Chapter 7
Spring 2010 Edition

Memory Cells and Analysis of Sequential Circuits

A *SUMMARY* of what you learned in Chapter 6:
Binary decoders
74x138: a 3-to-8 binary decoder
BCD code: upsides and downsides
Seven-segment code and seven-segment displays
BCD-to-seven-segment decoders
Binary encoders
Priority encoders
74x148: an 8-to-3 priority encoder
Selectors or multiplexers (muxes)
Transmission gates and how to use them to design a mux
Mux expansion in 2 dimensions: number of inputs (channels) and width of each input
Read-only-memories (ROMs)
Logic circuits as ROMs and vice versa
Muxs look like ROMs
How to use muxs to realize logic functions
74x151: an 8-input 1-bit multiplexer
Demultiplexers
Partial and full comparators
Signed and unsigned comparators
A repetitive approach to design full comparators
Unsystematic design methodology

Introduction
The digital circuits that we have studied so far (in the previous chapters) are called *combinational*. A major characteristic of a combinational circuit is that its current output depends only on its current input combination and not past inputs. This means that a combinational circuit never remembers anything from the past, in other words it is *memoryless*. More specifically, if we apply, for example, 3 and 9 to a 4-bit adder, we will *always* receive a 12, no matter whether we apply these inputs before we have applied another pair of inputs, say, 15 and 6, or after that. This behavior is totally consistent with the definition of a *function* in your math courses in which you learned no input (applied to a function) may generate two different outputs. In a *sequential* circuit, however, the current output necessarily depends on some *previous* inputs, while it may depend on the current input as well. In other words, a sequential circuit keeps a history from its past: **it has memory**. The length of history that a sequential circuit keeps is determined by the number of memory cells that the circuit has. The more memory cells it has, the longer

its history. To further clarify this issue, consider a circuit that receives random numbers and is supposed to detect and count prime numbers as they are received. A (properly-designed) combinational circuit is able to detect prime numbers; however it is unable to *remember* the number of prime numbers seen so far. In other words, a combinational circuit cannot *count* the prime numbers that are received because the (current) output of a combinational circuit is not affected by any input received in the past. To get this job done we need a sequential circuit. Since the number of memory cells utilized in a sequential circuit is always limited, the number of prime numbers remembered by this circuit or the length of history that this (or any) circuit may keep is obviously limited as well.[1]

As another example, suppose that in a chemical plant if the pressure goes beyond a threshold *before* the temperature reaches a limit, an alarm must go off, but not if the temperature reaches the limit first. Therefore, in addition to the current inputs, the *sequence* in which the two events occur is decisive as well. In other words, the alarm system only reacts to a specific sequence of events; hence, it is a sequential circuit.

The extra information (in addition to the current input) that determines the current output of a sequential circuit is called the current *state* of that circuit. In other words, the state of a circuit at each instant of time is the history that the circuit keeps and remembers at that specific instant of time. The state of a circuit is stored in the (internal) memory of that circuit and changes as the circuit receives additional inputs over time.

A single memory cell can be imagined as a tiny box that is filled (or written) with a 1 or 0. The cell will then hold this value (or *state*) forever unless it is overwritten (or the power turns off). The *content* of the box is the *output* of the box as well. When a memory cell is filled with a 1 or 0, the cell is said to be *set* or *reset* (*cleared*), respectively. There are different types of memory cells (with of course different names) with different rules for the set and reset operations. In this chapter a few types of memory cells are first introduced, then the concept of analysis of sequential circuits is explained and a systematic methodology for this analysis is presented.

r-s latch

Figure 1*a* and Figure 1*b* show a logic circuit and a logic symbol, respectively, for a memory cell called an r-s latch with two inputs, r (reset) and s (set) and two outputs, Q and Q'. It will soon become clear why the inputs are called r and s. Let's apply all possible input combinations to this circuit (similar to what we have done so far in previous chapters), and follow the input signals to reach the output nodes, hence determine signal values at the outputs for each input combination.

(1) Reset: starting with r s = 1 0 in Figure 1*a*, a 1 at the r input of NOR 2 pulls down the gate's output, Q. Now NOR 1 has two inputs at logic low (s = 0, Q = 0) pulling up this gate's output, Q'. When r s = 1 0 we say the latch is in the reset mode (Q = 0) where it operates as a combinational circuit with Q = r' = 0 and Q' = (Q + s)' = 1.

(2) Hold: change the inputs to r s = 0 0. Although r is now low, the output (Q) of NOR 2 remains low because the second input (Q') of this gate is still high. The 1 at Q' is consistent with the two low inputs of NOR 1. When r s = 0 0, we say the latch is in the hold mode, as the latch holds its previous output. Now the latch is not a combinational circuit anymore. We will see more about this mode shortly.

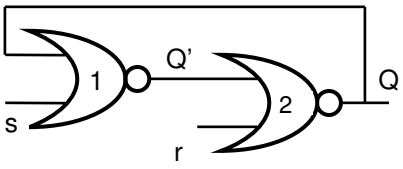
(3) Set: apply r s = 0 1 to the latch. A 1 at the s input of NOR 1 pulls down the gate's output, Q'. Now NOR 2 has two inputs at logic low (r = 0, Q' = 0) pulling up this gate's output, Q. When r s = 0 1 we say the latch is in the set mode (Q = 1) where it again operates as a combinational circuit but now Q' = s' = 0 and Q = (Q' + r)' = 1.

[1] The concept of memory used in this context should not be mistaken for the (main) memory in a personal computer, although they both can be written in and read out.

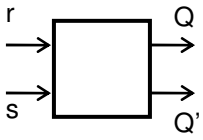
(4) Hold: change the inputs to r s = 0 0. This is again the hold mode; although s is now low, the output (Q') of NOR 1 remains low because the second input (Q) of this gate is still high. The 1 at Q is consistent with the two low inputs of NOR 2.

(5) Not recommended: assert both r and s. Now each NOR gate has one input at logic high, resulting in Q = Q' = 0. Here again the latch operates as a combinational circuit.

These observations have been listed in the truth or *characteristic* table of this latch shown in Figure 1*c*. Read this table from top to bottom. Can you determine the fundamental difference between an r-s latch and the combinational circuits studied in the previous chapters? If not, take a closer look at the second and fourth rows (or hold rows) of this truth table; the same input combination, 00, in these two rows corresponds to two different output values, Q = 0 and Q = 1, while a combinational circuit always generates the same output for the same input combination.



(a)



(b)

r s	Q	Q'	mode
1 0	0	1	reset
0 0	**0**	**1**	**hold**
0 1	1	0	set
0 0	**1**	**0**	**hold**
1 1	0	0	Don't use

(c)

Figure 1. NOR-based r-s latch: (a) logic circuit, (b) logic symbol, (c) characteristic table (read it in order from top to bottom)

The second and fourth rows in the characteristic table of Figure 1*c* can be merged to simplify this table as shown in Figure 2. This simplified table has four rows and does not have to be read in order anymore. The first row (of this table) represents the hold mode and reads "if both inputs are at logic low, the output does not change (NC stands for no change)". The other three rows have not been changed.

r s	Q	Q'	mode
0 0	NC	NC	Hold
0 1	1	0	set
1 0	0	1	reset
1 1	1	1	Don't use

Figure 2. Four-row characteristic table for r-s latch

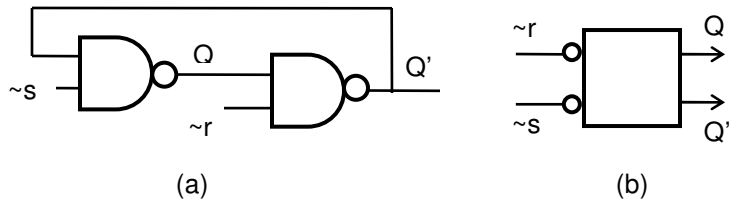
The hold mode (where r = s = 0) deserves special attention: Since inputs tied to logic 0 may be removed from NOR gates, in the hold mode (where r = 0 and s = 0) the latch of Figure 1*a* would become equivalent to the two-inverter loop shown in Figure 3*a*. In this circuit the value at Q (or Q') cannot be determined by r and/or s. (As a matter of fact, neither r nor s exists in this circuit any longer!) So, in this mode the latch is not a combinational circuit. Now the question is, "How is the value of Q determined in this mode?" The answer is, "The value of Q in Figure 3*a* is identical to the value of Q during the mode right before the hold mode". For example, if the previous mode was the set mode (i.e., the previous Q was 1), then the value of Q during the current hold mode would still be 1. This 1 has in fact been trapped in the stable loop (or technically speaking the *basic memory cell*) shown in Figure 3*a*. We call this loop stable because the values of all nodes in this loop will eventually stabilize. More specifically, Q = 1 applied to inverter 1 pulls output Q' down. And this 0 (applied to inverter 2) will in turn pull Q up. In a similar way we may show that the loop is stable for Q = 0 and Q' = 1 as well. (In general, loops made up of an even number of inverters are stable, while those made up of an odd number of inverters are

unstable.) The basic memory cell in Figure 3*a* is often shown as a cross-coupled inverter pair as illustrated in Figure 3*b*.

Figure 3. Equivalent circuit for a latch in hold mode: (a) cascaded inverters, (b) cross-coupled inverters

In summary, the behavior of an r-s latch may be described as follows: when either r or s is asserted the circuit behaves as a combinational logic, and output Q takes a 0 or 1, respectively. With r s = 00 the latch would enter the hold mode in which Q = 0 or Q = 1 if the latest asserted input was r or s, respectively. In other words, in the hold mode the latch holds (stores) the output that it had in the preceding mode. What this sequential circuit remembers in the hold mode (when both inputs are deasserted) is the latest asserted input, r or s. For all three input combinations r s = 0 0, 01 and 10, Q' is the complement of Q. Under the fourth input combination (r s = 1 1) the latch again behaves as a combinational circuit, but with inconsistent outputs, as now both outputs are pulled down. This input combination (r s = 1 1) should be avoided. In the Critical Thinking Section of Homework Assignment 7, you have been challenged to find out the possible catastrophic outcome of this input combination.

The logic circuit, logic symbol and characteristic table of the NAND-based counterpart of the r-s latch are illustrated in Figure 4*a*, Figure 4*b* and Figure 4*c*, respectively. Notice that now the inputs are active low.



~r	~s	Q	Q'	mode
0	0	1	1	Don't use
0	1	0	1	reset
1	0	1	0	set
1	1	NC	NC	hold

(c)

Figure 4. NAND-based r-s latch: (a) logic circuit, (b) logic symbol, (c) characteristic table

Example 1. Consider a single-pole/single-throw (SPST) push-button and a SPST switch (that we normally use to turn on and off lights) used in the LED control circuits shown in Figure 5*a* and Figure 5*b*, respectively. (In practice we may need to put a resistor in series with each LED.) Unlike the SPST switch, the push-button is not able to keep the light on (unless somebody stays there and continuously keeps his/her finger on the push-button!) This major difference between a SPST switch and a push-button stems from the fact that a SPST switch has a (mechanical) memory but a push-button does not; so that the SPST switch always remembers the last finger press that turned the off light on or vice versa. In this example we want to add an electronic memory to two (memoryless) push-buttons to let the resulting system remember the most recent button that was pushed, hence be able to turn an off LED on or vice versa, by pressing the right button.

Figure 5*c* illustrates two push-buttons, two resistors (R1 and R2), an electronic memory, which is in fact the r-s latch explained on pages 2-3, and an LED, which is driven by output Q of the latch. Let's see how this digital circuit works. When the power is turned on (and both switches are released) the r-s latch takes a non-deterministic state, i.e., Q could be 1 or 0. (This is also the case with other types of memory cells to be covered in this chapter.) As long as both switches are released, both r and s inputs of the latch are connected to ground (or GND for short) through the two resistors, R1 and R2. And this logically means

that these two inputs are tied to logic 0. Let's assume that the initial state of the latch (when the power is turned on) is Q = 0 (see row 2 in Figure 1*c*), which in turn means that the LED is off. As shown in Figure 5*c* the ON and OFF push-buttons are connected to the s and r inputs, respectively. In order to turn the light on we need to set the latch. And to set the latch we need to (momentarily) assert the s input as explained on pages 2-3; and this can be carried out by pressing the ON push-button. Pay close attention to how this works: when the ON button is pushed, the s input is *directly* connected to Vcc, while s is still (permanently) connected to GND through a resistor, R1. This means that node s is now influenced by both Vcc and GND. However, because of R1, Vcc wins, letting node s reach logic 1, and eventually setting the latch (see row 3 in Figure 1*c*). Now the ON push-button can be released. This will send the latch to the hold mode (see row 4 in Figure 1*c*). In other words, the latch will remain set (and the LED will remain on) as long as the OFF push-button has not been touched. In a similar way, if the OFF button is pushed, the latch toggles (changes its state from Q = 1 to Q = 0) resulting in an OFF LED. ◊

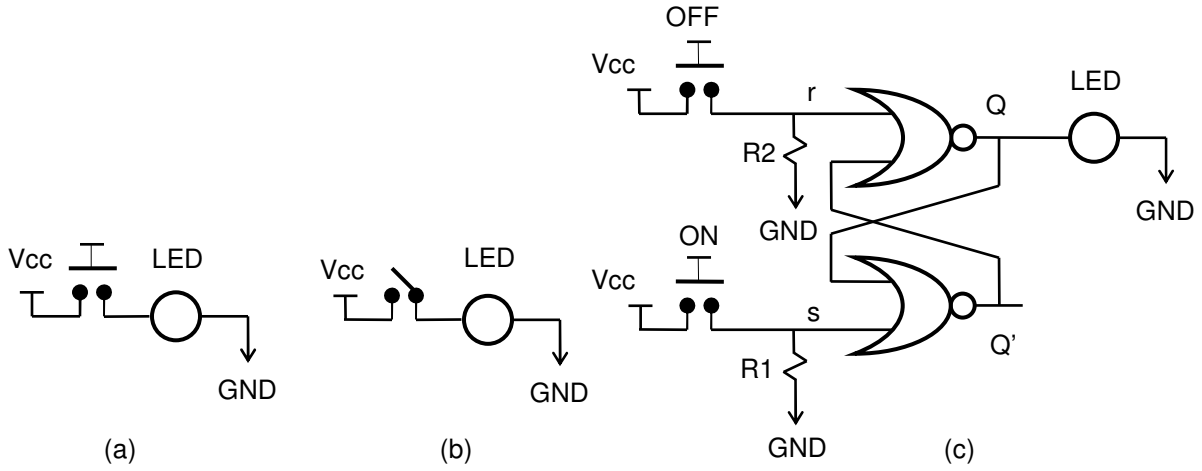


Figure 5. An LED controlled by: (a) a push-button, (b) a SPST switch, (c) two push-buttons

Sequential circuits are further classified as *synchronous*, to be covered in this book, and *asynchronous*. Commercial systems are mostly synchronous. In this book by sequential circuits we mean synchronous circuits unless otherwise specified. In synchronous circuits all write operations into memory cells are carried out simultaneously to solve a major problem to be addressed later in Chapter 8. In order to satisfy this concurrency requirement, we introduce a global synchronizing signal called *clock* (or *Clk* for short) shown in Figure 6, and also new memory cells to operate with this signal. The clock signal in a digital system is (normally) a periodic square wave to trigger all memory cells, and facilitate simultaneous write operations into these cells. Cycle time (T), frequency (f), falling edges, rising edges, positive pulses, negative pulses, and duty cycle of the clock signal have also been defined in Figure 6.

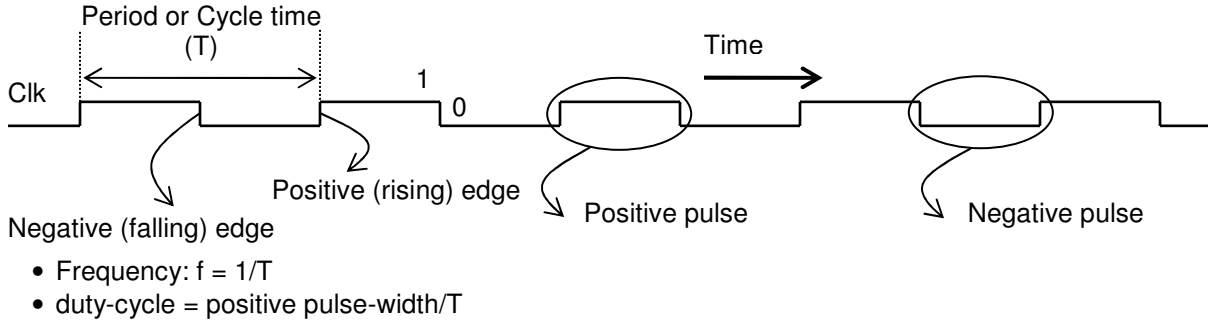


Figure 6. Clock signal

R-S latch As a first attempt to reach a clocked memory cell we add the clock signal to an r-s latch as illustrated in Figure 7*a* and get a clocked R-S latch in which the (control) inputs are called S and R. The characteristic (definition) table of an R-S latch is shown in Figure 7*b*, which is the same as a non-clocked r-s latch has. However, keep in mind that there is an implicit input signal, Clk, not shown in this table. Considering the two newly introduced AND gates shown in Figure 7*a*, any changes to the content of the latch may only take place while Clk is asserted (Clk = 1). Deasserted Clk (Clk = 0) keeps the memory cell in the hold mode, no matter what values S and R take. For example, the third row in the characteristic table reads if S = 0 and R = 1 while Clk = 1, the memory cell is reset, i.e., Q and Q' become 0 and 1, respectively. Additionally, if Clk is deasserted the reset memory cell remains reset, no matter what are applied to S and R inputs. As a second example consider the first row, which reads "if both R and S are deasserted no matter whether or not Clk is asserted, the content of this memory cell remains unchanged." A logic symbol for this memory cell is shown in Figure 7*c*.

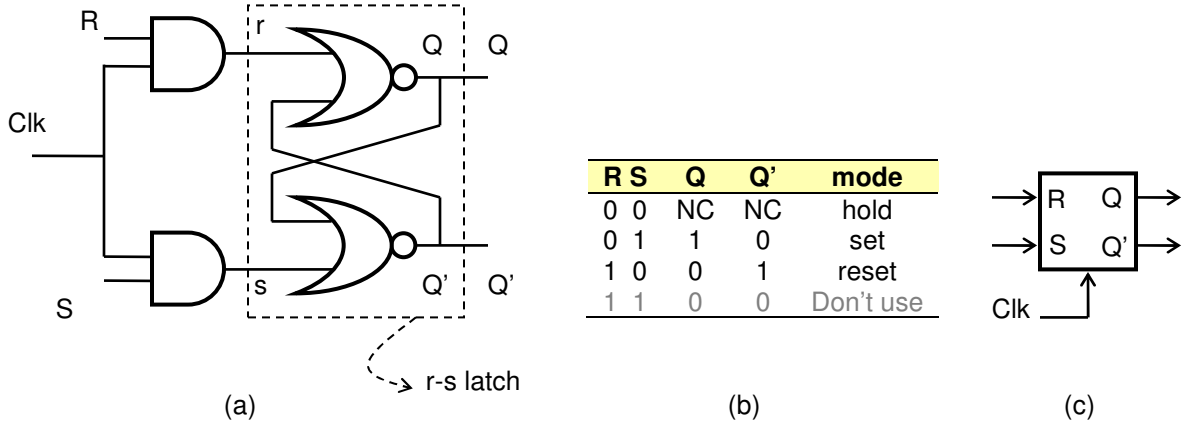


Figure 7. Clocked R-S latch: (a) logic circuit, (b) characteristic table, (c) logic symbol

D-latch As we know, what makes memory cells different from each other is the way that they are written in. Figure 8*a* illustrates a different memory cell called a D-latch, which has been built on the R-S latch developed in Figure 7*a*. In Figure 8*a* S = R'; so if we remove the irrelevant rows 1 and 4 (in which R = S) from Figure 7*b*, the 2-row characteristic table shown in Figure 8*b* will be obtained for a D-latch. A logic symbol for this memory cell is depicted in Figure 8*c*.

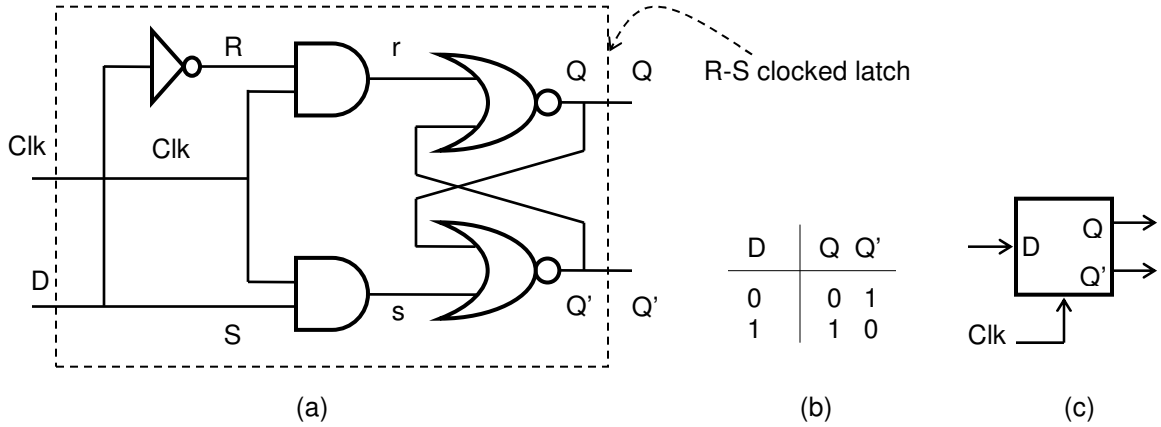


Figure 8. Clocked D-latch: (a) gate level circuit, (b) definition table, (c) logic symbol

Figure 9 illustrates a typical timing diagram for the D-latch of Figure 8*a*.

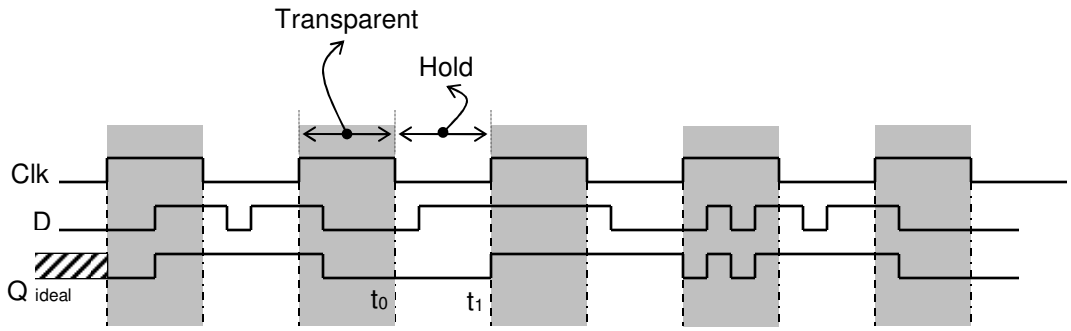


Figure 9. Ideal D-latch timing diagram

There are two operation modes for a D-latch, namely *transparent* and *hold*, as shown in this diagram. While Clk is asserted (i.e., during positive clock pulses for the D-latch in Figure 8*a*), the latch is in the transparent mode, in which output Q exactly follows input D. In this mode the latch obviously behaves as a combinational circuit. With a falling edge of the clock signal, say at instant t_0 in Figure 9, the latch leaves the transparent mode, and at the same time samples and holds input D. This is the beginning of the hold mode, in which the latch is converted to a memory cell, which contains the logic value sampled during the previous high-to-low transition of the clock signal. This content or state (which is as usual the output of the latch as well) remains unchanged until the next transparent mode, no matter how input D changes, as illustrated in Figure 9. More specifically, while the clock signal undergoes a high-to-low transition at t_0, the logic value of input D is 0, which is sampled by this high-to-low clock transition, and then kept in the latch during the following hold mode, which starts at t_0 and ends at t_1. Before the first positive edge of the clock signal we assume that we do not know the content (or state) of this latch. The shaded region in the output waveform shown in Figure 9 signifies this uncertainty.

Propagation delay times have not been taken into consideration in Figure 9. But similar to any other physical circuits, a real latch has some propagation delay time as shown in Figure 10.

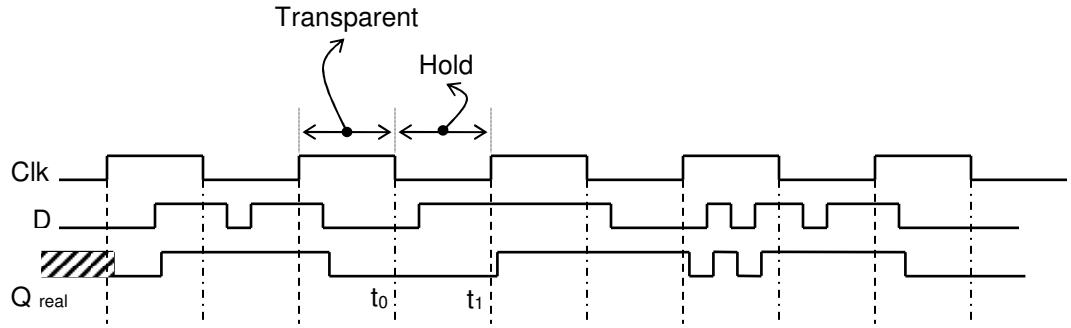


Figure 10. Real D-latch timing diagram

While D-latches have their own applications in digital systems, the transparent mode or *level* sensitivity to Clk causes a major problem in most synchronous logic circuits to be addressed later in this book. To solve this problem we utilize memory cells called *flip flops*, *FFs* or *flops* for short, that are written with clock edges only (either positive or negative) and not clock levels anymore. Before FFs are elaborated on, read the following facts thoroughly:

1- A FF is a memory cell, so it can be read from and written to as well.

2- As long as the output of a FF is accessible, that FF is in the read mode; i.e., we don't have to apply any signal to read a FF, unlike write operations to be elaborated on in this chapter. Remember that what makes the three types of FFs (mentioned above) different is the way that they are written in.

3- Each type of FF is further categorized into positive-edge triggered and negative-edge triggered. A positive-edge triggered FF needs a positive edge of the clock signal to perform a write operation. (We will see the details shortly.) Similarly, a negative-edge triggered FF needs a negative edge of the clock signal to perform a write operation. Therefore, if the clock signal is gated (remember signal gating from Chapter 2), clock edges will be prevented from reaching the FF, and therefore the FF's content will never change. The edge to which the FF is sensitive, is called the *active* edge of clock. In this book by a clock edge we mean an active edge unless otherwise specified. We also assume that positive edges are always active (i.e., all FFs are positive-edge triggered) unless otherwise specified.

4- Since Clk is a global signal, an immediate consequence of edge sensitivity of FFs is that all write operations into FFs in a digital system are carried out simultaneously, and synchronized with Clk's active edges.

5- The output of a FF shows the content or *state* of that FF; so, these three terms (output, content and state) may be used interchangeably.

6- The *state* of a digital system at a given time is comprised of the states of all FFs in that system at that specific instant of time. Therefore, the state of a system may only change with an active edge of Clk. (We know that the state of each individual FF may only change with an active edge of Clk.)

7- The memory cells that we consider here are *volatile*, i.e., they lose their contents when the power turns off. Additionally, when the power turns on a memory cell takes an unpredictable content.

Flip Flops: Edge-Triggered Memory Cells

The content of a positive or negative-edge-triggered flip flop may only change with a positive or negative edge, respectively, of the clock signal; i.e., a clock edge is necessary to perform a write operation into FFs. In other words, FFs are only sensitive to the positive (or negative) edges of clock and not to the clock's levels anymore. Therefore, *during each clock period only one write operation (change of state) is possible in a FF, as there is exactly one positive (and also one negative) edge in every clock period.* More specifically, since the output of FF may only change with a clock edge, the output stays unchanged until the next clock edge, i.e., **time is *quantized* as long as states of FFs are concerned**. That is why instead of considering time as a continuous variable, and specifying different instants of time in terms of, say, nanoseconds (1 nanosecond = 10^{-9} second), we usually talk about the *current (present) state, previous state* or *next state* of a FF. Each state will last *at least* one clock period or one time quantum.

In this section three different FFs (namely, D-FFs, T-FFs and JK-FFs) are introduced as three extra building blocks for synchronous logic. Remember that at any instant of time the states of all FFs in a sequential circuit comprise the state of the whole system at that instant of time. And the state of the whole system at any instant of time is the history that the system remembers from the past at that instant of time.

D-FFs

Figure 11*a* shows a logic symbol for a D-FF, which has one (data) input, D, a clock input, Clk, and two outputs, Q and inverted Q (Q').

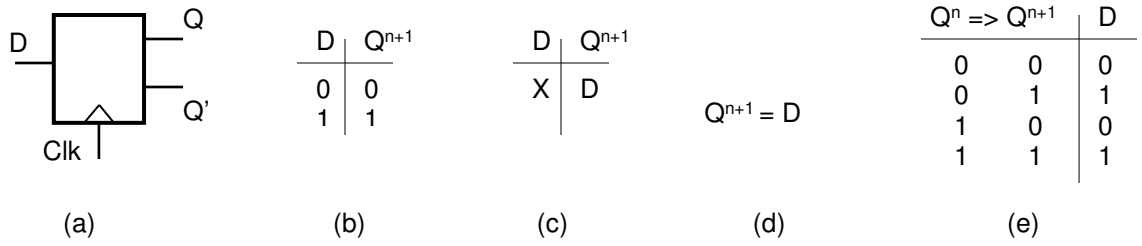


Figure 11. D-FF: (a) symbol, (b) characteristic table, (c) compressed characteristic table, (d) characteristic equation, (e) excitation table

The characteristic table of a D-FF is shown in Figure 11*b*. This table tells us how the output changes for a given D (upon the arrival of the next clock edge). Note that Q^{n+1} means the output during clock cycle n + 1; hence Q^n means the output during clock cycle n; i.e., Q^n and Q^{n+1} represent the output during two consecutive clock cycles. Without the loss of generality this statement may be reworded as follows: Q^n (or just Q) is the current state, and Q^{n+1} is the next state of the memory cell. By definition, the clock edge that comes between cycle n and n + 1 is called clock edge n. So, clock edge 6, for example, comes right after clock cycle 6 and right before clock cycle 7.

Let's take a closer look to understand the characteristic table:

The first row reads "if a clock edge arrives while D = 0, the FF will store this value; i.e., if its content was 0 before the clock edge, it would remain 0 after the clock edge. And, if it was 1 before the clock edge it would change to 0 after the clock edge."

The second row reads "if a clock edge arrives while D = 1, the FF will store this value; i.e., if its content was 1 before the clock edge, it would remain 1 after the clock edge. And, if it was 0 before the clock edge, it would change to 1 after the clock edge."

Figure 11*c* shows a compressed characteristic table for a D-FF. The algebraic version of the characteristic table (of D-FF) is called the *characteristic equation* (of D-FF), which is shown in Figure 11*d*. D (or D^n) means the current input.

Figure 12 shows a timing diagram for a positive-edge-triggered D-FF, i.e., the Q waveform for a given D waveform. A timing diagram (for a FF) is obtained by considering the definition (or characteristic table) of that specific FF and also the input waveforms applied to the FF.

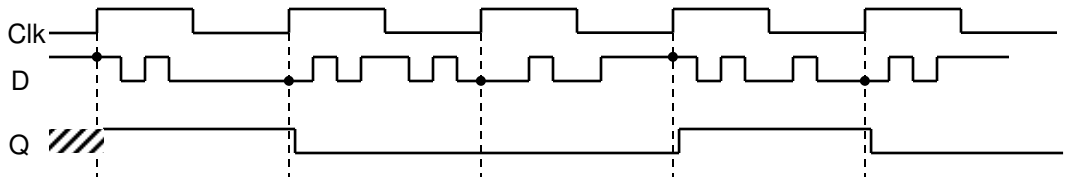


Figure 12. Timing diagram for positive-edge-triggered D-FF

Figure 13 shows a timing diagram for a negative-edge-triggered D-FF, with the same D waveform employed in Figure 12.

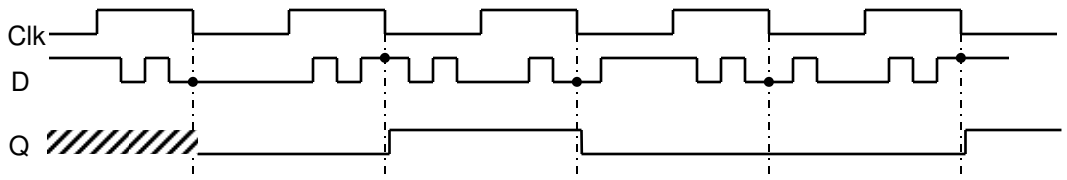


Figure 13. Timing diagram for negative-edge-triggered D-FF

The excitation table of a D-FF is shown in Figure 11*e*. This table is the reverse of the characteristic table and determines the correct D input required for each individual output change. Let's take a closer look to see how this table reads:

The first row reads "If the output during the current cycle is 0, and we want it to stay 0 during the next cycle, then D has to be 0 when the following clock edge arrives".

The second row reads "If the output during the current cycle is 0, and we want it to become 1 during the next cycle, then D has to be 1 when the following clock edge arrives".

The third reads "If the output during the current cycle is 1, and we want it to become 0 during the next cycle, then D has to be 0 when the following clock edge arrives".

The last row reads "If the output during the current cycle is 1, and we want it to stay 1 during the next cycle, then D has to be 1 when the following clock edge arrives".

It should be highlighted here that the characteristic table, characteristic equation and excitation table of a D- (or any) FF contain the same amount of information but in different formats. In other words, each of these three data structures can be obtained from either of the other two. We use characteristic tables (or equations) for circuit analysis (as elaborated on in the following section) and excitation tables for circuit design to be covered in Chapter 8.

A D-FF may be built up of two cascaded D-latches, as shown in Figure 14. In this design the Q output of the first latch (master) is tied to the D input of the second latch (slave). The D input of the master is the D input of the flop, and the Q output of the slave is the Q output of the flop. The clock signals applied to the two latches are the complements of each other. The resulting flop is also called a *master-slave* D-FF. The flop shown in this figure is negative-edge triggered. If the inverter is moved to the clock input of the master, a positive-edge-triggered D-flop would be obtained.

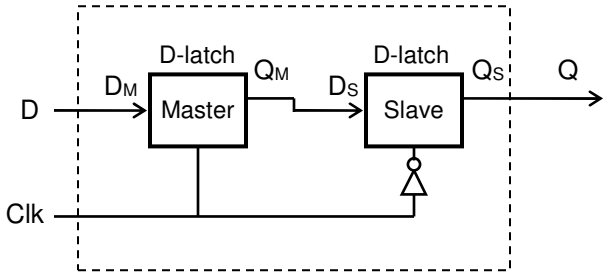


Figure 14. Negative-edge-triggered D-FF comprised of a master latch and a slave latch

Figure 15 shows a timing diagram for these two latches, hence for the flop. As long as Clk is high, the master is in the transparent mode and the slave is in the hold mode. So, the logic value at input D appears at Q_m (or D_s), but cannot pass through the slave, as it is in the hold mode. When Clk goes down, the logic value at input D (at the instant of this falling edge of Clk) is trapped in the master, passes through the

slave (because the slave is now in the transparent mode) and appears at the output of the slave, after some propagation delay time.

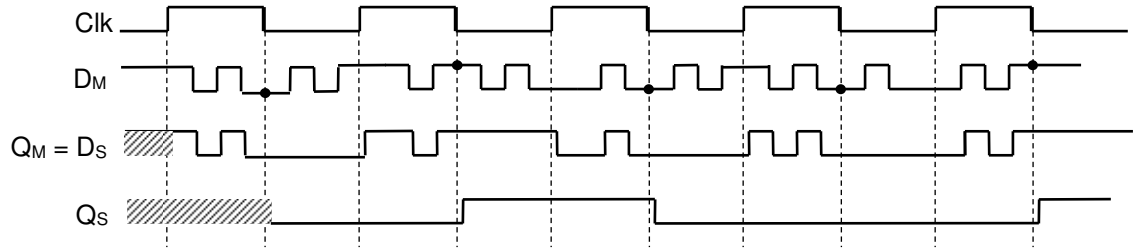


Figure 15. Timing diagram for a master-slave D-FF

T-FFs

Figure 16*a* shows a logic symbol for a toggle FF or T-FF for short, which has one (control) input, T, a clock input, Clk, and two outputs, Q and inverted Q (Q').

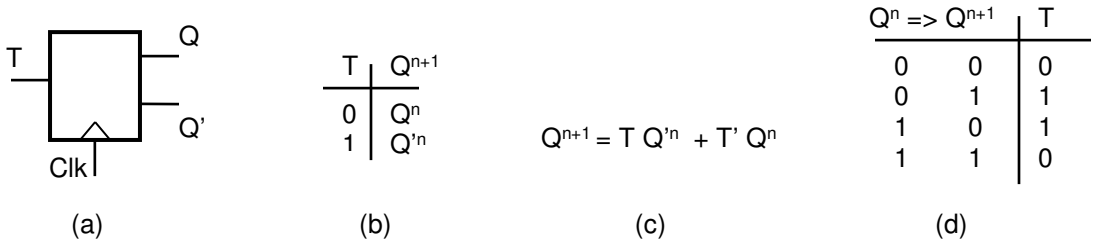


Figure 16. T-FF: (a) symbol, (b) characteristic table, (c) characteristic equation, (c) excitation table

The characteristic table of a T-FF is shown in Figure 16*b*. This table tells us how the output changes for a given T (upon the arrival of the next clock edge). Q^n or just Q is the current state, and Q^{n+1} is the next state of the memory cell.

Let's take a closer look to see how the characteristic table reads:

The first row reads "If a clock edge arrives while T = 0, then the content of FF remains unchanged; i.e., if it was 0 before the clock edge, it would remain 0 after the clock edge. And, if it was 1 before the clock edge, it would remain 1 after the clock edge".

The second row reads "If a clock edge arrives while T = 1, then the content of FF will change or toggle; i.e., if it was 0 before the clock edge, it would change to 1 after the clock edge. And, if it was 1 before the clock edge, it would change to 0 after the clock edge".

The algebraic version of the characteristic table (of T-FF) is called the *characteristic equation* (of T-FF), which is shown in Figure 16*c*. T (or T^n) means the current input.

Figure 17 shows a timing diagram for a positive-edge-triggered T-FF, i.e., the Q waveform for a given T waveform. In this example we assume that the initial state of the FF is known (say logic low); otherwise if the initial state was unknown, the T-FF's state would remain unknown for ever!

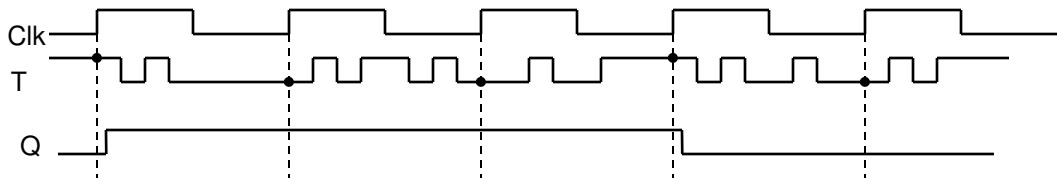


Figure 17. Timing diagram for positive-edge-triggered T-FF

Figure 18 shows a timing diagram for a negative-edge-triggered T-FF, with the same T waveform employed in Figure 17.

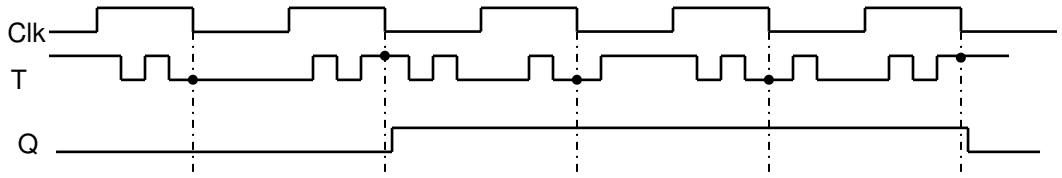


Figure 18. Timing diagram for negative-edge-triggered T-FF

The excitation table of a T-FF is shown in Figure 16*d*. This table is the opposite of the characteristic table and determines the correct T input required for each individual output change. Let's take a closer look to see how this table reads:

The first row reads "If the output during the current cycle is 0, and we want it to stay 0 during the next cycle, then T has to be 0 when the following clock edge arrives".

The second row reads "If the output during the current cycle is 0, and we want it to become 1 during the next cycle, then T has to be 1 when the following clock edge arrives".

The third row reads "If the output during the current cycle is 1, and we want it to become 0 during the next cycle, then T has to be 1 when the following clock edge arrives".

The last row reads "If the output during the current cycle is 1, and we want it to stay 1 during the next cycle, then T has to be 0 when the following clock edge arrives".

JK-FFs

Figure 19*a* shows a logic symbol for a JK-FF, which has two (control) inputs, J and K , a clock input, Clk, and two outputs, Q and inverted Q (Q').

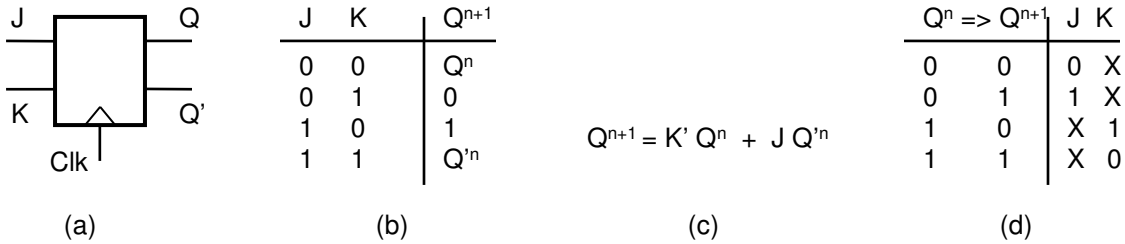


Figure 19. JK-FF: (a) symbol, (b) characteristic table, (c) characteristic equation, (c) excitation table

The characteristic table of a JK-FF is shown in Figure 19*b*. This table tells us how the output changes for each individual combination of J and K (upon the arrival of the next clock edge). Q^n or just Q is the current state, and Q^{n+1} is the next state of the memory cell.

Let's take a closer look to see how the characteristic table reads:

The first row reads "If a clock edge arrives while J = K = 0, then the content of FF will not change; i.e., if it was 0 before the clock edge, it would remain 0 after the clock edge. Similarly, if the content of FF was 1 before the clock edge, it would remain 1 after the clock edge".

The second row reads "If a clock edge arrives while J = 0 and K = 1, then the content of FF after the clock edge will be 0, no matter what it was before the clock edge".

The third row reads "If a clock edge arrives while J = 1 and K = 0, then the content of FF after the clock edge will be 1 no matter what it was before the clock edge".

The last row reads "If a clock edge arrives while J = 1 and K = 1, then the content of FF after the clock edge will be the complement of what it was before the clock edge (toggle mode)".

The algebraic version of the characteristic table (of JK-FF) is called the *characteristic equation* (of JK-FF), which is shown in Figure 19*c*. J K (or J^n K^n) mean the current inputs.

Figure 20 shows a timing diagram for a JK-FF, i.e., the Q waveform for a given pair of J and K waveforms.

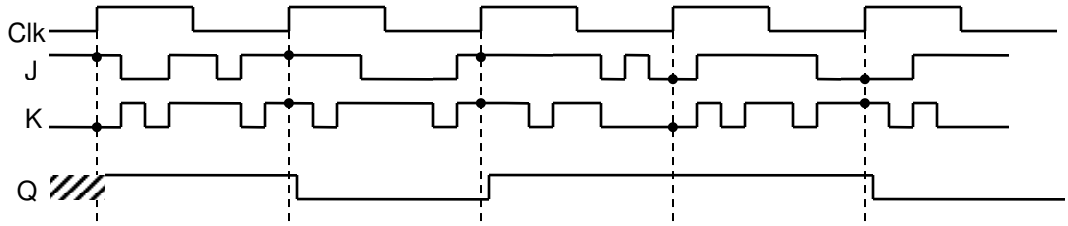


Figure 20. Timing diagram for positive-edge-triggered JK-FF

Figure 21 shows a timing diagram for a negative-edge-triggered JK-FF with the same J and K waveforms employed in Figure 20.

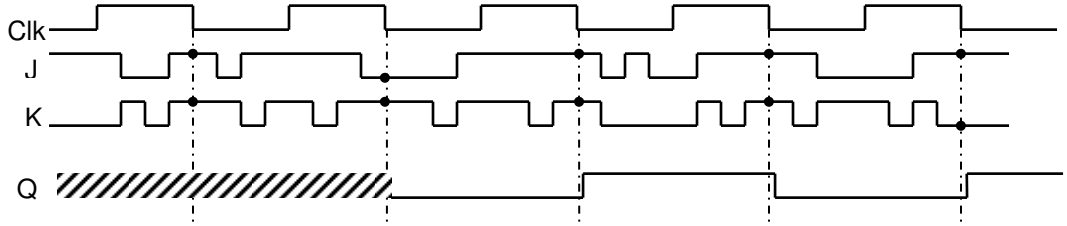


Figure 21. Timing diagram for negative-edge-triggered JK-FF

The excitation table of a JK-FF is shown in Figure 19*d*. This table is the opposite of the characteristic table and determines the correct combination of J and K inputs required for each individual output change. Let's take a closer look to see how this table reads:

The first row reads "If during the current cycle the output is 0, and we want it to stay 0 during the next cycle, then J has to be 0 when the following clock edge arrives, but K could be either 0 or 1".

The second row reads "If during the current cycle the output is 0, and we want it to become 1 during the next cycle, then J has to be 1 when the following clock edge arrives, but K could be either 0 or 1".

The third row reads "If during the current cycle the output is 1, and we want it to become 0 during the next cycle, then K has to be 1 when the following clock edge arrives, but J could be either 0 or 1".

The last row reads "If during the current cycle the output is 1, and we want it to stay 1 during the next cycle, then K has to be 0 when the following clock edge arrives, but J could be either 0 or 1".

Circuit Analysis

By "analysis of a digital circuit" we mean a procedure to reach the highest possible level of description for that circuit. As you remember the outcome of the analysis of a combinational circuit is usually a truth table. In the manual design of combinational circuits truth tables are normally considered the highest level of description reached right after the natural-language description of the problem in hand. Now the question is "What are we supposed to obtain as the highest level of description for sequential circuits"? We will shortly reach the answer after some background has been developed.

If a sequential circuit is modeled as shown in Figure 22, then it is usually called a *finite state machine*, *state machine* or FSM for short.

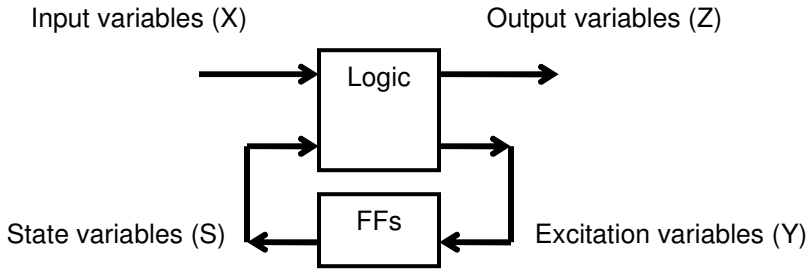


Figure 22. Generic model for finite state machines

In this figure you see two separate blocks, one containing combinational logic circuits (top) and the other one containing one or more FFs. The variables in this model are categorized into four different types: 1) Input variables (coming from the outside world) that are in fact the feedback received by this finite state machine from the environment; 2) Output variables that are the means by which this system impacts on the outside world; 3) State variables that are the outputs of the FFs, or the history that this machine keeps from the past; and, 4) excitation variables that are the inputs to the memory cells. The FFs (memory cells) need excitation variables to get properly updated over time.

Pay close attention to see how/where different signals are produced and how/where they are consumed: the combinational logic circuit receives input variables (from the outside world) and also state variables (from the memory), and produces output variables and also excitation variables. Output variables are consumed in the outside world; excitation variables, however, are internal signals applied to the FFs to provide them with the right signals for write operations. Look at this generic block diagram in Figure 22 again; each output variable is in general a function of input variables and state variables[2]. Similarly, each excitation variable is also (in general) a function of input variables and state variables.

Remember that the content of a FF at each instant of time is called the state of that FF at that instant of time. Additionally, the set of states of all FFs at any given time is called the state of the underlying digital circuit at that specific instant of time. On the other hand, with each individual active clock edge one or more FFs might be updated (overwritten). This means that the state of the whole system changes over time as illustrated in Figure 23, for example. Additionally, these changes are synchronized with active

[2] In a subset of FSMs, the output variables of a FSM depend only on the state variables, as you will see in Chapter 8.

edges of the clock signal; no state changes may occur without the arrival of an active clock edge. Now the question is **"**For a given digital circuit and following one specific state (current state), which state is reached next (next state)"? If we manage to answer this question for each individual current state, then we have in fact been able to *analyze* the state machine in hand. The graph that shows the next state for each individual current state is called a *state graph* or *state diagram*. Therefore, in the analysis of a digital circuit the goal is to obtain a state graph (or state diagram) for the circuit as elaborated on shortly. This graph illustrates how the corresponding circuit steps through all possible states. Following the description at the natural-language level, state graphs are considered the highest level of description for state machines.

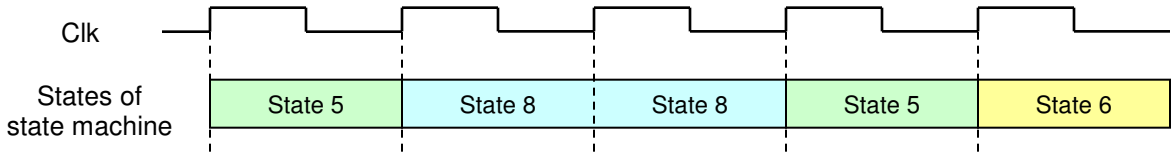


Figure 23. An example: state changes in a state machine

A classic methodology for the analysis of D-FF-based state machines will be detailed in Example 2 to obtain a state diagram (or state graph). Then similar procedures will be repeated for a T-type and for a JK-type state machine in Example 3 and Example 4, respectively. Notice that different types of FFs may be used in the same state machine. For the analysis of this type of state machine each type of FF must be processed in accordance with its own specific procedure.

Example 2. Analyze the circuit illustrated in Figure 24 and obtain a state graph for the state machine represented by this circuit.

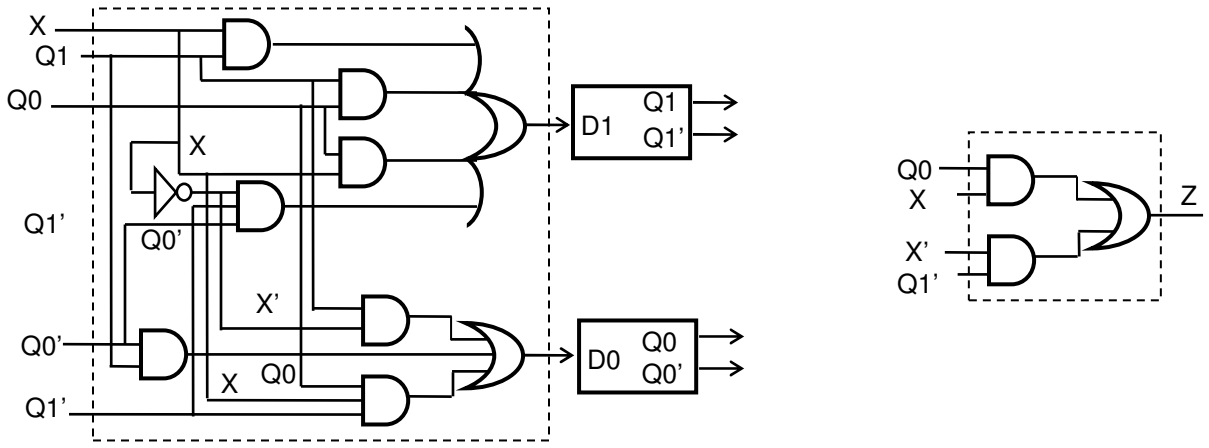


Figure 24. Digital circuit in Example 2

As shown in Figure 24, this state machine has two D-FFs, some combinational logic surrounded by two dotted boxes, one input variable, X, and one output variable, Z. Figure 25 illustrates the same state machine but in the generic form of Figure 22. The details of the dotted box (the combinational circuit) have not been shown in Figure 25 for the sake of clarity. As clearly illustrated in this figure, Q1 and Q0 are the two state variables, and D1 and D0 are the two excitation variables of this machine.

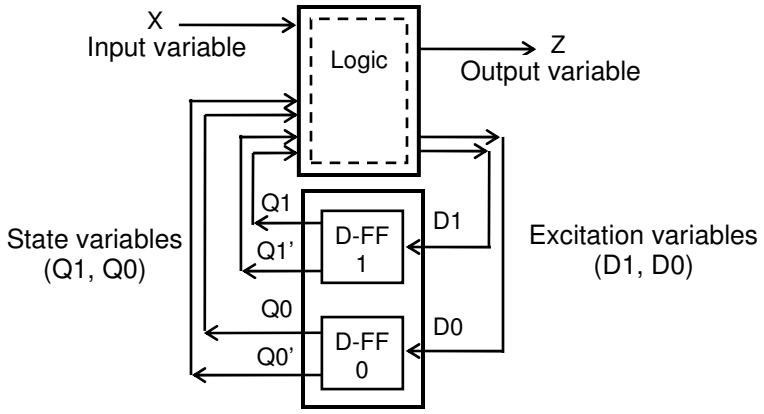


Figure 25. State machine of Example 2 redrawn in generic format

Definition. Excitation equation: An *excitation* equation expresses an excitation variable in terms of state and input variables.

1) Obtain excitation equations and output equation: Inspect the combinational circuit of the state machine (shown in Figure 24) to obtain the excitation equations and output equation:

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

Definition. Excitation map: The K-map corresponding to an excitation equation (of an excitation variable) is called the *excitation map* of that variable. (So, state variables and input variables are the coordinate variables of excitation maps.)

2) Obtain excitation maps: Use the excitation equations to obtain excitation maps for D1 and D0, as illustrated in Figure 26*a* and Figure 26*b*, respectively. Also use the output equation to obtain the output K-map (or simply output map), as shown in Figure 26*c*. In each of these maps list the state variables and input variables vertically and horizontally, respectively.

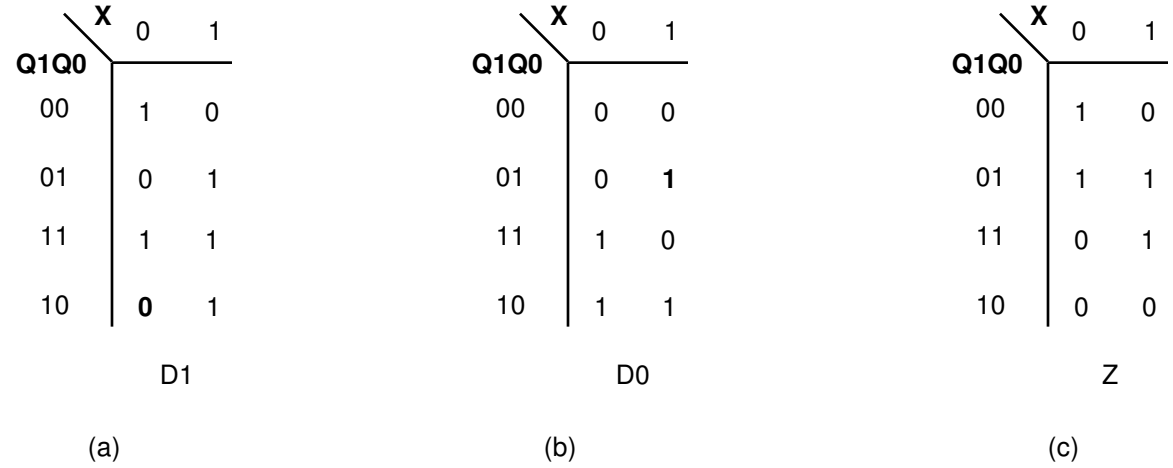


Figure 26. For Example 2: (a) excitation map for D1, (b) excitation map for D0, (c) output map

Definition. Partial transition table: In a state machine every state variable (or FF) has its own partial transition table with the same size and coordinates as the corresponding excitation map. A partial transition table shows the FF's next state for each individual combination of current state (of the finite-state machine) and input value. (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.)

3) Obtain partial transition tables: Considering the characteristic equation of a D-FF, namely $Q^{n+1} = D$, the excitation map and partial-transition table of a D-FF are always the same. Figure 27*a* and Figure 27*b* show the partial transition tables of D-FF1 and D-FF0, respectively. (But this is not the case with other two FFs to be studied shortly.)

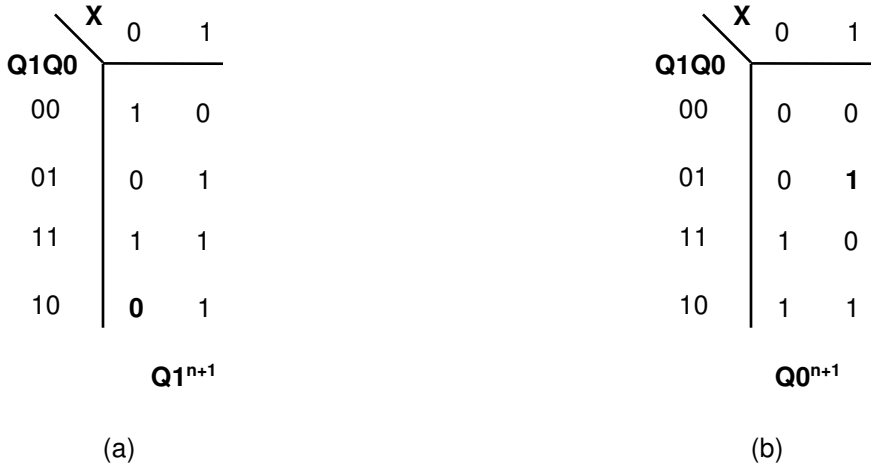


Figure 27. For Example 2: (a) partial transition table for FF1, (b) partial transition table for FF0

4) Obtain a transition table: Combine the two partial transition tables developed above and also include the output data from the output map shown in Figure 26*c*, to obtain a *transition table* for the whole circuit, as shown in Figure 28*a*.

5) Obtain a state table: Assign a name (such as a, b , …) to each numeric state in the transition table to obtain a *state table*, as illustrated in Figure 28*b*.

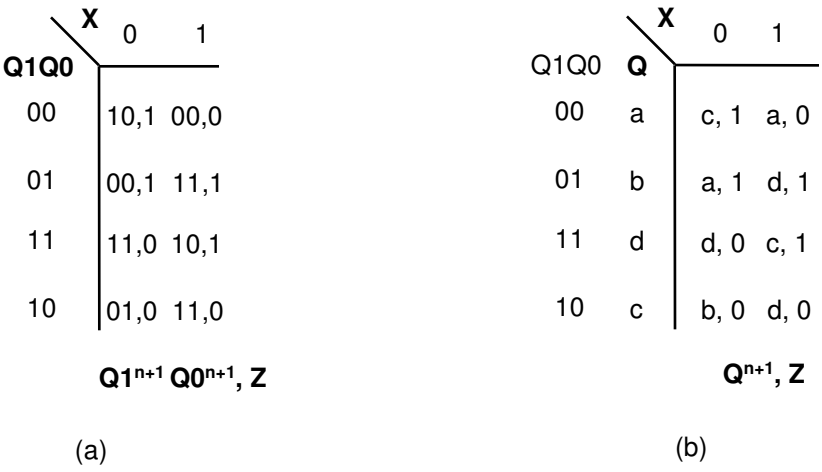


Figure 28. For Example 2: (a) transition table, (b) state table

6) Obtain a transition diagram and a state diagram: A transition table can easily be converted to its graphical version, namely *transition diagram* or *transition graph*. This graph has as many nodes as the number of states in the corresponding transition table. Additionally, there is a directed edge from a state such as Si to another state, such as Sj if Sj can be reached directly from Si. Figure 29*a* shows a transition diagram based on the transition table in Figure 28*a*. Output information is normally added to transition diagrams as well. Therefore, as you see in Figure 29*a* there are two different values attached to each edge in a transition diagram: X (input) and Z (output), represented as X/Z. The value corresponding to X is the condition that has to be satisfied to follow that edge and reach the corresponding destination state. For example, starting with state 11 in Figure 29*a* the graph reads "If X = 1 and an active edge of clock arrives, then the circuit will jump to state 10. However, if X = 0 when the next active edge of clock arrives, then the circuit will remain in the same state, 11".

The interpretation of the output signal deserves a special attention: again starting with state 11 the transition diagram reads "if X = 1 then Z becomes 1, **BEFORE** the following active edge of clock arrives, i.e., Z is the *current* output. If an active clock edge is received while X = 1 , then the circuit jumps to state 10, in which a 0 output will be generated should input X still equals 1."

When symbolic codes (such as a, b, …) are assigned to numeric states in a transition diagram, a state diagram is obtained as shown in Figure 29*b*. A state diagram may also be reached from the state table in the same way that a transition diagram is reached from the transition table.

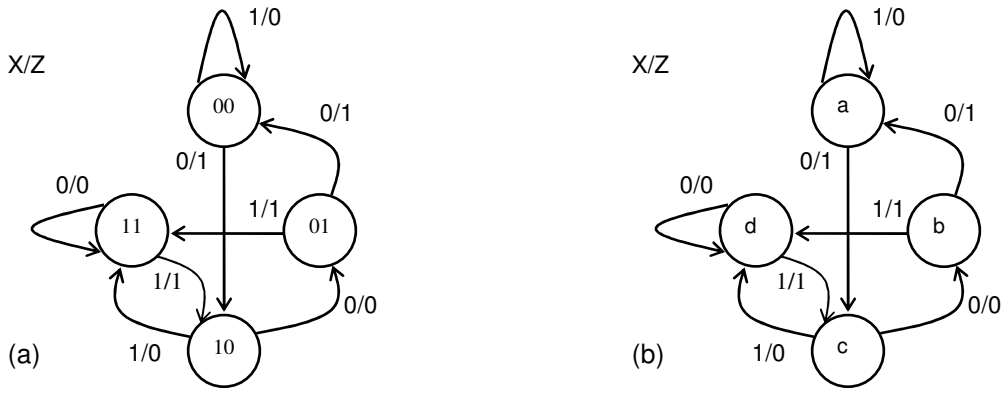


Figure 29. For Example 2: (a) transition diagram, (b) state diagram

Example 3. Analyze the circuit illustrated in Figure 30 and obtain a state graph for the state machine represented by this circuit.

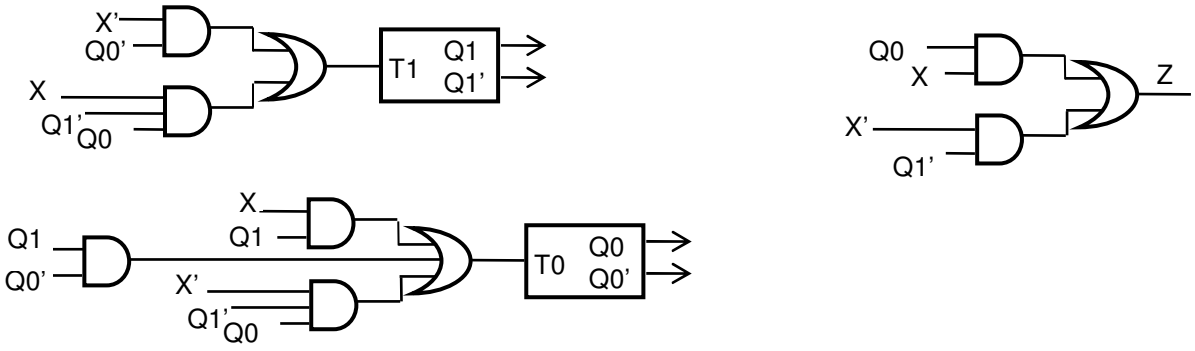


Figure 30. Digital circuit in Example 3

The definitions used in Example 2 are valid for T-type-based and JK-type-based circuits as well.

1) Inspect the combinational circuit of the state machine (shown in Figure 30) to obtain the excitation equations and output equation:

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

2) Obtain excitation maps: Use the excitation equations to obtain excitation maps for T1 and T0 as shown in Figure 31*a* and Figure 31*b,* respectively. Also use the output equation to obtain the output map, as illustrated in Figure 31*c*.

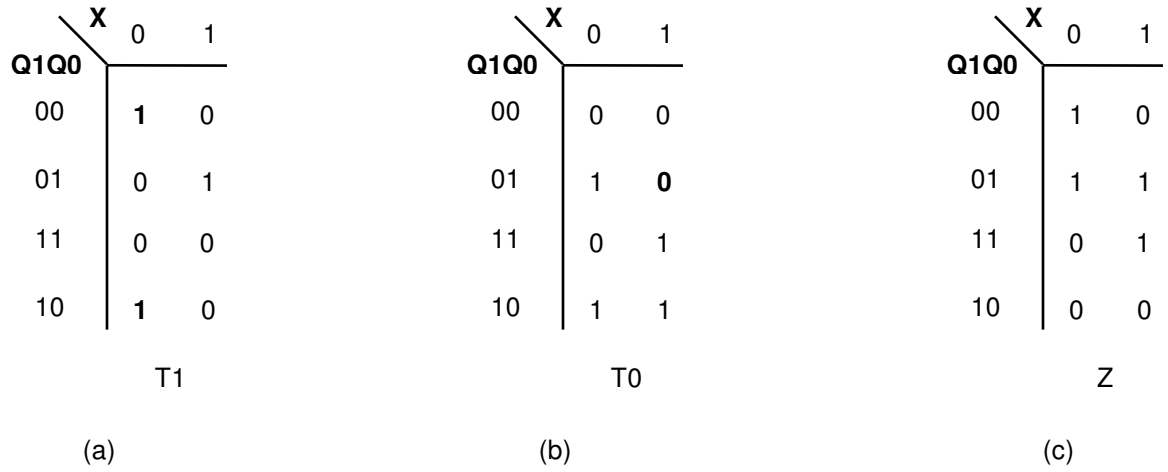


Figure 31. For Example 3: (a) excitation map for T1, (b) excitation map for T0, (c) output map

3) Obtain partial transition tables: Use the characteristic table of a T-FF, and also the excitation maps developed above to obtain a partial transition table for each of the excitation variables, T1 and T0. Remember that every state variable (or FF) has its own partial transition table with the same size and coordinates as the corresponding excitation map. A partial transition table shows the FF's next state (Q^{n+1}) for each individual combination of current state (of the finite-state machine) and input value. Two partial transition tables for this example are shown in Figure 32*a* and Figure 32*b* for T-FF1 and T-FF0, respectively. Pay close attention to how these tables are filled out: each position is filled in by examining the same position in the excitation map of the same FF shown in Figure 31. For example, position {row 00, column 0} in the excitation map of Figure 31*a* reads "If X = 0 and Q1 Q0 = 00, then T1 = 1". Now the same position in the partial transition table shown in Figure 32*a* is filled out as follows: since in this position T1 = 1, T-FF1 will toggle with the next clock edge, as prescribed by the characteristic table of a T-FF. In this position $Q1^n$ (the current state of T-FF1) = 0, as shown in the coordinates of this position; therefore position {row 00, column 0} in the partial transition table in Figure 32*a* receives a 1.

As another example, consider position {row 01, column 1} in the excitation map of Figure 31*b*, which reads "if X = 1 and Q1 Q0 = 01, then T0 = 0". Now the same position in the partial transition table shown in Figure 32*b* is filled out as follows: since in this position T0 = 0, T-FF0 will remain unchanged with the next clock edge, as prescribed by the characteristic table of a T-FF. But in this position $Q0^n$ (the current state of T-FF0) = 1, as shown in the coordinates of this position; therefore position {row 01, column 1} in the partial transition table of Figure 32*b* receives a 1.

As a third example, consider position {row 10, column 0} in the excitation map in Figure 31*a*, which reads "if X = 0 and Q1 Q0 = 10, then T1 = 1". Now the same position in the partial transition table

shown in Figure 32*a* is filled out as follows: since in this position T1 = 1, T-FF1 will toggle with the next clock edge, as prescribed by the characteristic table of a T-FF. But in this position $Q1^n$ (the current state of T-FF1) = 1, as shown in the coordinates of this position; therefore position {row 10, column 0} in the partial transition table in Figure 32*a* receives a 0.

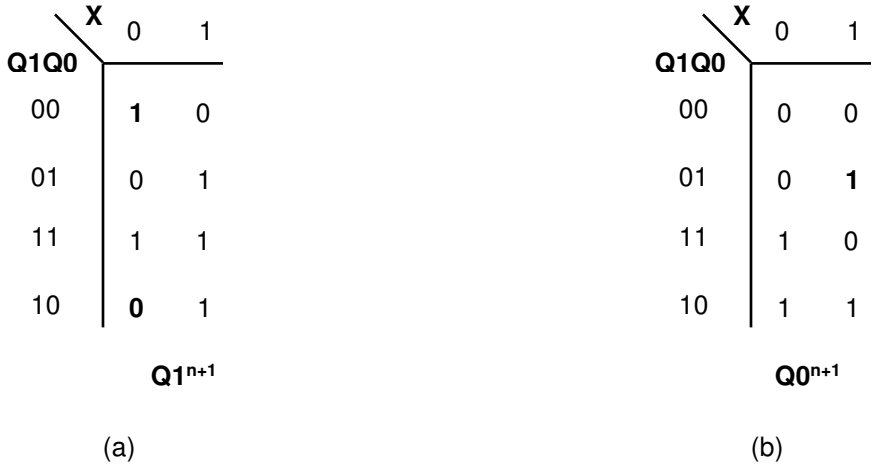


Figure 32. For Example 3: (a) partial transition table for FF1, (b) partial transition table for FF0

4) Obtain a transition table: Combine the two partial transition tables developed above and also include the output data form Figure 31*c* to obtain a transition table for the whole circuit as shown in Figure 33*a*.

5) Obtain a state table: Assign a name (such as a, b, …) to each numeric state in the transition table to obtain a state table, as illustrated in Figure 33*b*.

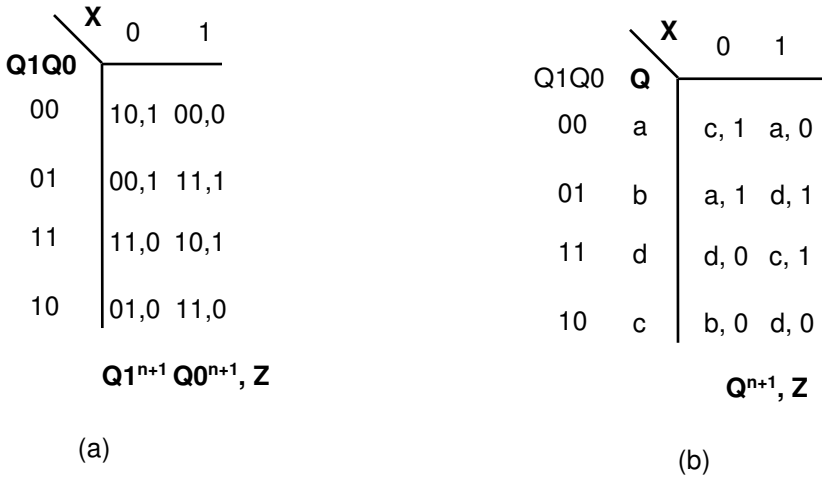


Figure 33. For Example 3: (a) transition table, (b) state table

6) Obtain a transition diagram and a state diagram: A transition table and also a state table can be converted to their graphical versions, namely a transition diagram and a state diagram, respectively, as explained in Example 2, and illustrated in Figure 34*a* and Figure 34*b*, respectively.

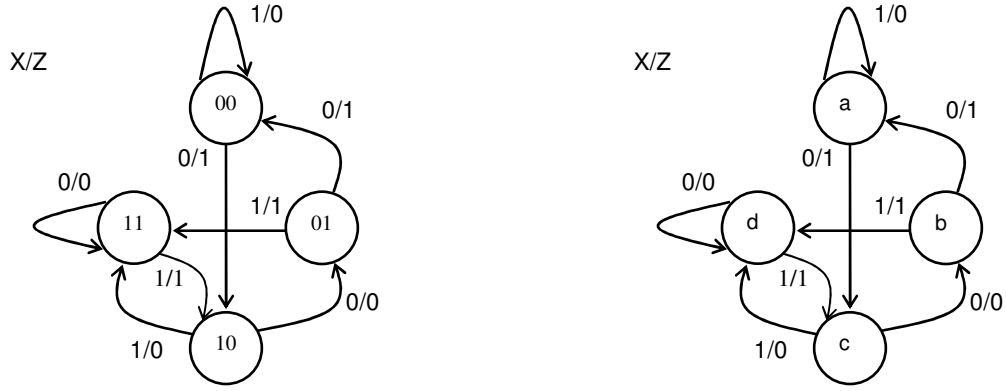


Figure 34. For Example 3: (a) transition diagram, (b) state diagram

Example 4. Analyze the circuit illustrated in Figure 35 and obtain a state graph for the state machine represented by this circuit.

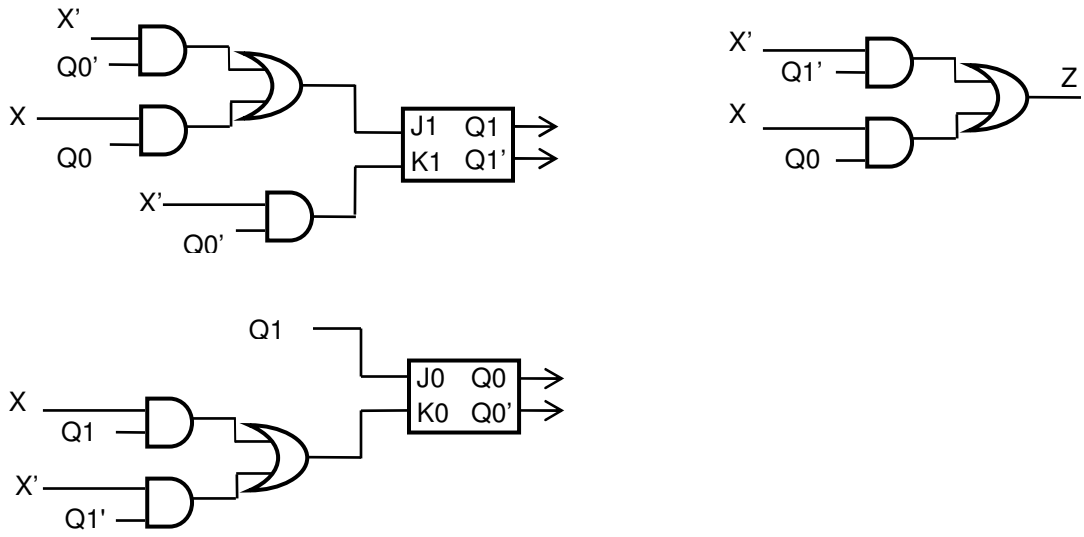


Figure 35. Digital circuit in Example 4

1) Inspect the combinational circuit of the state machine (shown in Figure 35) to obtain the excitation equations and output equation. Now there are four excitation variables, hence four excitation equations:

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

2) Use the excitation equations to obtain excitation maps for J1, K1, J0 and K0 as illustrated in Figure 36*a,* Figure 36*b,* Figure 36*c* and Figure 36*d*, respectively. Also use the output equation to obtain the output map, as shown in Figure 36*e*.

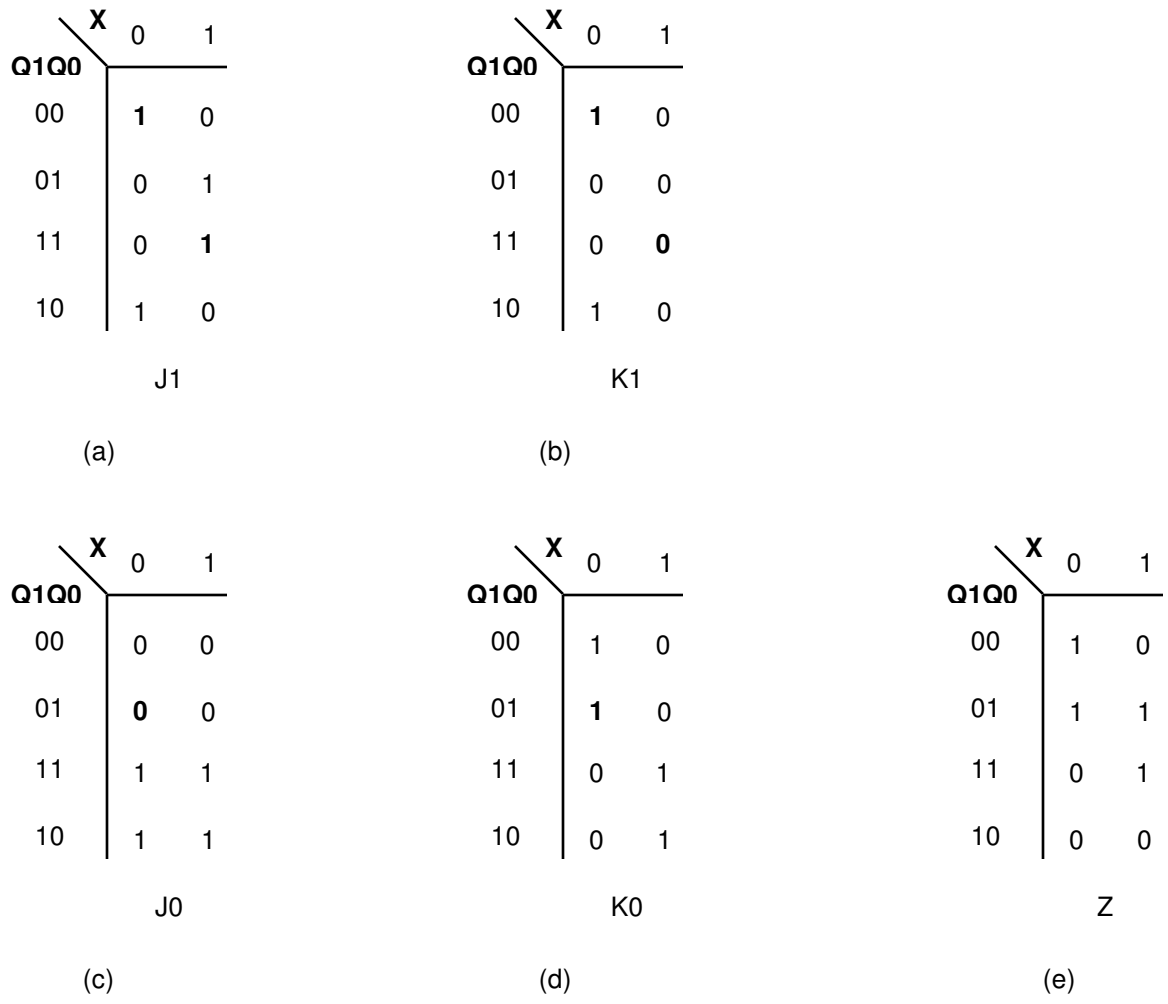


Figure 36. Excitation maps for: (a) J1, (b) K1, (c) J0, (d) K0, and (e) output map in Example 4

3) Use the characteristic table of a JK-FF, and also the excitation maps developed above to obtain partial-transition tables for excitation variables J1, K1, J0 and K0. Remember that each partial transition table shows the next state of the corresponding FF ($Q1^{n+1}$ or $Q0^{n+1}$) for each individual combination of current state (of the finite-state machine) and input value, as illustrated in Figure 37*a* and Figure 37*b* for JK-FF1 and JK-FF0, respectively. Pay close attention to see how these tables are filled out: each position in a partial transition table is filled in by examining the same positions in the two excitation maps of the same FF shown in Figure 36. For example, positions {row 00, column 0} in the excitation maps of J1 and K1 in Figure 36*a* and Figure 36*b*, respectively, read "if X = 0 and Q1 Q0 = 00, then J1 = 1 and K1 = 1". Now the same position in the partial transition table shown in Figure 37*a* is filled out as follows: since in this position J1 = 1 and K1 = 1, JK-FF1 will toggle with the next clock edge, as prescribed by the characteristic table of a JK-FF. In this position $Q1^n$ (the current state of JK-FF1) = 0, as shown in the coordinates of this position; therefore position {row 00, column 0} in the partial transition table in Figure 37*a* receives a 1.

As another example, consider positions {row 11, column 1} in the excitation maps of J1 and K1 in Figure 36*a* and Figure 36*b*, respectively, which read "if X = 1 and Q1 Q0 = 11 then J1 = 1 and K1 = 0". Now the same position in the partial transition table shown in Figure 37*a* is filled out as follows: since in this position J1 = 1 and K1 = 0, JK-FF1 will be set with the next clock edge, no matter what its current state

is, as prescribed by the characteristic table of a JK-FF. Therefore, position {row 11, column 1} in the partial transition table of Figure 37*a* receives a 1.

As a third example, consider positions {row 01, column 0} in the excitation maps of J0 and K0 in Figure 36*c* and Figure 36*d*, respectively, which read "if X = 0 and Q1 Q0 = 01 then J0 = 0 and K0 = 1". Now the same position in the partial transition table shown in Figure 37*b* is filled out as follows: since in this position J0 = 0 and K0 = 1, JK-FF0 will be reset with the next clock edge, no matter what its current state is, as prescribed by the characteristic table of a JK-FF. Therefore, position {row 01, column 0} of the partial transition table in Figure 37*b* receives a 0.

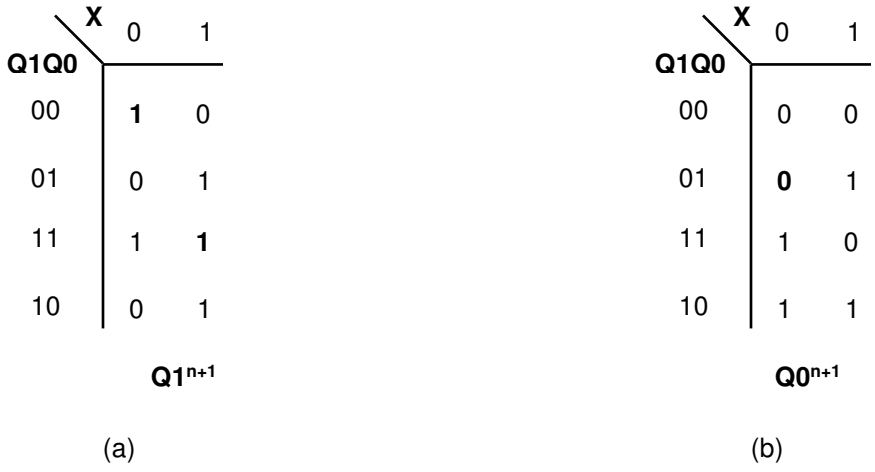


Figure 37. For Example 4: (a) partial transition table for FF1, (b) partial transition table for FF0

4) Combine the two partial transition tables developed above and also include the output data from the output map shown in Figure 36*e*, to obtain a transition table for the whole circuit as shown in Figure 38*a*.

5) Assign a name to each numeric state in the transition table to obtain a state table, as illustrated in Figure 38*b*.

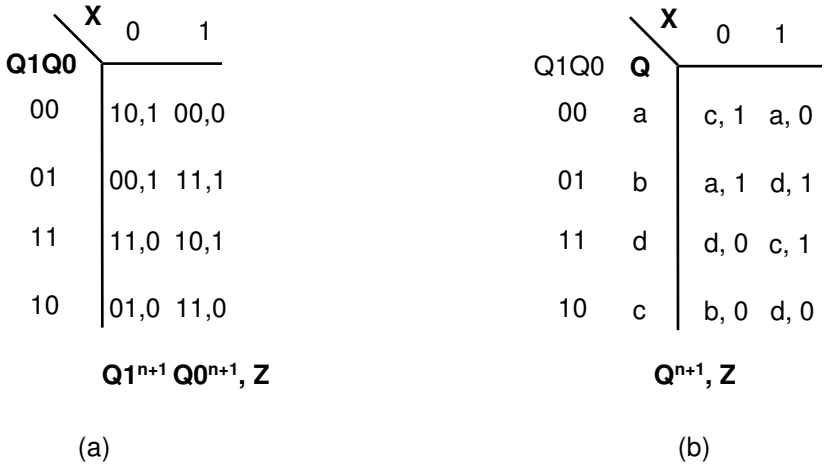


Figure 38. For Example 4: (a) transition table, (b) state table

6) A transition table and also a state table can be converted to their graphical versions, namely a transition diagram and a state diagram, respectively, as explained in Example 2, and illustrated in Figure 39*a* and Figure 39*b*, respectively.

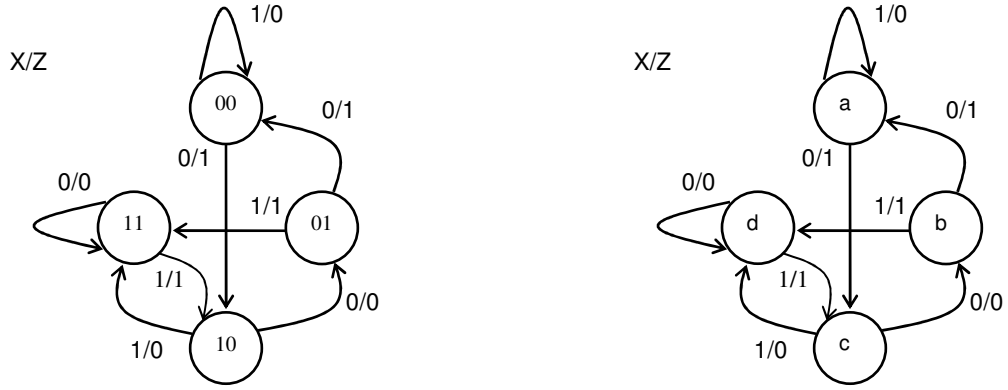


Figure 39. For Example 4: (a) transition diagram, (b) state diagram