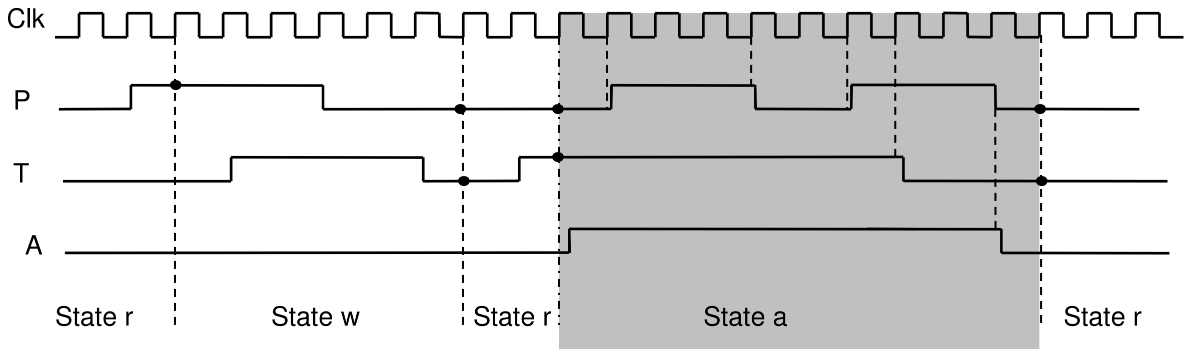


**Figure 6. Timing diagram for Example 3**

**Example 4.** In this example we make the same change to Example 3 that we made to Example 1 and reached Example 2; so that now the input combination before PT = 01 is unimportant. In other words, now the alarm must sound if the pressure goes up after the temperature goes up, no matter what the inputs were before this specific input combination. From now on the alarm must continue to sound if the temperature, pressure or both are high. When the temperature and pressure both become normal the system is reset, the alarm turns off and the above sequence is repeated.

A state graph for this control system is illustrated in Figure 7. The only difference between this graph and the one shown in Figure 4 is that now only one input combination, PT = 00, may cause a jump back to state r; the other input combination, PT = 10, now keeps the machine in state a with an asserted output.

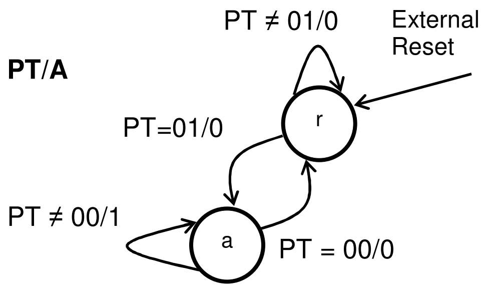


**Figure 7. State diagram in Example 4**

**Example 5.** Repeat Example 1, but now in order to sound the alarm the pressure has to go up *at least four cycles* after the temperature has become high.

A state diagram for this alarm system is illustrated in Figure 8. Again the system leaves state r with the first pattern of 01 on the PT lines, but it does not enter state a until this pattern has been repeated three more times. Input combinations PT = 00 and PT = 1X occurring in states r1, r2 or r3 return the system to the reset state and wait state, respectively. A timing diagram for this state machine is shown in Figure 9.

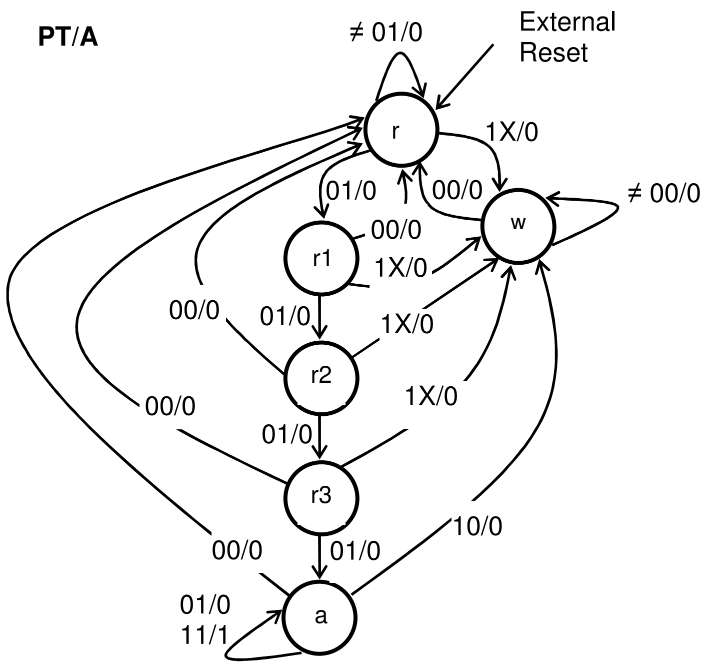


**Figure 8. State diagram for Example 5**

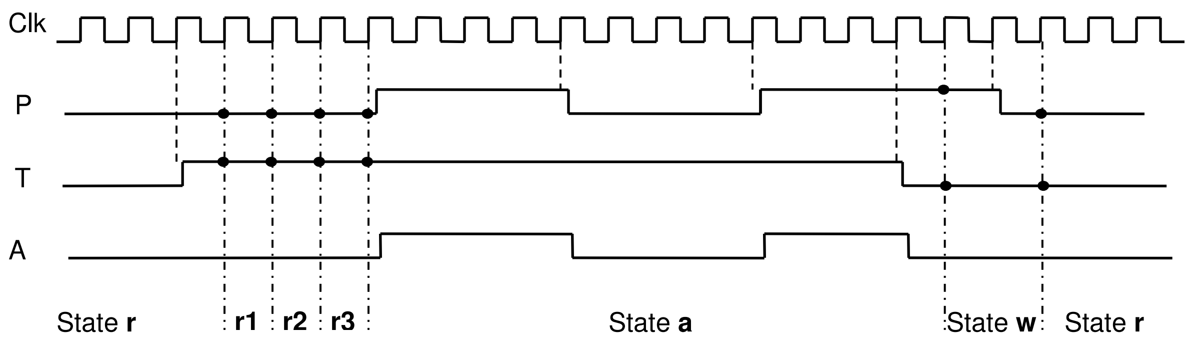


**Figure 9. Timing diagram for Example 5**

**Example 6.** Repeat Example 2, but now in order to sound the alarm the pressure has to go up *at least four cycles* after the temperature has become high.

A state diagram for this alarm system is illustrated in Figure 10. Again the system leaves state r with the first pattern of 01 on the PT lines, but it does not enter state a until this pattern has been repeated three more times. Each of the input combinations PT = 00, PT = 10 and PT = 11 occurring in states r1, r2 or r3 will return the machine to the reset state.

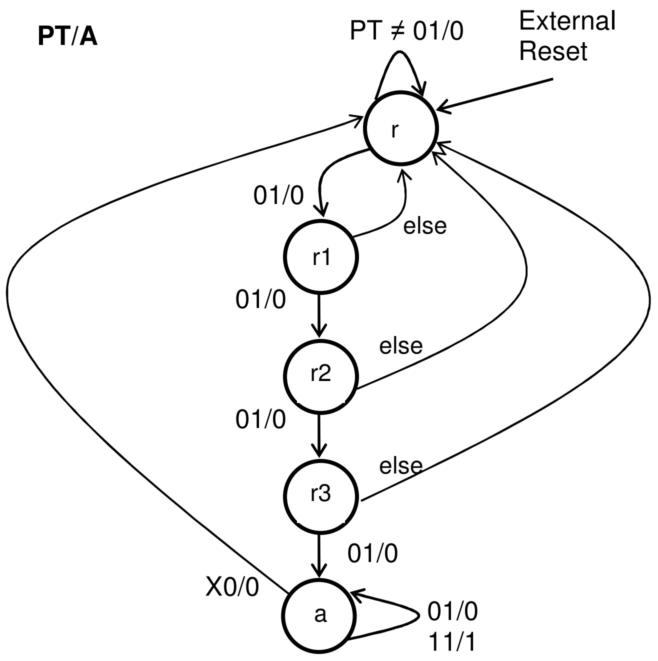


**Figure 10. State diagram for Example 6**

**Example 7.** Develop a state diagram for an LED controller such that when a push-button is pushed (P = 1) and then released (P = 0) the OFF light (L = 0) will turn ON (L = 1) and remains ON until the push-button is pushed and released again. A top-level block diagram and also a typical timing diagram for this system are shown in Figure 11 and Figure 12, respectively[1].

[1] Here we assume that the push-button is bounce- free, i.e., once the button is pushed and contact is made, contact will not be broken (even momentarily) until the button is released. And, once the button is released and contact is broken, contact will not be made (even momentarily) until the button is pushed again. In case of a bouncy push-button we may first use a switch debouncer (see Lab Exercise 07).

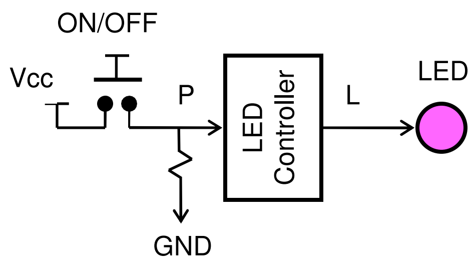


**Figure 11. Block diagram for LED system in Example 7**

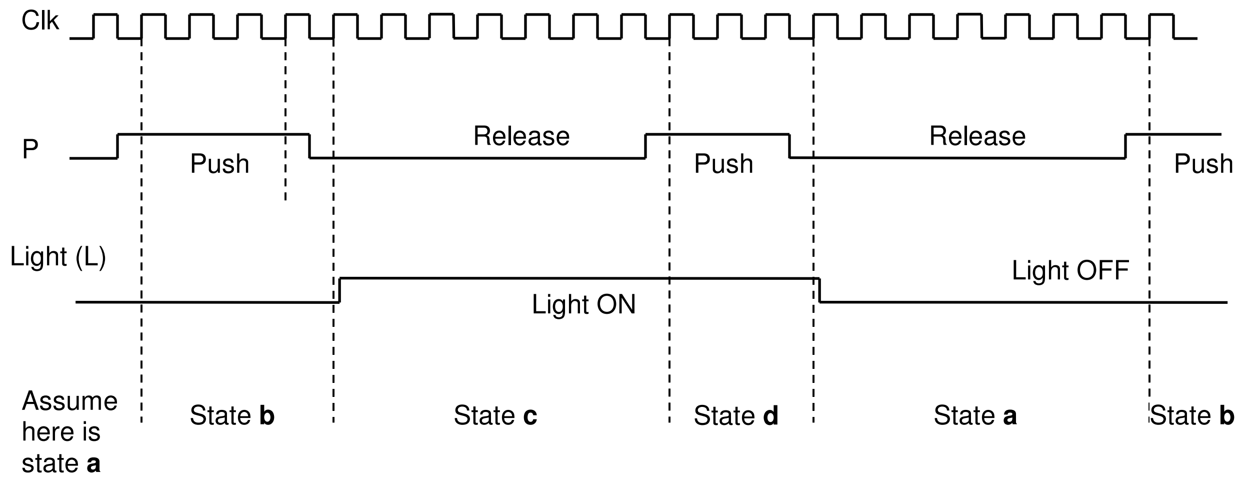


**Figure 12. Timing diagram for a typical operation of LED controller in Example 7**

Notice that push and release periods each may take an unknown number of clock cycles. Figure 13 shows a state diagram for this system.

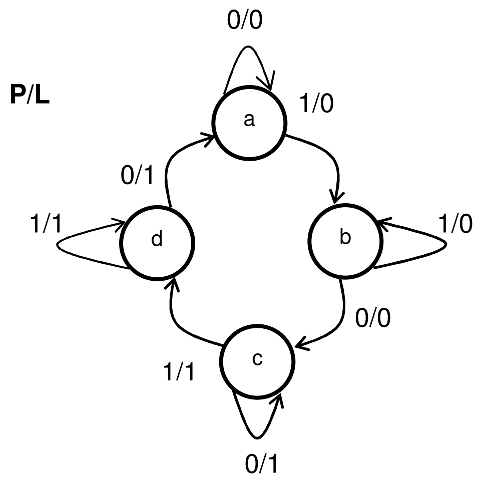


**Figure 13. State diagram for LED controller in Example 7**

Starting with state a in which the light is off, the state of the controller remains unchanged as long as the button has not been pushed. With the first clock edge (after the button is pushed) the controller goes to state b (where the light is still off) and remains there until the button is released. The next clock edge takes the controller to state c, where the light turns on. The state of the system remains unchanged as long as the button has not been pushed. With the first clock edge (after the button is pushed) the controller goes to state d (where the light is still on) and remains there until the button is released. The next clock edge takes the controller to state a, where the light turns off again. Different states are also shown in the timing diagram of Figure 12.

In the state machine modeled in Figure 13, the light turns on or off *after* the button is released. We may of course change the design to turn the light on or off *after* the button is pushed and before it is released. Also notice that in this example P is asynchronous, i.e., it can be asserted or deasserted at any time.

**Example 8.** Input data synchronized with a clock signal is received one bit at a time on a single line, A, so that each input bit lasts one cycle (see Figure 14). Develop a state diagram with one input, A, and one output, Z, such that *Z* will be pulled up only during the cycle of the fourth '1' of any group of four consecutive '1's on A (see Figure 14). Sequences may overlap. A typical input bit pattern on A and the corresponding output bit pattern on Z are shown below as a guideline:

A: 0 1 1 1 **1 1 1** 0 1 1 1 **1 1**

Z: 0 0 0 0 1 1 1 0 0 0 0 1 1

A timing diagram and a state diagram for this sequence detector are shown in Figure 14 and Figure 15, respectively.

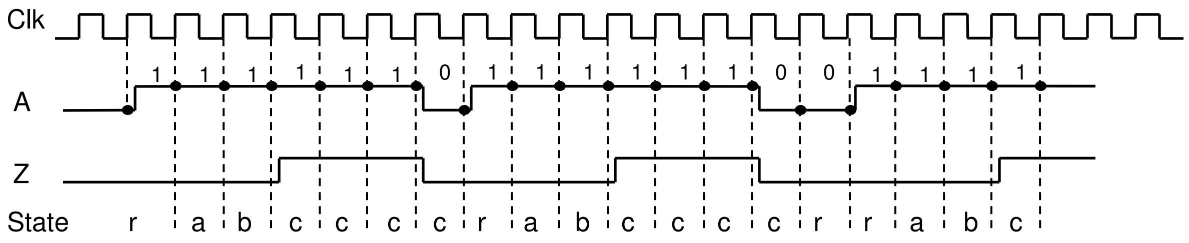


**Figure 14. Timing diagram for sequence detector in Example 8**

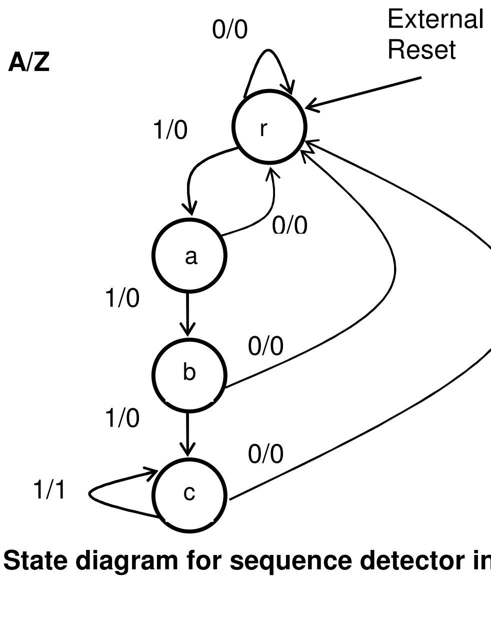


**Figure 15. State diagram for sequence detector in Example 8**

**Hint**: Suppose that during the process of developing a state diagram, you want to draw an outgoing arrow from a state. The first question that you have to answer is "Where should this outgoing arrow go to?" In other words, "Do I need to add a new state to the partial state graph developed so far, or should this outgoing arrow go to one of the existing states?" The answer may not always be obvious. In such a situation you should introduce a new state to stay on the safe side. If your guess happens to be incorrect, this redundant state will not cause any problems; it will be identified and removed in the state *minimization* step to be mentioned shortly in this chapter.

**A Design Methodology for State Machines Starting with State Tables**

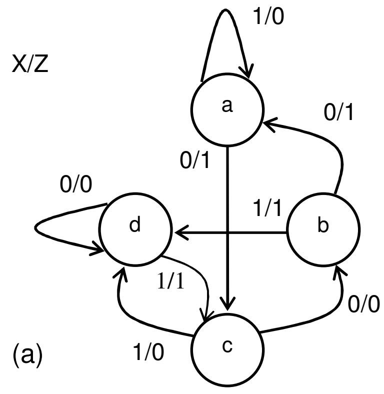
The first step in the design methodology (or translating a problem in natural language to a state table) was examined in the previous examples. In this section the full design cycle of synchronous circuits starting with a state diagram (step 2) is presented, utilizing D-FFs, T-FFs and JK-FFs in Example 9, Example 10 and Example 11, respectively.

**Example 9.** Use D-FFs to realize the state diagram shown in Figure 16*a*.

A state diagram is already available; so we start from step 2 as follows:

**2)** Translate the state diagram into a state table.

A state table for this state diagram is shown in Figure 16*b*. We performed this transformation in the analysis of sequential logic in Chapter 7 as well, but in reverse order.



(a)

(b)

| Q \ X | 0 | 1 |
|---|---|---|
| a | c, 1 | a, 0 |
| b | a, 1 | d, 1 |
| d | d, 0 | c, 1 |
| c | b, 0 | d, 0 |

$Q^{n+1}$, Z

**Figure 16. For Example 9: (a) state diagram, (b) state table**

As an example, consider state d. Starting with this state both the state diagram and state table read "if the current state is d and the input is 0, the output will be 0. Additionally, the circuit will stay in this state even if a clock edge arrives. On the other hand, if the current state is d and the input is 1, the output will be 1. Additionally, the circuit will stay in this state as long as no clock edge arrives. But if a clock edge arrives, then the circuit will jump to state c".

**3)** State minimization

After a state table is developed, the next step is state minimization. In this step all redundant states (if any) that may have been introduced in the development of the state diagram (step 1) are identified and removed, resulting in fewer states and possibly fewer state variables or FFs. For this step there is no counterpart in the analysis procedure of sequential circuits. State-table minimization is a mature area, hence well documented in the literature; however, it will not be addressed in this book.

Let's go back to Example 9. As a matter of fact, the state table obtained in Figure 16*b* is already minimized.

**4)** State assignment to obtain a transition table
Once the minimized state table is obtained, state assignment is performed. In this step a unique numeric code is assigned to each symbolic state to obtain a transition table. Although this step does have a counterpart in the analysis of finite state machines, state assignment in the design procedure is not trivial anymore. Different state assignments may result in different amounts of hardware for the realization of the combinational part of a finite state machine. Optimal state assignment is not covered in this book.

The state assignment used in this example is shown in Figure 17*a*. The resulting transition table is also illustrated in this figure.

| Q | Q1Q0 \ X | 0 | 1 |
|---|---|---|---|
| a | 00 | 10, 1 | 00, 0 |
| b | 01 | 00, 1 | 11, 1 |
| d | 11 | 11, 0 | 10, 1 |
| c | 10 | 01, 0 | 11, 0 |

$Q1^{n+1}\,Q0^{n+1}$, Z

(a)

| Q1 \ X | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

$Q1^{n+1}$

(b)

| Q0 \ X | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |

$Q0^{n+1}$

(c)

**Figure 17. For Example 9: (a) state assignment and transition table, (b) partial transition table for FF1, (c) partial transition table for FF0**

**5)** Partial transition tables
To make step 7 less error-prone, we split the transition table into two partial transition tables for FF1 and FF0, as shown in Figure 17*b* and Figure 17*c*, respectively.

**6)** Choosing one or more types of flip-flops
We may choose one or different types of FFs to realize a state machine. However, D-FFs are the most common type, as they are the least expensive and most convenient to use, although this choice may result in more expensive combinational logic depending on the specific design in hand. In this example we choose D-FFs.

**7)** Excitation and output maps
In this step, and in general, we use the excitation table of each type of FF being used, and also the partial transition tables obtained in step 5 to reach one excitation map for each excitation variable. The output map is also generated in this step. But remember that considering the characteristic equation of a D-FF ($Q^{n+1} = D$), the excitation map and partial-transition table of a D-FF are always the same.

Going back to Example 9, Figure 18*a* and Figure 18*b* show excitation maps for D1 and D0, respectively. Notice that similar to the partial transition tables, the coordinate variables of each excitation map are the state variables (Q1 and Q0) and the input variable (X) of the finite-state machine. In other words, an excitation map shows the logic value of the corresponding excitation variable (D1 or D0 in this example) for each individual combination of input value and current state of the finite-state machine. Remember that the current state of a finite-state machine during a clock cycle is made up of the states of all FFs (in the finite-state machine) in that cycle.

An output map is directly obtained from the state table or state diagram, as shown in Figure 18*c*. Remember that the output map and also the excitation maps have the same coordinate variables, namely Q1, Q0 and X.

**8)** Logic minimization for excitation and output variables

In this step the excitation maps and also the output map are minimized using the K-map theory. Since we have used Gray code in these maps, they are already K-maps. Figure 19 shows these K-maps with the proper circles.

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 0 | 1 |
| 11 | 1 | 1 |
| 10 | **0** | 1 |

**D1**

(a)

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 0 | 1 |
| 11 | 1 | 0 |
| 10 | 1 | 1 |

**D0**

(b)

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 1 | 1 |
| 11 | 0 | 1 |
| 10 | 0 | 0 |

**Z**

(c)

**Figure 18. Step 7 for Example 9: (a) excitation map of D1, (b) excitation map of D0, (c) output map**

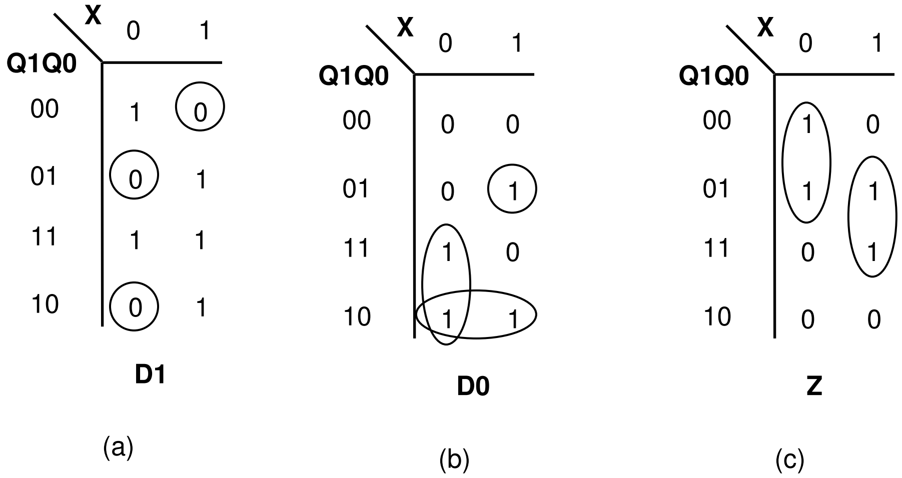


**Figure 19. K- maps for (a) D1, (b) D0, (c) Z in Example 9**

Excitation and output equations are derived from the maps of Figure 19.

D1 = (X'+Q1+Q0) . (X+Q1+Q0') . (X+Q1'+Q0)
D0 = X'.Q1 + Q1.Q0' + X.Q1'.Q0
Z = X'.Q1' + X.Q0

The above expressions are then translated to (combinational) circuits to generate excitation and output variables as shown in Figure 20**.** The design is now complete and ready for breadboard prototyping.

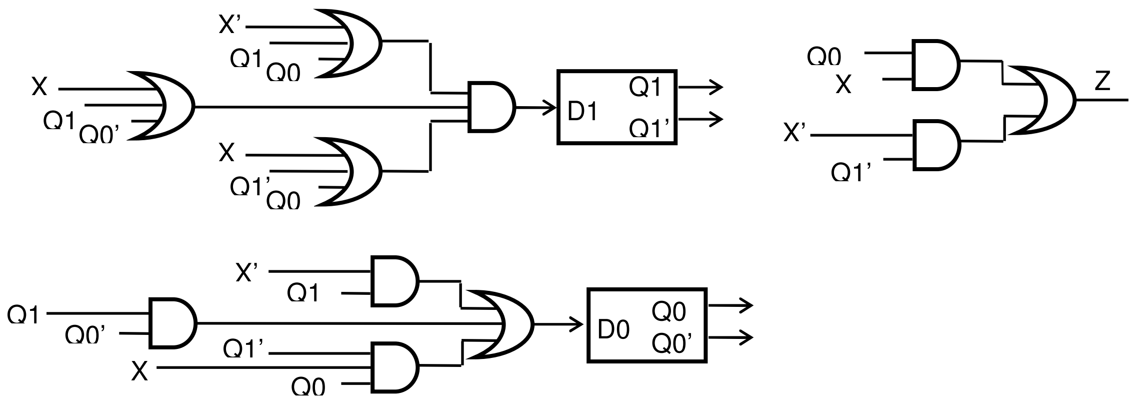
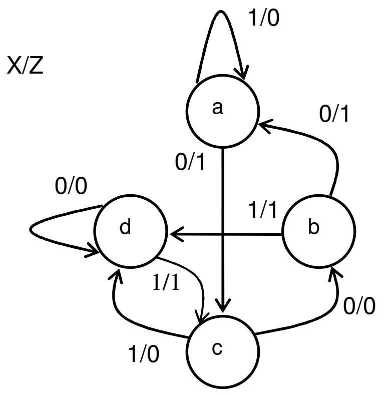


**Figure 20. Circuit diagram for Example 9**

**Example 10.** Use T-FFs to realize the state diagram shown in Figure 21*a*. Remember that step 1 (translating the natural-language description of problem to a state diagram) has already been carried out, so we start from step 2. Also notice that the next 4 steps of the design methodology are independent of the specific type of FF that we use. Here these steps are repeated for ease of reference.

**2)** Translate the state diagram into a state table.
A state table for this state diagram is shown in Figure 21*b*. We performed this transformation in the analysis of finite state machines in Chapter 7 as well, but in reverse order.



(a)

| Q \ X | 0 | 1 |
|---|---|---|
| a | c, 1 | a, 0 |
| b | a, 1 | d, 1 |
| d | d, 0 | c, 1 |
| c | b, 0 | d, 0 |

**$Q^{n+1}$,Z**

(b)

**Figure 21. For Example 10: (1) state diagram, (b) state table**

As an example, consider state d. Starting with this state both the state diagram and state table read "if the current state is d and the input is 0, the output will be 0. Additionally, the circuit will stay in this state even if a clock edge arrives. On the other hand, if the current state is d and the input is 1, the output will be 1. Additionally, the circuit will stay in this state as long as no clock edge arrives. But if a clock edge arrives, then the circuit will jump to state c".

**3)** State minimization
After a state table is developed the next step is state minimization. In this step all redundant states (if any) that may have been introduced in developing the state diagram (step 1) are identified and removed, resulting in fewer states and possibly fewer state variables or FFs. State minimization is not addressed in this book.

Going back to Example 10, the state table obtained in Figure 16*b* is already minimized.

**4)** State assignment to obtain a transition table
Once the minimized state table is obtained, state assignment is performed. In this step a unique numeric code is assigned to each symbolic state to obtain a transition table. Although this step does have a counterpart in the analysis of finite state machines, state assignment in the design procedure is not trivial anymore. Different state assignments may result in different amounts of hardware for the realization of the combinational part of a finite state machine. Optimal state assignment will not be covered in this book.

The state assignment used in this example is shown in Figure 22*a*. The resulting transition table is also illustrated in this figure.

| Q | Q1Q0 \ X | 0 | 1 |
|---|---|---|---|
| a | 00 | 10, 1 | 00, 0 |
| b | 01 | 00, 1 | 11, 1 |
| d | 11 | 11, 0 | 10, 1 |
| c | 10 | 01, 0 | 11, 0 |

$Q1^{n+1}\,Q0^{n+1}$, Z

(a)

| Q1 \ X | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

$Q1^{n+1}$

(b)

| Q0 \ X | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |

$Q0^{n+1}$

(c)

**Figure 22. For Example 10: (a) state assignment and transition table, (b) partial transition table for FF1, (c) partial transition table for FF0**

**5)** Partial transition tables
To make step 7 less error-prone, we split the transition table into two partial transition tables for FF1 and FF0, as shown in Figure 22*b* and Figure 22*c*, respectively.

**6)** Choosing one or more types of flip-flops
We may choose one or different types of FFs to realize a state machine. In this example we choose T-FFs. In general different types of FFs may result in different amounts of combinational hardware to realize the corresponding excitation variables.

**7)** Excitation and output maps
In this step we use the excitation table of a T-FF (shown again in Figure 23*a*), and also the partial transition tables from step 5 (shown again in Figure 23*b* and Figure 23*c*) to obtain one excitation map for each excitation variable. The output map is also generated in this step.

The excitation maps of T1 and T0 are shown in Figure 24*a* and Figure 24*b*, respectively. Notice that similar to the partial transition tables, the coordinate variables of each excitation map are the state variables (Q1 and Q0) and the input variable (X) of the finite-state machine. In other words, an excitation map shows the logic value of the corresponding excitation variable (T1 or T0 in this example) for each individual combination of input value and current state of the finite-state machine. (Remember that the current state of a finite-state machine during a clock cycle is made up of the states of all FFs (in the finite-state machine) in that cycle.) Here is the procedure to fill in each position in an excitation map:

| $Q^n$ => | $Q^{n+1}$ | T |
|---|---|---|
| 0 | 0 | 0 |
| **0** | **1** | **1** |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(a)

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 0 | **1** |
| 11 | 1 | 1 |
| 10 | 0 | 1 |

(b) **$Q1^{n+1}$**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 0 | 1 |
| 11 | 1 | 0 |
| 10 | 1 | 1 |

(c) **$Q0^{n+1}$**

**Figure 23. For Example 10: (a) excitation table of a T-FF, (b) partial transition table of FF1, (c) partial transition table of FF0**

Examine the position in the partial transition table corresponding to the position in the excitation map that you want to fill in. You will see a state transition (either a change or no change) for that position in the partial transition table. Now check the excitation table of a T-FF, obtain the logic value for the T input required for that specific state transition and place this value in the same position in the excitation map. For example, consider position {row 0 1 column 1} in the partial transition table of FF1 shown in Figure 23*b*. It reads "if X = 1, and Q1 Q0 = 0 1, then state of FF1 must change from $Q_1^n = 0$ to $Q_1^{n+1} = 1$ with the next clock edge". So, the question is "What must we apply to T1 to let this FF undergo a 0-to-1 state transition"? And the answer is in the excitation table of this FF as explained below.

To fill in the same position (X = 1, and Q1 Q0 = 0 1) in the excitation map shown in Figure 24*a*, we proceed as follows: for this position we need a $Q_1^n = 0$ to $Q_1^{n+1} = 1$ state transition. To find out the required value for the excitation variable, T1, we refer to the excitation table of this FF shown in Figure 23*a*. Row 01 of this table prescribes that for this transition ($Q_1^n = 0$ to $Q_1^{n+1} = 1$) we need a 1 for T1. Therefore, position {row 01 column 1} in the excitation map of T1 is filled in with a 1, as shown in Figure 24*a*. Other positions are filled out in a similar way, to eventually come up with the excitation maps shown in Figure 24*a* and Figure 24*b*, for T1 and T0, respectively.

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 0 | **1** |
| 11 | 0 | 0 |
| 10 | 1 | 0 |

(a) **T1**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 1 | 0 |
| 11 | 0 | 1 |
| 10 | 1 | 1 |

(b) **T0**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 1 | 1 |
| 11 | 0 | 1 |
| 10 | 0 | 0 |

(c) **Z**

**Figure 24. For Example 10: (a) excitation map of FF1 (b) excitation map of FF0, (c) output map**

An output map is directly obtained from the state table or state diagram, as shown in Figure 24*a*. Remember that the output map and also the excitation maps have the same coordinate variables, namely Q1, Q0 and X.

**8)** Logic minimization for excitation and output equations

In this step the excitation maps and also the output map are minimized using the K-map theory. Since we have used Gray code in these maps, they are already K-maps. Figure 25 shows these K-maps with the proper circles.

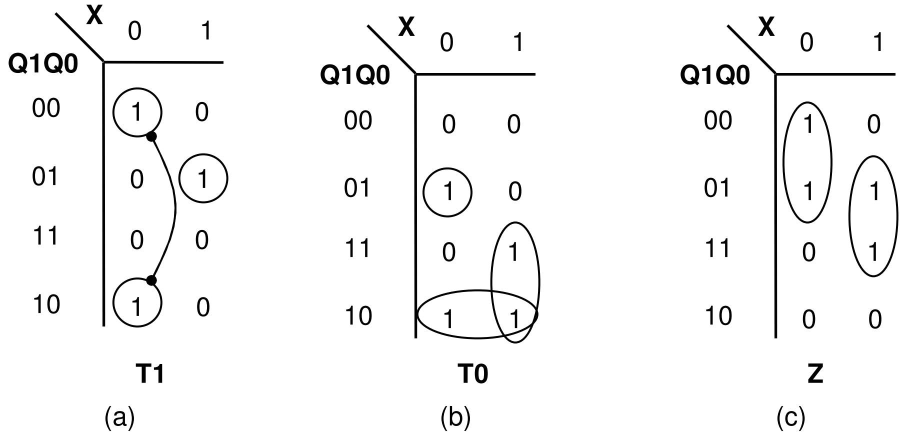


**Figure 25. K- maps for (a) T1, (b) T0, (c) Z in Example 10**

Excitation and output equations are derived from the maps of Figure 25.

T1 = X’.Q0’ + X.Q1’.Q0  
T0 = X.Q1 + Q1.Q0’ + X’.Q1’.Q0  
Z = X’.Q1’ + X.Q0

The above expressions are then translated to (combinational) circuits to generate excitation and output variables as shown in Figure 26**.** The design is now complete and ready for breadboard prototyping.

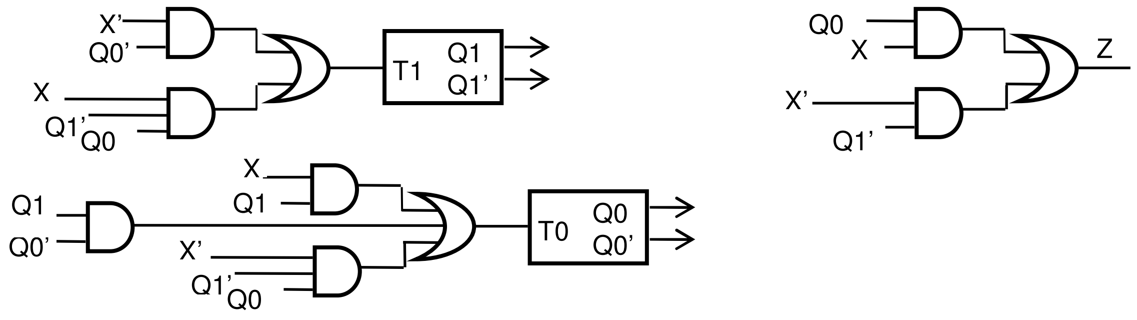
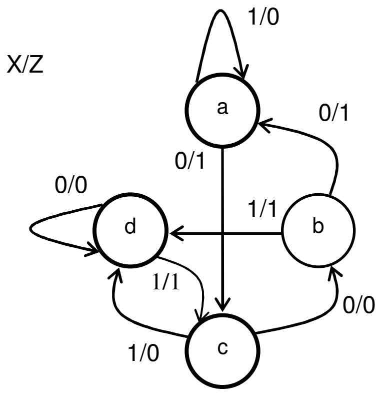


**Figure 26. Circuit diagram for Example 10**

**Example 11.** Use JK-FFs to realize the state diagram shown in Figure 27*a*. Remember that step 1 (translating the natural-language description of problem to a state diagram) has already been carried out, so we start from step 2. Also notice that the next 4 steps of the design methodology are independent of the specific type of FF that we use. Here these steps are repeated for ease of reference.

**2)** Translate the state diagram into a state table.

A state table for this state diagram is shown in Figure 27*b*. We performed this transformation in the analysis of finite state machines in Chapter 7 as well, but in reverse order.



(a)

| Q \ X | 0 | 1 |
|---|---|---|
| a | c, 1 | a, 0 |
| b | a, 1 | d, 1 |
| d | d, 0 | c, 1 |
| c | b, 0 | d, 0 |

$Q^{n+1}$, Z

(b)

**Figure 27. For Example 11: (1) state diagram, (b) state table**

As an example, consider state d. Starting with this state both the state diagram and state table read "if the current state is d and the input is 0, the output will be 0. Additionally, the circuit will stay in this state even if a clock edge arrives. On the other hand, if the current state is d and the input is 1, the output will be 1. Additionally, the circuit will stay in this state as long as no clock edge arrives. But if a clock edge arrives, then the circuit will jump to state c".

**3)** State minimization
After a state table is developed the next step is state minimization. In this step all redundant states (if any) that may have been introduced in developing the state diagram (step 1) are identified and removed, resulting in fewer states and possibly fewer state variables or FFs. State minimization is not addressed in this book.

Going back to Example 11, the state table obtained in Figure 27*b* is already minimized.

**4)** State assignment to obtain a transition table
Once the minimized state table is obtained, state assignment is performed. In this step a unique numeric code is assigned to each symbolic state to obtain a transition table. Although this step does have a counterpart in the analysis of finite state machines, state assignment in the design procedure is not trivial anymore. Different state assignments may result in different amounts of hardware for the realization of the combinational part of a finite state machine. Optimal state assignment will not be covered in this book.

The state assignment used in this example and also the resulting transition table are shown in Figure 28*a*.

**5)** Partial transition tables
To make step 7 less error-prone, we split the transition table into two partial transition tables for FF1 and FF0, as shown in Figure 28*b* and Figure 28*c*, respectively.

**6)** Choosing one or more types of flip-flops
We may choose one or different types of FFs to realize a state machine. In this example we choose JK-FFs. In general different types of FFs may result in different amounts of combinational hardware to realize the corresponding excitation variables.

| Q | Q1Q0 \ X | 0 | 1 |
|---|---|---|---|
| a | 00 | 10, 1 | 00, 0 |
| b | 01 | 00, 1 | 11, 1 |
| d | 11 | 11, 0 | 10, 1 |
| c | 10 | 01, 0 | 11, 0 |
| | | **Q1$^{n+1}$ Q0$^{n+1}$, Z** | |

(a)

| **Q1** \ X | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| | **Q1$^{n+1}$** | |

(b)

| **Q0** \ X | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| | **Q0$^{n+1}$** | |

(c)

**Figure 28. For Example 11: (a) state assignment and transition table, (b) partial transition table for FF1, (c) partial transition table for FF0**

**7)** Excitation and output maps
In this step we use the excitation table of a JK-FF (shown again in Figure 29*a*), and also the partial transition tables from step 5 (shown again in Figure 29*b* and Figure 29*c*) to obtain one excitation map for each excitation variable. The output map is also generated in this step.

| $Q^n$ => | $Q^{n+1}$ | J | K |
|---|---|---|---|
| 0 | 0 | 0 | d |
| 0 | 1 | 1 | d |
| **1** | **0** | **d** | **1** |
| 1 | 1 | d | 0 |

(a)

| **Q1**Q0 \ X | 0 | 1 |
|---|---|---|
| **0**0 | 1 | 0 |
| **0**1 | 0 | **1** |
| **1**1 | 1 | 1 |
| **1**0 | 0 | 1 |
| | **Q1$^{n+1}$** | |

(b)

| Q1**Q0** \ X | 0 | 1 |
|---|---|---|
| 0**0** | 0 | 0 |
| 0**1** | **0** | 1 |
| 1**1** | 1 | 0 |
| 1**0** | 1 | 1 |
| | **Q0$^{n+1}$** | |

(c)

**Figure 29. For Example 11: (a) excitation table of JK-FF, (b) partial transition table for FF1, (c) partial transition table for FF0**

The excitation maps of J1, K1, J0 and K0 are shown in Figure 30*a*, Figure 30*b*, Figure 30*c* and Figure 30*d*, respectively. Notice that similar to the partial transition tables, the coordinate variables of each excitation map are the state variables (Q1 and Q0) and the input variable (X) of the finite-state machine. In other words, an excitation map shows the logic value of the corresponding excitation variable (J1, K1, J0 or K0 in this example) for each individual combination of input value and current state of the finite-state machine. (Remember that the current state of a finite-state machine during a clock cycle is made up of the states of all FFs (in the finite-state machine) in that cycle.) Here is the procedure to fill in each position in an excitation map:

Examine the position in the partial transition table corresponding to the position in the excitation map that you want to fill in. You will see a state transition (either a change or no change) for that position in the partial transition table. Now check the excitation table of a JK-FF (see Figure 29*a*) and obtain the logic value for the J and K inputs required for that specific state transition. Place these values in the corresponding positions in the excitation maps of the J input and K input, respectively. For example consider position {row 01 column 0} in the partial transition table of FF0 shown in Figure 29*c*. It reads if x = 0, and Q1 Q0 = 0 1, then state of FF0 must change from $Q_0^n = 1$ to $Q_0^{n+1} = 0$ with the next clock edge. So, the question is what we must apply to J1 and K1 to let this FF undergo a 1-to-0 state transition. And the answer is in the excitation table of this FF as explained below.

To fill in the same positions (x = 0, and Q1 Q0 = 0 1) in the excitation maps shown in Figure 30*c* and Figure 30*d*, we proceed as follows: for this position we need a $Q_0^n = 1$ to $Q_0^{n+1} = 0$ state transition. To find out the required values for the corresponding excitation variables (J0 and K0) we refer to the excitation table of this FF shown in Figure 29*a*. Row 1 0 (the fourth row) of this table prescribes that for this transition ($Q_0^n = 1$ to $Q_0^{n+1} = 0$) we need a 1 for K0, but J0 can be a 1 or a 0. Therefore, positions {row 01 column 0} in the excitation maps of J0 and K0 are filled in with a d (don't care) and 1, respectively, as shown in Figure 30*c* and Figure 30*d,* respectively. Other positions are filled out in a similar way, to eventually come up with the excitation maps shown in Figure 30*a,* Figure 30*b,* Figure 30*c* and Figure 30*d* for J1, K1, J0 and K0, respectively.

(a) **J1**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 0 | 1 |
| 11 | d | d |
| 10 | d | d |

(b) **K1**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | d | d |
| 01 | d | d |
| 11 | 0 | 0 |
| 10 | 1 | 0 |

(c) **J0**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 0 | 0 |
| 01 | **d** | d |
| 11 | d | d |
| 10 | 1 | 1 |

(d) **K0**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | d | d |
| 01 | **1** | 0 |
| 11 | 0 | 1 |
| 10 | d | d |

**Figure 30. Excitation maps in Example 11: (a) J1, (b) K1, (c) J0, (d) K0**

Remember that in a K-map or truth table, a don't care may be assigned to the output for two different reasons: 1) to indicate an impossible input combination, and 2) to show this fact that for the corresponding input combination both output values, 0 and 1, are consistent with the specifications of the problem, i.e., the designer has two output options, 0 or 1, for that input combination. For the first category of output don't cares a couple of examples were presented in Chapter 4. A practical example for the second category is shown in Figure 30, in which either logic value (0 or 1) may substitute for any don't care (d).

An output map is directly obtained from the state table or state diagram, as shown in Figure 31. Remember that the output map and also the excitation maps have the same coordinate variables, namely Q1, Q0 and X.

**Z**

| Q1Q0 \ X | 0 | 1 |
|---|---|---|
| 00 | 1 | 0 |
| 01 | 1 | 1 |
| 11 | 0 | 1 |
| 10 | 0 | 0 |

**Figure 31. For Example 11: output map**

**8)** Logic minimization for excitation and output equations

In this step the excitation maps and also the output map are minimized using the K-map theory. Since we have used Gray code in these maps, they are already K-maps. Figure 32 shows these K-maps with the proper circles. In order to distinguish don't cares from the input variable, X, in this figure we represent don't cares with letter d.

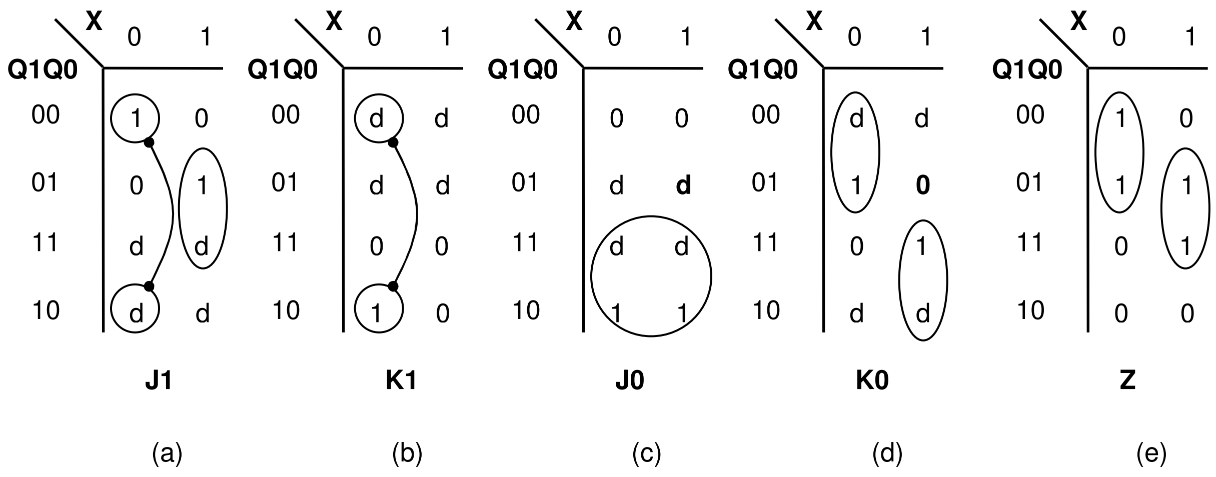


**Figure 32. K- maps for (a) J1, (b) K1, (c) J0, (d) K0, (e) Z in Example 11**

The excitation and output equations are derived from the maps of Figure 32.

J1 = X'.Q0' + X.Q0
K1 = X'.Q0'
J0 = Q1
K0 = X'.Q1' + X.Q1
Z = X'.Q1' + X.Q0

The above expressions are then translated to (combinational) circuits to generate excitation and output variables as shown in Figure 33**.** The design is now complete and ready for breadboard prototyping.

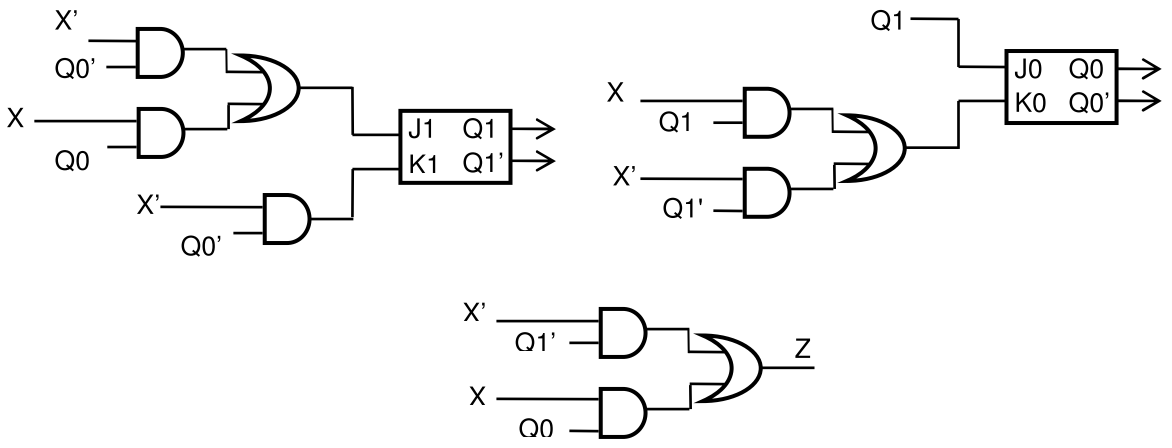


**Figure 33. Circuit diagram for Example 11**

**Mealy and Moore Machines**

A generic model for finite state machines (Figure 22, Chapter 7) is again shown in Figure 34 for ease of reference. There are two different versions of finite state machines, namely Mealy and Moore machines, modeled in Figure 35*a* and Figure 35*b*, respectively.

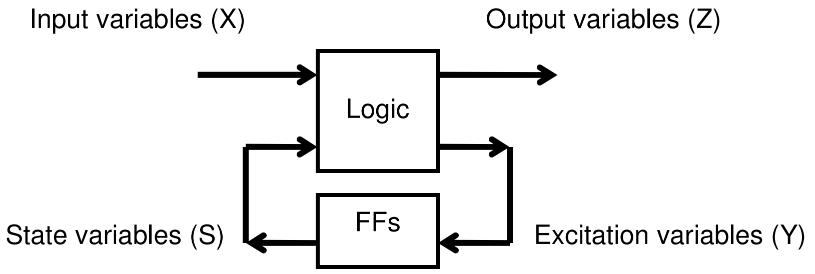


**Figure 34. Generic model for finite state machines (Figure 20, Chapter 7)**

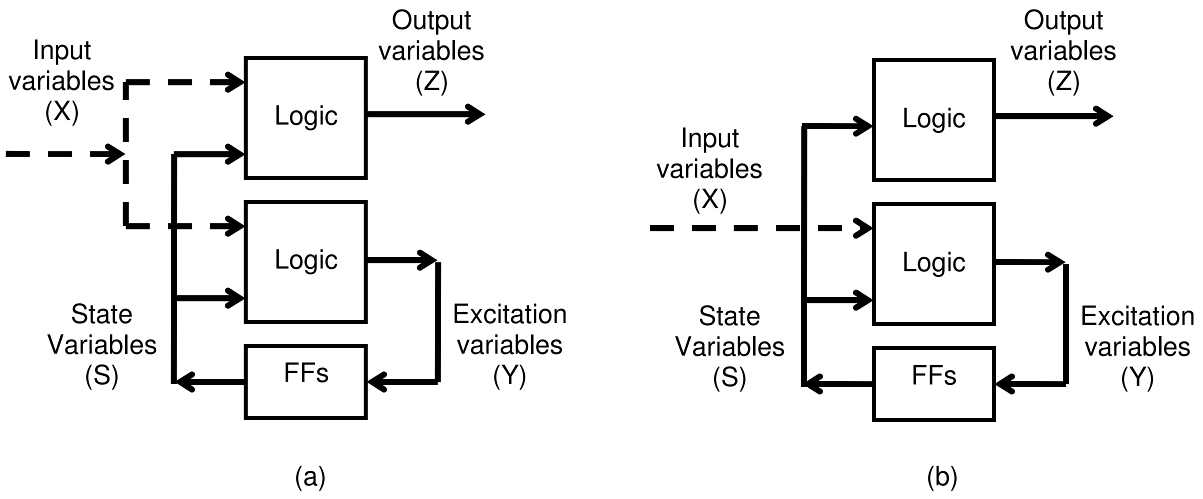


**Figure 35. Two categories of finite state machines: (a) Mealy, (b) Moore**

In a Mealy machine the output is a function of the state and input variables, while in a Moore machine the output only depends on the state (and not the input(s) anymore). Therefore, the transition diagrams shown in Figure 36*a* and Figure 36*b* represent a Mealy machine and a Moore machine, respectively.

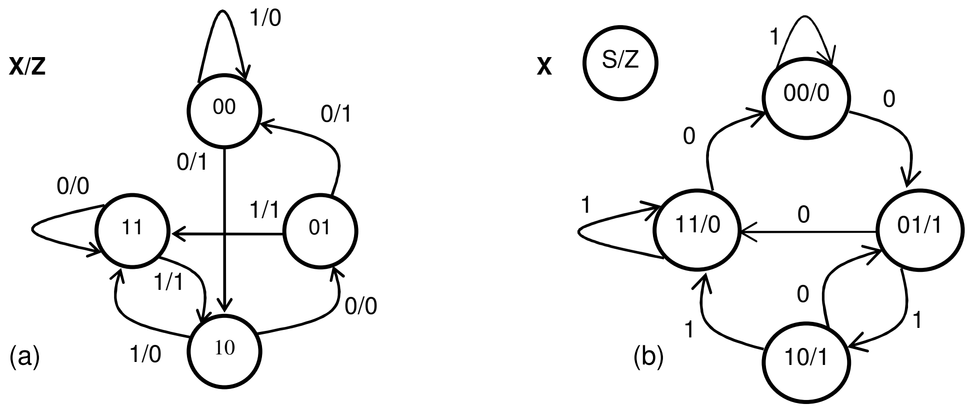


**Figure 36. Transition diagram for: (a) Mealy machine, (b) Moore machine**

In Figure 36*b* the values of the output variable have been moved into the state circles, as now the output is a function of the state only, and not the input anymore.

When there is more than one output, it is possible to consider them independently and reword the above concept as follows: A *Moore-type output* depends only on the state, while a *Mealy-type output* is a function of both the input and state.