3 Sequential Logic

CHAPTER MAP

2 Logic elements and Boolean algebra

We begin our introduction to the computer with the basic building block from which we construct all computers, the gate. A combinational digital circuit such as an adder is composed of gates and its output is a Boolean (logical) function of its inputs only.

3 Sequential logic

The output of a sequential circuit is a function of both its current inputs and its past inputs; that is, a sequential circuit has memory. The building blocks used to construct devices that store data are called flip-flops. In this chapter we look at basic sequential elements and the counters, registers, and shifters that are constructed from flip-flops.

4 Computer arithmetic

Computer arithmetic concerns the representation of numbers in a computer and the arithmetic used by digital computers. We look at how decimal numbers are converted into binary form and the properties of binary numbers and we demonstrate how operations like addition and subtraction are carried out. We also look at how computers deal with negative numbers and fractional numbers.

5 The instruction set architecture

In this chapter we introduce the computer's instruction set architecture (ISA), which describes the low-level programmer's view of the computer. The ISA describe the type of operations a computer carries out. We are interested in three aspects of the ISA: the nature of the instructions, the resources used by the instructions (registers and memory), and the ways in which the instructions access data (addressing modes). The 68K microprocessor is used to illustrate the operation of a real device.

INTRODUCTION

We now introduce a new type of circuit that is constructed from devices that *remember* their previous inputs. The logic circuits in Chapter 2 were all built with *combinational elements* whose outputs are functions of their inputs *only*. Given a knowledge of a combinational circuit's inputs and its Boolean function, we can always calculate the state of its outputs. The output of a *sequential circuit* depends not only on its current inputs, but also on its *previous* inputs. Even if we know a sequential circuit's Boolean equations, we can't determine its output state without knowing its past history (i.e. its previous internal states). The basic building blocks of sequential circuits are the *flip-flop, bistable*, and *latch* just as the basic building block of the combinational circuit is the gate.

It's not our intention to deal with sequential circuits at anything other than an introductory level, as their full treatment forms an entire branch of digital engineering. Sequential circuits can't be omitted from introductory texts on computer hardware because they are needed to implement registers, counters, and shifters, all of which are fundamental to the operation of the central processing unit.

Figure 3.1 describes the conceptual organization of a sequential circuit. An input is applied to a combinational circuit using AND, OR, and NOT gates to generate an output that is fed to a memory circuit that holds the value of the output. The information held in this memory is called the *internal state* of the circuit. The sequential circuit uses its previous output together with its current input to generate the *next* output. This statement contains a very important implicit concept, the idea of a *next state*. Sequential circuits have a clock input, which triggers the transition from the *current* state to the *next* state. The *counter* is a good example of a sequential machine because it stores the current count that is updated to become the next count. We ourselves are state machines because our future behavior depends on our past

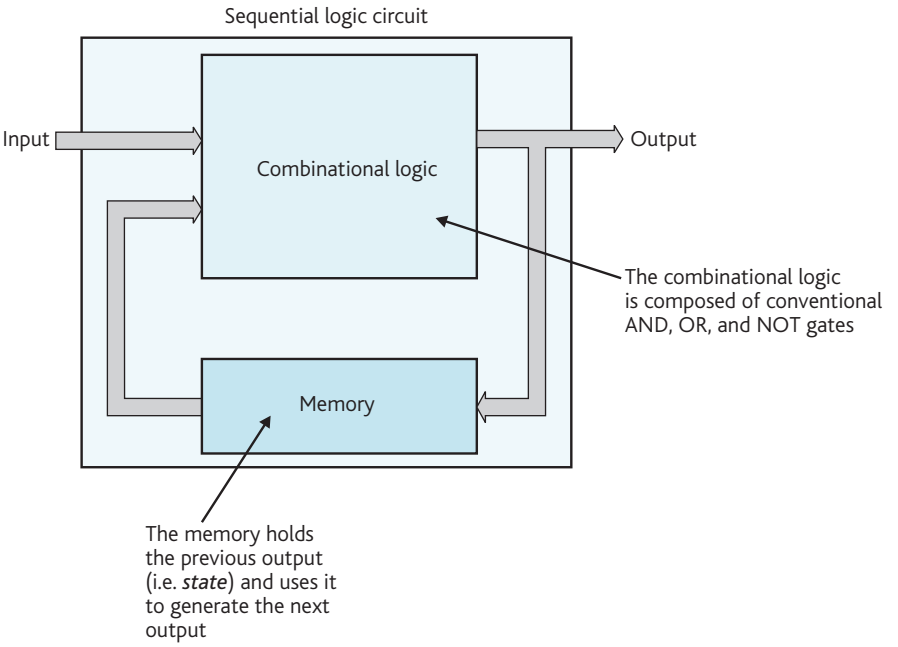


Figure 3.1 The sequential circuit.

WHAT IS SEQUENTIAL LOGIC?

Sequential logic elements perform as many different functions as combinational logic elements; however, they do carry out certain well-defined functions, which have been given names.

Latch A latch is a 1-bit memory element. You can capture a single bit in a latch at one instant and then use it later; for example, when adding numbers you can capture the carry-out in a latch and use it as a carry-in in the next calculation.

Register The register is just *m* latches in a row and is able to store an *m*-bit word; that is, the register is a device that stores one memory word. A computer's memory is just a very large array of registers.

Shift register A shift register is a special-purpose register that can move the bits of the word it holds left or right; for example the 8-bit word 00101001 can be shifted left to give 01010010.

Counter A counter is another special-purpose register that holds an *m*-bit word. However, when a counter is triggered (i.e. clocked) its contents increase by 1; for example, if a counter holding the binary equivalent of 42 is clocked, it will hold the value 43. Counters can count up or down, by 1 or any other number, or they can count through any arbitrary sequence.

State machines A state machine is a digital system that moves from one state to another each time it is triggered. You can regard a washing machine controller as a state machine that steps though all the processes involved in washing (at a rate depending on the load, the temperature, and its preselected functions). Ultimately, the computer itself is a nothing more than a state machine controlled by a program and its data.

inputs—if you burn yourself getting something out of the oven, you approach the oven with more care next time.

We begin our discussion of sequential circuits with the *bistable* or *flip-flop*. A bistable is so called because its output can remain in one of two stable states indefinitely, even if the input changes. For a particular input, the bistable's output may be high or low, the actual value depending on the

previous inputs. Such a circuit remembers what has happened to it in the past and is therefore a form of memory element. A more detailed discussion of memory elements is given in Chapter 8. A bistable is the smallest possible memory cell and stores only a single bit of information. The term *flip-flop,* which is synonymous with bistable, gives the impression of the circuit going *flip* into one state and then *flop* into its complement. Bistables were constructed from electromagnetic relays that really did make a flip-flop sound as they jumped from one state into another.

The term *latch* is also used to describe certain types of flip-flop. A latch is a flip-flop that is unclocked (i.e. its operation isn't synchronized with a timing signal called a *clock*). The RS flip-flop that we describe first can also be called a latch.

Sequential systems can be divided into two classes: *synchronous* and *asynchronous*. Synchronous systems use a master clock to update the state of all flip-flops periodically. The speed of a synchronous system is determined by its slowest device and all signals must have settled to steady-state values by the time the system is clocked. In an asynchronous system a change in an input signal triggers a change in another circuit and this change ripples through the system (an asynchronous system is rather like a line of closely spaced dominoes on edge—when one falls it knocks its neighbor over and so on). Reliable asynchronous systems are harder to design than synchronous systems, although they are faster and consume less power. We will return to some of these topics later.

We can approach flip-flops in two ways. One is to demonstrate what they do by defining their characteristics as an abstract model and then show how they are constructed. That is, we say this is a flip-flop and this is how it behaves—now let's see what it can do. The other way of approaching flip-flops is to demonstrate how they can be implemented with just two gates and then show how their special properties are put to work. We intend to follow the latter path. Some readers may prefer to skip ahead to the summary of flip-flops at the end of this section and then return when they have a global picture of the flip-flop.

3.1 The RS flip-flop

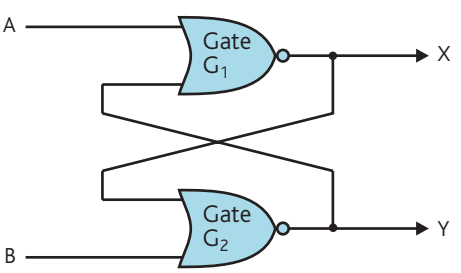
We begin our discussion of the flip-flop with the simplest member of the family, the RS flip-flop. Consider the circuit of Fig. 3.2. What differentiates this circuit from the combinational circuits of Chapter 2 is that the gates are *cross-coupled* and the output of a gate is fed back to its input. Although Fig. 3.2 uses no more than two two-input NOR gates, its operation is not immediately apparent.

The circuit has two inputs, A and B, and two outputs, X and Y. A truth table for the NOR gate is provided alongside Fig. 3.2 for reference. From the Boolean equations governing the NOR gates we can readily write down expressions for outputs X and Y in terms of inputs A and B.

1. $X = \overline{A + Y}$
2. $Y = \overline{B + X}$

If we substitute the value for Y from equation (2) in equation (1), we get

3. $X = \overline{A + \overline{B + X}}$
   $= \overline{A} \cdot \overline{\overline{B + X}}$ By de Morgan's theorem
   $= \overline{A} \cdot (B + X)$ Two negations cancel
   $= \overline{A} \cdot B + \overline{A} \cdot X$ Expand the expression



A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Figure 3.2 Two cross-coupled NOR gates.

Because Boolean algebra doesn't define the operations of division or subtraction we can't simplify this equation any further and are left with an expression in which the *output* is a function of the *output*; that is, the value of X depends on X. Equation (3) is correct but its meaning isn't obvious. We have to look for another way of analyzing the behavior of cross-coupled gates. Perhaps a better approach to understanding this circuit is to assume a value for output X and for the inputs A and B and then see where it leads us.

3.1.1 Analyzing a sequential circuit by assuming initial conditions

Figure 3.3(a) shows the cross-coupled NOR gate circuit with the initial condition X = 1 and A = B = 0 and Fig. 3.3(b) shows the same circuit redrawn to emphasize the way in which data flows between the gates.

Because the inputs to gate G_2 are X = 1, B = 0, its output, $Y = \overline{X + B}$, must be 0. The inputs to gate G_1 are Y = 0 and A = 0, so that its output, X, is $\overline{Y + A}$, which is 1. Note that this situation is *self-consistent*. The output of gate G_1 is X = 1, which is fed back to the input of gate G_1 to keep X in a logical 1 state. That is, the output actually maintains itself. It should now be a little clearer why equation (3) has X on *both* sides (i.e. $X = \overline{A} \cdot B + \overline{A} \cdot X$).

Had we assumed the initial state of X to be 0 and inputs A = B = 0, we could have proceeded as follows. The inputs to G_2 are X = 0, B = 0 and therefore its output is $Y = \overline{X + B} = \overline{0 + 0} = 1$. The inputs to G_1 are Y = 1 and A = 0, and its output is $X = \overline{Y + A} = \overline{1 + 0} = 0$. Once more we can see that the circuit is self-consistent. The output can remain indefinitely in either a 0 or a 1 state for the inputs A = B = 0.

The next step in the analysis of the circuit's behavior is to consider what happens if we change inputs A or B. Assume that the X output is initially in a logical 1 state. If input B to gate G_2 goes high while input A remains low, the output of gate G_2 (i.e. Y) is unaffected, because the output of a NOR gate is low if either of its inputs are high. As X is already high, the state of B has no effect on the state of Y.

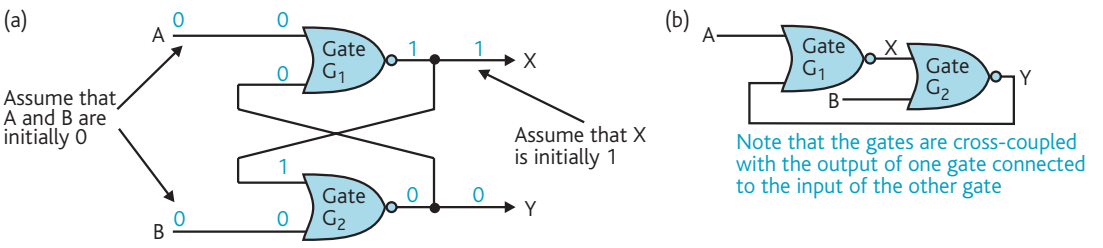
If now input A goes high while B remains low, the output, X, of gate G_1 must fall to a logical 0 state. The inputs to gate G_2 are now both in logical 0 states and its output Y rises to a logical 1. However, because Y is fed back to the input of gate G_1, the output X is *maintained* at a logical 0 even if A returns to a 0 state.

The effect of setting A to a 1 causes output X to flip over from a 1 to a 0 and to *remain* in that state when A returns to a 0. We call an RS flip-flop a *latch* because of its ability to capture a signal. Table 3.1 provides a truth table for the circuit of Fig. 3.2. Two tables are presented—one appropriate to the circuit we have described and one with its inputs and outputs relabeled.

Table 3.1(a) corresponds exactly to the two-NOR gate circuit of Fig. 3.2 and Table 3.1(b) to the idealized form of this circuit that's called an RS *flip-flop*. There are two differences between Tables 3.1(a) and 3.1(b). Table 3.1(b) uses the conventional labeling of an RS flip-flop with inputs R and S and an output Q. The other difference is in the entry for the case in which A = B = 1 and R = S = 1. The effect of these differences will be dealt with later.

We've already stated that Fig. 3.2 defines its output in terms of itself (i.e. $X = \overline{A} \cdot B + \overline{A} \cdot X$). The truth table gets round this problem by creating a new variable, X^+ (or Q^+), where X^+ is the *new* output generated by the *old* output X and the current inputs A and B. We can write $X^+ = \overline{A} \cdot B + \overline{A} \cdot X$. The input and output columns of the truth table are now not only separated in space (e.g. input on the left and output on the right) but also in *time*. The *current* output X is combined with inputs A and B to generate a *new* output X^+. The value of X that produced X^+ no longer exists and belongs only to the past.

Labels R and S in the Table 3.1(b) correspond to *reset* and *set*, respectively. The word *reset* means *make 0* (*clear* has the same meaning) and *set* means *make 1*. The output of all flip-flops is called Q by a historical convention. Examining the truth table reveals that whenever R = 1, the output Q is reset to 0. Similarly, when S = 1 the output is set to 1.



Analyzing the circuit by assuming initial conditions.

An alternative view of the circuit.

Figure 3.3 Analyzing the behavior of cross-coupled NOR gates.

(a) Truth table for Fig. 3.2.

Inputs			Output
A	B	X	X^+
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0
		↑	↑
		Old X	New X

(b) Truth table for relabeled Fig. 3.2.

Inputs			Output	
R	S	Q	Q^+	
0	0	0	0	No change
0	0	1	1	No change
0	1	0	1	Set
0	1	1	1	Set
1	0	0	0	Clear
1	0	1	0	Clear
1	1	0	?	Undefined
1	1	1	?	undefined
		↑	↑	
		Old Q	New Q	

The truth table is interpreted as follows. The output of the circuit is currently X (or Q) and the new inputs to be applied to the input terminals are A, B (or R, S). When these new inputs are applied to the circuit, its output is given by X^+ (or Q^+). For example, if the current output X is 1 and the new values of A and B are A = 1, B = 0, then the new output, X^+, will be 0. This value of X^+ then becomes the next value of X when new inputs A and B are applied to the circuit.

Table 3.1 Truth table for the circuit in Fig. 3.2.

When R and S are both 0, the output does not change; that is, $Q^+ = Q$.

If both R and S are simultaneously 1, the output is conceptually undefined (hence the question marks in Table 3.1(b), because the output can't be set and reset at the same time. In the case of the RS flip-flop implemented by two NOR gates, the output X does, in fact, go low when A = B = 1. In practice, the user of an RS flip-flop should avoid the condition R = S = 1.

The two-NOR gate flip-flop of Fig. 3.2 has two outputs X and Y. An examination of the circuit for all inputs except A = B = 1 reveals that X and Y are *complements*. Because of the symmetric nature of flip-flops, almost all flip-flops have two outputs, Q and its complement $\overline{Q}$. The complement of Q may not always be available to the user of the flip-flop because many commercial devices leave $\overline{Q}$ buried on the chip and not brought out to a pin. Figure 3.4 gives the circuit representation of an RS flip-flop.

We can draw the truth table of the RS or any other flip-flop in two ways. Up to now we've presented truth tables with two output lines for each possible input, one line for Q = 0 and one for Q = 1. An alternative approach is to employ the algebraic value of Q and is illustrated by Table 3.2.

When R = S = 0 the new output Q^+ is simply the old output Q. In other words, the output doesn't change state and remains in its previous state as long as R and S are both 0. The inputs R = S = 1 result in the output $Q^+ = X$. The symbol X is used in truth tables to indicate an *indeterminate* or *undefined* condition. In Chapter 2 we used the same symbol

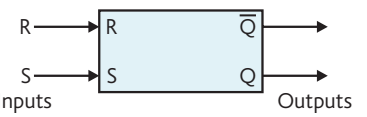


Figure 3.4 Circuit representation of the RS flip-flop as a black box.

Inputs		Output	Description
R	S	Q^+	
0	0	Q	No change
0	1	1	Set output to 1
1	0	0	Reset output to 0
1	1	X	Forbidden

Table 3.2 An alternative truth table for the RS flip-flop.

to indicate a *don't care* condition. An indeterminate condition is one whose outcome can't be calculated, whereas a don't care condition is one whose outcome does not matter to the designer.

3.1.2 Characteristic equation of an RS flip-flop

We have already demonstrated that you can derive an equation for a flip-flop by analyzing its circuit. Such an equation is called the flip-flop's characteristic equation. Instead of using an actual circuit, we can derive a characteristic equation from

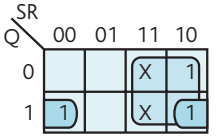


Figure 3.5 Karnaugh map for the characteristic equation of an RS flip-flop.

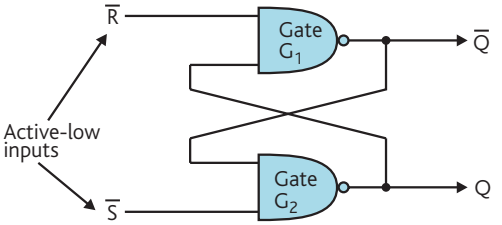


Figure 3.6 RS flip-flop constructed from two cross-coupled NAND gates.

the flip-flop's truth table. Figure 3.5 plots Table 3.1(b) on a Karnaugh map. We have indicated the condition R = S = 1 by X because it is a forbidden condition. From this truth table we can write $Q^+ = S + Q \cdot \overline{R}$.

Note that this equation is slightly different from the one we derived earlier because it treats R = S = 1 as a don't care condition.

3.1.3 Building an RS flip-flop from NAND gates

An RS flip-flop can be constructed from two cross-coupled NAND gates just as easily as from two NOR gates. Figure 3.6 illustrates a two-NAND gate flip-flop whose truth table is given in Table 3.3.

The only significant difference between the NOR gate flip-flop of Fig. 3.2 and the NAND gate flip-flop of Fig. 3.6 is that the inputs to the NAND gate flip-flop are active-low. If we were to place inverters at the R and S inputs to the NAND gate flip-flop, it would then be logically equivalent to the NOR gate flip-flop of Fig. 3.2.

The *no change* input to the NAND gate flip-flop is R, S = 1, 1; the output is cleared by forcing R = 0 and set by forcing S = 0; the forbidden input state is R, S = 0, 0. Suppose that we did set the inputs of a NAND gate RS flip-flop to 0, 0 and then released the inputs to 1,1 to enter the no change state. What would happen? The answer is that we can't predict the final outcome. Initially, when both inputs are 0s, *both* outputs of the RS flip-flop must be 1s (because the output of a NAND gate is a 1 if either of its inputs are a 0). The real problem arises when the inputs change state from 0, 0 to 1, 1. Due to tiny imperfections, either one or the other input would go high before its neighbor and cause the flip-flop to be set or reset.

Inputs		Output	Comment
R	S	Q^+	
0	0	X	Forbidden
0	1	1	Reset output to 0
1	0	0	Set output to 1
1	1	Q	No change

Table 3.3 Truth table for an RS flip-flop constructed from NAND gates.

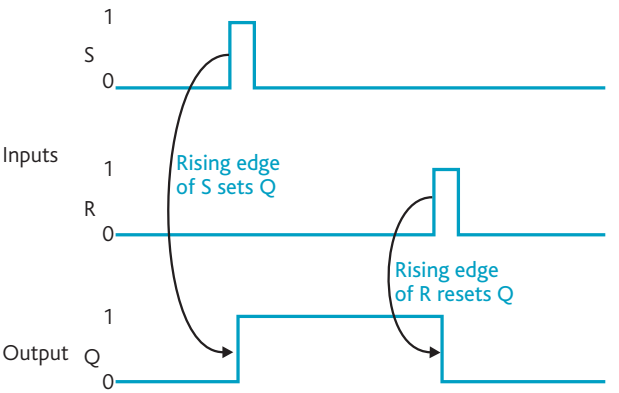


Figure 3.7 Timing diagram of the effect of pulses on an RS flip-flop's inputs.

Real applications of RS flip-flops may employ either two NAND or two NOR gates depending only on which gates provide the simpler solution. In practice, the majority of RS flip-flops are often constructed from NAND gates because most circuits use active-low signals. We began our discussion of RS flip-flops with the NOR gate circuit (unlike other texts that introduce first the more common NAND gate flip-flop) because we have discovered that many students find it hard to come to terms with negative logic (i.e. logic in which the low state is the active state).

3.1.4 Applications of the RS flip-flop

An important application of RS flip-flops is in the recording of short-lived events. If the Q output of a flip-flop is in a zero state, a logical 1 pulse at its S input (assuming the R input is 0) will cause Q to be set to a 1, and to remain at a 1, until the R input resets Q. The effect of a pulse at the S input followed by a pulse at the R input of an RS flip-flop is illustrated in Fig. 3.7.

Consider the following application of RS flip-flops to an indicator circuit. If an aircraft is flown outside its performance envelope no immediate damage may be apparent, but its structure might be permanently weakened. To keep things

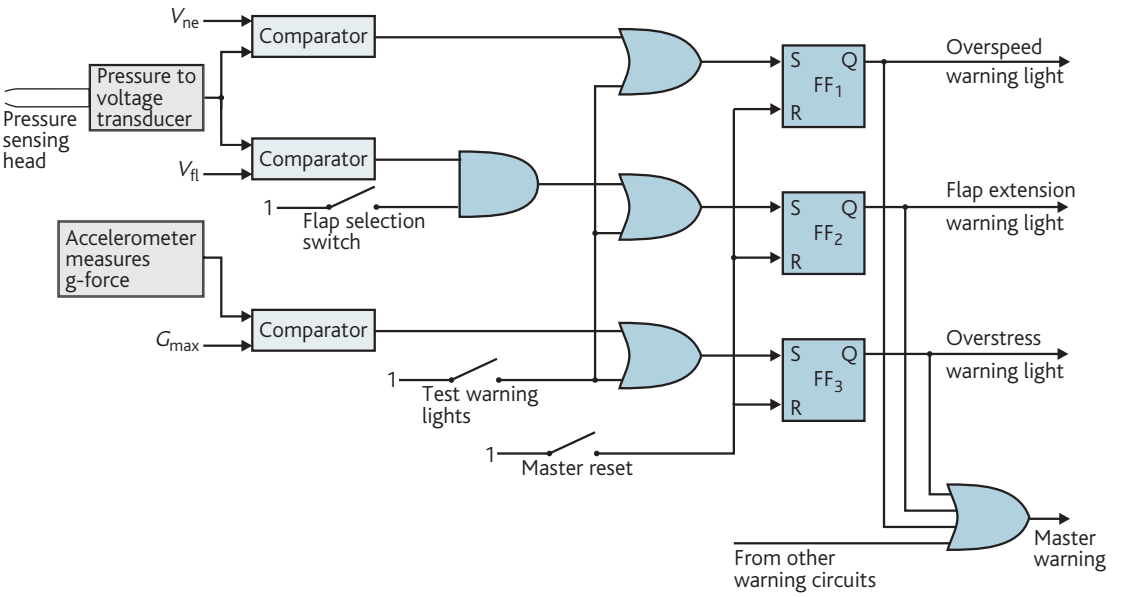


Figure 3.8 Application of RS flip-flops in a warning system.

simple, we will consider three possible events that are considered harmful and might endanger the aircraft.

1. Exceeding the maximum permissible speed V_{ne}.
2. Extending the flaps above the flap-limiting speed V_{fl}. That is, the flaps must not be lowered if the aircraft is going faster than V_{fl}.
3. Exceeding the maximum acceleration (*g*-force) G_{max}.

If any of the above parameters are exceeded (even for only an instant), a lasting record of the event must be made.

Figure 3.8 shows the arrangement of warning lights used to indicate that one of these conditions has been violated. Transducers that convert acceleration or velocity into a voltage measure the acceleration and speed of the aircraft. The voltages from the transducers are compared with the three threshold values (V_{ne}, V_{fl}, G_{max}) in comparators, whose outputs are true if the threshold is exceeded, otherwise false. In order to detect the extension of flaps above the flap-limiting speed, the output of the comparator is ANDed with a signal from the flap actuator circuit that is true when the flaps are down.

The three signals from the comparators are fed, via OR gates, to the S inputs of three RS flip-flops. Initially, on switching on the system, the flip-flops are automatically reset by applying a logical 1 pulse to all R inputs simultaneously. If at any time one of the S inputs becomes true, the output of that flip-flop is set to a logical 1 and triggers an alarm. All outputs are ORed together to illuminate a master warning light. A master alarm signal makes it unnecessary for the pilot to have to scan all the warning lights periodically. An additional feature of the circuit is a test facility. When the warning test button is pushed, all warning lights should be illuminated and remain so until the reset button is pressed. A test facility verifies the correct operation of the flip-flops and the warning lights.

A pulse-train generator

Figure 3.9 gives the circuit of a pulse-train generator that generates a sequence of *N* pulses each time it is triggered by a positive transition at its START input. The value of *N* is user supplied and is fed to the circuit by three switches to select the values of C_c, C_b, C_a. This circuit uses the counter that we will meet later in this chapter.

The key to this circuit is the RS flip-flop, G_6, used to start and stop the pulse generator. Assume that initially the R and S inputs to the flip-flop are R = 0 and S = 0 and that its output Q is a logical 0. Because one of the inputs to AND gate G_1 is low, the pulse train output is also low.

When a logical 1 pulse is applied to the flip-flop's START input, its Q output rises to a logical 1 and enables AND gate G_1. A train of clock pulses at the second input of G_1 now appears at the output of the AND gate. This gated pulse train is applied to the input of a counter (to be described later), which counts pulses and generates a three-bit output on Q_a, Q_b, Q_c, corresponding to the number of pulses counted in the range 0 to 7. The outputs of the counter are fed to an equality detector composed of three EOR gates, G_2 to G_4, plus NOR gate G_5. A second input to the equality detector is the user-supplied count value C_a, C_b, C_c. The outputs of the EOR gates are combined in NOR gate G_5 (notice that it's drawn in negative logic form to emphasize that the output is 1 if all its inputs are 0).

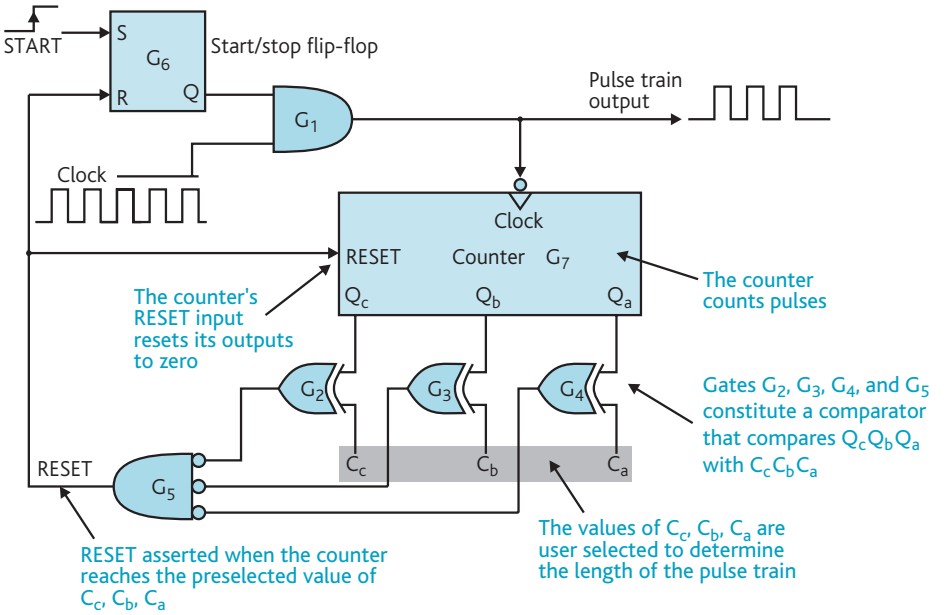


Figure 3.9 Pulse train generator.

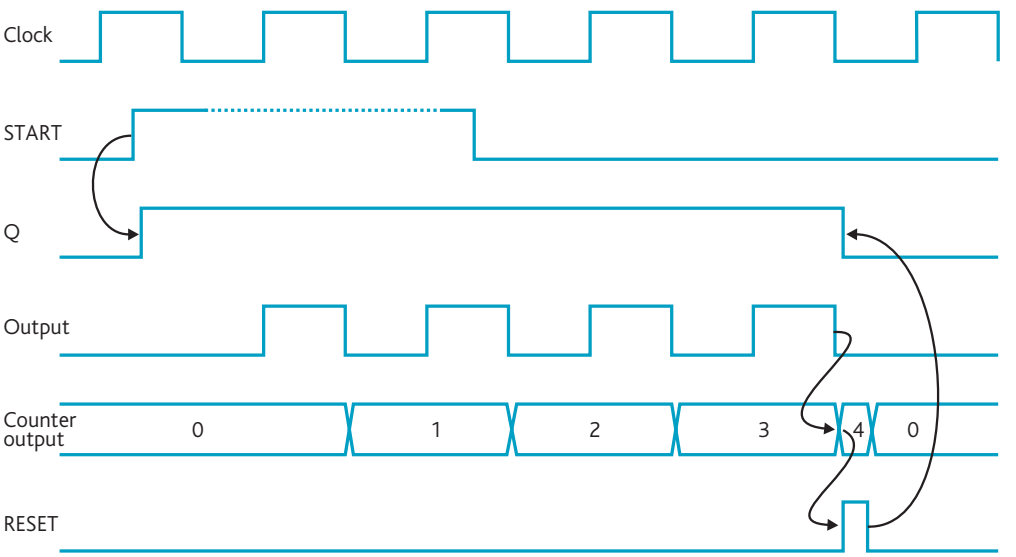


Figure 3.10 Timing diagram of pulse train generator.

Figure 3.10 gives a timing diagram for the pulse generator. Initially the counter is held in a reset state ($Q_a = Q_b = Q_c = 0$). When the counter is clocked, its output is incremented by 1 on the falling edge of each clock pulse. The counter counts upward from 0 and the equality detector compares the current count on Q_a, Q_b, Q_c output with the user-supplied inputs C_a, C_b, C_c. When the output of the counter is equal to the user-supplied input, the output of gate G_5 goes high and resets both the counter and the RS flip-flop. Resetting the counter forces the counter output to 0. Resetting the RS flip-flop disables AND gate G_1 and no further clock pulses appear at the output of G_1. In this application of the RS flip-flop, its S input is triggered to start an action and its R input is triggered to terminate the action.

3.1.5 The clocked RS flip-flop

The RS flip-flop of Fig. 3.2 responds to signals applied to its inputs according to its truth table. There are situations when

we want the RS flip-flop to ignore its inputs until a particular time. The circuit of Fig. 3.11 demonstrates how this is accomplished by turning the RS flip-flop into a clocked RS flip-flop.

A normal, unmodified, RS flip-flop lies in the inner box in Fig. 3.11. Its inputs, R′ and S′, are derived from the external inputs R and S by ANDing them with a clock input C—some texts call these two AND gates '*steering gates*'. As long as C = 0, the inputs to the RS flip-flop, R′ and S′, are forced to remain at 0, no matter what is happening to the external R and S inputs. The output of the RS flip-flop remains constant as long as these R′ and S′ inputs are both 0.

Whenever C = 1, the external R and S inputs to the circuit are transferred to the flip-flop so that R′ = R and S = S, and the flip-flop responds accordingly. The clock input may be thought of as an *inhibitor*, restraining the flip-flop from acting until the right time. Figure 3.12 demonstrates how we can build a clocked RS flip-flop from NAND gates. Clocked flip-flops are dealt with in more detail later in this chapter.

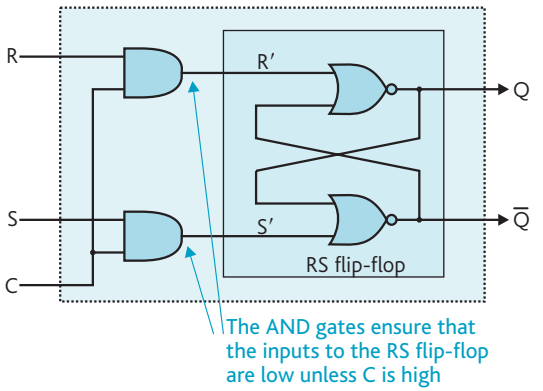


Figure 3.11 The clocked RS flip-flop.

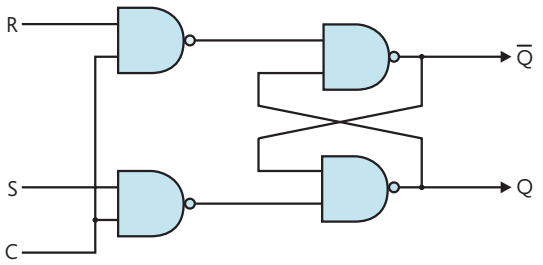


Figure 3.12 Building a clocked RS flip-flop with NAND gates.

3.2 The D flip-flop

Like the RS flip-flop, the D flip-flop has two inputs, one called D and the other C. The D input is referred to as the *data* input and C as the clock input. The D flip-flop is, by its nature, a clocked flip-flop and we will call the act of pulsing the C input high and then low clocking the D flip-flop.

When a D flip-flop is clocked, the value at its D input is transferred to its Q output and the output remains constant until the next time it is clocked. The D flip-flop is a staticizer because it records the state of the D input and holds it constant until it's clocked. Others call it a delay element because, if the D input changes state at time *T* but the flip-flop is clocked *t* seconds later, the output Q doesn't change state until *t* seconds after the input. I think of the D flip-flop as a *census taker* because it takes a census of the input and remembers it until the next census is taken. The truth table for a D flip-flop is given in Table 3.4.

The circuit of a D flip-flop is provided in Fig. 3.13 and consists of an RS flip-flop plus a few gates. The two AND gates turn the RS flip-flop into a clocked RS flip-flop. As long as the C input to the AND gates is low, the R and S inputs are clamped at 0 and Q cannot change.

Full form						Algebraic form		
Inputs			Output Q^+			Inputs		Output Q^+
C	D	Q				C	D	
0	0	0	0	$Q^+ \leftarrow Q$	No change	0	0	Q
0	0	1	1	$Q^+ \leftarrow Q$	No change	0	1	Q
0	1	0	0	$Q^+ \leftarrow Q$	No change	1	0	0
0	1	1	1	$Q^+ \leftarrow Q$	No change	1	1	1
1	0	0	0	$Q^+ \leftarrow D$				
1	0	1	0	$Q^+ \leftarrow D$				
1	1	0	1	$Q^+ \leftarrow D$				
1	1	1	1	$Q^+ \leftarrow D$				

Table 3.4 Truth table for a D flip-flop.

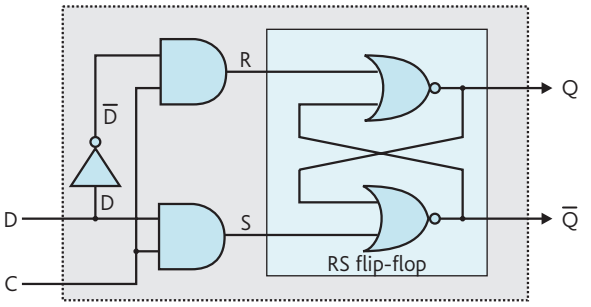


Figure 3.13 Circuit of a D flip-flop.

When C goes high, the S input is connected to D and the R input to $\overline{D}$. Consequently, (R, S) must either be (0, 1) if D = 1, or (1, 0) if D = 0. Therefore, D = 1 sets the RS flip-flop, and D = 0 clears it.

3.2.1 Practical sequential logic elements

Just as semiconductor manufacturers have provided combinational logic elements in single packages, they have done the same with sequential logic elements. Indeed, there are more special-purpose sequential logic elements than combinational logic elements. Practical flip-flops are more complex than those presented hitherto in this chapter. Real circuits have to cater for real-world problems. We have already said that the output of a flip-flop is a function of its current inputs and its previous output. What happens when a flip-flop is first switched on? The answer is quite simple. The Q output takes on a random state, assuming no input is being applied that will force Q into a 0 or 1 state.

Random states may be fine at the gaming tables in Las Vegas; they're less helpful when the control systems of a nuclear reactor are first energized. Many flip-flops are provided with special control inputs that are used to place them in a known state. Figure 3.14 illustrates the 74LS74, a dual positive-edge triggered D flip-flop that has two active-low control inputs called *preset* and *clear* (abbreviated $\overline{PRE}$ and $\overline{CLR}$). In normal operation both $\overline{PRE}$ and $\overline{CLR}$ remain in logical 1 states. If $\overline{PRE}$ = 0 the Q output is set to a logical 1 and if $\overline{CLR}$ = 0 the Q output is cleared to a logical 0. As in the case of the RS flip-flop, the condition $\overline{PRE} = \overline{CLR} = 0$ should not be allowed to occur.

These preset and clear inputs are *unconditional* in the sense that they override all activity at the other inputs of this flip-flop. For example, asserting $\overline{PRE}$ sets Q to 1 irrespective of the state of the flip-flop's C and D inputs. When a digital system is made up from many flip-flops that must be set or cleared at the application of power, their $\overline{PRE}$ or $\overline{CLR}$ lines are connected to a common $\overline{RESET}$ line and this line is momentarily asserted active-low by a single pulse shortly after the power is switched on.

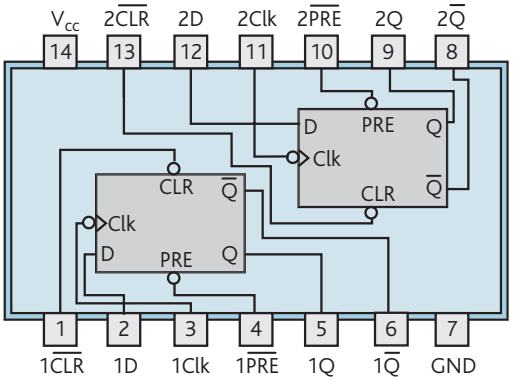


Figure 3.14 The 74LS74 D flip-flop.

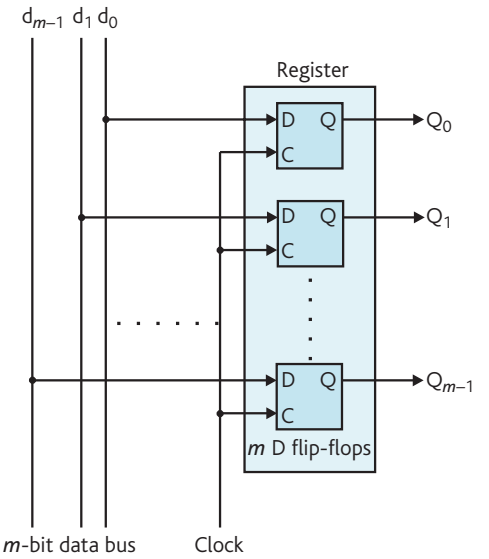


Figure 3.15 Using D flip-flops to create a register.

3.2.2 Using D flip-flops to create a register

Later we shall discover that a computer is composed of little more than combinational logic elements, buses, and groups of flip-flops called registers that transmit data to and receive data from buses. A typical example of the application of D flip-flops is provided by Fig. 3.15 in which an *m*-bit wide data bus transfers data from one part of a digital system to another. Data on the bus is constantly changing as different devices use it to transmit their data from one register to another.

The D inputs of a group of *m* D flip-flops are connected to the *m* lines of the bus. The clock inputs of all flip-flops are

connected together, allowing them to be clocked simultaneously. As long as C = 0, the flip-flops ignore data on the bus and their Q outputs remain unchanged. Suppose some device wishes to transfer its data to the flip-flops. It first puts its data on the bus and then the flip-flops are clocked, latching the data into them. When the clock has returned to zero, the data remains frozen in the flip-flops.

3.2.3 Using Digital Works to create a register

We are now going to use Digital Works to create a simple bused system using D flip-flops. Although Digital Works implements both RS and D flip-flops, we'll construct a D flip-flop from basic gates. Figure 3.16 shows a single 1-bit cell in a register (we can construct an *m*-bit register by using *m* identical elements in parallel).

If you examine Fig. 3.16 you will find that the flip-flop is more complex than the simple D flip-flop of Fig. 3.13. We have added a tri-state gate to the Q output to allow the flip-flop to drive a bus or to be disconnected from the bus. Furthermore, we've added an input multiplexer to allow the D input to be connected to one of two sources A and B. The inputs and output of Fig. 3.16 are

- A input
- B input
- A/B select input
- Clock input
- Enable output
- Q output.

In Fig. 3.17 we've used Digital Work's macro facility to convert the circuit in Fig. 3.16 into a black box macro that can be used as a circuit element to build more complex systems.

Figure 3.18 provides a test bed for three of the register slices. We have provided one bit of three registers and three buses (input bus A, input bus B, and output bus C). Each register slice is connected to all three buses. We've added input devices to all the control inputs to enable us to experiment with this circuit.

The system in Fig. 3.18 can't do a lot, but what it can do is very important. Because we've added input devices to buses A and B, we can force our own data on bus A and B. We can select whether each register slice gets its input from bus A or bus B by setting the value of the *Input select* input to 1 (bus A) or 0 (bus B). Data is clocked into any of the register slices by clocking it (i.e. setting its clock input to 1 to capture the data and then setting the clock input to 0 to latch and retain the data). Finally, data from any of the three register slices can be put on bus C by asserting the appropriate output.

This circuit is important because it forms the heart of a computer. All we need to create an ALU (arithmetic and logic unit) are the circuits that take data from bus C, process it, and copy the result to the A or B bus.

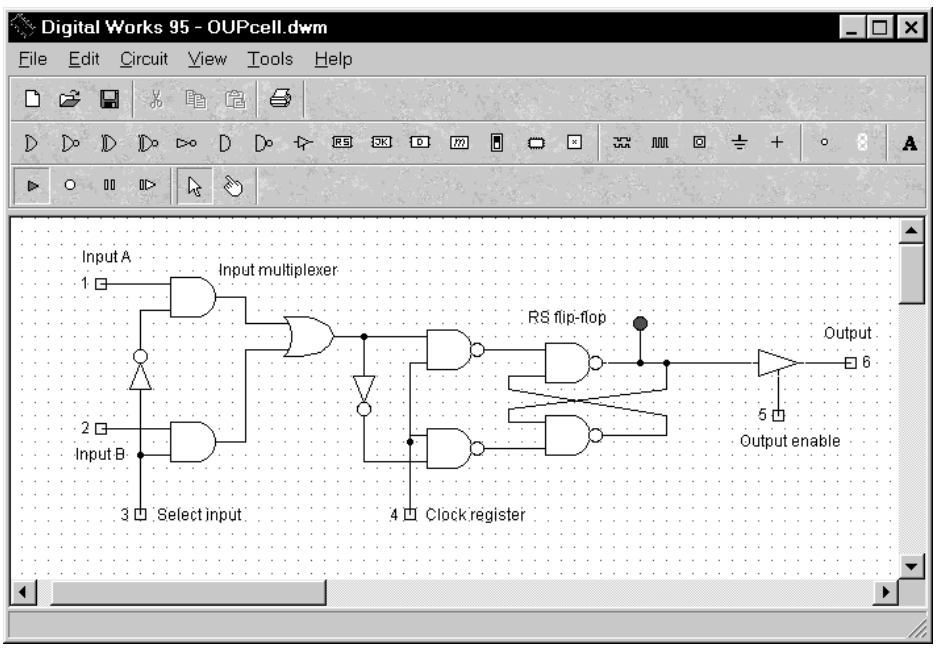


Figure 3.16 Using D flip-flops to create one cell of a register.

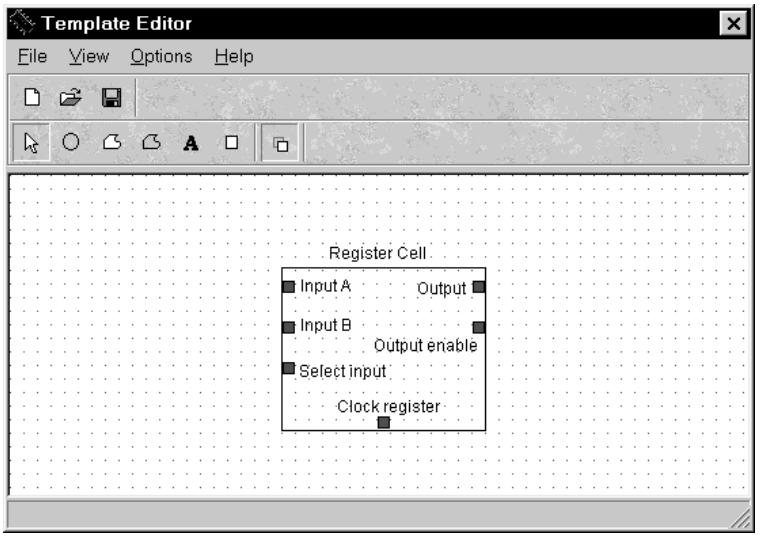


Figure 3.17 Converting the circuit of Fig. 3.16 into a macro (i.e. black box representation).

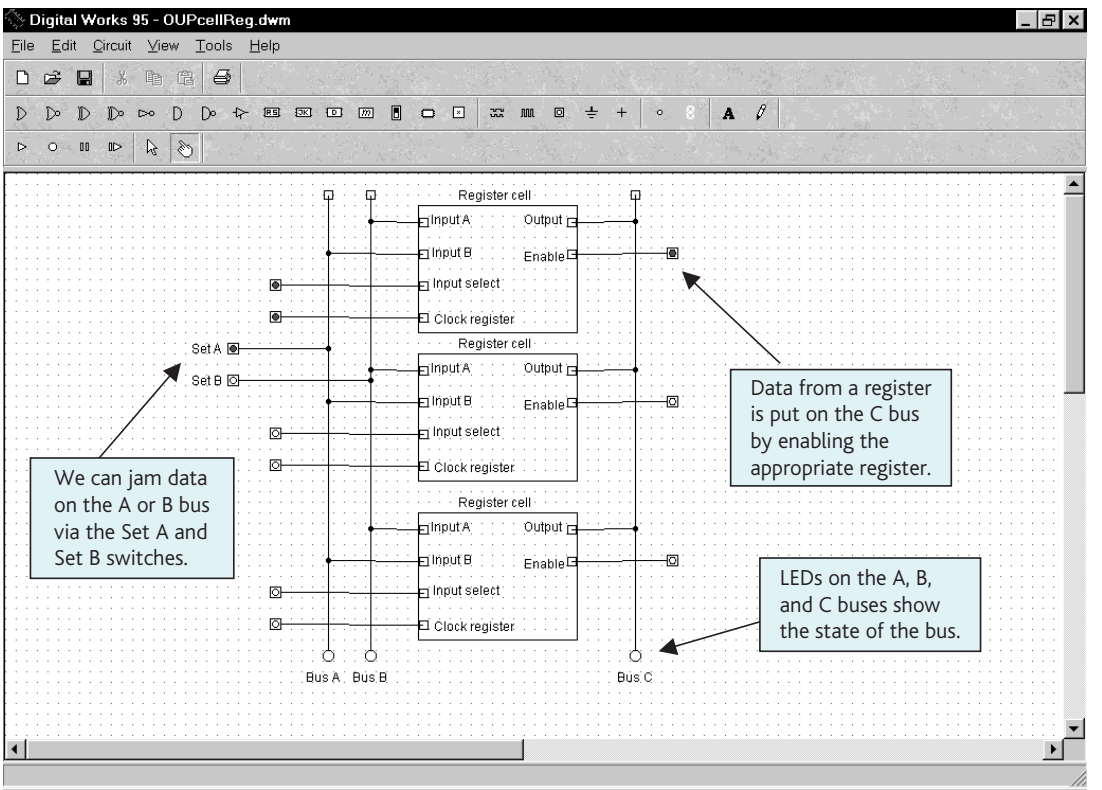


Figure 3.18 Using D flip-flops to create a register in Digital Works.

3.2.4 A typical register chip

You can obtain a single package containing the flip-flops that implement a register. Figure 3.19 illustrates the 74LS373, an octal register composed of D flip-flops that is available in a 20-pin package with eight inputs, eight outputs, two power supply pins, and two control inputs. The clock input, G, is a level-sensitive clock, which, when high, causes the value at D_i to be transferred to Q_i. All eight clock inputs are connected together internally so that the G input clocks each flip-flop simultaneously.

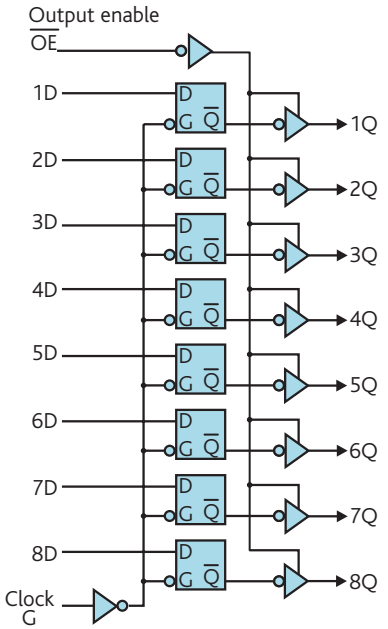


Figure 3.19 The 74LS373 octal register.

The 74LS373's other control input is active-low $\overline{OE}$ (output enable), which controls the output of all flip-flops. When $\overline{OE} = 0$, the flip-flop behaves exactly as we would expect. When $\overline{OE} = 1$, the eight Q outputs are internally disconnected from the output pins of the device; that is, the 74LS373 has tri-state outputs and $\overline{OE}$ is used to turn off the chip's output circuits when it is not driving a bus.

Figure 3.20 demonstrates the 74LS373 octal register in a digital system where four registers are connected to a common data bus. Each register is arranged with *both* its outputs and its inputs connected to the same bus allowing it to transmit data onto the bus or to receive data from it.

Each register has tri-state outputs controlled by an output enable pin. When $\overline{OE}$ is asserted low, the corresponding register drives the bus. Registers are clocked by an active-high clock input labeled G.

IC5a is a 2-line to 4-line decoder; that is, a demultiplexer of the type we described in Chapter 2. When this device is enabled, the 2-bit binary *source code* at the input of IC5a causes one of its output lines, $\overline{Y}_0$ to $\overline{Y}_3$, to go low. These outputs are connected to the respective $\overline{OE}$ inputs of the four registers. Each of the four possible source codes enables one of the registers; for example, if the source code at the input to IC5a is 01, the output of register 1 is enabled and the contents of register 1 are placed on the bus. The outputs of all other registers remain internally disconnected from the bus.

The 74LS139 contains two complete 2-line to 4-line decoders in a single 16-pin package. The second half of this package acts as a destination decoder. Each of the four outputs from IC5b is connected to one of the clock inputs, G, of the four registers. Because the clock inputs are active-high and the outputs of the decoder are active-low, it's necessary to invert these outputs. Four inverters, IC6, perform this function. When IC5b is enabled, one of its outputs is asserted and the corresponding register clocked. Clocking a register latches data from the data bus.

Suppose the contents of register 1 are to be copied into register 3. The source code at IC5a is set to 01 and the destination code at IC5b is set to 11. This puts the data from register 1 on the bus and latches the data into register 3. We can easily relate the example of Fig. 3.20 to the digital computer. One of the most fundamental operations in computing is the assignment that can be represented in a high-level language as `B = A` and in a low-level language as `MOVE A, B`. The action `MOVE A, B` (i.e. transfer the contents of A to B) is implemented by specifying A as the source and B as the destination. Note that throughout this text we put the destination of a data transfer in bold font to stress the direction of data transfer.

3.3 Clocked flip-flops

Before we introduce the JK flip-flop we look more closely at the idea of *clocking* sequential circuits. Clocked circuits allow logic elements to respond to their inputs only when the inputs are valid. Some writers use the term *trigger* rather than clock, because *triggering a flip-flop* gives the impression of causing an event to take place at a discrete instant. We begin by examining the effect of delays on the passage of signals through systems.

Figure 3.21 demonstrates the effect of circuit delays on a system with two inputs, A and B, that are acted upon by *processes* A, B, and C to produce an output. The nature of the processes is not important because we're interested only in the way in which they delay signals passing through them. Imagine that at time $t = 0$, the inputs to processes A and B become valid (i.e. these are the correct inputs to be operated on by the processes). Assume that process A in Fig. 3.21 introduces a two-unit delay, process B a one-unit delay, and process C a two-unit delay.

Although the output from process B becomes valid at $t = 1$, it's not until $t = 2$ that the output of process A has become valid. The outputs of processes A and B are fed to process C, which has a two-unit delay. Clearly, the desired output from C due to inputs A and B is not valid until at least *four* time units after $t = 0$. The output from process C changes at least once before it settles down to its final value (Why? Because of the different delays through processes A and B). This poses a problem. How does an observer at the output of process C know when to act upon the data from C?

What we need is some means of capturing data only when we know that it's valid—see Fig. 3.22. If a D flip-flop is placed

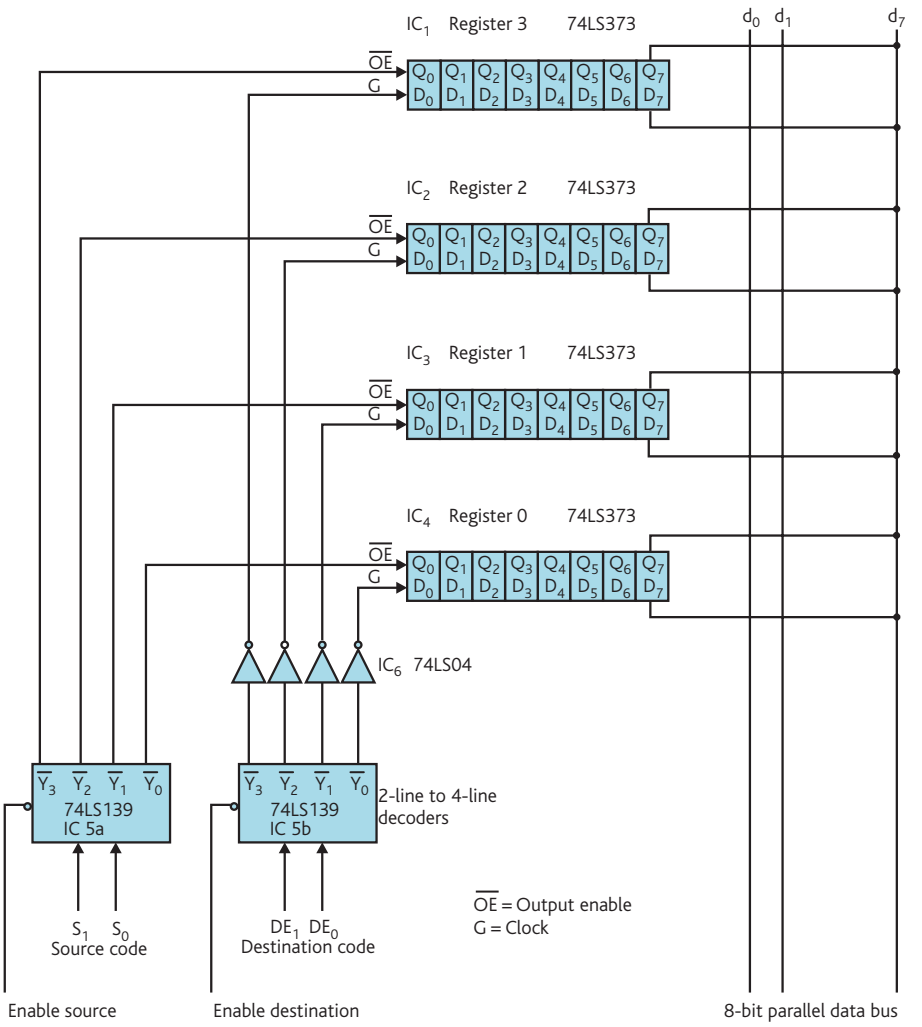


Figure 3.20 Using the 74LS373 octal register in a bused system.

at the output of process C and is clocked four units of time after $t = 0$, the desired data will be latched into the flip-flop and held constant until the next clock pulse. Clocked systems hold digital information constant in flip-flops while the information is operated on by groups of logic elements, analogous to the processes of Fig. 3.21. Between clock pulses, the outputs of the flip-flops are processed by the logic elements and the new data values are presented to the inputs of flip-flops.

After a suitable time delay (longer than the time taken for the slowest process to be completed), the flip-flops are clocked. The outputs of the processes are held constant until the next time the flip-flops are clocked. A clocked system is often called *synchronous*, as all processes are started simultaneously on each new clock pulse. An *asynchronous* system is one in which the end of one process signals (i.e. triggers) the start of the next. Obviously, an asynchronous system must be faster than the corresponding synchronous system. Asynchronous systems are more complex and difficult to design than synchronous systems and popular wisdom says that they are best avoided because they are inherently less reliable than synchronous circuits. The 1990s saw a renewed interest in asynchronous systems because of their speed and lower power consumption.

3.3.1 Pipelining

Now consider the effect of placing D flip-flops at the *outputs* of processes A, B, and C in the system of Fig. 3.23. Figure 3.23 shows the logical state at several points in a system as a function of time. The diagram is read from left to right (the direction of time flow). Signals are represented by parallel lines to demonstrate that the signal values may be 1s or 0s (we don't care). What matters is the time at which signals change. Changes are shown by the parallel lines crossing over. Lines with arrowheads are drawn between points to demonstrate cause and effect; for example, the line from *Input* A to *Output* A shows that a change in *Input* A leads to a change in *Output* A.

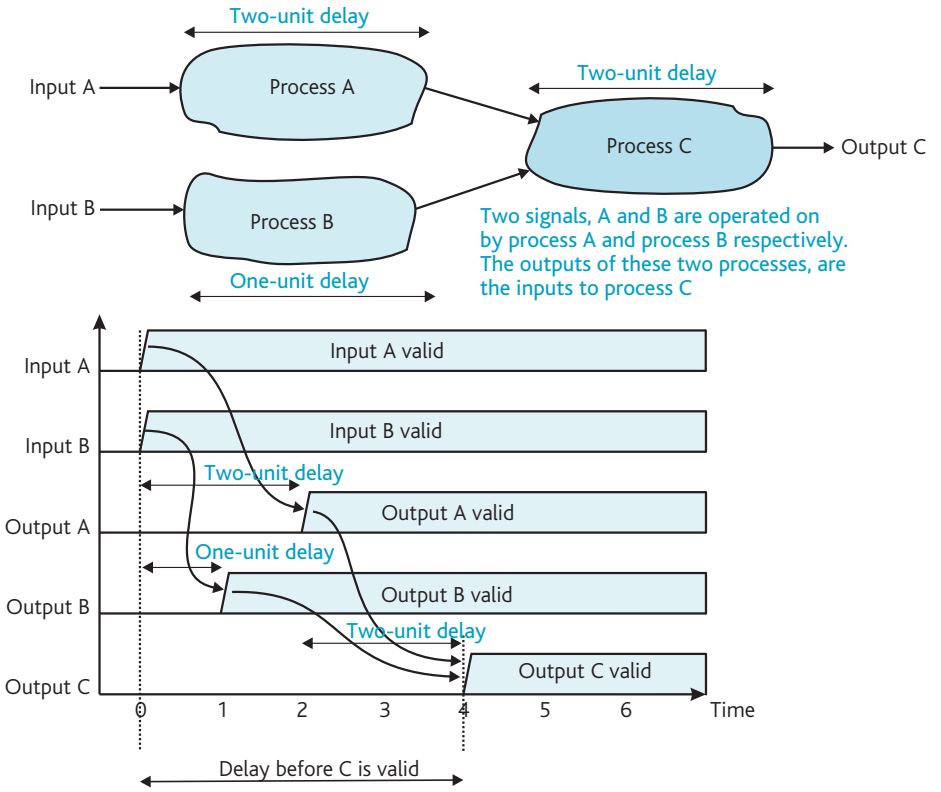


Figure 3.21 Processes and delays.

In this example we assume that each of the processes introduces a single unit of delay and the flip-flops are clocked simultaneously every unit of time. Figure 3.23 gives the timing diagram for this system. Note how a new input can be accepted every unit of time, rather than every two units of time as you might expect. The secret of our increase in throughput is called *pipelining* because we are operating on different data at different stages in the pipeline. For example, when process A and process B are operating on data *i*, process C is operating on data $i - 1$ and the latched output from process C corresponds to data $i - 2$.

When we introduce the RISC processor we will discover that pipelining is a technique used to speed up the operation of a computer by overlapping consecutive operations.

3.3.2 Ways of clocking flip-flops

A clocked flip-flop captures a digital value and holds it constant. There are, however, three ways of clocking a flip-flop.

1. Whenever the clock is asserted (i.e. a level-sensitive flip-flop).
2. Whenever the clock is changing state (i.e. an edge-sensitive flip-flop).
3. Capture data on one edge of the clock and transfer it to the output on the following edge (i.e. a master–slave flip-flop).

A *level-sensitive* clock triggers a flip-flop whenever the clock is in a particular logical state (some flip-flops are clocked by a logical 1 and some by a logical 0). The clocked RS flip-flop of Fig. 3.11 is level sensitive because the RS flip-flop responds to its R and S inputs whenever the clock input is high. A level-sensitive clock is unsuitable for certain applications. Consider the system of Fig. 3.24 in which the output of a D flip-flop is fed through a logic network and then back to the flip-flop's D input. If we call the output of the flip-flop the *current* Q, then the current Q is fed through the logic network to generate a *new* input D. When the flip-flop is clocked, the value of D is transferred to the output to generate Q^+.

If the clock is level sensitive, the new Q^+ can rush through the logic network and change D and hence the output. This chain of events continues in an oscillatory fashion with the *dog chasing its tail*. To avoid such unstable or unpredictable behavior, we need an infinitesimally short clock pulse to capture the output and hold it constant. As such a short pulse can't easily be created, the edge-sensitive clock has been introduced to solve the feedback problem. Level-sensitive clocked D flip-flops are often perfectly satisfactory in applications such as registers connected to data buses, because the duration of the clock is usually small compared to the time for which the data is valid.

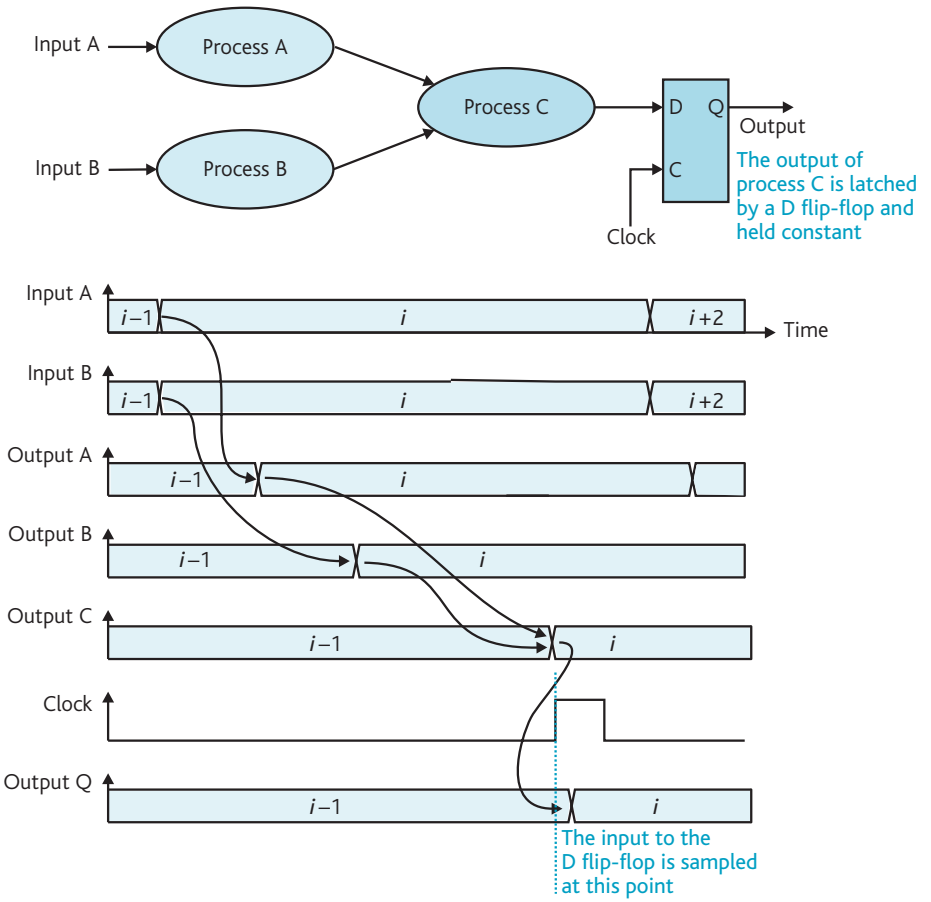


Figure 3.22 Latching the output of a system.

3.3.3 Edge-triggered flip-flops

An edge-triggered flip-flop is clocked not by the level or *state* of the clock (i.e. high or low), but by the *transition* of the clock signal from zero to one, or one to zero. The former case is called a positive or rising-edge sensitive clock and the latter is called a negative or falling-edge sensitive clock. As the rising or falling edge of a pulse may have a duration of less than 1 ns, an edge-triggered clock can be regarded as a level-sensitive clock triggered by a pulse of an infinitesimally short duration. A nanosecond (ns) is a thousand millionth (10^{-9}) of a second. The feedback problem described by Fig. 3.24 ceases to exist if you use an edge-sensitive flip-flop because there's insufficient time for the new output to race back to the input within the duration of a single rising edge.

There are circumstances when edge-triggered flip-flops are unsatisfactory because of a phenomenon called clock *skew*. If, in a digital system, several edge-triggered flip-flops are clocked by the same edge of a pulse, the exact times at which the individual flip-flops are clocked vary. Variation in the arrival time of pulses at each clock input is called *clock skew* and is caused by the different paths by which clock pulses reach each flip-flop. Electrical impulses move through circuits at somewhat less than the speed of light, which is 30 cm/ns. Unless each flip-flop is located at the same distance from the source of the clock pulse and unless any additional delays in each path due to other logic elements are identical, the clock pulse will arrive at the flip-flops at different instants. Moreover, the delay a signal experiences going through a gate changes with temperature and even the age of the gate. Suppose that the output of flip-flop A is connected to the input of flip-flop B and they are clocked together. Ideally, at the moment of clocking, the old output of A is clocked into B. If, by bad design or bad luck, flip-flop A is triggered a few nanoseconds before flip-flop B, B sees the new output from A, not the old (i.e. previous) output—it's as if A were clocked by a separate and earlier clock.

Figure 3.25 gives the circuit diagram of a positive edge-triggered D flip-flop that also has unconditional preset and clear inputs. Edge triggering is implemented by using the active transition of the clock to clock latches 1 and 2 and then feeding the output of latch 2 back to latch 1 to cut off the clock in the NAND gate. That is, once the clock has been detected, the clock input path is removed.

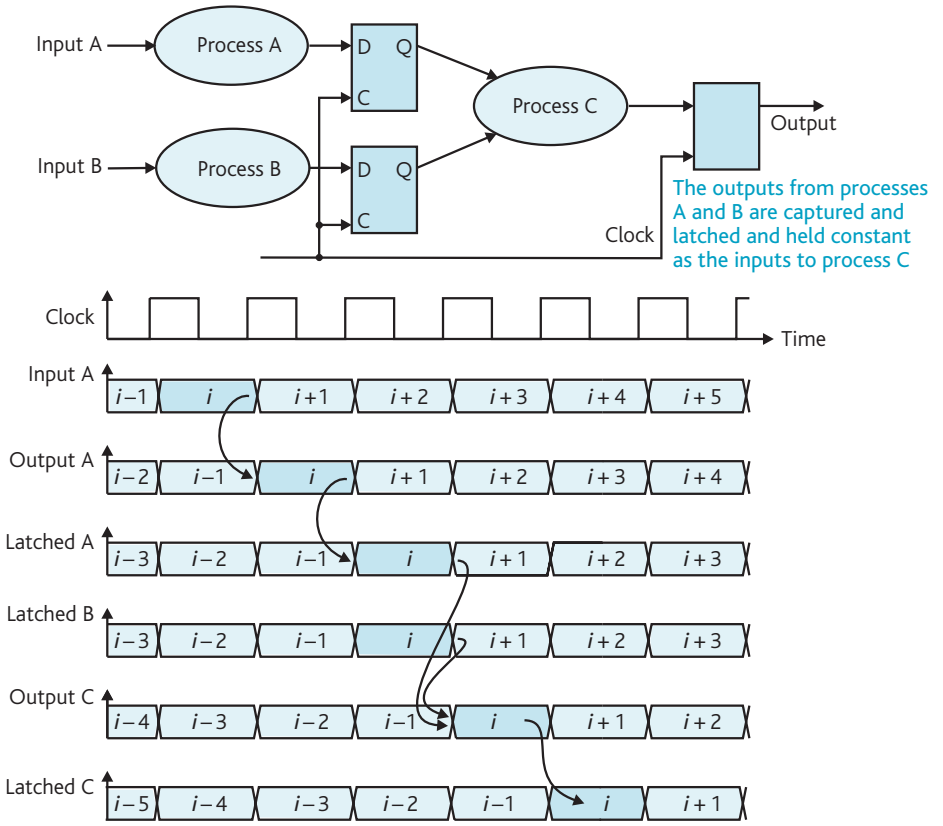


Figure 3.23 Latching the input and output of processes to implement pipelining.

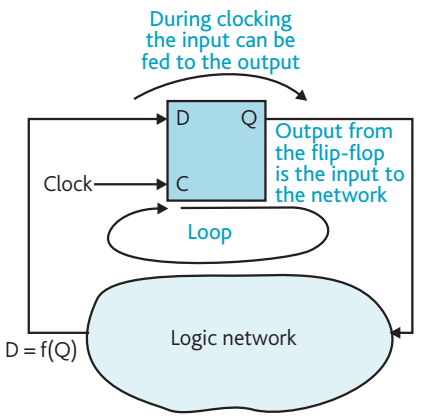


Figure 3.24 Feedback and the level-sensitive clock.

3.3.4 The master–slave flip-flop

The master–slave (MS) flip-flop has the external appearance of a single flip-flop, but internally is arranged as two flip-flops operating in series. One of these flip-flops is called the *master* and the other the *slave*. The term *slave* is used because the slave flip-flop follows the *master*. Figure 3.26 describes a simple RS master–slave flip-flop composed of two RS flip-flops in series. Note that the master flip-flop is enabled when the clock is high and the slave flip-flop is enabled when the clock is low.

When the clock pulse goes high, the input data at the R and S input terminals of the master flip-flop is copied into the master flip-flop. At this point, the output terminals of the master–slave flip-flop aren't affected and don't change state because the output comes from the slave flip-flop that is in a hold state because its clock is low.

Because the master flip-flop of Fig. 3.26 uses a level-sensitive RS flip-flop, the master responds to data at its RS inputs as long as the clock is asserted high. The data at the RS inputs of the master is latched by the master at the instant the clock input goes low. On the falling edge of the clock, the slave's clock input goes high and data from the master flip-flop's outputs is copied into the save flip-flop. Only now may the output terminals change state. Figure 3.27 provides a timing diagram for the master–slave RS flip-flop.

Master–slave flip-flops totally isolate their input terminals from their output terminals simply because the output of the slave flip-flop does not change until *after* the input conditions have been sampled and latched internally in the master. Conceptually, the master–slave flip-flop behaves like an air

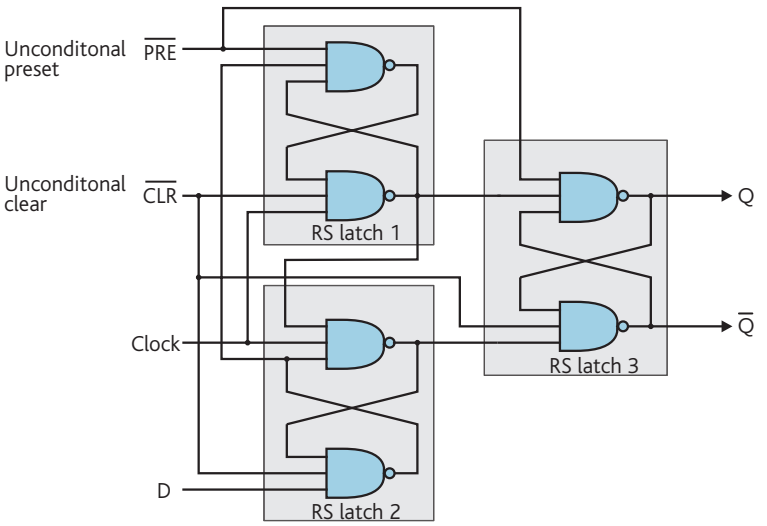


Figure 3.25 Circuit of an edge-triggered flip-flop.

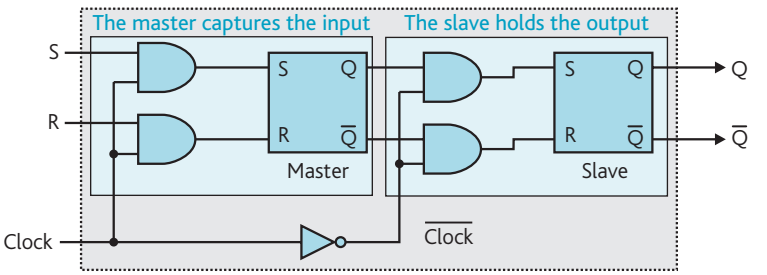


Figure 3.26 The master–slave RS flip-flop.

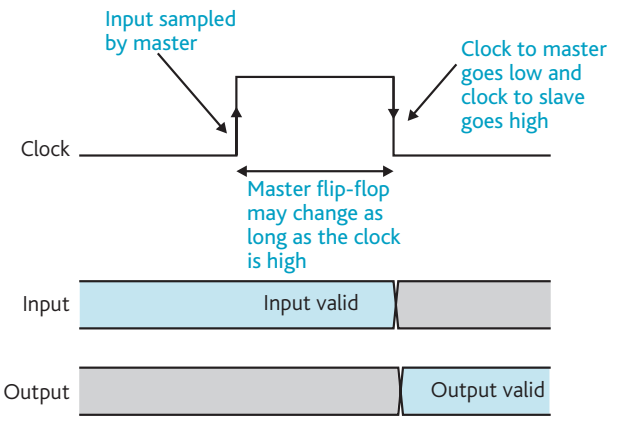


Figure 3.27 Timing diagram of a master–slave RS flip-flop.

lock in a submarine or spacecraft. An air lock exists to transfer people between regions of different pressure (air-to-vacuum or air-to-water) without ever permitting a direct path between the two pressure regions. A flip-flop is analogous to an air lock because its output must not be fed directly back to its input. To operate an air lock in a submarine, divers in the water open the air lock, enter, and close the door behind them. The divers are now isolated from both the water outside and the air inside. When the divers open the door into the submarine, they step inside and close the air lock door behind them.

In order to demonstrate how the different types of clocked flip-flop behave, Fig. 3.28 presents the output waveforms for four clocked D flip-flops when presented with the same input.

3.3.5 Bus arbitration—an example

We now look at a more advanced application of flip-flops in a *bus arbitration circuit* that decides which of two processors get to access a block of common memory, called dual-ported RAM, when both processors want the memory at the same time. Students may omit this section on a first reading.

Let's look at a system with two processors that communicate via a common block of RAM called DPRAM (dual-ported RAM). Figure 3.29 describes such an arrangement. You could regard the DPRAM as a bridge between two buses.

Because both processors 1 and 2 operate independently, either processor may access the common memory at any time. We need a means of requesting control of the common memory and getting access to the memory even if both processors make near-simultaneous requests.

Figure 3.30 describes an arbiter with a clock input, two request inputs, and two grant outputs. The request and grant inputs and outputs are all active-low. The memory-request inputs, $\overline{\text{Request1}}$ and $\overline{\text{Request2}}$, are sampled by two positive-edge triggered latches. The arbiter clocks latch 1a on the rising edge of the clock and latch 2a on the falling edge of the clock. This arrangement ensures that the two request inputs are not sampled simultaneously.

Figure 3.31 provides a timing diagram for the case in which both processors request the bus simultaneously. As we can see, processor 2 wins the request and processor 1 must wait until processor 2 has relinquished the bus. That is, processor 1 does not have to try again—it simply waits for the memory to become free. Processor 1 determines that the bus is once more free.

Initially, the arbiter is in an idle state with both request inputs inactive-high. Therefore, both D inputs to latches 1a and 2a are high and in a steady-state condition. Outputs AA, BB, Grant1, and Grant2 are all high.

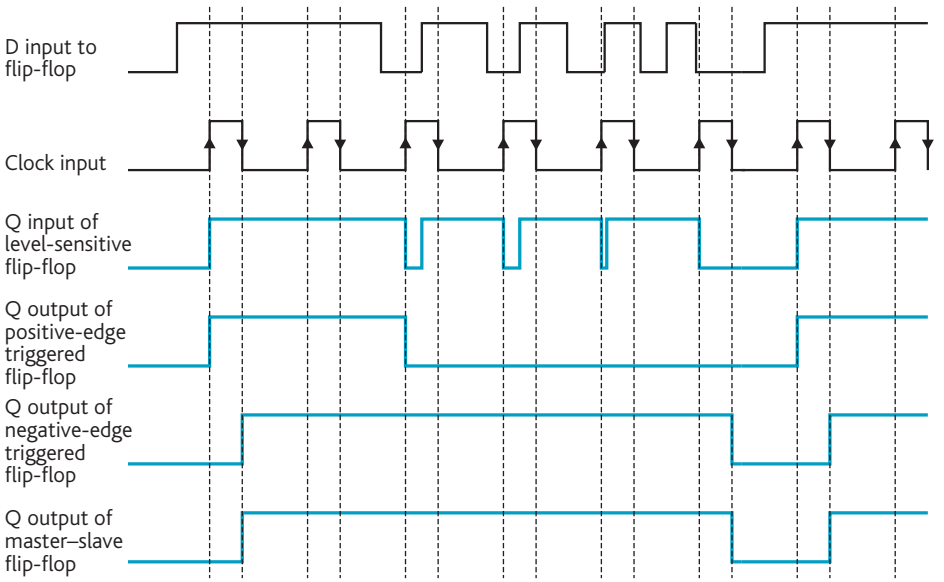


Figure 3.28 Comparison of flip-flop clocking modes.

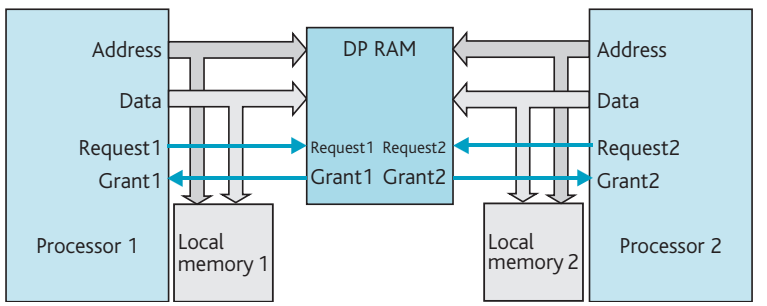


Figure 3.29 Two processors communicating via dual-ported RAM.

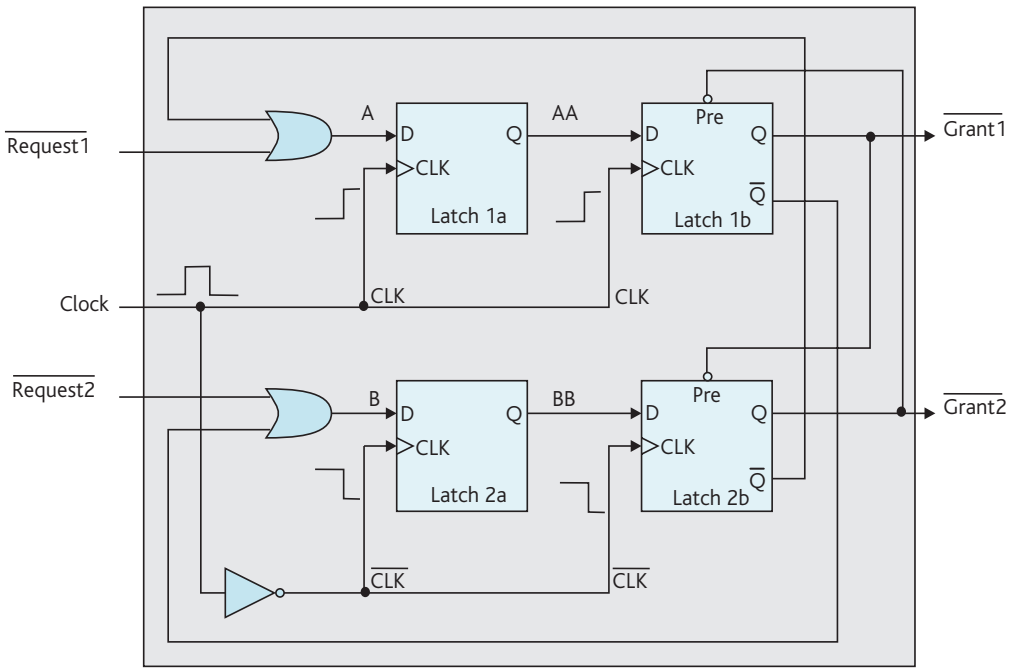


Figure 3.30 An arbiter circuit.

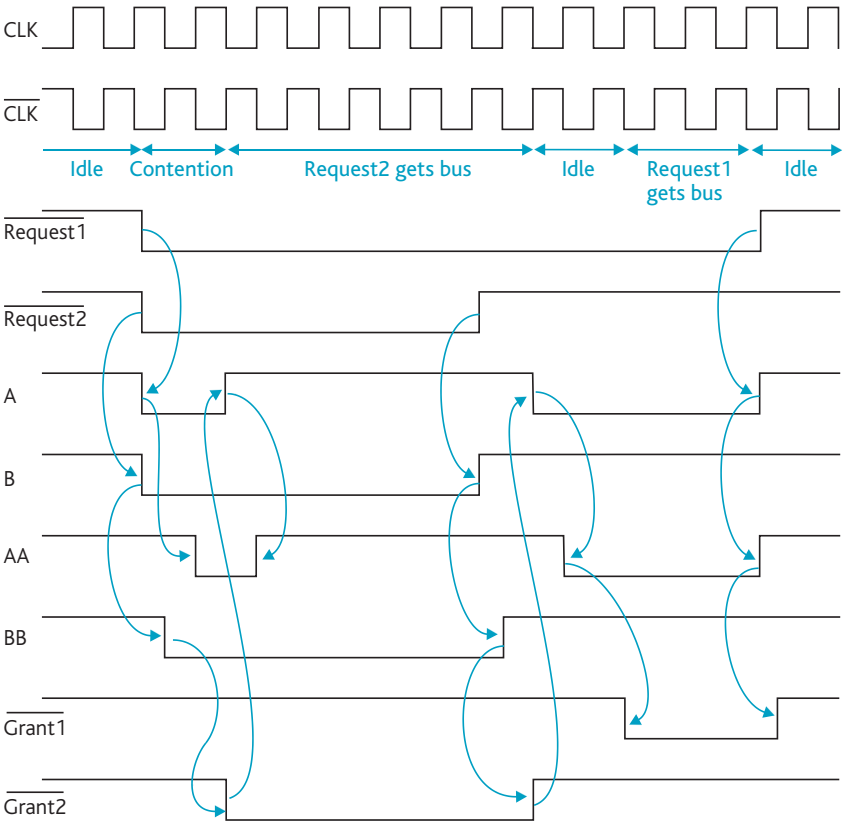


Figure 3.31 Timing diagram for Fig. 3.30.

Suppose that Request1 and Request2 are asserted almost simultaneously when the clock is in a high state. This results in the outputs of both OR gates (A and B) going low simultaneously. The cross-coupled feedback inputs to the OR gates (Grant1 and Grant2) are currently both low.

On the next rising edge of the clock, the Q output of latch 1a (i.e. AA) and the Q output of latch 2a (i.e. BB) both go low. However, as latch 2a sees a rising edge clock first, its Q output goes low one half a clock cycle before latch 1's output also goes low.

When a latch is clocked at the moment its input is changing, it may enter a metastable[1] state lasting for up to about 75 ns before the output of the latch settles into one state or the other. For this reason a second pair of latches is used to sample the input latches after a period of 80 ns.

One clock cycle after Request2 has been latched and output BB forced low, the output of latch 2b, Grant2 goes low. Its complement, Grant2 is fed back to OR gate 1, forcing input A high. After a clock cycle AA also goes high. Because Grant2 is connected to latch 1b's active-low preset input, latch 1b is held in a high state.

At this point, Grant1 is negated and Grant2 asserted, permitting processor 2 to access the bus.

When processor 1 relinquishes the memory, Request2 becomes inactive-high, causing first B, then BB and finally Grant2 to be negated as the change ripples through the arbiter. Once Grant2 is high, Grant2 goes low, causing the output of OR gate 1 (i.e. A) to go low. This is clocked through latches 1a and 1b to force Grant1 low and therefore permit processor 1 to access the memory. Of course, once Grant1 is asserted, any assertion of Request2 is ignored.

3.4 The JK flip-flop

The JK flip-flop can be configured, or programmed, to operate in one of two modes. All JK flip-flops are clocked and the majority of them operate on the master–slave principle. The truth table for a JK flip-flop is given in Table 3.5 and Fig. 3.32 gives its logic symbol. A bubble at the clock input to a flip-flop indicates that the flip-flop changes state on the *falling* edge of a clock pulse.

Table 3.5 demonstrates that for all values of J and K, except J = K = 1, the JK flip-flop behaves exactly like an RS flip-flop with J acting as the set input and K acting as the reset input. When J and K are both true, the output of the JK flip-flop

[1] If a latch is clocked at the exact moment its input is changing state, it can enter a metastable state in which its output is undefined and it may even oscillate for a few nanoseconds. You can avoid the effects of metastability by latching a signal, waiting for it to settle, and then capturing it in a second latch.

Full form					Algebraic form			
Inputs				Output	Inputs			Output
J	K	Q		Q^+	J	K		Q^+
0	0	0	0	No change	0	0	Q	No change
0	0	1	1	No change	0	1	0	Clear
0	1	0	0	Reset Q	1	0	1	Set
0	1	1	0	Reset Q	1	1	$\overline{Q}$	Toggle
1	0	0	1	Set Q				
1	0	1	1	Set Q				
1	1	0	1	$Q^+ \leftarrow \overline{Q}$				
1	1	1	0	$Q^+ \leftarrow \overline{Q}$				

Table 3.5 Truth table for a JK flip-flop.

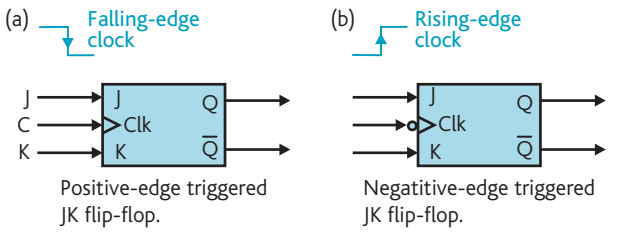


Figure 3.32 Representation of the JK flip-flop.

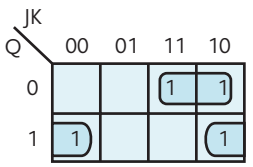


Figure 3.33 Deriving the characteristic equation of a JK flip-flop.

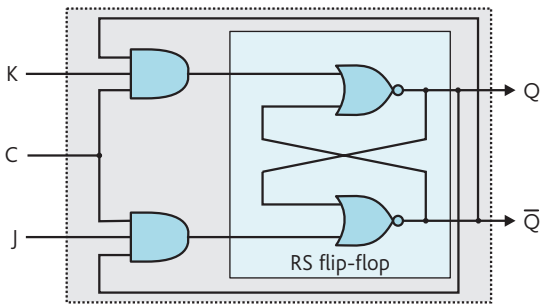


Figure 3.34 Construction of a basic JK flip-flop.

toggles, or changes state, each time the flip-flop is clocked. That is, if Q was a 0 it becomes a 1 and vice versa. It is this property that puts the JK flip-flop at the heart of many counter circuits, the operation of which is dealt with in the next section. Note that the *T flip-flop* is a JK flip-flop with J = K = 1, which changes state on each clock pulse (we don't deal with T flip-flops further in this text).

We can derive the characteristic equation for a JK flip-flop by plotting Table 3.5 on a Karnaugh map, Fig. 3.33. This gives $Q^+ = J \cdot \overline{Q} + \overline{K} \cdot Q$.

Figure 3.34 demonstrates how a JK flip-flop can be constructed from NAND gates and Fig. 3.35 describes a master–slave JK flip-flop.

3.5 Summary of flip-flop types

To understand flip-flops, it's necessary to appreciate that, unlike combinational circuits, they have internal states as well as external inputs; that is, the output of a flip-flop depends on the previous inputs of the flip-flop. Flip-flops are therefore *memory elements*. The most common forms of flip-flop are the D flip-flop, the RS flip-flop, and the JK flip-flop. Each flip-flop has two outputs, Q and its complement $\overline{Q}$, although the complementary output is not always connected to a pin in an integrated circuit. Most flip-flops are clocked and have a clock input that is used to trigger the flip-flop. Flip-flops often have unconditional preset and clear inputs that can be used the set or clear the output, respectively. The term unconditional means that these inputs override any clock input.

The D flip-flop D flip-flops have two inputs, a D (data) input and a C (clock) input. The output of a D flip-flop remains in its previous state until its C input is clocked. When its C input is clocked, the Q output becomes equal to D *until the next time it is clocked*.

The RS flip-flop An RS flip-flop has two inputs, R (reset) and S (set). As long as both R and S are 0, the Q output of the RS flip-flop is constant and remains in its previous state. When R = 1 and S = 0, the Q output is forced to 0 (and

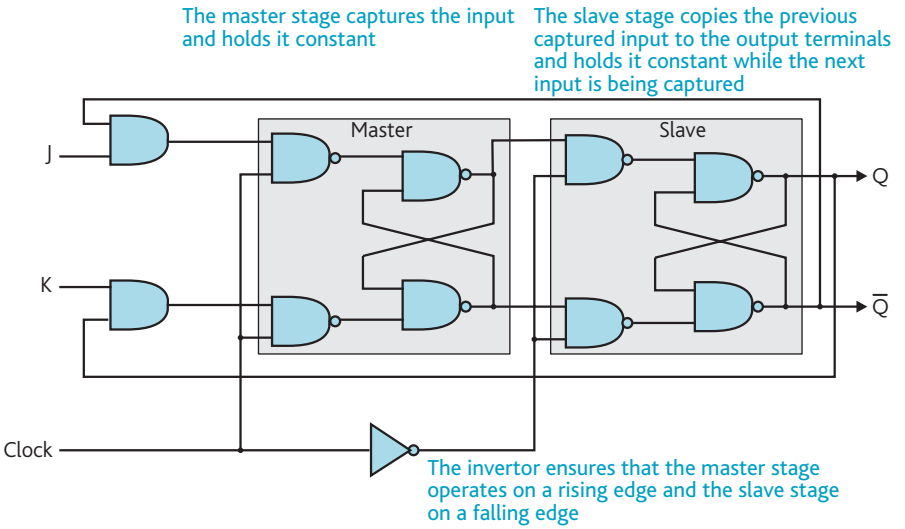


Figure 3.35 Circuit diagram of a master–slave JK flip-flop.

remains at zero when R returns to 0). When S = 1 and R = 0, the Q output is forced to one (and remains at one when S returns to 0). The input conditions R = S = 1 produce an *indeterminate state* and should be avoided. Clocked RS flip-flops behave as we have described, except that their R and S inputs are treated as zero until the flip-flop is clocked. When the RS flip-flop is clocked, its Q output behaves as we have just described.

The JK flip-flop The JK flip-flop always has three inputs, J, K, and a clock input C. As long as a JK flip-flop is not clocked, its output remains in the previous state. When a JK flip-flop is clocked, it behaves like an RS flip-flop (where J = S, K = R) for all input conditions except J = K = 1. If J = K = 0, the output does not change state. If K = 1 and J = 0, the Q output is reset to zero. If J = 1 and K = 0, the Q output is set to 1. If both J and K are 1, the output changes state (or *toggles*) each time it is clocked.

The T flip-flop The T flip-flop has a single clock input. Each time it is clocked, its output *toggles* or changes state. A T flip-flop is functionally equivalent to a JK flip-flop with J = K = 1.

3.6 Applications of sequential elements

Just as the logic gate is combined with other gates to form combinational circuits such as adders and multiplexers, flip-flops can be combined together to create a class of circuits called *sequential circuits*. Here, we are concerned with two particular types of sequential circuit: the *shift register*, which moves a group of bits left or right and the *counter*, which steps through a sequence of values.

3.6.1 Shift register

By slightly modifying the circuit of the register we can build a *shift register* whose bits can be moved one place right every time the register is clocked. For example, the binary pattern

	01110101
becomes	00111010 after the shift register is clocked once
and	00011101 after it is clocked twice
and	00001110 after it is clocked three times, and so on.

Note that after the first shift, a 0 has been shifted in from the left-hand end and the 1 at the right-hand end has been lost. We used the expression *binary pattern* because, as we shall see later, the byte 01110101 can represent many things. However, when the pattern represents a binary number, shifting it one place right has the effect of dividing the number by two (just as shifting a decimal number one place right divides it by 10). Similarly, shifting a number one place *left* multiplies it by 2. Later we will see that special care has to be taken when shifting signed two's complement binary numbers right (the sign-bit has to be dealt with).

Figure 3.36 demonstrates how a shift register is constructed from D flip-flops. The Q output of each flip-flop is connected to the D input of the flip-flop on its right. All clock inputs are connected together so that each flip-flop is clocked simultaneously. When the ith stage is clocked, its output, Q_i, takes on the value from the stage on its left, that is, $Q_i \leftarrow Q_{i+1}$. Data presented at the input of the left-hand flip-flop, D_{in}, is shifted into the $(m-1)$th stage at each clock pulse. Figure 3.36 describes a *right*-shift register—we will look at registers that shift the data sequence left shortly.

The flip-flops in a shift register must either be edge-triggered or master-slave flip-flops, otherwise if a level-sensitive flip-flop were used, the value at the input to the left-hand

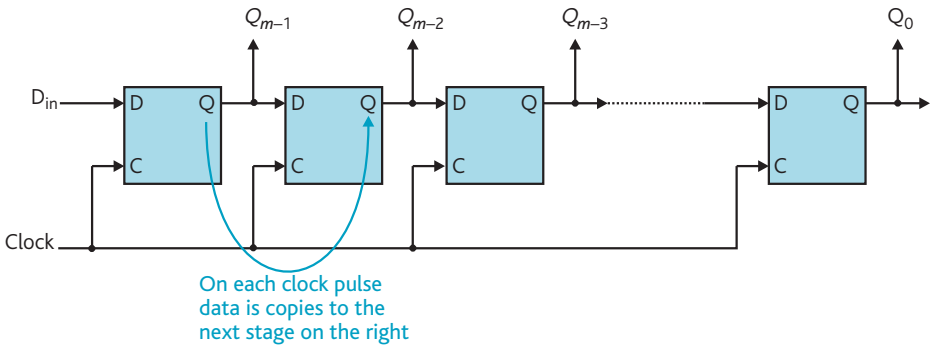


Figure 3.36 The right-shift register.

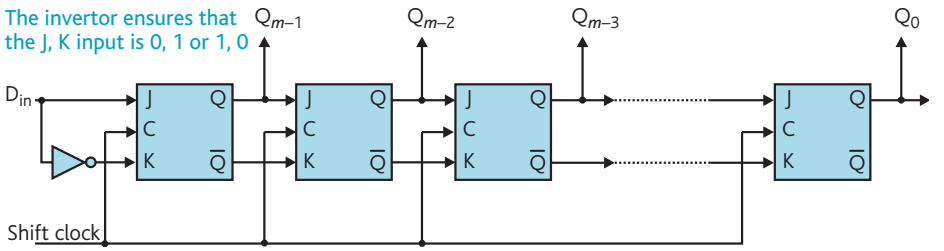
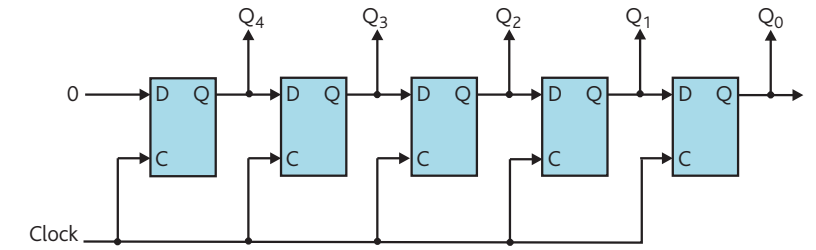


Figure 3.37 Shift register composed of JK flip-flops.



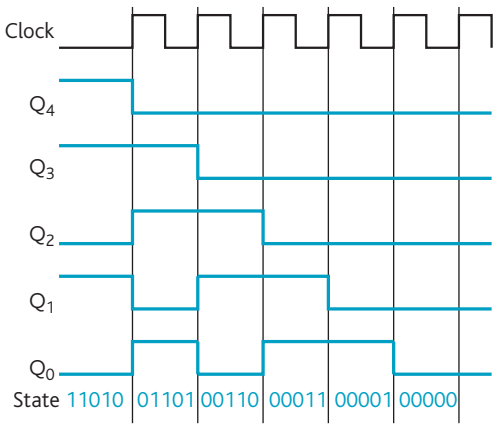


Figure 3.38 Example of a five-stage shift-right register.

stage would ripple through all stages as soon as the clock went high. We can construct a shift register from JK flip-flops just as easily as from RS flip-flops as Fig. 3.37 demonstrates.

Figure 3.38 shows a five-stage shift register that contains the initial value 01101. At each clock pulse the bits are shifted right and a 0 enters the most-significant bit stage. This figure also provides a timing diagram for each of the five Q outputs. The output of the right-hand stage, Q_0, consists of a series of five sequential pulses, corresponding to the five bits of the word in the shift register (i.e. 11010).

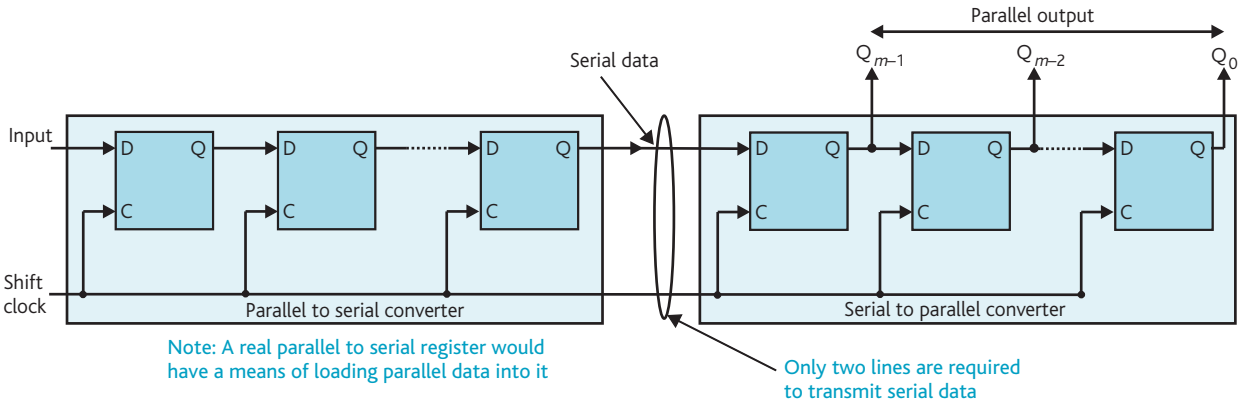


Figure 3.39 Serial to parallel converter.

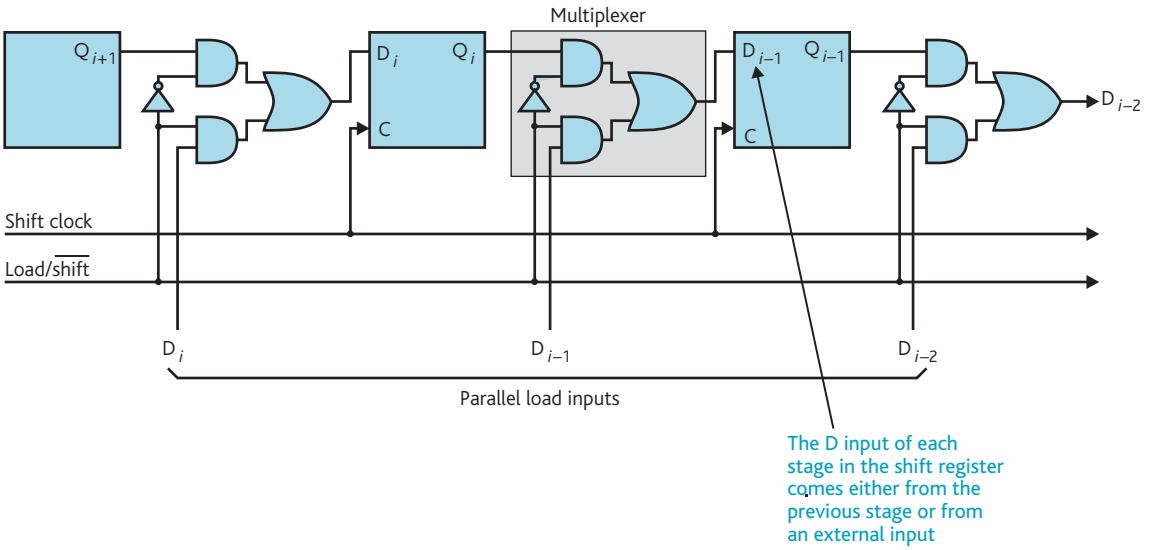


Figure 3.40 Shift register with a parallel load capability.

A shift register can be used to convert a parallel word of m bits into a serial word of m consecutive bits. Such a circuit is called a *parallel to serial converter*. If the output of an m-bit parallel to serial converter is connected to the D_{in} input of an m-bit shift register, after m clock pulses the information in the parallel to serial converter has been transferred to the second (right-hand) shift register. Such a shift register is called a *serial to parallel converter* and Fig. 3.39 describes a simplified version. In practice, a means of loading parallel data into the parallel-to-serial converter is necessary (see Fig. 3.40). There is almost no difference between a parallel to serial converter and a serial to parallel converter.

A flaw in our shift register (when operating as a parallel to serial converter) is the lack of any facilities for loading it with m bits of data at one go, rather than by shifting in m bits through D_{in}. Figure 3.40 shows a right-shift register with a parallel load capacity. A two-input multiplexer, composed of two AND gates, an OR gate, and an inverter switches a flip-flop's D input between the output of the previous stage to the left (*shift* mode) and the load input (*load* mode). The control inputs of all multiplexers are connected together to provide the mode control, labeled load/$\overline{\text{shift}}$. When we label a variable name1/$\overline{\text{name2}}$, we mean that when the variable is high it carries out action *name1* and when it is low it carries out action *name2*. If load/$\overline{\text{shift}}$ = 0 the operation performed is a shift and if load/$\overline{\text{shift}}$ = 1 the operation performed is a load.

Constructing a left-shift register with JK flip-flops

Although we've considered the right-shift register, a left-shift register is easy to design. The input of the ith stage, D_i, is

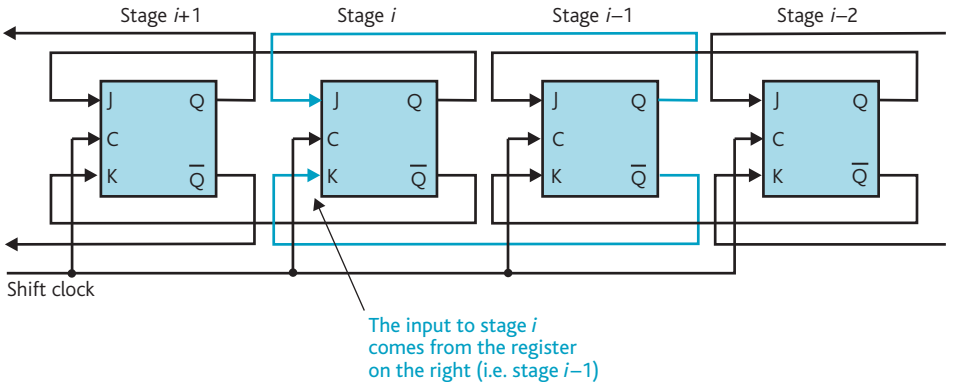


Figure 3.41 The left-shift register.

Shift type	Shift left	Shift right
Original bit pattern before shift	11010111	11010111
Logical shift	10101110	01101011
Arithmetic shift	10101110	11101011
Circular shift	10101111	11101011

Table 3.6 The effect of logical, arithmetic, and circular shifts.

connected to the output of the $(i-1)$th stage so that, at each clock pulse, $Q_i \leftarrow D_{i-1}$. In terms of the previous example

01110101
becomes 11101010 after one shift left
and 11212100 after two shifts left

The structure of a left-shift register composed of JK flip-flops is described in Fig. 3.41.

When we introduce the instruction set of a typical computer we'll see that there are several types of shift (logical, arithmetic, circular). These operations all shift bits left or right—the only difference between them concerns what happens to the bit shifted in. So far we've described the *logical* shift where a 0 is shifted in and the bit shifted out at the other end is lost. In an *arithmetic* shift the sign of 2's complement number is preserved when it is shifted right (this will become clear when we introduce the representation of negative numbers in the next chapter). In a *circular* shift the bit shifted out of one end becomes the bit shifted in at the other end. Table 3.6 describes what happens when the 8-bit value 11010111 undergoes three types of shift.

A typical shift register

Figure 3.42 gives the internal structure of a 74LS95 parallel-access bidirectional shift register chip. You access the shift register through its pins and cannot make connections to the internal parts of its circuit. Indeed, its actual internal implementation may differ from the published circuit. As long as it behaves like its published circuit, the precise implementation of its logic function doesn't matter to the end user. The 74LS95 is a versatile shift register and has the following functions.

Parallel load The four bits of data to be loaded into the shift register are applied to its parallel inputs, the mode control input is set to a logical one, and a clock pulse applied to the clock 2 input. The data is loaded on the falling edge of the clock 2 pulse.

Right-shift A shift right is accomplished by setting the mode control input to a logical zero and applying a pulse to the clock 1 input. The shift takes place on the falling edge of the clock pulse.

Left-shift A shift left is accomplished by setting the mode control input to a logical one and applying a pulse to the clock 2 input. The shift takes place on the falling edge of the clock pulse. A left shift requires that the output of each flip-flop be connected to the parallel input of the previous flip-flop and serial data entered at the D input.

Table 3.7 provides a function table for this shift register (taken from the manufacturer's literature). This table describes the behavior of the shift register for all combinations of its inputs. Note that the table includes don't care values of inputs and the effects of input *transitions* (indicated by ↓ and ↑).

Designing a versatile shift register—an example

Let's design an 8-bit shift register to perform the following operations.

(a) Load each stage from an 8-bit data bus	(parallel load)
(b) Logical shift left	(0 in, MSB lost)
(c) Logical shift right	(0 in, LSB lost)
(d) Arithmetic shift left	(same as logical shift left)
(e) Arithmetic shift right	(MSB replicated, LSB lost)
(f) Circular shift left	(MSB moves to LSB position)
(g) Circular shift right	(LSB moves to MSB position)

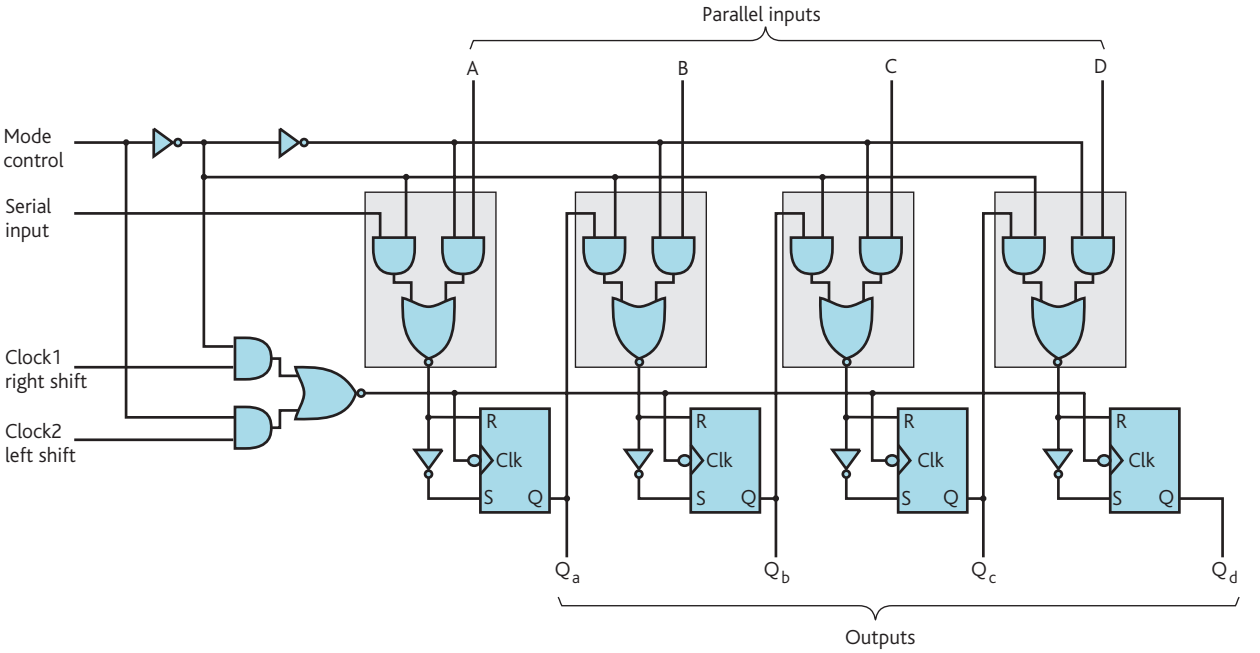


Figure 3.42 The left-shift register.

Inputs								Outputs			
Mode control	Clocks		Serial	Parallel inputs							
	2 (L)	1(R)		A	B	C	D	Q_a	Q_b	Q_c	Q_d
1	1	x	x	x	x	x	x	Q_{a0}	Q_{b0}	Q_{c0}	Q_{d0}
1	↓	x	x	A	B	C	D	A	B	C	D
1	↓	x	x	Q_b	Q_c	Q_d	D	Q_{bn}	Q_{cn}	Q_{dn}	D
0	0	1	x	x	x	x	x	Q_{a0}	Q_{b0}	Q_{c0}	Q_{d0}
0	x	↓	1	x	x	x	x	1	Q_{an}	Q_{bn}	Q_{cn}
0	x	↓	0	x	x	x	x	0	Q_{an}	Q_{bn}	Q_{cn}
↑	0	0	x	x	x	x	x	Q_{a0}	Q_{b0}	Q_{c0}	Q_{d0}
↓	0	0	x	x	x	x	x	Q_{a0}	Q_{b0}	Q_{c0}	Q_{d0}
↓	0	1	x	x	x	x	x	Q_{a0}	Q_{b0}	Q_{c0}	Q_{d0}
↑	1	0	x	x	x	x	x	Q_{a0}	Q_{b0}	Q_{c0}	Q_{d0}
↑	1	1	x	x	x	x	x	Q_{a0}	Q_{b0}	Q_{c0}	Q_{d0}

Notes
1. Left-shift operations assume that Q_b is connected to A, Q_c to B, and Q_d to C.
2. x = don't care.
3. ↓ and ↑ indicate high-to-low and low-to-high transitions, respectively.
4. Q_{a0} indicates the level at Q_a before the indicated inputs were established.
5. Q_{an} indicates the level of Q_a before the ↓ transition of the clock.

Table 3.7 Function table for a 74LS95 shift register.

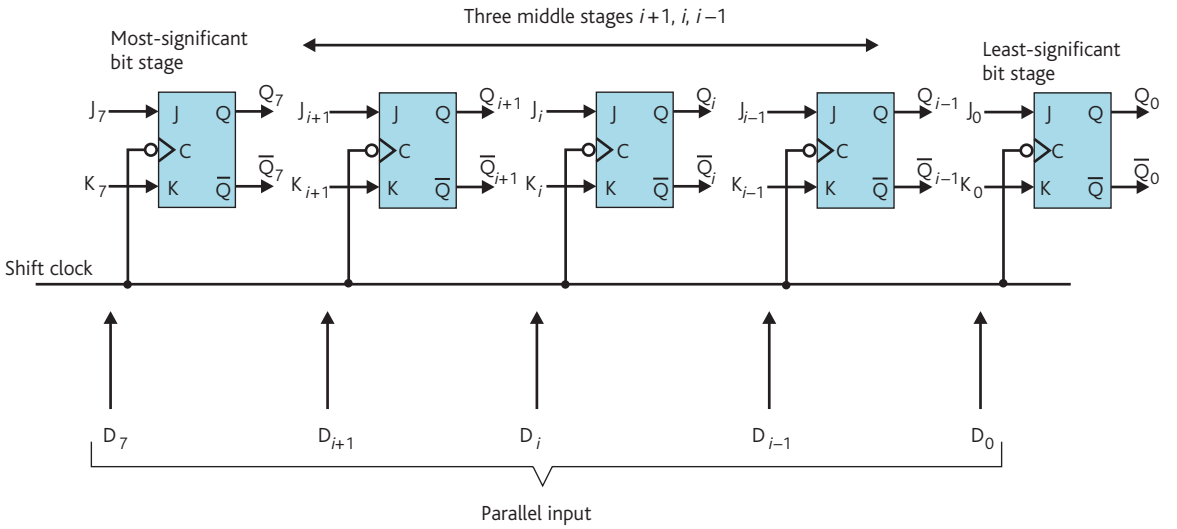


Figure 3.43 End and middle stages of a shift register.

The circuit is composed of eight master–slave JK flip-flops and has a clock input that causes operations (a)–(g) above to be carried out on its falling edge. The circuit has five control inputs:

R When R = 1 shift right, when R = 0 shift left.
S When S = 1 perform a shift operation, when S = 0 a parallel load.
L When L = 1 perform a logical shift (if S = 1).
A When A = 1 perform an arithmetic shift (if S = 1).
C When C = 1 perform a circular shift (if S = 1).

Assume that illegal combinations of L, A, and C cannot occur because only one type of shift can be performed at a time. Therefore, more than one of L, A, and C, will never be true simultaneously.

For all eight stages of the shift register obtain algebraic expressions for J and K in terms of control inputs R, S, L, A, and C and the outputs of the flip-flops.

Figure 3.43 illustrates five stages of the shift register. These are the end stages Q_7 and Q_0, the most-significant and least-significant bit stages, respectively. A non-end stage Q_i, together with its left-hand neighbor Q_{i+1} and its right-hand neighbor Q_{i-1}, must also be considered.

All stages except 0 and 7 perform the same functions: parallel load, shift right, and shift left. As the JK flip-flops always load from an external input or another stage, only the inputs J = 1, K = 0, or J = 0, K = 1 have to be considered. Consequently, $J = \overline{K}$ and we need only derive expressions for J, as the corresponding values for K can be obtained from an inverter.

Stage i

Parallel load	$J_i = D_i$	$S = 0$
Shift right	$J_i = Q_{i+1}$	$S = 1,\ R = 1$
Shift left	$J_i = Q_{i-1}$	$S = 1,\ R = 0$

Therefore, $J_i = \overline{S}\cdot D_i + S(R\cdot Q_{i+1} + \overline{R}\cdot Q_{i-1})$

Stage 0 (LSB)

Parallel load		$J_0 = D_0$	$S = 0$
Shift right	logical	$J_0 = Q_1$	$S = 1,\ R = 1,\ L = 1$
	arithmetic	$J_0 = Q_1$	$S = 1,\ R = 1,\ A = 1$
	circular	$J_0 = Q_1$	$S = 1,\ R = 1,\ C = 1$
Shift left	logical	$J_0 = 0$	$S = 1,\ R = 0,\ L = 1$
	arithmetic	$J_0 = 0$	$S = 1,\ R = 0,\ A = 1$
	circular	$J_0 = Q_7$	$S = 1,\ R = 0,\ C = 1$

$$\begin{aligned}\text{Therefore, } J_0 &= \overline{S}\cdot D_0 + S(R\cdot L\cdot Q_1 + R\cdot A\cdot Q_1 + R\cdot C\cdot Q_1 \\ &\quad + \overline{R}\cdot L\cdot 0 + \overline{R}\cdot A\cdot 0 + \overline{R}\cdot C\cdot Q_7) \\ &= \overline{S}\cdot D_0 + S(R\cdot L\cdot Q_1 + R\cdot A\cdot Q_1 + R\cdot C\cdot Q_1 \\ &\quad + \overline{R}\cdot C\cdot Q_7) \\ &= \overline{S}\cdot D_0 + S(R\cdot Q_1(L + A + C) + \overline{R}\cdot C\cdot Q_7) \\ &\quad \text{Note: } L + A + C = 1 \\ &= \overline{S}\cdot D_0 + S(R\cdot Q_1 + \overline{R}\cdot C\cdot Q_7).\end{aligned}$$

Stage 7 (MSB)

Parallel load		$J_7 = D_7$	$S = 0$
Shift right	logical	$J_7 = 0$	$S = 1,\ R = 1,\ L = 1$
	arithmetic	$J_7 = Q_7$	$S = 1,\ R = 1,\ A = 1$
	circular	$J_7 = Q_0$	$S = 1,\ R = 1,\ C = 1$
Shift left	logical	$J_7 = Q_6$	$S = 1,\ R = 0,\ L = 1$
	arithmetic	$J_7 = Q_6$	$S = 1,\ R = 0,\ A = 1$
	circular	$J_7 = Q_6$	$S = 1,\ R = 0,\ C = 1$

$$\begin{aligned}\text{Therefore, } J_7 &= \overline{S}\cdot D_7 + S(R\cdot L\cdot 0 + R\cdot A\cdot Q_7 + R\cdot C\cdot Q_0 \\ &\quad + R\cdot L\cdot Q_6 + \overline{R}\cdot A\cdot Q_6 + R\cdot C\cdot Q_6) \\ &= \overline{S}\cdot D_7 + S(R\cdot A\cdot Q_7 + R\cdot C\cdot Q_0 \\ &\quad + \overline{R}\cdot Q_6\,(L + A + C)) \\ &= \overline{S}\cdot D_7 + S(R\,(A\cdot Q_7 + C\cdot Q_0) + \overline{R}\cdot Q_6)\end{aligned}$$

3.6.2 Asynchronous counters

A counter is a sequential circuit with a clock input and *m* outputs. Each time the counter is clocked, one or more of its outputs change state. These outputs form a sequence with *N* unique values. After the *N*th value has been observed at the counter's output terminals, the next clock pulse causes the counter to assume the same output as it had at the start of the sequence; that is, the sequence is *cyclic*. For example, a counter may display the sequence 01234501234501 . . . or the sequence 9731097310973 . . .

A counter composed of *m* flip-flops can generate an arbitrary sequence with a length of not greater than 2^m cycles before the sequence begins to repeat itself.

One of the tools frequently employed to illustrate the operation of sequential circuits is the *state diagram*. Any system with internal memory and external inputs such as the flip-flop can be said to be in a state that is a function of its internal and external inputs. A state diagram shows some (or all) of the possible states of a given system. A labeled circle represents each of the states and the states are linked by unidirectional lines showing the paths by which one state becomes another state.

Figure 3.44 gives the state diagram of a JK flip-flop that has just two states, S_0 and S_1. S_0 represents the state $Q = 0$ and S_1 represents the state $Q = 1$. The transitions between states S_0 and S_1 are determined by the values of the JK inputs at the time the flip-flop is clocked. In Fig. 3.44 we have labeled the flip-flop's input states C_1 to C_4. Table 3.8 defines the four possible input conditions, C_1, C_2, C_3, and C_4, in terms of J and K.

From Fig. 3.44 it can be seen that conditions C_3 or C_4 cause a transition from state S_0 to state S_1. Similarly, conditions C_2 or C_4 cause a transition from state S_1 to state S_0. Condition C_4 causes a change of state from S_0 to S_1 and also from S_1 to S_0. This is, of course, the condition $J = K = 1$, which causes the JK flip-flop to toggle its output. Some conditions cause a state to change to *itself*; that is, there is no overall change. Thus, conditions C_1 or C_2, when applied to the system in state S_0, have the effect of leaving the system in state S_0.

The binary up-counter

The state diagram of a simple 3-bit binary up-counter is given in Fig. 3.45 (an up-counter counts upward 0, 1, 2, 3, . . . in contrast with a down-counter, which counts downward . . . , 3, 2, 1, 0). In this state diagram, there is only a single path from each state to its next higher neighbor. As the system is clocked, it cycles through the states S_0 to S_7 representing the natural binary numbers 0 to 7. The actual design of counters in general can be quite involved, although the basic principle is to ask 'What input conditions are required by the flip-flops to cause them to change from state S_i to state S_{i+1}?'

The design of an *asynchronous* natural binary up-counter is rather simpler than the design of a counter for an arbitrary sequence. Figure 3.46 gives the circuit diagram of a 3-bit binary counter composed of JK flip-flops and Fig. 3.47 provides its timing diagram. The J and K inputs to each flip-flop

J	K	Condition
0	0	C_1
0	1	C_2
1	0	C_3
1	1	C_4

Table 3.8 Relationship between JK inputs and conditions C_1 to C_4.

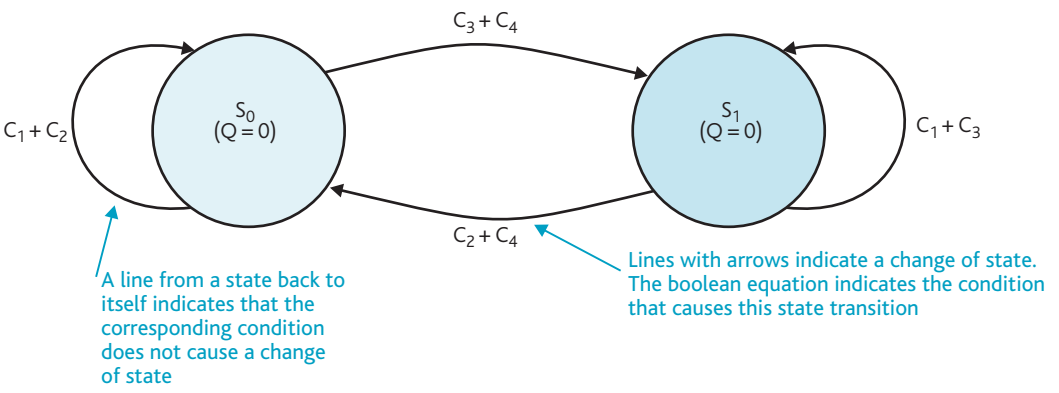


Figure 3.44 The state diagram of a JK flip-flop.

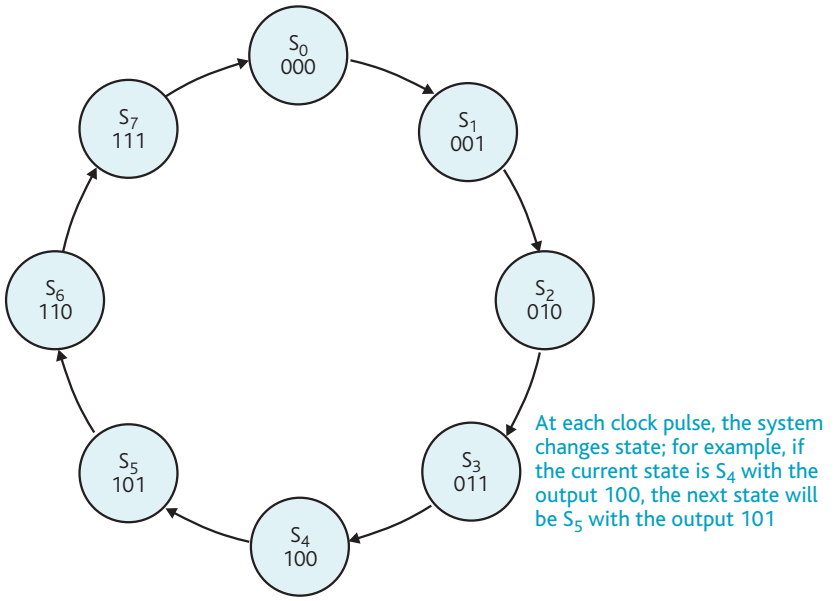


Figure 3.45 The state diagram of a binary 3-bit up-counter.

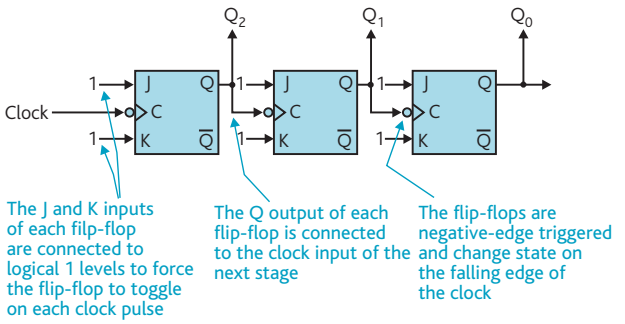


Figure 3.46 Circuit of an asynchronous binary up-counter.

are connected to constant logical 1 levels. Consequently, whenever a flip-flop is clocked, its output changes state. The flip-flops are arranged so that the Q output of one stage triggers the clock input of the next higher stage (i.e. the output Q_i of stage i triggers the clock input of stage $i+1$). The flip-flops in Fig. 3.46 are master–slave clocked and their outputs change on the negative edge of the clock pulse.

Consider the first stage of this counter. When the clock input makes a complete cycle (0 to 1 to 0), the Q output changes state on the falling edge of the clock. It takes two clock cycles to make the Q output execute one cycle; that is, the flip-flop divides the clock input by 2.

The asynchronous binary counter of Fig. 3.46 is called a *ripple counter* because the output of the first stage triggers the input of the second stage, the output of the second stage triggers the input of the third stage, and so on. Consequently, a change of state at the output of the first stage ripples through the counter until it clocks the final stage. The propagation delay through each stage of the counter determines its maximum speed of operation. The timing diagram of Fig. 3.47 doesn't show the ripple effect—when one stage changes state, there's a short delay before stages to its right change state.

Figure 3.48 demonstrates the construction of a four-stage binary up-counter in Digital Works. We have wired all J and K inputs together and connected them to V_{cc} (the positive power supply that provides a logical 1 to cause the JK flip-flops to toggle when clocked). We have labeled each of the Q outputs and used the **Logic History** function to capture the output waveform. Digital Works clears all flip-flops at the start of each run. However, the flip-flops have two unlabeled set and clear inputs that can be used to preset outputs to 1 or 0, respectively (these are not used in this application).

The binary down-counter

We can also create a binary down-counter that counts backwards from 7 to 0. Figure 3.49 demonstrates the effect of connecting the $\overline{Q}$ output of each stage in a ripple counter to the clock input of the next stage. You can also create a binary down-counter by using JK flip-flops that are clocked on the *positive* or rising edge of the clock pulse by connecting Q_i to Clk_{i+1}.

Designing an asynchronous decimal counter

Let's design a 4-bit asynchronous ripple-through decimal counter to count from 0 to 9 cyclically. We use JK master–slave flip-flops with an unconditional active-low clear input. A decimal counter can be derived from a binary counter by resetting the counter to zero at the appropriate point. A four-stage binary counter counts from 0000 to 1111

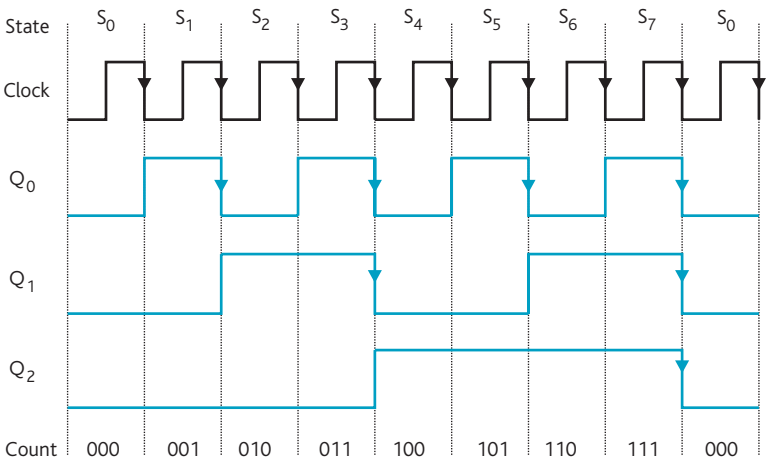


Figure 3.47 Timing diagram of an asynchronous 3-bit binary up-counter.

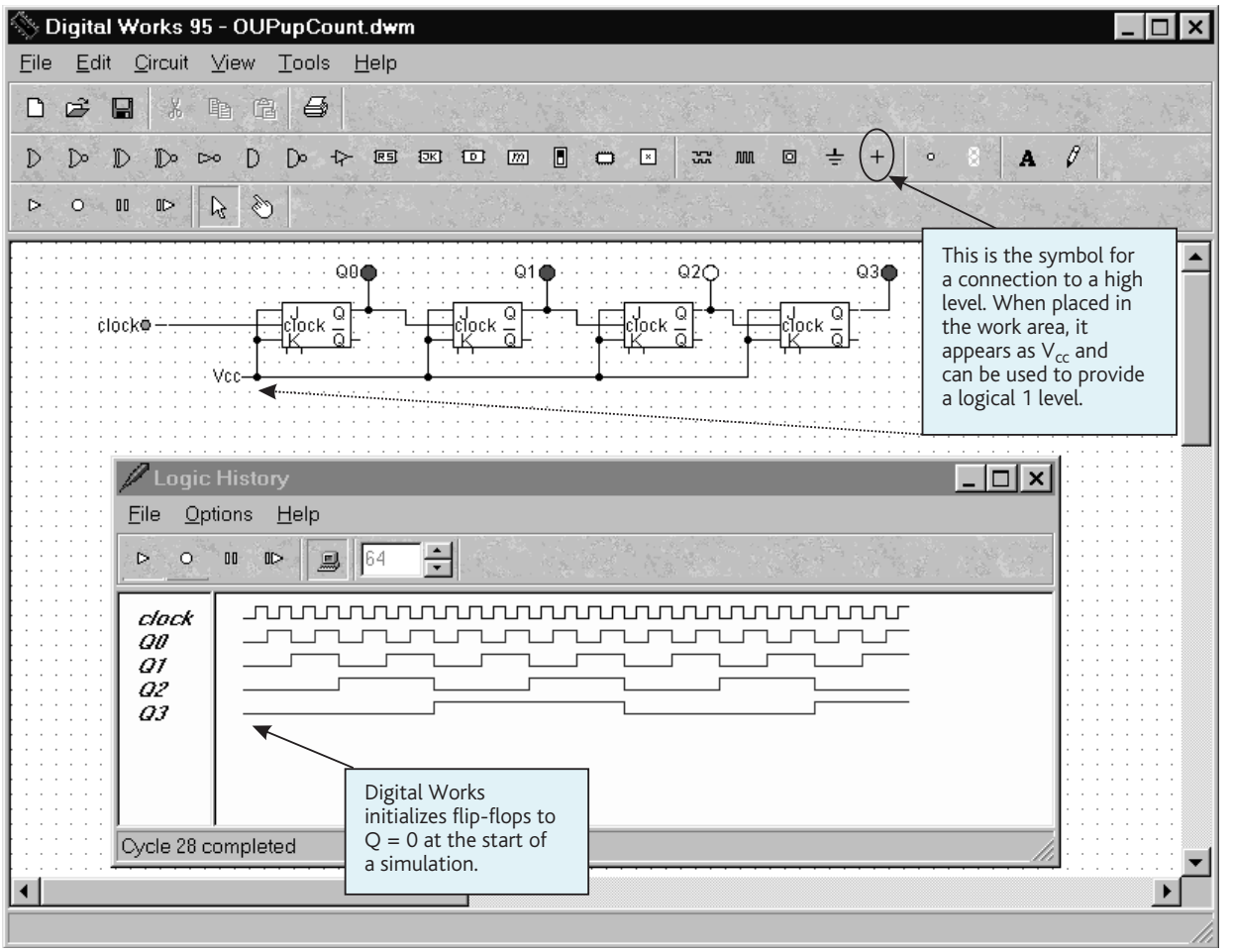


Figure 3.48 Using Digital Works to create a binary up-counter.

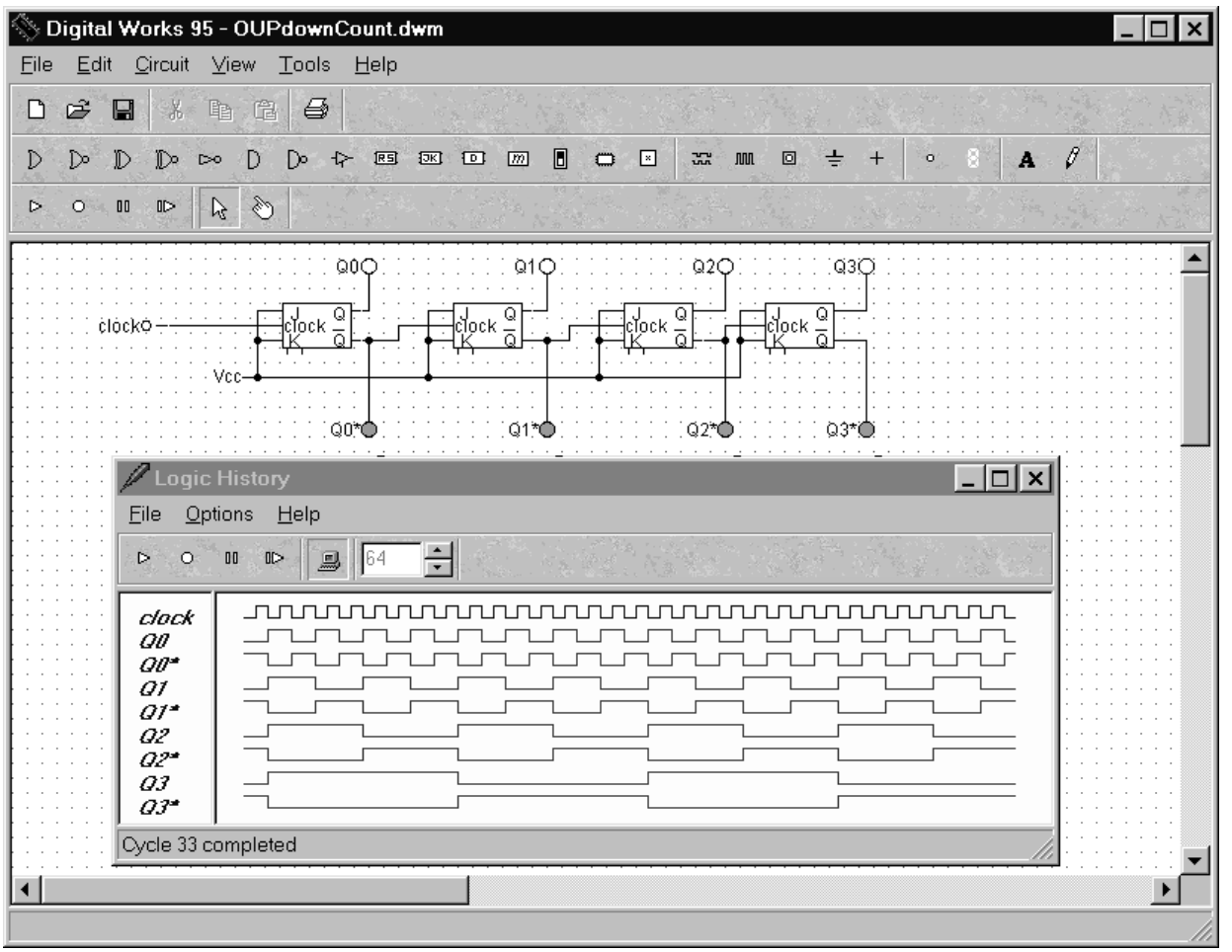


Figure 3.49 Using Digital Works to create a binary down-counter.

(i.e. 0 to 15). To create a decade counter the state 10 (1010) must be detected and used to reset the flip-flops. Fig. 3.50 provides a possible circuit.

The binary counter counts normally from 0 to 9. On the tenth count $Q_3 = 1$ and $Q_1 = 1$. This condition is detected by the NAND gate whose output goes low, resetting the flip-flops. The count of 10 exists momentarily as Fig. 3.51 demonstrates. We could have detected the state 10 with Q_3, Q_2, Q_1, $Q_0 = 1010$. However, that would have required a four-input gate and is not strictly necessary. Although $Q_3 = 1$ and $Q_1 = 1$ corresponds to counts 10, 11, 14, and 15, the counter never gets beyond 10.

The reset pulse must be long enough to reset all flip-flops to zero. If the reset pulse were too short and, say, Q_1 was reset before Q_3, the output might be reset to 1000. The counting sequence would now be: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (10), 8, 9, 8, 9, ... However, such a problem is unlikely to occur in this case, because the reset pulse is not removed until at least the output of one flip-flop and the NAND gate has changed state. The combined duration of flip-flop reset time plus a gate delay will normally provide sufficient time to ensure that all flip-flops are reset.

It is possible to imagine situations in which the circuit would not function correctly. Suppose that the minimum reset pulse required to guarantee the reset of a flip-flop were 50 ns. Suppose also that the minimum time between the application of a reset pulse and the transition $Q \leftarrow 0$ were 10 ns and that the propagation delay of a NAND gate were 10 ns. It would indeed be possible for the above error to occur. This example demonstrates the dangers of designing asynchronous circuits!

The pulse generator revisited

When we introduced the RS flip-flop we used it to start and stop a simple pulse generator that created a train of *n* pulses. Figure 3.52 shows a pulse generator in Digital Works. This system is essentially the same as that in Fig. 3.9, except that we've built the counter using JK flip-flops and we've added

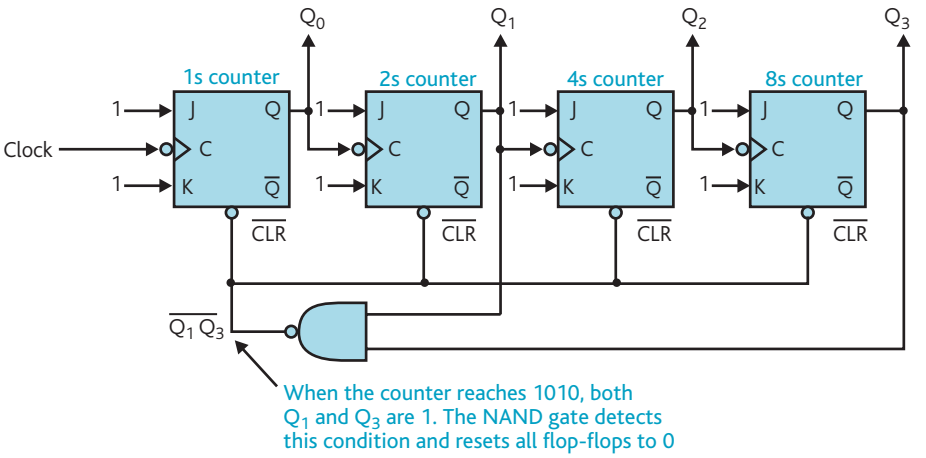


Figure 3.50 Circuit of a decimal counter.

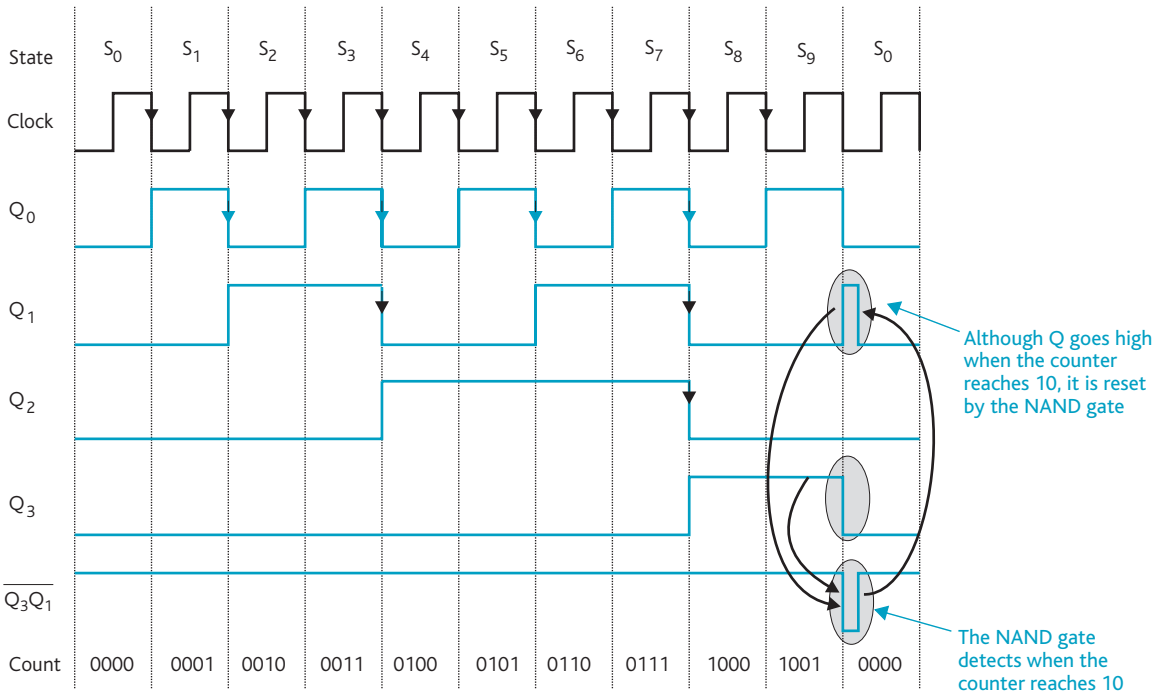


Figure 3.51 Timing diagram of a decimal counter.

LEDs to examine the signals produced when the system runs. Note also that the RS flip-flop can be set only when the flip-flop is in the reset mode.

3.6.3 Synchronous counters

Synchronous counters are composed of flip-flops that are all clocked at the same time. The outputs of *all* stages of a synchronous counter become valid at the same time and the ripple-through effect associated with asynchronous counters is entirely absent. Synchronous counters can be easily designed to count through any arbitrary sequence just as well as the natural sequence 0, 1, 2, 3, ...

We design a synchronous counter by means of a state diagram and the *excitation table* for the appropriate flip-flop (either RS or JK). An excitation table is a version of a flip-flop's truth table arranged to display the input states required to force a given output transition. Table 3.9 illustrates the excitation table of a JK flip-flop. Suppose we wish to force the Q output of a JK flip-flop to make the transition from 0 to 1 the next time it is clocked. Table 3.9 tells us that the J, K input should be 1, d (where d = don't care).

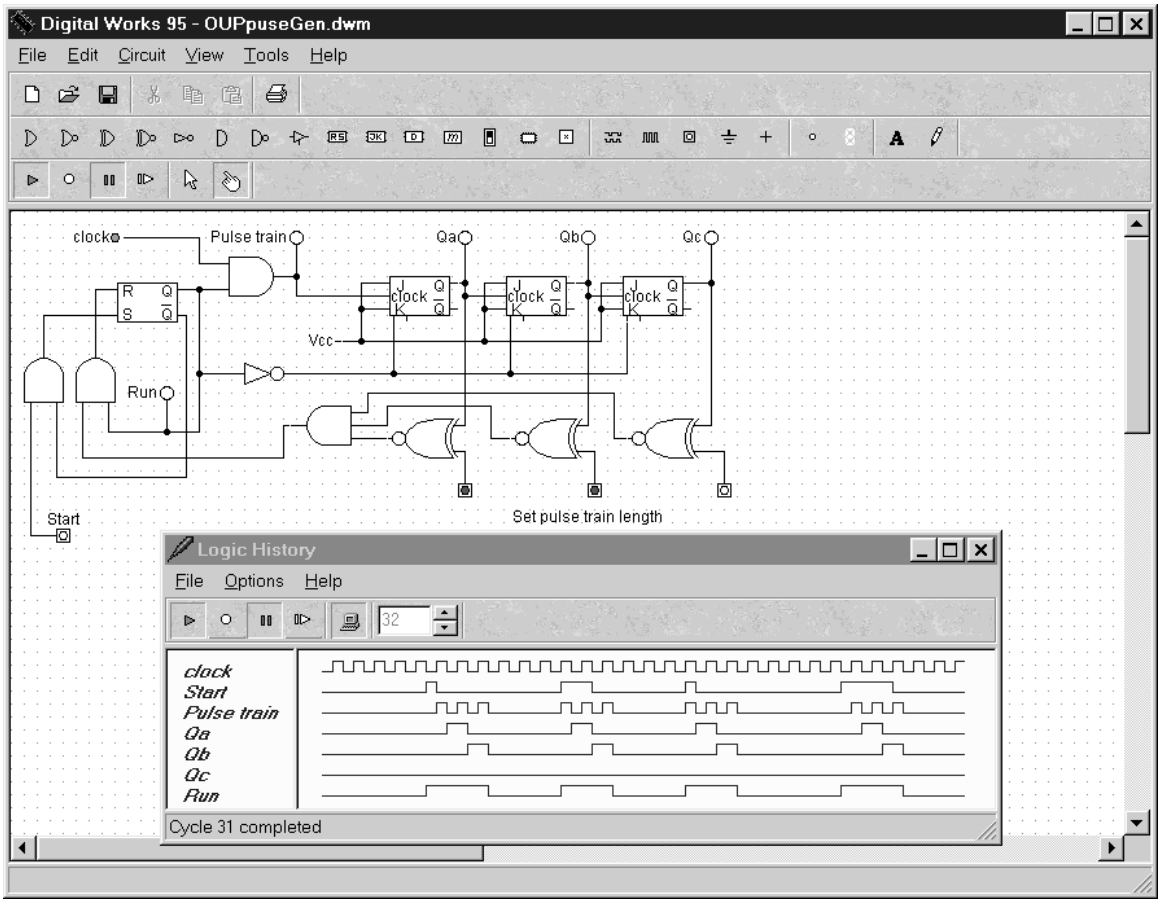


Figure 3.52 Using Digital Works to design a pulse generator.

Inputs		Transition
J	K	$Q \rightarrow Q^+$
0	d	$0 \rightarrow 0$
1	d	$0 \rightarrow 1$
d	1	$1 \rightarrow 0$
d	0	$1 \rightarrow 1$

Table 3.9 Excitation table of a JK flip-flop.

Why is the K input a *don't care condition* when we want a $0 \rightarrow 1$ transition? If we set J = 1 and K = 0, the flip-flop is set when it's clocked and Q^+ becomes 1. If we set J = 1 and K = 1, the flip-flop is toggled when it's clocked and the output Q = 0 is toggled to Q = 1. Clearly, the state of the K input doesn't matter when we wish to set Q^+ to 1 given that Q = 0 and J = 1. It should now be clear why all the transitions in the JK's excitation table have a don't care input—a given state can be reached from more than one starting point.

The next step in designing a synchronous counter is to construct a truth table for the system to determine the JK inputs required to force a transition to the required next state for each of the possible states in the table. It is much easier to explain this step by example rather than by algorithm.

Let's design a synchronous *binary-coded decimal* or modulo-10 counter to count through the natural sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, ... As there are 10 states, we require four JK flip-flops because $2^3 < 10 < 2^4$. Table 3.10 provides a truth table for this counter.

To understand Table 3.10 it's necessary to look along a line and to say, 'Given this state, what must the inputs of the flip-flops be to force the transition to the next state?' For example, in the first line the current state is 0, 0, 0, 0 and the next state is 0, 0, 0, 1. The values for the four pairs of J, K inputs are obtained from the excitation table in Table 3.9. Three of these outputs cause the transition $0 \rightarrow 0$ and one causes the transition $0 \rightarrow 1$. The J, K inputs required are 0, d for the 0 to 0 transitions and 1, d for the 0 to 1 transition.

From the truth table of the synchronous counter we can write down eight Karnaugh maps for the Js and Ks.

Count	Output				Next state				J, K inputs required to force transition							
	Q_d	Q_c	Q_b	Q_a	Q_d	Q_c	Q_b	Q_a	J_d	K_d	J_c	K_c	J_b	K_b	J_a	K_a
0	0	0	0	0	0	0	0	1	0	d	0	d	0	d	1	d
1	0	0	0	1	0	0	1	0	0	d	0	d	1	d	d	1
2	0	0	1	0	0	0	1	1	0	d	0	d	d	0	1	d
3	0	0	1	1	0	1	0	0	0	d	1	d	d	1	d	1
4	0	1	0	0	0	1	0	1	0	d	d	0	0	d	1	d
5	0	1	0	1	0	1	1	0	0	d	d	0	1	d	d	1
6	0	1	1	0	0	1	1	1	0	d	d	0	d	0	1	d
7	0	1	1	1	1	0	0	0	1	d	d	1	d	1	d	1
8	1	0	0	0	1	0	0	1	d	0	0	d	0	d	1	d
9	1	0	0	1	0	0	0	0	d	1	0	d	0	d	d	1
10	1	0	1	0	x	X	x	x	x	x	x	x	x	x	x	x
11	1	0	1	1	x	x	x	x	x	x	x	x	x	x	x	x
12	1	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x
13	1	1	0	1	x	x	x	x	x	x	x	x	x	x	x	x
14	1	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x
15	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x

The *d*s in the table correspond to *don't care* conditions in the excitation table of the JK flip-flop. The x's correspond to don't care conditions due to unused states; for example, the counter never enters states 1010 to 1111. There is, of course, no fundamental difference between x and d. We've chosen different symbols in order to distinguish between the origins of the don't care states.

Table 3.10 Truth table for a synchronous counter.

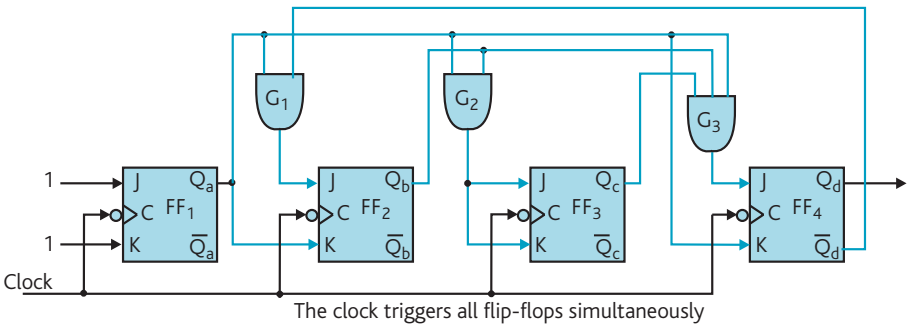


Figure 3.54 Circuit diagram for a 4-bit synchronous BCD counter.

Figure 3.53 gives the Karnaugh maps for this counter. These maps can be simplified to give

$J_d = Q_c \cdot Q_b \cdot Q_a$ $\quad K_d = Q_a$
$J_c = Q_b \cdot Q_a$ $\quad K_c = Q_b \cdot Q_a$
$J_b = \overline{Q}_d \cdot Q_a$ $\quad K_b = Q_a$
$J_a = 1$ $\quad K_a = 1$

We can now write down the circuit diagram of the synchronous counter (Fig. 3.54). Remember that d denotes a don't care condition and indicates that the variable marked by a d may be a 0 or a 1 state. The same technique can be employed to construct a counter that will step through any arbitrary sequence. We will revisit this technique when we look at state machines.

3.7 Introduction to state machines

No discussion of sequential circuits would be complete without at least a mention of state machines. The state machine offers the designer a formal way of specifying, designing, testing, and analyzing sequential systems. Because the detailed study of state machines is beyond the scope of this introductory text, we shall simply introduce some of the basic concepts here.

It would be impossible to find a text on state machines without encountering the general state machines called Mealy machines and Moore machines (after G. H. Mealy and E. Moore). Figure 3.55 illustrates the structure of a Mealy

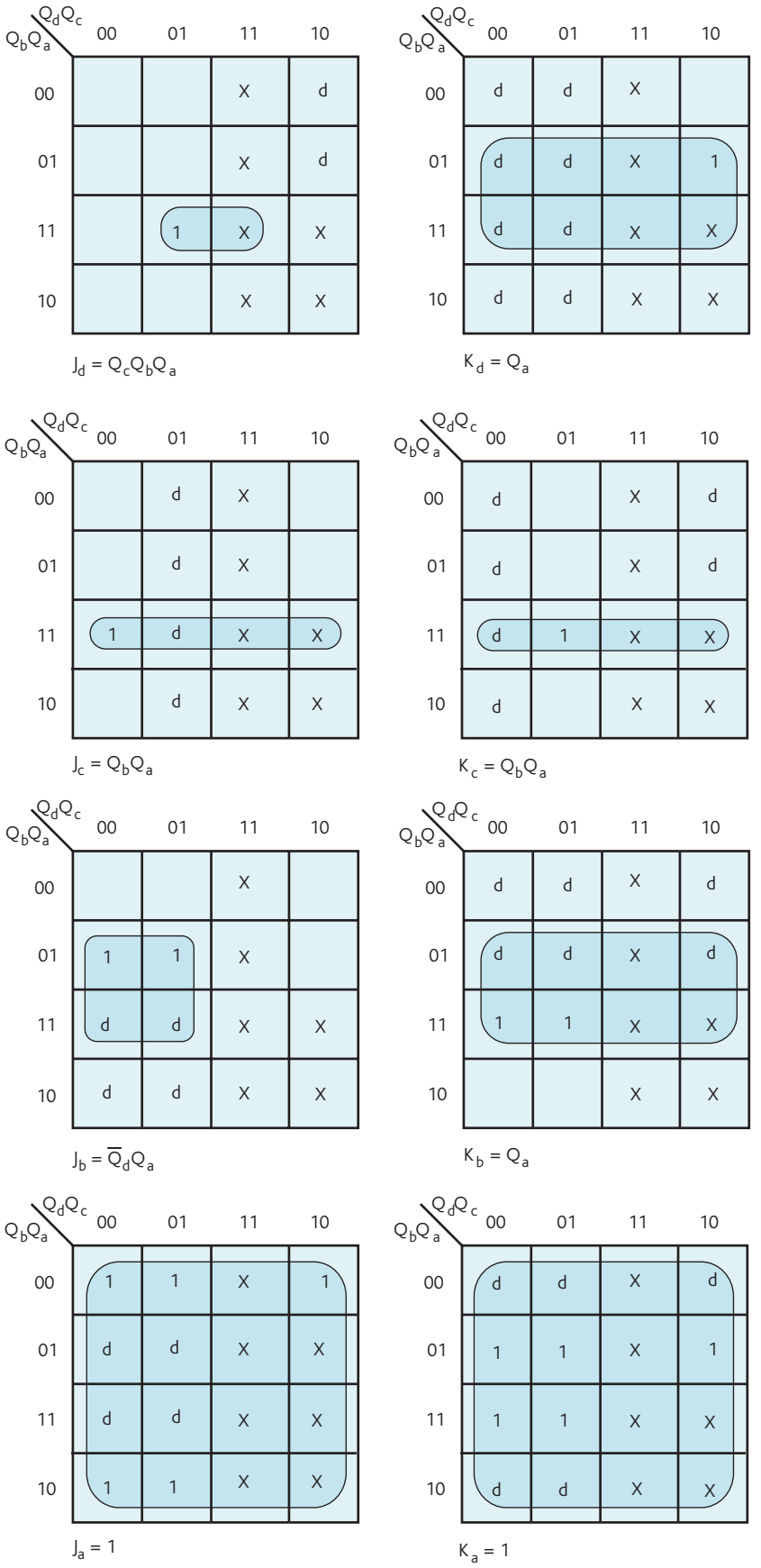


Figure 3.53 Karnaugh maps for a synchronous counter.

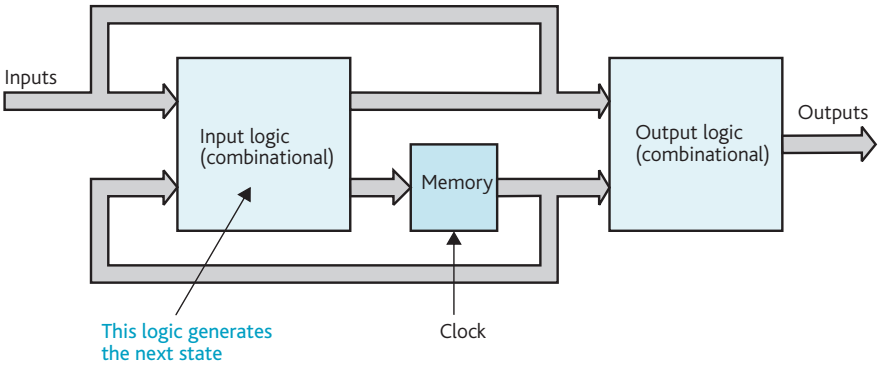


Figure 3.55 The Mealy state machine.

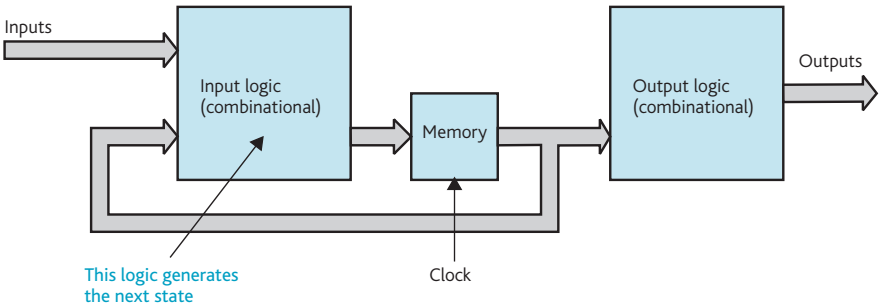


Figure 3.56 The Moore state machine.

state machine and Fig. 3.56 the structure of a Moore state machine. Both machines have a combinational network that operates on the machine's inputs and on its internal states to produce a new internal state. The output of the Mealy machine is a function of the current inputs and the internal state of the machine, whereas the output of a Moore machine is a function of the internal state of the machine only.

3.7.1 Example of a state machine

As we have already said, the state machine approach to the design of sequential circuits is by no means trivial. Here, we will design a simple state machine by means of an example.

Suppose we require a sequence detector that has a serial input X and an output Y. If a certain sequence of bits appears at the input of the detector, the output goes true. Sequence detectors are widely used in digital systems to split a stream of bits into units or *frames* by providing special bit patterns between adjacent frames and then using a sequence detector to identify the start of a frame.

In the following example we design a sequence detector that produces a true output Y whenever it detects the sequence 010 at its X input.

For example, if the input sequence is 00011001101011000101 1,
the output sequence will be 00000000000010000000 10

(the output generates a 1 in the state following the detection of the pattern).

Figure 3.57 shows a black box state machine that detects the sequence 010 in a bit stream. We have provided input and output sequences to demonstrate the machine's action.

We solve the problem by constructing a state diagram as illustrated in Fig. 3.58. Each circle represents a particular state of the system and transitions between states are determined by the current input to the system at the next clock pulse.

A state is marked name/value, where *name* is the label we use to describe the state (e.g. states A, B, C, and D in Fig. 3.58) and *value* is the output corresponding to that state. The transition between states is labeled *a/b*, where *a* is the input condition and *b* the output value after the next clock. For example, the transition from state A to state B is labeled 0/0 and indicates that if the system is in state A and the input is 0, the next clock pulse will force the system into state B and set the output to 0.

Figure 3.59 provides a partial state diagram for this sequence detector with details of the actions that take place during state transitions. State A is the initial state in Fig. 3.59. Suppose we receive an input while in state A. If input X is a 0 we may be on our way to detecting the sequence 010 and therefore we move to state B along the line marked 0/0 (the output is 0 because we have not detected the required sequence yet). If the input is 1, we return to state A because we have not even begun to detect the start of the sequence.

From state B there are two possible transitions. If we detect a 0 we remain in state B because we are still at the start of the desired sequence. If we detect a 1, we move on to state C (we have now detected 01). From state C a further 1 input takes us

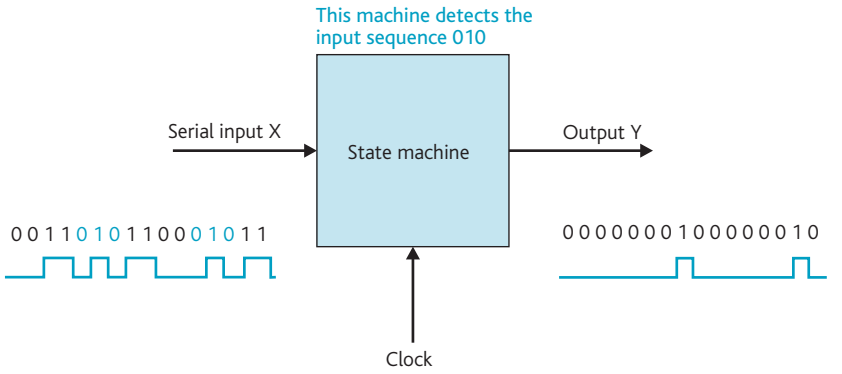


Figure 3.57 State machine to detect the sequence 010.

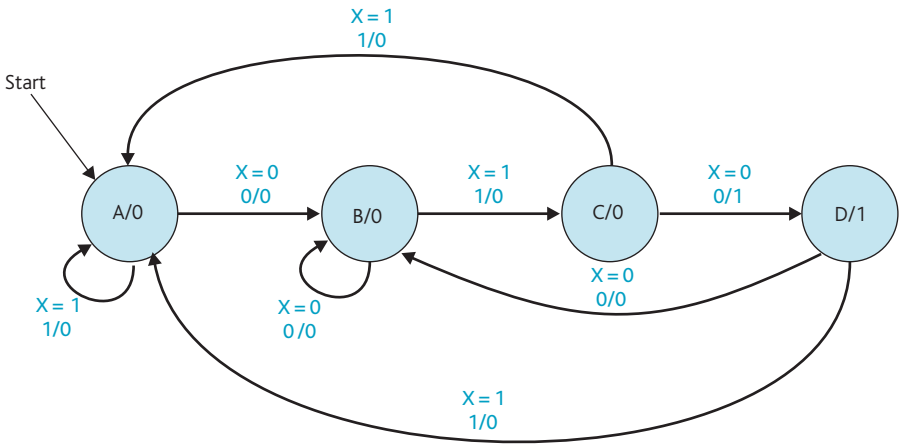


Figure 3.58 State diagram for a 010 sequence detector (X is the current input).

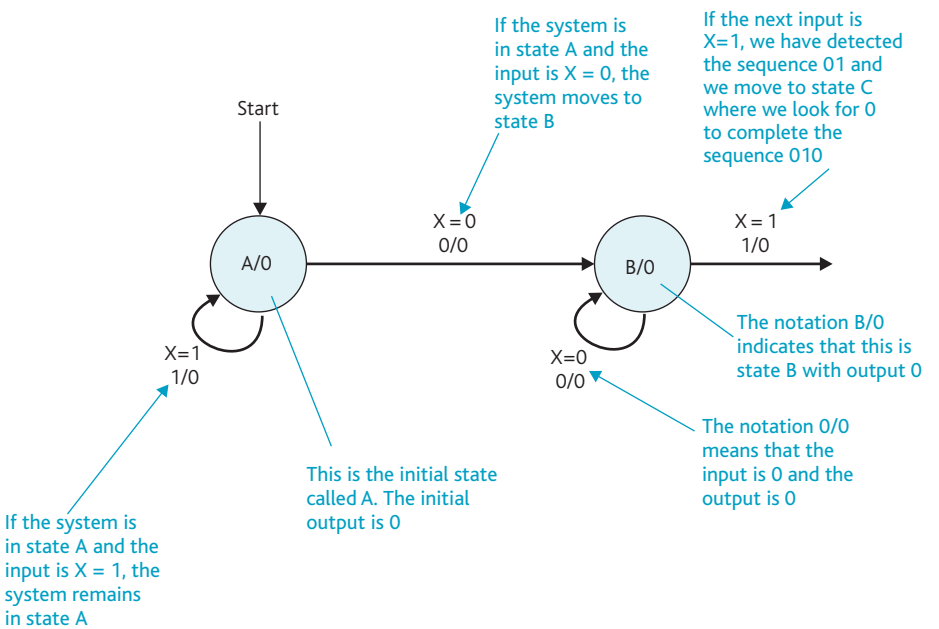


Figure 3.59 Details of the state counter diagram of Fig. 3.58.

Current state	Output	Next state	
		X = 0	X = 1
A	0	B	A
B	0	B	C
C	0	D	A
D	1	B	A

Table 3.11 State table for a 010 sequence detector.

Current state	Flip-flop outputs		Output	Next state	
	Q_1	Q_2		X = 0	X = 1
A	0	0	0	0,1	0,0
B	0	1	0	0,1	1,0
C	1	0	0	1,1	0,0
D	1	1	1	0,1	0,0

Table 3.12 Modified state table for a sequence detector.

Current state			Next state		Output			
Q_1	Q_2	X	Q_1	Q_2	J_1	K_1	J_2	K_2
0	0	0	0	1	0	d	1	d
0	0	1	0	0	0	d	0	d
0	1	0	0	1	0	d	d	0
0	1	1	1	0	1	d	d	1
1	0	0	1	1	d	0	1	d
1	0	1	0	0	d	1	0	d
1	1	0	0	1	d	1	d	0
1	1	1	0	0	d	1	d	1

$J_1 = Q_1 + Q_2 \cdot X$
$J_2 = \overline{X}$
$K_1 = Q_2 + X$
$K_2 = X$

Table 3.13 Determining the JK outputs of the sequence detector.

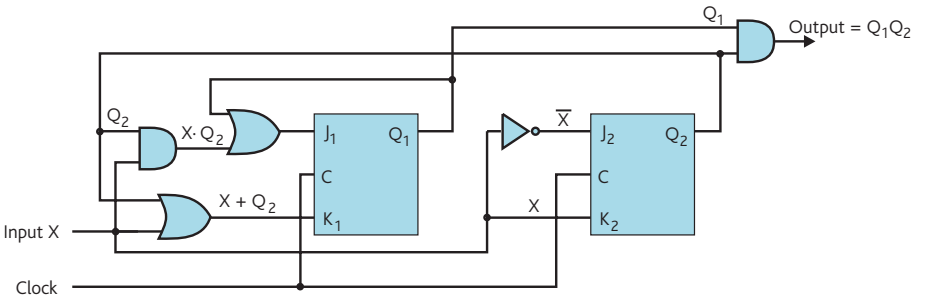


Figure 3.60 Circuit to detect the sequence 010.

right back to state A (because we have received 011). However, if we detect a 0 we move to state D and set the output to 1 to indicate that the sequence has been detected. From state D we move back to state A if the next input is a 1 and back to state B if it is a 0. From the state diagram we can construct a state table that defines the output and the next state corresponding to each current state and input. Table 3.11 provides a state table for Fig. 3.58.

3.7.2 Constructing a circuit to implement the state table

The next step is to go about constructing the circuit itself. If a system can exist in one of several *states*, what then defines the *current* state? In a sequential system flip-flops are used to hold state information—in this example there are four states, which requires two flip-flops.

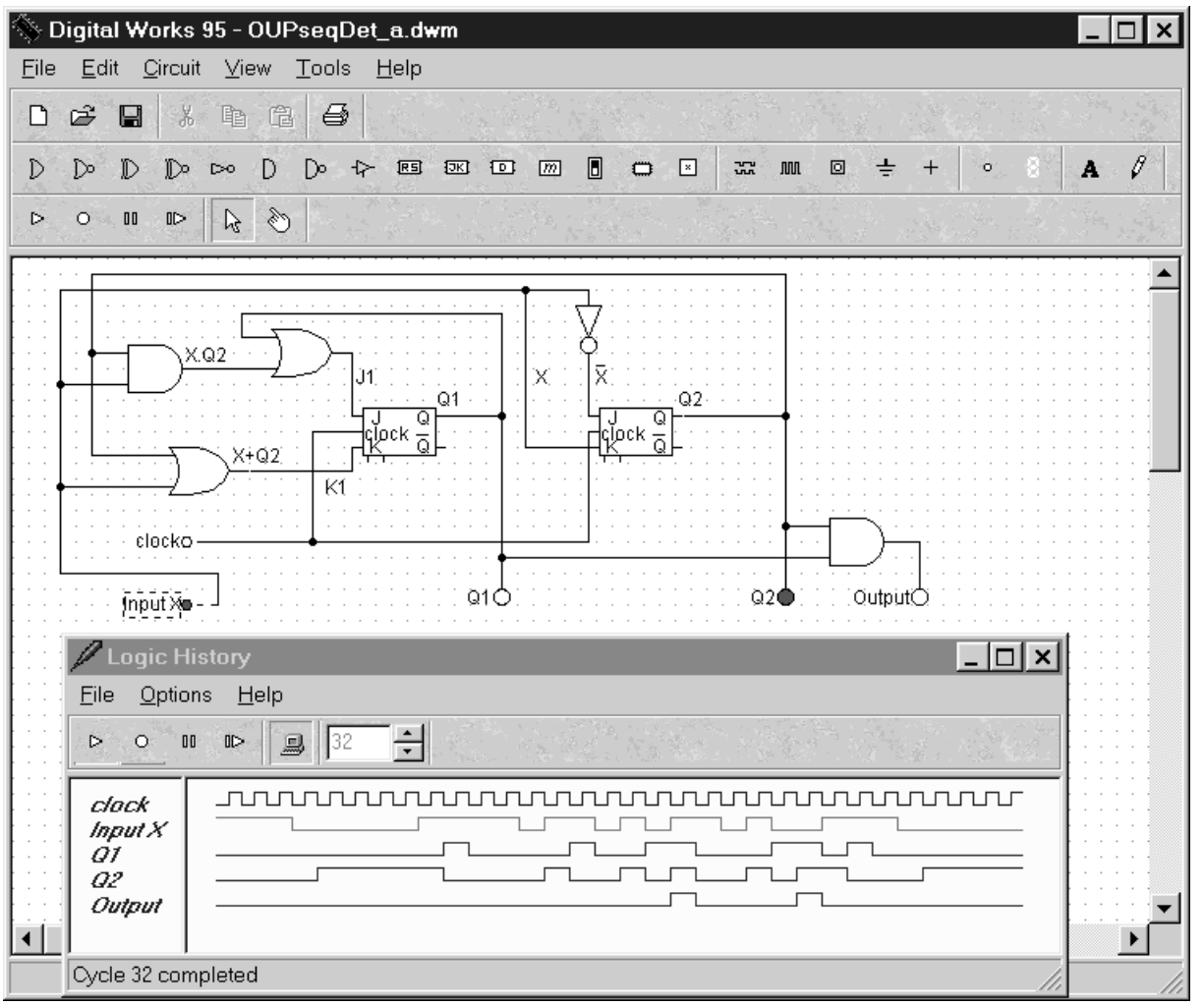


Figure 3.61 Using Digital Works to implement the sequence detector.

Table 3.12 expands Table 3.11 to represent internal states A to D by flip-flop outputs Q_1, Q_2 = 0, 0 to 1, 1. We next construct Table 3.13 to determine the JK input of each JK flip-flop that will force the appropriate state transition, given the next input X. Table 3.13 is derived by using the excitation table of the JK flip-flop (see Table 3.9). The final step is to create a circuit diagram from Table 3.13 (i.e. Fig. 3.60).

Figure 3.61 demonstrates the construction of the sequence detector in Digital Works. We've added LEDs to show the state of the flip-flop outputs and control signals and have provided an example of a run. Note the output pulse after the sequence 010. We used the programmable sequence generator to provide a binary pattern for the test.

SUMMARY

In this chapter we've looked at the flip-flop, which provides data storage facilities in a computer and which can be used to create counters and shift registers as well as more general forms of state machine. We have introduced the RS, D, and JK flip-flops. All these flip-flops can capture data and the JK flip-flop is able to operate in a *toggle* mode in which its output changes state each time it is clocked. Any of these flip-flops can be converted into the other two flip-flops by the addition of a few gates.

We have also introduced the idea of clocking or triggering flip-flops. A flip-flop can be triggered by a clock at a given level or by the change in state of a clock. The master–slave flip-flop latches data at its input when the clock is high (or low) and transfers data to the output (slave) when the clock changes state.

We have looked at the counter and shift register. The counter counts through a predetermined sequence such as the natural integers 0, 1, 2, 3, ... A shift register holds a word of data and is able to shift the bits one or more places left or right. Shift registers are used to divide and multiply by two and to manipulate data in both arithmetic and logical operations. Counters and shift registers can be combined with the type of

combinational logic we introduced in the previous chapter to create a digital computer.

Sequential machines fall into two categories. Asynchronous sequential machines don't have a master clock and the output from one flip-flop triggers the flip-flop it's connected to. In a synchronous sequential machine all the flip-flops are triggered at the same time by means of a common master clock. Synchronous machines are more reliable. In this chapter we have briefly demonstrated how you can construct a synchronous counter and a machine that can detect a specific binary pattern in a stream of serial data.

PROBLEMS

3.1 What is a *sequential circuit* and in what way does it differ from a combinational circuit?

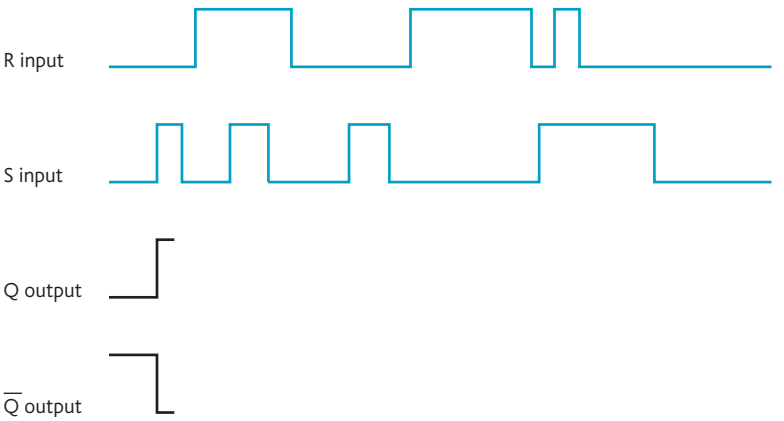


Figure 3.62 R and S inputs to an RS flip-flop.

3.2 Explain why it is necessary to employ *clocked* flip-flops in sequential circuits (as opposed to unclocked flip-flops)?

3.3 What are the three basic flip-flop clocking modes and why is it necessary to provide so many clocking modes?

3.4 The behavior of an RS flip-flop is not clearly defined when R = 1 and S = 1. Design an RS flip-flop that does not suffer from this restriction. (Note: What assumptions do you have to make?)

3.5 For the waveforms in Fig. 3.62 draw the Q and $\overline{Q}$ outputs of an RS flip-flop constructed from two NOR gates (as in Fig. 3.2).

3.6 For the input and clock signals of Fig. 3.63, provide a timing diagram for the Q output of a D flip-flop. Assume that the flip-flop is

(a) Level sensitive
(b) positive edge triggered
(c) negative-edge triggered
(d) a master–slave flip-flop

3.7 What additional logic is required to convert a JK flip-flop into a D flip-flop?

3.8 Assuming that the initial state of the circuit of Fig. 3.64 is given by C = 1, D = 1, P = 1, and Q = 0, complete the table. This question should be attempted by calculating the effect of the new C and D on the inputs to both cross-coupled pairs of NOR gates and therefore on the outputs P and Q. As P and Q are also inputs to the NOR gates, the change in P and Q should be taken into account when calculating the effect of the next inputs C and D. Remember that the output of a NOR is 1 if both its inputs are 0, and is 0 otherwise.

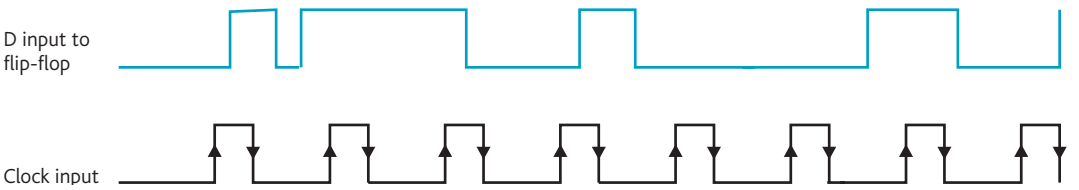


Figure 3.63 Timing diagram of a clock and data signal.

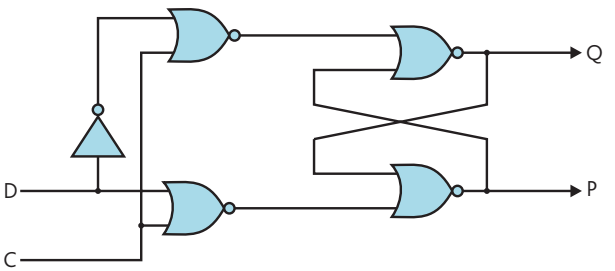


Figure 3.64 Circuit for Question 3.8.

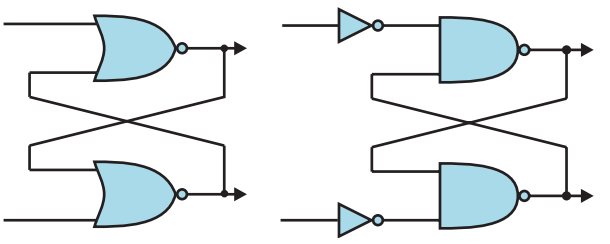


Figure 3.65 Circuit for Question 3.9.

C	D	P	Q
1	1	1	0
1	0		
0	0		
1	1		
0	1		
1	1		
0	1		
0	0		
1	0		

Modify the circuit to provide a new input S which, when 1, will at any time set P to 1 and Q to 0. Provide another input R that will similarly set P to 0 and Q to 1. Note that R and S cannot both be a 1 at the same time and therefore the condition R = S = 1 need not be considered.

3.9 Demonstrate that the flip-flops in Fig. 3.65 are equivalent. Are they exactly equivalent?

3.10 Many flip-flops have unconditional *preset* and *clear* inputs. What do these inputs do and why are they needed in sequential circuits?

3.11 A T flip-flop has a single clock input and outputs Q and $\overline{Q}$. Its Q output toggles (changes state) each time it is clocked. The T flip-flop behaves exactly like a JK flip-flop with its J and K inputs connected permanently to a logical one. Design a T flip-flop using a D flip-flop.

3.12 Why haven't D and RS flip-flops been replaced by the JK flip-flop, because the JK flip-flop can, apparently, do everything a D flip-flop or an RS flip-flop can do?

3.13 What is a *shift register* and why is it so important in digital systems?

3.14 Design a shift register that has two inputs, a clock input and a shift input. Whenever this register receives a pulse at its shift input, it shifts its contents *two* places right.

3.15 Analyze the operation of the circuit of Fig. 3.66 by constructing a timing diagram (assume that Q_0 and Q_1 are initially 0). Construct the circuit using Digital Works and observe its behavior.

3.16 Analyze the operation of the circuit of Fig. 3.67 by constructing a timing diagram (assume any initial value for Q_0 to Q_3). Construct the circuit using Digital Works and observe its behavior. This type of circuit is an important circuit in digital systems because it can be used to generate a pseudo random sequence; that is, the sequence of bits at its Q_0 output look (to an

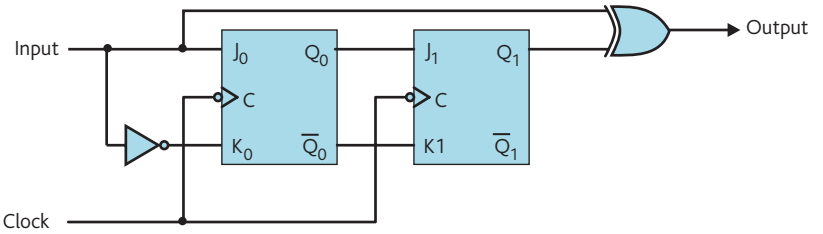


Figure 3.66 Circuit diagram for Question 3.15.

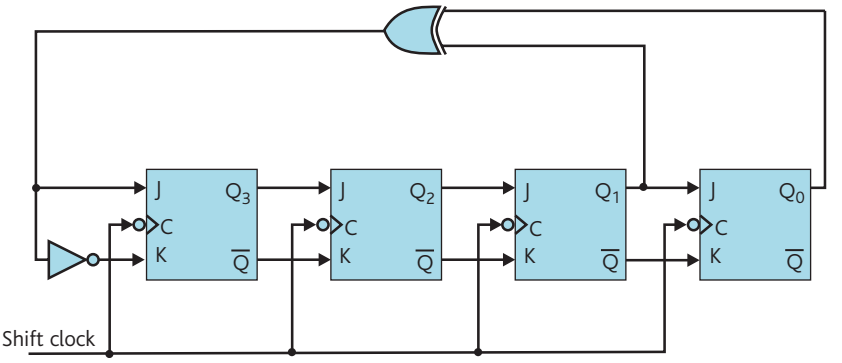


Figure 3.67 Circuit diagram for Question 3.16.

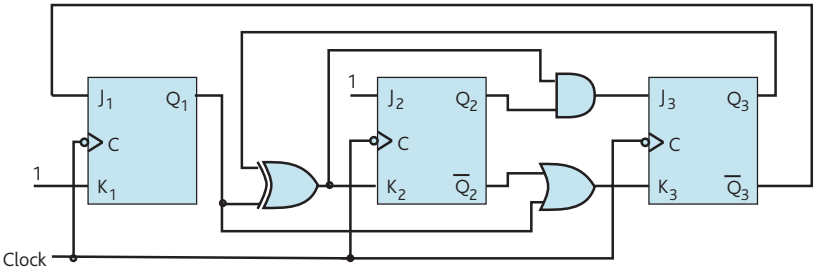


Figure 3.68 Circuit diagram for Question 3.17.

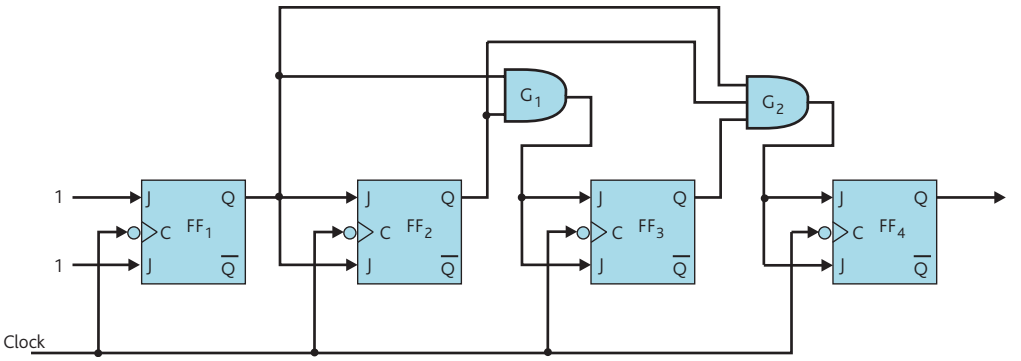


Figure 3.69 Circuit diagram for Question 3.18.

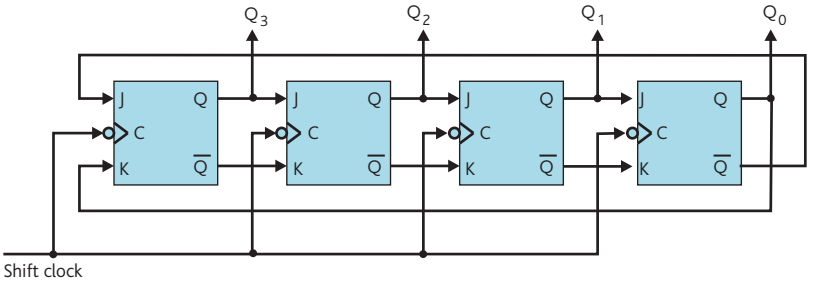


Figure 3.70 Circuit diagram of a Johnson counter.

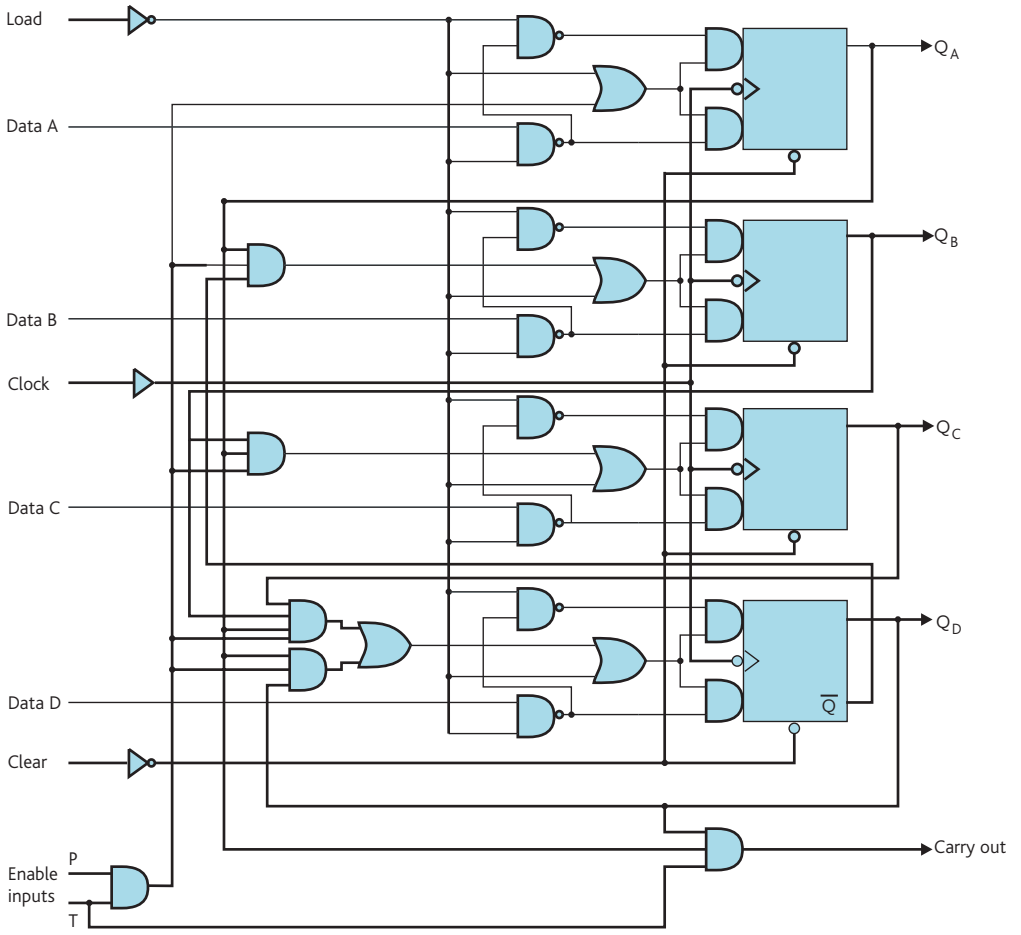


Figure 3.71 Organization of a 74162 synchronous decade counter.

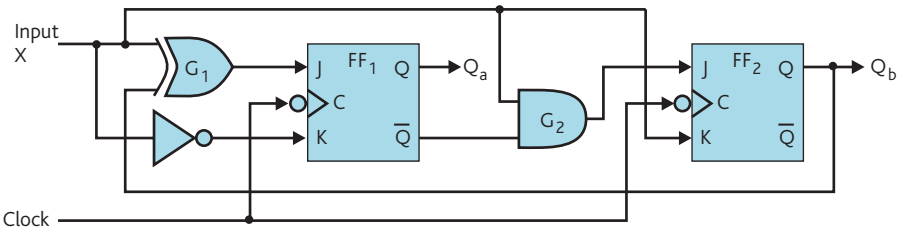


Figure 3.72 Circuit diagram of a sequence processor.

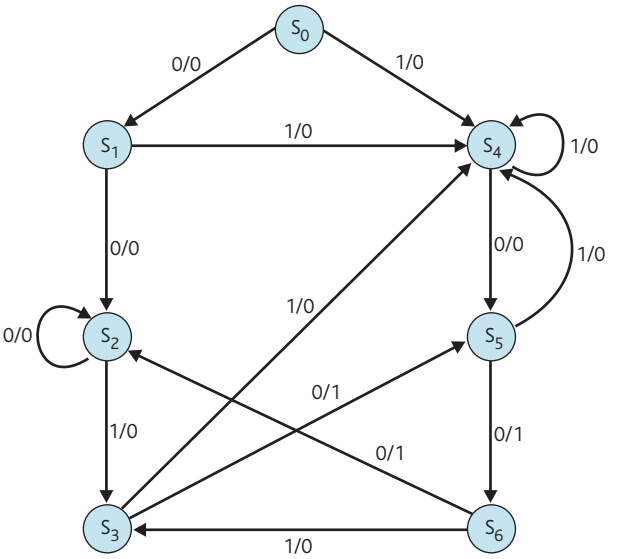


Figure 3.73 Circuit diagram of a sequence processor.

observer) as if they constitute a random series of 1s and 0s. Longer sequences of random numbers are generated by increasing the number of stages in the shift register. The input is the exclusive OR of two or more outputs.

3.17 Use Digital Works to construct the circuit of Fig. 3.68 and then investigate its behavior.

3.18 Investigate the behavior of the circuit in Fig. 3.69.

3.19 Explain the meaning of the terms *asynchronous* and *synchronous* in the context of sequential logic systems. What is the significance of these terms?

3.20 Design an asynchronous base 13 counter that counts through the natural binary sequence from 0 (0000) to 12 (1100) and then returns to zero on the next count.

3.21 Design a synchronous binary duodecimal (i.e. base 12) counter that counts through the natural binary sequence from 0 (0000) to 11 (1011) and then returns to zero on the next count. The counter is to be built from four JK flip-flops.

3.22 Design a synchronous modulo 9 counter using

(a) JK flip-flops
(b) RS flip-flops (with a master–slave clock).

3.23 Design a *programmable* modulo 10/modulo 12 synchronous counter using JK flip-flops. The counter has a control input, TEN/$\overline{\text{TWELVE}}$, which when high, causes the counter to count modulo 10. When low, TEN/$\overline{\text{TWELVE}}$ causes the counter to count modulo 12.

3.24 How would you determine the maximum rate at which a synchronous counter could be clocked?

3.25 The circuit in Fig. 3.70 represents a *Johnson counter*. This is also called a *twisted ring* counter because feedback from the last (rightmost) stage is fed back to the first stage by crossing over the Q and $\overline{\text{Q}}$ connections. Investigate the operation of this circuit.

3.26 Design a simple digital time of day clock that can display the time from 00:00:00 to 23:59:59. Assume that you have a clock pulse input derived from the public electricity supply of 50 Hz (Europe) or 60 Hz (USA).

3.27 Figure 3.71 gives the internal organization of a 74162 synchronous decade (i.e. modulo 10) counter. Investigate its operation. Explain the function of the various control inputs. Note that the flip-flops are master–slave JKs with asynchronous (i.e. unconditional) clear inputs.

3.28 Design a modulo 8 counter with a clock and a control input UP. When UP = 1, the counter counts 0, 1, 2, . . . , 7. When UP = 0, the counter counts down 7, 6, 5, . . . 0. This circuit is a programmable up-/down-counter.

3.29 Design a counter using JK flip-flops to count through the following sequence.

Q_2	Q_1	Q_0	
0	0	1	
0	1	0	
0	1	1	
1	1	0	
1	1	1	
0	0	1	sequence repeats

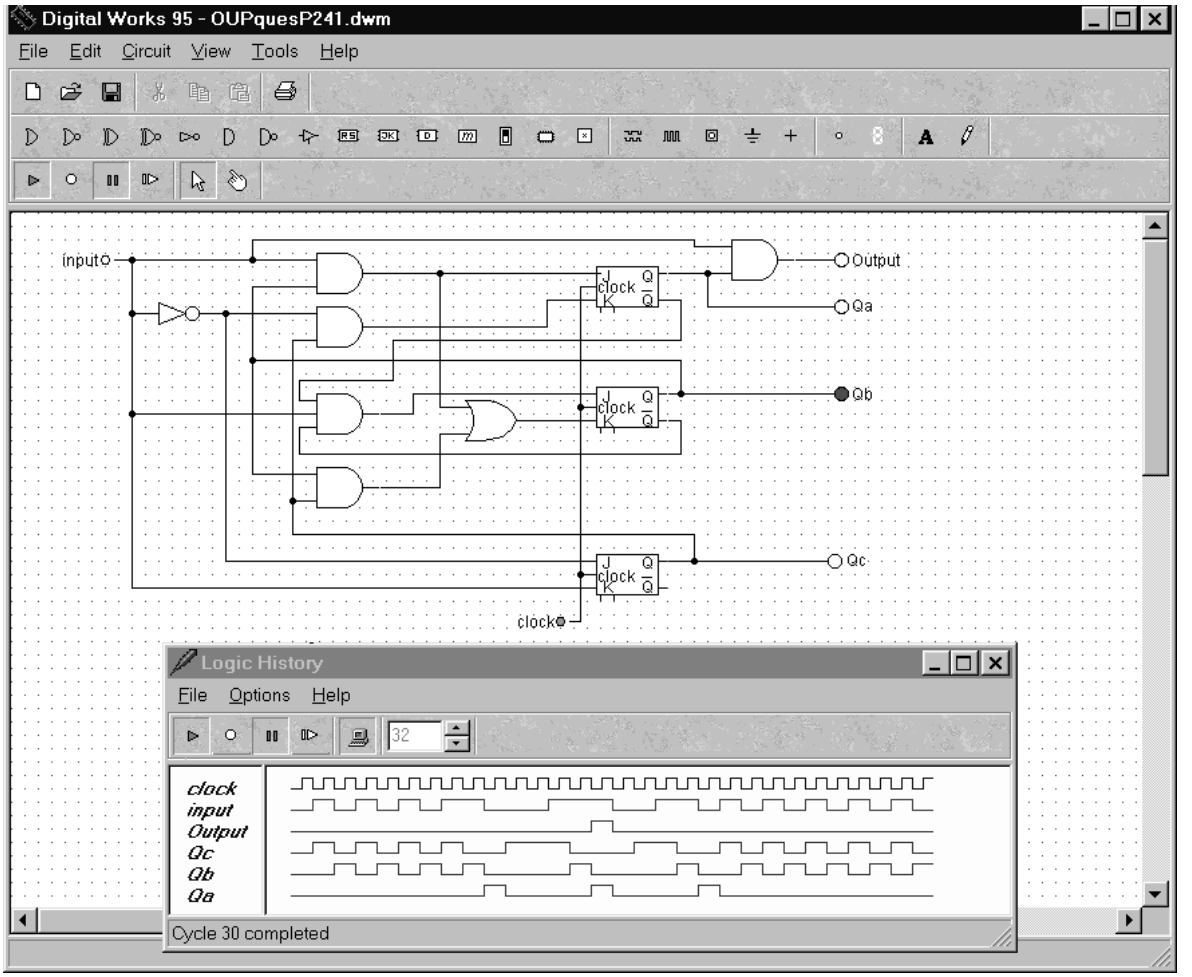


Figure 3.74 A sequential circuit constructed with Digital Works.

3.30 Investigate the action of the circuit in Fig. 3.72 when it is presented with the input sequence 111000001011111, where the first bit is the rightmost bit. Assume that all flip-flops are reset to Q = 0 before the first bit is received.

3.31 Design a state machine to implement the state diagram defined in Fig. 3.73.

3.32 Figure 3.74 provides a screen shot of a session using Digital Works. Examine the behavior of the circuit both by constructing it and by analyzing it.