Chapter 5

Building Blocks With Memory

In the preceding chapters, we tacitly assumed that electronic devices are infinitely fast and that they generate outputs that depend only on the present input values. In this chapter, we explore the interesting consequences of violating these assumptions.

First, we examine what can happen when there are finite propagation delays within gates. Output signals from assemblies of gates sometimes have spurious short pulses that are not predicted by standard Boolean algebra. These spurious pulses are seldom useful, but we must contend with them, usually by waiting until they have gone away.

Next, we explore gate circuits that include feedback. Some of these circuits exhibit *memory*, which is an essential tool for the system designer. We consider useful sequential (memory) building blocks: flip-flops, registers, counters, and so on. These are basic tools for developing the digital architectures in the coming chapters of Part II.

We then discuss large memory arrays—RAMs, ROMs, and allied solid-state memories. Then, after a treatment of programmable logic devices, we conclude with a description of some timing devices that are needed when designing with these large memories.

5.1 The Time Aspect

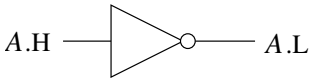
5.1.1 Hazards

The outputs of real gates cannot change instantaneously when an input is changes. Integrated circuits operate by the movement of holes and electrons within some physical material, usually silicon. Not even very light particles such a electrons can move at infinite speeds, and their movement will always involve delays. The time between a change in an input signal and a correspond-

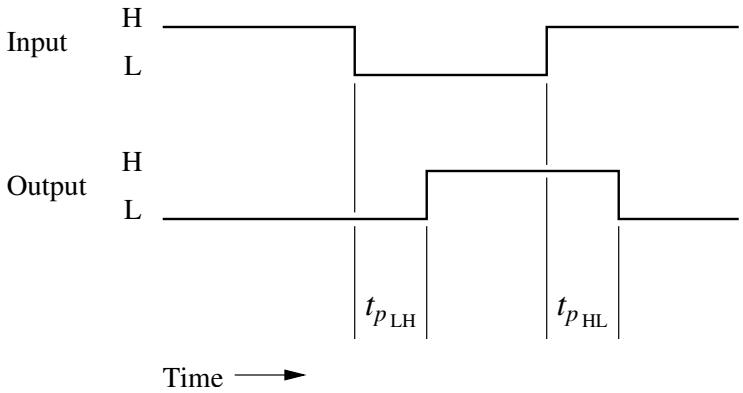
ing change in an output is called the *propagation delay* of the circuit. When inputs change, an output may undergo a change from L to H of from H to L. The corresponding propagation delays are denoted $t_{p\mathrm{LH}}$ and $t_{p\mathrm{HL}}$. Propagation delays depend on the input waveforms, temperatures, output loadings, operating power, logic family and a host of other parameters. Single-gate propagation delays are about 5 nanoseconds in TTL low-power Schottky devices.

Another source of delay is the wire carrying signals between gates. Electricity in a wire can travel only about 8 inches in a nanosecond, so when wires become long, the *interconnection delays* may become serious.

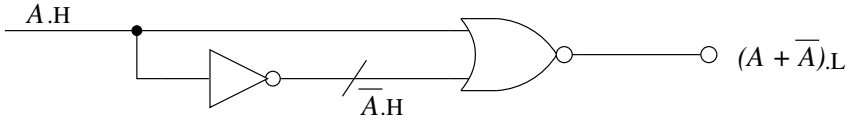
Our purpose here is to show how these delays can create spurious outputs called *hazards*. Consider the following simple circuit that changes the voltage polarity of a signal:



Assume that the voltage at the input A has been stable for a long time. The output will also be stable and the opposite voltage level. If the voltage at the input changes, the output will change a short time later. When an input changes from L to H, the output will change from H to L after a propagation delay $t_{p\mathrm{HL}}$. Figure 4–1 is a *timing diagram*, a graph of input and output values (either voltage or logic) as a function of time. Each variable's graph is called a .



To see what can happen when we introduce time into Boolean algebra, consider the following circuit, whose output is $A + \overline{A}$:



Of course we know that $A + \overline{A} = T$ regardless of the logic value of A, and we predict, from Boolean algebra, that the output of the circuit will always be

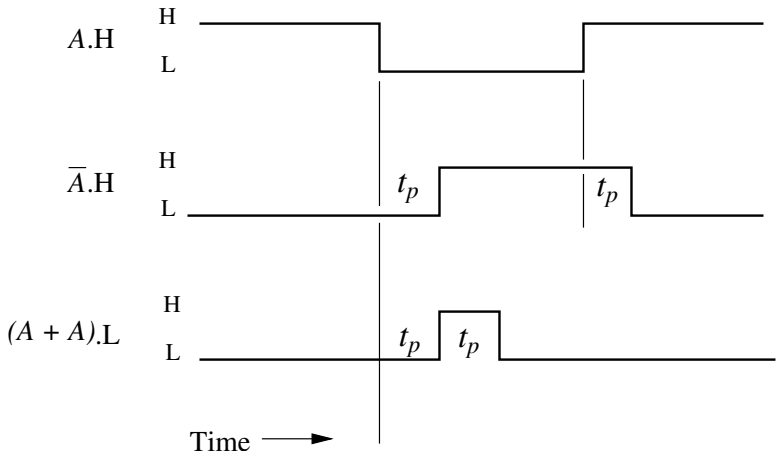
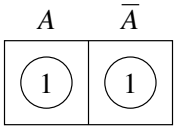


Figure 5.1: A hazard caused by propagation delay in an inverter

L. B:ut assume that each circuit element has a propagation delay t_p for any transition. IF A changes from T to F, the voltage pattern in Fig. 5.1 will prevail; there is a spurious high-voltage (F) output that lasts for one gate delay.

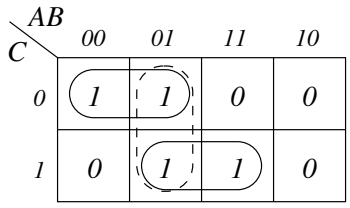
These spurious outputs of combinational circuits, called *hazards* or *glitches*, are common in digital systems. Fortunately, given sufficient time they will die out and the outputs of gates will assume the values predicted by classical Boolean algebra.

Occasionally, it is necessary to generate gate outputs that are *clean*—that have no hazards. It can be shown that a function*may* have a hazard if the function's Karnaugh map has adjacent 1's not enclosed in the same circle. The preceding example, when plotted on a one-variable K-map becomes



The two adjacent 1's do not share a common circle, and indeed the circuit has a hazard. If we circle both 1's in the K-map, we have the TRUE function, which is hazard-free.

The following function is a more complex example.



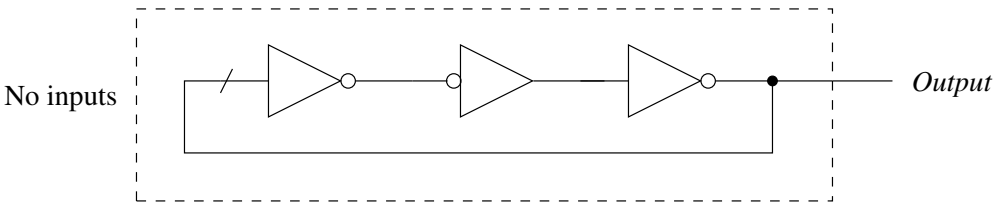
The theory is that a circuit based on the two solid loops may or may not contain a hazard; however, if we build a circuit that includes the dashed loop, we can be sure that the circuit will have no hazards. Using the dashed loop requires extra hardware (additional AND and OR gates), a necessary penalty when we cannot tolerate hazards.

This technique of eliminating hazards works in simple sum-of-products circuits derived from K-maps. In more general circuits, the elimination of hazards is quite complex, and therefore we must use finesse instead of brute force. Rather than use design techniques that require hazard-free signals, we will make our designs *insensitive* to the hazards that occur when combinational inputs are changing. A standard technique is to wait a fixed time after the inputs of the gates change, during which time the hazards will die out. We may then proceed to use the stable signals. This idea is the basis of synchronous (clocked) design, which we introduce in Chapter 5.

5.1.2 Circuits with Feedback

In the preceding section, we discussed purely combinational circuits. Except for momentary hazards, the behavior of the circuits is adequately describe by the Boolean algebraic or truth-table methods used in previous chapters. After a sufficient time to "settle," the circuit's outputs become a function only of the inputs. We now consider another class of circuits, in which the value of the outputs ofter the settling time depends not only on the external inputs but also on the original value of the outputs. Such circuits exhibit *feedback*; the output feeds back to contribute to the inputs of earlier elements of the circuit.

Feedback yields curious results in some circuits. The following circuit, which has no external inputs, consists of three inverters and feedback:



The voltage a the output is fed back to the input where, after a short time, it appears inverted on the output. Then new voltage causes a similar inversion: the output voltage *oscillates* rapidly.

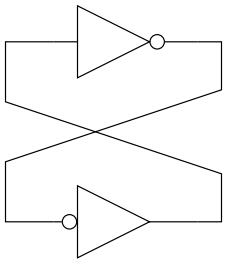
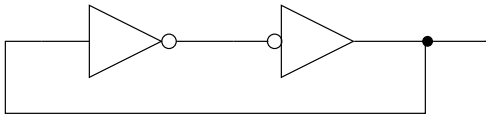


Figure 5.2: Memory displayed by a circuit with feedback.

Remove on inverter from this circuit, producing the following circuit:



If you construct this circuit with real inverters and apply operating power, the output voltages of each inverter will go through a period of instability, during which one output will settle at a high level and the other at a low level. Although there is no way to predict which output will be high and which low, the circuit will remain stable after the settling time. You can verify the stability by tracing voltages around the circuit. Redrawing the circuit, as in Fig. 5.2, helps to illustrate the stability. Since neither of the inverter feedback circuits shown above has external inputs, Boolean algebra is powerless to descrive the circuit's behavior

5.2 Sequential Circuits

The circuit in Fig. 5.2 exhibits a primitive form of memory: the circuit "remembers" the resolution of the initial voltage conflict. Without external inputs, this memory is useless. In contrast, certain feedback circuits with external inputs not only exhibit memory, but also allow the designer to control the value stored in the memory. Controllable memory is the digital designer's most powerful tool. Digital systems with memory are called *sequential circuits*.

Sequential devices may be synthesized from gates, but this procedure is not within the scope of this book, except that it show the typical structure of some simple memory elements. Manufacturers have packaged proven gate designs of various sequential circuits, and we can use these a building blocks once we know their behavior. Sequential building blocks have names such as *latch*, *flip-flop*, and *register*.

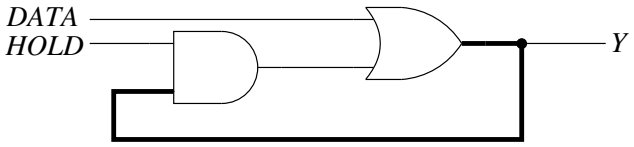


Figure 5.3: A latch circuit; the heavy line is the feedback path.

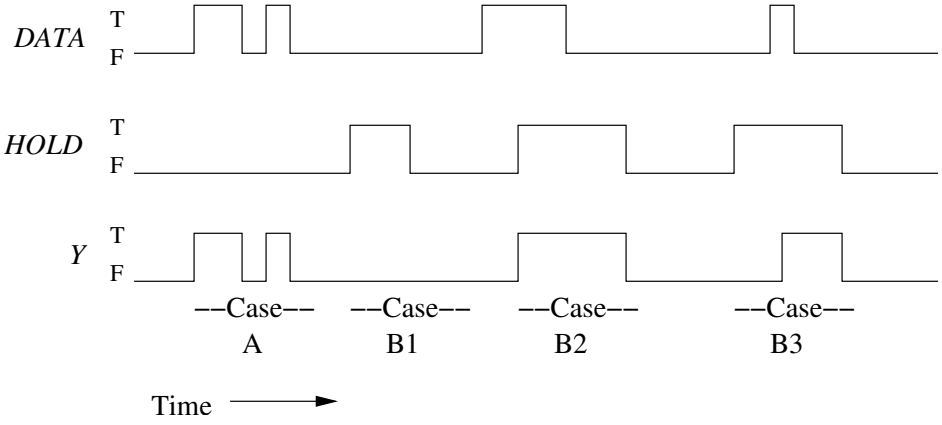


Figure 5.4: A timing diagram for a latch. Note the 1's catching behavior.

5.2.1 Unclocked Sequential Circuits

The latch . The latch is the simples data storage element. Its logic diagram is in Fig 5.3. To describe the action of the latch, we must introduce time as a parameter. This was not necessary in combinational logic, but it is always necessary in sequential logic. The timing diagram is frequently used to portray sequential circuit behavior. To analyze the latch circuit, consider the several cases shown in the timing diagram, Fig 5.4.

Case A: HOLD = F. In this case, Y = DATA.

Case B: HOLD = T. Any occurrence of DATA = T will be captured, and the output will thereafter remain true until HOLD becomes false. We consider three sub-cases:

Case B1: DATA is false throughout the period when HOLD is true. Then Y is false.

Case B2: DATA is true when HOLD is true. When HOLD becomes true, the latch captures the (true) value of DATA and stores it as

longs as HOLD remains true. (After HOLD becomes false, case A applies.)

Case B3: DATA is false when HOLD becomes true. At the beginning, Y is false. The first occurrence of a true signal on the DATA line will cause Y to become true; the output will *remain true* until HOLD becomes false.

The latch has the property of passing true input data to its output immediately. This behavior is sometimes useful in digital design, but it can be quite dangerous. Suppose that while HOLD is true, a glitch or noise pulse on the DATA line causes DATA to become true momentarily. This momentary true, or 1, will cause output Y to become true and remain true as long a HOLD is true. This behavior is sometimes called *1's catching*; it is useful only rarely.

The latch circuit in Fig. 5.4 is not frequently used, and it is not generally available as an SSI integrated circuit. A true latch is a memory element that exhibits combinational behavior at some values of its inputs. There are other varieties of latch; unfortunately, designers use the term loosely to describe various signal-capturing events. We will soon develop more satisfactory memory devices.

Timing diagrams may be used to show gross voltage or logic behavior, or to show fine detail. The timing diagrams in Figs **??** and **??** sow the fine detail of gate delays. On the other hand, the timing diagram in Fig. 5.4 shows only the gross behavior of the latch circuit and is accurate only with the time scale is sufficiently large. On a fine time scale, the output Y in Fig 5.4 would be shifted slightly to the right to account for the delays incurred while changes in DATA or HOLD are absorbed by the gates in the circuit.

The asynchronous RS flip-flop. The feedback circuit in Fig. **??** exhibits a peculiar form of memory; it remembers which inverter had a low output after "power-up." The circuit has two stable states, and is indeed a memory, albeit a useless one, since there is no way to change it from one state to the other. By changing the inverters to two-input NOR gates, we obtain a useful device known as the *asynchronous RS flip-flop* (See Fig 5.5).

The RS flip-flop is a *bistable* device, which means that in the absence of any inputs it can assume either of two stable states. To see this, assume thatn $R = S =$ L, and assume that the output X of gate 1 is L. Gate 2 will then present a high voltage level to Q. When this H feeds back to the input of gate 1 it will produce an L a X, which is consistent with out original assumption about the polarity of X. We can describe this behavior by saying that the the circuit is in a *stable state*, it will remain there as long as there are no changes in the R and S inputs.

There is another stable state during which gate 1 outputs H and gate 2 outputs L. We could predict this from the symmetry of the circuit, but you should verify it by tracing signals as we just did.

We have shown that the circuit of two cross-coupled NOR gates can exist in two stable states. We call one of the stable states the *set state* and the other

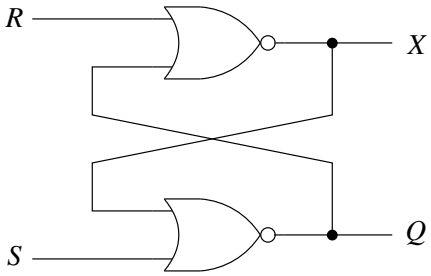
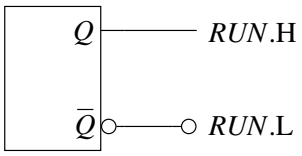


Figure 5.5: A asynchronous RS flip-flop constructed with NOR gates.

the *reset state*. By convention, the set state corresponds to Q = H, and the reset state to Q = L.

The conventional representation of a flip-flop is a rectangle from which Q.H emerges as at the upper right side. Most flip-flops produce two voltages of opposite polarity and the second output appears below the O.H output. In data books, the second output is usually called $\overline{Q}$. Since this output behaves like Q with a voltage inversion, mixed logicians prefer to designate the signal as Q.L, the alternative voltage form of Q.H. Nevertheless, the nomenclature within the flip-flop symbol, like our other building blocks, must conform to normal data-book usage so that there will be no confusion about the interpretation of the pins of the chip. The interior of the symbol serves to identify pin functions; the extern notations for inputs and outputs represent specific signals in a logic design. Thus, if we have a flip-flop whose output is a logic variable RUN, our standard notation for the output is



Now we will consider the S and R inputs to the RS flip-flop. We know that as long as S and R are low, the flip-flop remains in its present state. We may use the S and R lines to force the flip-flop into either state. S is a control input that places the RS flip-flop into the set state (Q = H) whenever S = H. Analogously, R = H resets the flip-flop by making Q = L. The obvious association of truth and volage is T = H at S, R, and Q, so that we set the flip-flop by making $S = T$, and we reset by making $R = T$. This leads us to our usual mixed logic notation for an RS flip-flop constructed of NOR gates:

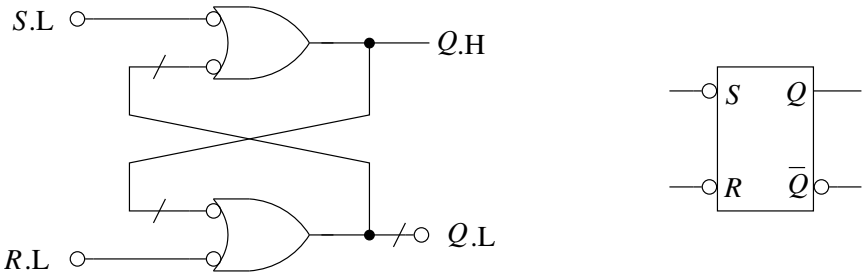


Figure 5.6: A asynchronous RS flip-flop constructed with NAND gates.

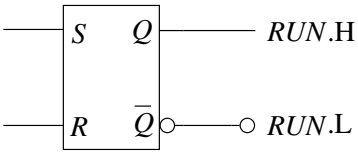


Figure 5.6 is a similar asynchronous RS flip-flop designed with NAND gates. This figure, a mixed-logic diagram of the cross-couple gates, emphasizes that $\mathsf{T} = \mathrm{L}$ at the input of this flip-flop.

The term *asynchronous* associated with the RS flip-flop implies that there is no master clocking signal that governs the activity of the flip-flop; suitable changes of S or R cause the outputs to react immediately. Asynchronous means *unclocked*. Its counterpart is a *clocked*, or *synchronous*, circuit. (workers refer to all the unclocked storage elements as latches; we will not adopt this practice.) The asynchronous RS flip-flop is sensitive to noise, or glitches, at the S input when in the reset state, and at the R input when in the set state. This sensitivity is occasionally useful, but in general you should avoid using asynchronous devices, since glitches are undesirable byproducts of gate delays and noise is usually unpredictable in digital systems. Part of our goal is to develop design techniques that bypass these inevitable problems. Therefore, one of our dictums will be: don't use asynchronous RS flip-flops as a general design tool.

Switch Debouncing. There is one standard use of the RS flip-flop—as a *switch debouncer*. It is an unfortunate fact that mechanical switches do not make of break contact cleanly. At closure there will be several separate contacts over a period of many microseconds. The same is true during switch opening. The witch bounces. Since we do not wish to use a bouncy or spiky signal in our digital designs, we need a way to clean up the switch output.

Whenever a mechanical switch changes its position, we wish the associated digital signal to undergo one smooth change of voltage level. The asynchronous RS flip-flop is well suited for this. Figure 5.7 contains two switch debouncing

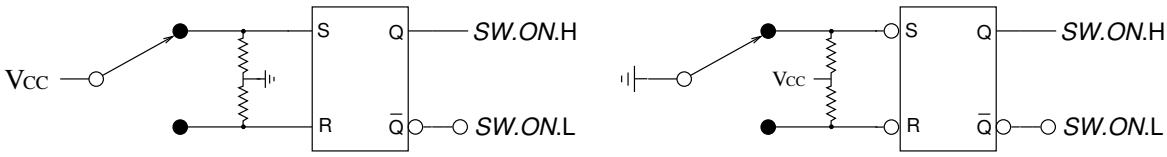


Figure 5.7: Mechanical switch debouncing circuits using asuchronous RS flip-flops.

circuits. In Chapter 12 we discuss the electrical details of the input circuits; here we will be satisfied to state that the resistors keep the control inputs inactive unless the voltage from the switch forces on input to become active. When the switch is off, it is constantly resetting the flip-flop, producing a constant F output. As the switch moves toward the on position, there will be a period of oscillation or bounce on the R input, caused by the mechanical switch breaking and making its contact with its off terminal. The S input is false throughout all of this, and the repeated resetting does not affect the false output of the flip-flop. There follows a "long" period when the switch moves between its off and on positions, during which time both S and R are false. Then the switch begins its bouncy contact with the on terminal. The first contact causes S to become true, which sets the flip-flop to its true state, where it remains throughout the on-position bounce and until the switch is returned to off.

Ambibuous behavior in the RS flip-flop. Of the four voltage combinations of th S and R inputs, we have used three: to hold, set, and reset. What happens when S and R are simultaneously true? In the NOR-gate version, the voltages at both outputs will be low—a distrubing situation. In the NAND-gate version, both will be high. Although this deviation from voltage complementarity is unwelcome, it nevertheless repreents a well-defined and stable configuration of the flip-flop. BUt watch what happens when we try to retreat from this configuration of inputs. If we change only one of the inputs, the flip-flop enters either the set or reset state, without difficulty. But if we try to change botrh the inputs simultaneously (in an attempt to move to the hold state), the flop-flop is in deep trouble. Consider the NOR-gate version of the RS flip-flop, Fig. **??**. If the volgages at S and R are both high, then they are low at both Z and Q. If the voltages as S and R both become low simultaneously, then after on gate delay both gates in the flip-flop will produce high outputs. Thses high outputse, feeding back to the inputs of the NOR gates, will result in low gate outputs after one more gate delay. And so on. The ciruit oscillates rapidly, at leas in the biginning, with both outputs producing either high or low voltage levels "in phase." The resulting changes occur so repidly that the flip-flop is forced out of the digital mode of operation for which it was designed, and the output voltages quickly cease to conform to reliable digital voltage levels—an

example of idxmetastable behavior that is discussed in Chapter 12. Eventually, the slight differences in the phisical properties of the two gates will allow the flip-flop to drop into the set state or the reset state. The time required for the voltages to settle and the final result are uncertain. so this behavior is of no use to designers. Therefore, it is considered improper design practice to allow R and S to be asserted at the same time.

Exitation tables. Timing diagrams are usefule for displaying the time-dependent characteristics of sequential circuits, but for most purposes a tabular form is better. The *Exitation table* is the sequential counterpart of the truth table or voltage table for combinational circuits. The exitation table looks much like a truth table, but it contains the element of time. In a sequential circuit, the new outputs depend on the present inputs and also on the present values of the outputs. We can display the behavior of the RS flip-flop of Fig **??** in the following exitation table:

S	R	$Q_{(t)}$	$X_{(t)}$	$Q_{(t+\delta)}$	$X_{(t+\delta)}$	
L	L	q	x	q	x	Hold
L	H	q	x	L	H	Reset
H	L	q	x	H	L	Set
H	H	q	x			*Disallowed*

$Q_{(t)}$ is the value of output Q at time t; $Q_{(t+\delta)}$ is the value of output Q at some small time δ after t, where δ is sufficiently long for the effects of the gate delays to settle down.

The exitation table is also useful for displaying the logical behavior of sequential cirucits. For instance, the following exitation table describes the logical behavior of RS flip-flops using a modification of the previous notation:

S	R	Q	Q'	
F	F	q	q	Hold
F	T	q	F	Reset
T	F	q	T	Set
T	T	q		*Disallowed*

In the literature, notations for exitation tables vary greatly and in this chapter we will use a variety of forms. You should be able to recognize the notational differences.

5.2.2 Clocked Sequential Circuits

Asynchronous flip-flops are 1's catchers. A more useful class of flip-flops is available for general digital design. In these flip-flops, outputs will not change unless another signal, called the *clock*, is asserted. Since the activity is sunchronized with the clock signal, these flip-flops are called *synchronous*. Digital systems usually have a repetitive clock with a square waveform. The clock signal alternates between its H and L signal levels. Depending on the application, we may

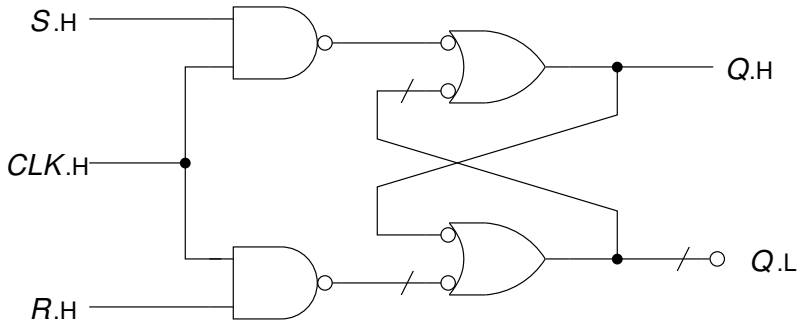


Figure 5.8: A clocked RS flip-flop circuit.

view either H or L as repreenting truth on the clock line, although in almost all our applications we shall use the T-H assignment for clock signals. In chapter 12 we dicusss ways of generating this important signal, and you will encounter clocked circuits throughtout the remainder of this book.

Clocked RS flip-flop. We can derive a clocked flip-flop from an asynchronous RS flip-flop by gating the R and S input signal to restrict the time during which they are actie, as in Fig. 5.9 The flip-flop outputs may change whenever the cloci is true—a potentially risky situation similar to the 1's catching of the latch circuit. In digital systems, flip-flop ouputs oftwn contribute to combinational circuits that produce inputs to othyer flip-flops. Shortly after the rise of the clock, the system is in "shock" owing to the changing of flip-flops. Doring this period of shock, hazards may be present that can feed erroneous signals into flip-flop inputs while the clock is *still true*, resulting in false setting or resetting of the flip-flops.

It is natural to try to avoid this problem by making the true portion of the clock signal as narrow as possible. Unfortunately, this is not a good solution, since the system's behavior is critically dependent on the quality of the clock and narrow clock signals are difficult to generate and distribute.

The aim is to reduce the time during which the flip-flop outputs respond to the inputs. Since altering the clock waveform leads to difficulties, can we achieve the tgoak fy further mofification fo the flip-flop circuit itself? Can we devise a flip-flop that will recognize R and S only at a single instant and ignor the inputs at other times? Such behavior would be desirable because all the flipflops would change at precisely the same time if they were clocked from the same source. This would mean that we could arrange for all the R and S inputs on all flip-flops to be stable at the time of clocking, and the flipflops would not be influenced by the chocke of the changes induced just after clocking.

Flipflops that allow output changes to occur only at a single clocked instant are called *edge-driven* or *edge-triggered*. An *edge* is a voltage transition on the

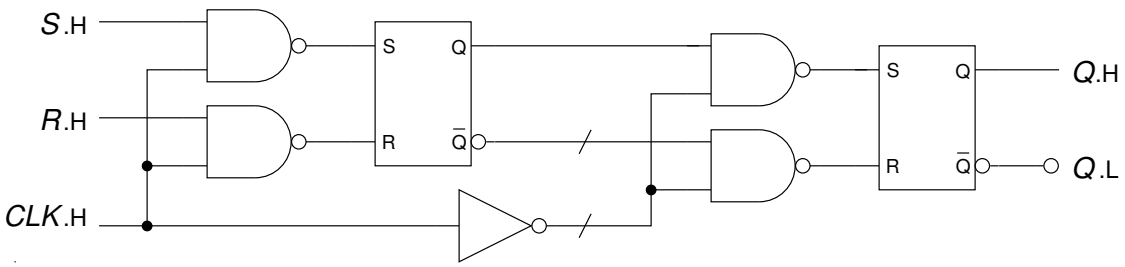


Figure 5.9: A master-slave clocked RS flip-flop.

clock signal, and may be either a postive edge ($L \rightarrow H$) or a negative edge ($H \rightarrow L$). The clocked circuit in FIg. 5.9 is *level-driven*, since its outputs may change at any time during the true part of the clock cycle. In your designs of clocked sequential circuits, use only edge driven devices.

Master-slave flip-flop. The master-slave flip-flop is a relec from the early days of integrated circuit technology, but is still widely used becaus of its pseudo-edge-driven characteristics. It is a relatively simple device that we can easily discuss at the gate level, sow we will show how one is derived by extending the clocked RS flip-flop. Figure **??** is a master-slave flip-flop schematic. The master flip-flop will responde to inputs S and R as long as the clock signal is high. This period must be long enough to ensure that S and R are stable when the clock goes from high to low. This $H \rightarrow L$ transition, the negative clock edge, isloates the master flip-flop form the inputs S and R. The master flip-flop will now remain unchanged until the next positive clock edge.

Because of the voltage inverter, the slave flip-flop does not become sensitive to its input until one gate delay after the negative clock edge. At that time, it receives its S and R inputs from a stable master flip-flop. The net effect is that the outputs of the mssster-slave combination change only on the negative clock edge rather than during a clock level.

Pure edge-driven flipflop. The master-slave flip-flop appears to be an attractive edge-driven device. Why are we not content with this design? Becuase the master flip-flop is still a 1's catcher during the positive half of the clock cycle. This means that R and S must stabilize during the negative half of the clock, since the master flip-flop will react to any T glitches during the positive clock phase. We could greatly simplify our digital circuits if we could eliminate the 1's catching behavior. We need a flip-flop that samples its input only on a clock edge and ganges its outputs only as a result of the clock edge. Such a device is called a *pure edge-driven flipflop*. The $F \rightarrow T$ clock transition is called the *active edge*. It may be either the $H \rightarrow L$ or $L \rightarrow H$ transition, although in the most useful integrated circuits the $L \rightarrow H$ transition is the active edge.

The property of changing state and sensing inputs only at a given instant gives the designer a powerful tool for combatting glitches and noise. We can now choos the time to look at signals and can fix that time to allow adequate stabilization of the system. We will make constant use of pure edge-driven sequential curcuits in our designs. The internal structure of these devices is rather complex, but for purposes of digital system design it is not necessary for us to examine their construction in detail. Hereafter, in all our discussions of clocked sequential circuits, we will assume the use of pure edge driven devices.

Exitation tables for edge-driven flip-flops. Assume that the edge-driven flip-flop is subjected to a steady stream of active clock edges. Each clock edge will cause the flip-flop to enter either its set or its reset state, in accordance with the faluce of its inputs and the current value stored in the flip-flop. After the flip-flop has received n clock triggters, the value stored in the flip-flop is $Q_{(n)}$. If the flip-flop is in the set state after the nth clock edge, then $Q_{(n)} = T$; if the reset state, $Q_{(n)} = F$ After the appearence of the next clock edge, the value of Q will be $Q_{(n+1)}$. The exitation table for edge-driven devices is a tabulation of $Q_{(n+1)}$ for all combinations of the exiting variables.

In the remainder of this chapter we will use exitation tables to classify flip-flops. For the excitation table to be valid, we muse ensure that the control inputs are stable for a short time befor the active clock edge (the *setup* time), and perhaps for a short time after the active clock edge (the *hold* time). The input voltages may go through wild excursions prior to the onset of the setup time and after the hold time, as long as they remain stable during the steup and hold times. (See Chapter 12 for a discussion of setup and hold times).

5.3 Clocked Building Blocks

In this section we present the common SSI building blocks for clocked digital design. Table **??**, at the end of Chapter 12, contains a selected list of useful integrated circuits as well as more complex building blocks.

The JK Flip-flop

Whereas the RS flip-flop displays ambiguous behavior if both R and S are true simultaneously, the JK flip-flop produces unambiguous results in all combinations of its inputs. A logical exitation table for the basic JK flip-flop is:

Clock	J	K	$Q_{(n)}$	$Q_{(n+1)}$	
F	X	X	q	q	
T	X	X	q	q	
$\uparrow$	F	F	q	q	Hold
$\uparrow$	F	T	q	F	Reset
$\uparrow$	T	F	q	T	Set
$\uparrow$	T	T	q	$\overline{q}$	Toggle(complement)

J is the counterpart of the S input of an RS flip-flop, and K is the counterpart of R. The first two lines of the exitation table demonstrate the edge-triggered behavior of the flip-flop; when the clock signal is stable false or true, the output of the flop-flop is insensitive to the other inputs. Often these lines do not appear in the exitation table, since such bnehavior is expected of an edge-triggered device. The remaining four lines in the table describe the flip-flop behavor when the clock under goes its active ($f \to T$) transition. The first thre of these lines are analogous to the RS flip-flop. The last line shows that, if both control inputs are true when the clock fires, the flip-flop will complement its output. This behavior is called *toggling*.

Commercial flip-flops come in various forms. The most interesting variations are:

(a) Active clock edge: positive or negative. On all clocked devices we show the clock input as a small wedge ($\triangleright$) inside the device symbol. A negative edge-triggered flipflop has a small circle (representing $\mathsf{T} = \mathrm{L}$) on the clock input ($\circ\!\triangleright$).

(b) Active voltage level for J and K. We find flip-flops with both J and K active-high ($\mathsf{T} = \mathrm{H}$), and also a flip-flop with J active-high and K active-low. In the latter form, the K input has a small circle on the circuit symbol.

(c) Availability of asynchronous R and S inputs. These are often called *direct clear* or *preclear* and *direct set* or *preset*. One, both, or neither may be present on the chip. Direct set usually appears at the top of the flip-flop symbol, and direct clear at the bottom. Truth is usually a low voltage level, in which case the inputs will bear small circles. As long as an asynchronous input is asserted, it will override the normal synchronous behavior of the flip-flop.

The 74LS109. The 74LS109 Dual JK Flip-Flop is the most-used SSI package. It is positive-edge-triggered, compatible with the standard MSI sequential building blocks, with a high-active J input and a low-active K input. As usual, when designing with JK flip-flops we think in terms of logical operations rather than voltages. It is useful to describe the primary logical operations on the JK flip-flop as the "set" and "clear," setting Q to T and clearing Q to F. The 74LS109 flip-flop has two useful mixed-logic representations, shown in Fig. 5.10 with appropriate input and output signals. Port pr is the asyunchronous preset input; clrs preclear. The symbol in Fig. 5.10a is conventional and causes no difficulty. The circuit shown in Fig 5.10b is less easy to derive, but it gives us a degree of flexibitlity that repays our efforts. The voltage exitation table for the

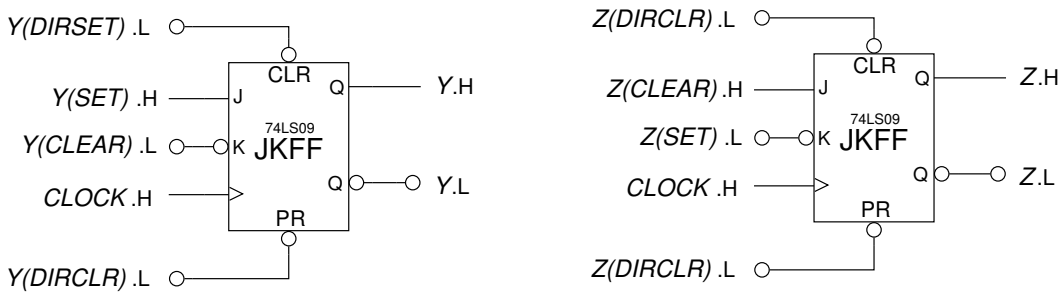


Figure 5.10: Two mixed-logic uses of the LS109 Dual JK Flip-Flop.

74LS109 is

Preset	Preclear	Clock	J	K	$Q_{(n+1)}$	Action if Q is active-high
L	H	X	X	X	H	Directset
H	L	X	X	X	L	Directclear
L	L	X	X	X	–	*disallowedconfiguration*
H	H	↑	L	L	L	Clear(Reset)
H	H	↑	L	H	$Q_{(n)}$	Hold
H	H	↑	H	L	$\widetilde{Q_{(n)}}$	Toggle
H	H	↑	H	H	H	Set

Here, $\widetilde{Q_{(n)}}$ means the voltage inversion of $Q_{(n)}$; as usual, X means "don't care." This excitation table contains yet another variation of notation, in which the monotonous input column for the present value of the flip-flop's current value is omitted.

When T = H at Q, we derive the mixed-logic symbole in Fig 5.10a, the usual form. If T = L at Q, the logical act of setting the flip-flop must rewult in an L output at $\overline{Q}$; logical clearing must yield Q = H. In order to maatch this behavior with the voltage exitation table, we are led to the conclusion that we must *set* the flip-flop with the K input and *clear* with the J input. In turn, this causes the preset input pr to perform as a *logical direct clear*, and the preclear input clr to perform as a *logical direct set*.

The advantage of the form used in Fig. 5.10b is its versatility, since we use a different voltage confention for setting and clearing than we do with the conventional symbol. This mixed-logic symbol for the 74LS109 is the most difficult of the common building blocks to derive, yet having onced derived and mastered it, we may use either symbol for the 74LS109, as our use dictates, without further thought.

This exercise in mixed logic illustrates one aspect of good design: We try to define the general behavior of common circuit elements, and arrive at general solutions to common design problems. We move these recurring but purhaps

difficulet items up front, where we face them squarely, so that having dealt with their intricacies once, we may thereafter use the starndarde results in our design work. You will see this principle invoked many times in this book; it is the essence of top-down design.

The JK flip-flop is our most powerful SSI storage element, and you must master its use. There are several ways of using a single flip-flop, and later you will see many larger constructions based on this flexible element.

JK flip-flop as controlled storage. The most general use of the JK flip-flop, and the one that gives it such power and flexibility, as as a storage element under explicit control. In digital design, whenever we must set or clear or toffle a signal to form a specific value for later use, we usually think of a JK flip-flop. The penalty for this generality is the need to control two separate inputs.

JK flip-flop for storing data. The JK flop-flop is basically a controlled storage element. On occasion, we with to adopt a different posture and view the JK flip-flop as a medium for entering and stroing data. From the exitation table, we see that $Q_{(n+1)} = Q_{(n)}$ whenever $J = K = F$ at the clock edge. This is simply a data-storage mode. All that is necessary to continu holding data in the flip-flop is to ensure that $J = K = F$ during the setup time before each clock edge.

JK flip-flop for entering data. Th J and K inputs are not data lines; they are control linds for flip-flop storage. Nevertheless, we can view the JK flip-flop as a data-entry device. We can enter data in three ways:

(a) Clearing, followed by later setting if necessary.

(b) Setting, followed by later clearing if necessary.

(c) Forcing the data into the flip-flop in one clock cycle.

The rule for case (a) is:

> If you are sure that the flip-flop is cleared, you may enter data D into the FF on a clock edge by having J = D, independent of the value of K.

Case (be) is analogous to case (a). The rule is:

> If you are sure that the output is true, you may enter data D into the flip-flop on the clock edge by having $K = \overline{D}$, regardless of the value of J.

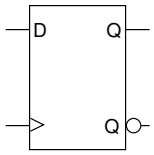
You should varify the rules for cases (a) and (b).

As for case (c), the designer often cannot guarantee that a flip-flop will be in a given state. Proceeding as we did in cases (a) and (be) would wast one clock cycle for the initial clearing or setting operation. It would be nice to have a mode that would force data to enter the flip-flop at a clock edge, regardless

of the present condition at the output. Such a data-entry mode is called a *jam transfer*, since the data is "jammed" into the flip-flop independent of prior conditions. Examination of the exitation table for the JK flip-flop shows that such a mode is indeed avaialbe. We enter data D as follows: If $D = F$, J must equal F and K must equal T. IF D = T, J must equal T and K must equal F. COmbining these conditions, we see that $Q_{(n_1)}$ will equal D whenever $J = D$ and K = $\overline{D}$. Now you see the utilit of having opposite voltage conventions for truth on the J and K inputs of the 74LS109 flip-flop. With this device we can connect the J and K inputs together to make $K = \overline{J}$ as required by the analysis above. Then by connecting the input data D to the joined inputs of the flip-flop, we will enter D into Q at each clock edge.

5.3.1 The D Flip-FLop

The D (Delay) flip-flop has a simpler exitation than the JK, and is used in applications that do not require the full power of the JK flip-flop. The symbol and exitation table for the D flip-flop are:



D	$Q_{(n+1)}$
0	0
1	1

As an SSI device, the D flip-flop appears in these common varieties:

(a) The active clock edge can be either positive ($L \rightarrow H$) or negative ($H \rightarrow L$), which is shown by either the presence or absence of a small circle on the clock terminal.

(b) Direct (asynchronous) set and clear inputs appear in these combinations: both, neither, or clear only. Almost always, these inputs, when present, are low-active and appear in the diagram with a small circle. These asynchronos inputs are 1's catchers, and you should use them only with great caution.

(c) Aome D flip-flops have only the Q output; others provide both polarities. Although it appears to be ideal for data storage, there are, in fact, just a few common uses of the D flip-flop in good design.

D flip-flop as a delay. As its name implies, the D flip-flop serves to delay the value of the signal by one clock time. You will see such a use in Chapter 6 when we discuss the single-pulser circuit for manual switch processing.

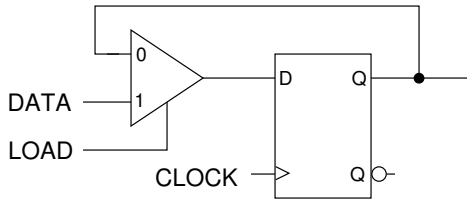
D flip-flop as a synchronizer. One natural application of the D flip-flop is as a synchrnonizer of an input signal. Clocked logic mus sometimes deal with input signals that have no fixed temporal relation to the master clock. An example is

a manual pushbutton such as a stop switch on a computer console. The operator my close this switch at any time, perhaps so near the next edge of the system clock that the effect of the changing signal cannot be fully propagated through the circuit before the clock edge arrives. If the inputs to clocked elements are not stable during their setup times, their behavior is not predictable after the clock edge; some of the outputs may change; others may not. We need some way to process this manual switch signal so that it changes only when the active clock edges appear. This is called *synchronization.* Since the output of a clocked element changes only in step with the system clock, we may use the D flip-flop as a synchronizer by feeding the unsynchronized signal to the flip-flop input. We deal with this matter more fully in later chapters.

D flip-flop for data storage. The D flip-flop appears to be well suited to data entry and storage. Unfortunately, designers use it far too often for this purpos. The problem is that every clock pulse will "load" new data and this is seldom wanted. We usually need a device that allows us to control when the flip-flop accepts new data, just as we could with the JK flip-flop. With the D flip-flop it seems natural to *gate the clock* b *and*ing with a control signal in order to produce a clock edge at the flip-flop only when we wish to load data. This is a dangerous practice, as you will see in later chapters. Clocked circuit design relies on a clean clock signal that arrives at all clock inputs simultaniously. We have the best chance of meeting these conditions if we use unmodified clock signals. This means that the devices will be clocked every c;ycle, so we must seek other ways of effecting the necessary control over the flip-flop activities.

The enabled D flip-flop. To alleviate the problems caused by gating the clock input to a D flip-flop, we will construct a new type of device called the *enabled D flip-flop.* FIgure **??** shows the principle. The circuit consists of a D flip-flop with a multiplexer on its input. A new control signal, LOAD appears, in addition to the customary data input.

The system clock goes directly to the clock input, thereby avoiding the problems of a gated clock. As long as LOAD is false, the data selector selects the current value of the flip-flop output as input to the flip-flop. The net effect is that Q recurculates unchanged: the flip-flop stores data. When LOAD = T, the multiplexer routes the external signal DATA into the D input, where it will be loaded into the flip-flop on the next clock edge. The loading process is a jam transfer.



The enabled D flip-flop is the element of choice for simple data storate applications. Although we can accomplish the same effect with the JK flip-flop, the

enabled D device provides a more natural way of handling data. Curiously, integrated circuit manufactureres were slow to produce SSI elements containing several independent enabled D flip-flops. However, the concept is widely used in arrays of storage elements, and we have available a good selection of tools for registers.

5.4 Register Building Blocks

A *register* is an ordered set of flip-flops. It is normally used for temporary storage of a related set of bits for some operations. This is a common activity in digital des9ign, especially when the system musp process byte- or word-organized data. You are familiar with the use of the word *register* in the context of digital computers, but the notion is more general than just accumulators and instruction registers. Multiple-bit storage is such a desirable architectural element that it is a natural candidate for building blocks. Integrated-circuit manufactureres have provided an assortment of useful devices at the MIS level of complexity.

5.4.1 Data Storage

Enabled D register. The most elegant data storage element for registers contains the enabled D flip-flop. Chips are available with four, six, or eight identical elements per package with common clock and enable inputs. The 74LS378 Hex D Register with Enable is typical of this building block. This 16-pin chip provides six aflip-flops, each with a data input and a single Q outpus. The 74LS379 Quad D Register with Enable has four flip-flops, each with both output signal polarities.

As you have seen, we favor the enabled D conafiguration because we may hook the system clock directly to the device's clock input. The apparently small point of not gating the clock is really of great importance to the reliability of the system, and you should adopt the practice routinely.

Pure D register. There are a few occasions when a register of pure D flip-flops is the element of choice. We can always achieve this behavior with the enabled variety by setting the enablin input to the true condition. Pure D registers are also available, usually with a common aschnchronous (direct) clear input. The only reason to choose su;ch an input is if you wan the direct clear feature; you know to be wary of its 1's catching properties.

5.4.2 Counters

Modulus countin. Counting is a necessary operation in digital design. Since all binary counters are modulus counters, we will explor the concept of modulus counting before we examine the hardware for it.

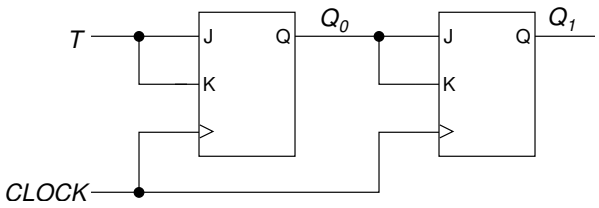


Figure 5.11: A two-bit binary counter. The least significant bit is on the left.

Countin the positive integers in as infinte process. We have a mathematical rule for writing down the integer $n + 1$ if we are given the integer n. This may caus the creation of a new column of digits; for example, if n is the three-digit number 999, then $n + 1$ is the four-digit number 1,000. In an abstract mathamatical sense, the creation of the fourth digit is trivial. Not so in hardware. Hardware counters are limited to a given number of columns of digits and thus there is a maximum number that a counter can represent. A three-digit decimal counter can represent exactly $1-^3$ different numbers, from 000 through 999. We define such a counter as a *modulus* (mod) 1000 counter. (A *number M modulo some modulus N*, written M mod N, is defined as the remainder after dividing M by N.) Another way of viewing this is that the counter will count normally from 000 through 999, and once more count will cause it to cycle back to 000. An automobile's odometer behavies much the same way.

Counting with the JK flip-flop. The JK flip-flop, operating in its toggle mode, goes through the following sequence

Clock pulse number:	0	1	2	3	4	5	6	...
Flip-flop output Q:	0	1	0	1	0	1	0	...

We see that the flip-flop behaves as a modulo-2 binary counter. Counters of higher moduli can be formed by concatenating other binary counters. For instance, a modulo-4 counter made from two mudulo-2 counters mus behave as follows

Clock pulse number:	0	1	2	3	4	5	6	7	8 ...	
Cunter outputs Q_1, Q_0:	00	01	10	11	00	01	10	11	00	...

Can we devise a logic configuration that will cause two JK flip-flops to count in this fashion? On answer is in Fig. **??**. Here, for drafting convenience, wed draw the least significant bit Q_0 on the left, whereas Q_0 appears on the right in the usual mathematical representation of the number $Q_1\,Q_0$. Q_0 alternates in value (toggles) at each clock. At alternate clock edges, Q_1 is clocked when $Q_0 = T$; at these times the value Q_1 toggles.

Figure **??** contains another solution that appears to give equivalent results. Again, Q_0 will toggle at each clock pulse since $J = K = T$ on that flip-flop.

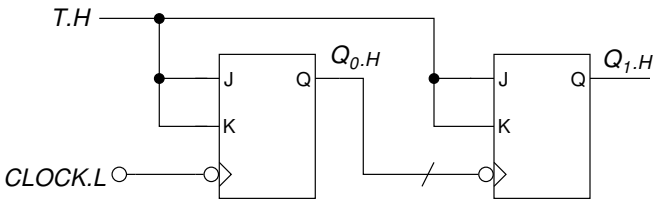


Figure 5.12: A binary ripple counter. The least significant bit is on the left. This circuit displays both logic and voltage, wherease the related Fig. **??** displays only logic.

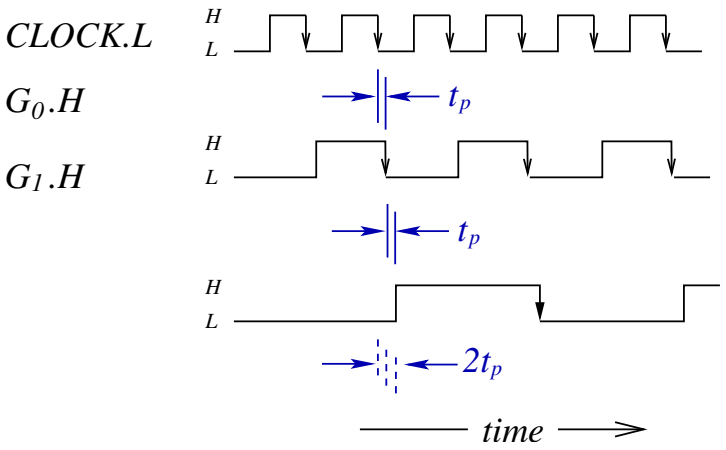


Figure 5.13: A timing diagram for a 2-bit ripple counter. Each stage suffers a cumulative propagation delay. In synchronous counters there is only on delay.

THis is necessary for a binary counting sequence. Every time Q_0 generates a $T \rightarrow F$ transition ($H \rightarrow L$ in this circuit), Q_1 will toggle since $J = K = T$ on that flip-flop alsow, and Q_0 provides the Q_1 *clock*. Figure **??** is a timing diagram for this circuit. The timing diagram for Fig. **??** is almost identical to Fig **??**; lthe difference is due to propatation delays. In **??**, if we assume that t_p is the flip-flop propatation delay, both Q_0 and Q_1 will change t_p nanoseconds after the clock edge, since J and K were stable during the setup time of both flip-flops. We define such counters as *synchronous*.

In contrast, Q_1 in Fig. **??** canot change until t_p nanoseconds after Q_0 has changed. Counters that change their outputs in this staggered fashion are called *asynchronous*, or *ripple*, counters , since a change in outpus mus ripple through all the lower-order bits before it can serve as a clock for a high-order bit.

Ripple counters are easily extensible to any number of bits. Thus, a modulo

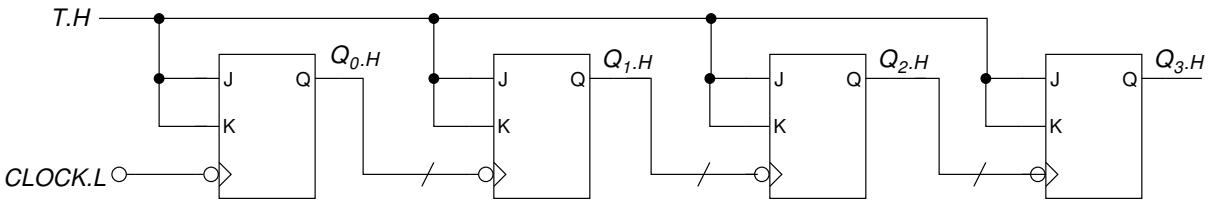


Figure 5.14: A 4-bit (modulo-16) ripple counter.

16 ripple counter would be as in Fig. **??** This simple configuration is useful if you are not interested in the relation of Q_3 to any lower-order bits. A common example is a digital watch, in which a 32,768 Hz (2^15 cycles per second) quartz crystal oscillator is the primary timing source. The watch display is driven at a rate of 1 Hz, using the output of a 15-stage ripple counter.

The *clock skew* caused by flip-flop propagation are negligible to the display and the simple function of the counter raises no other problems. In addition, as we shall see in a moment, the design is economical. However, without *extremely* careful analysis, we would ordinarily reject this design simply because, by gating the clock, it raises the need for analysis which we would ordinarily wish to avoid.

To discover the problems that can arise with ripple counters, consider a modulo-8 counter that is changing its count from 3 to 4. The ripple sequence from the initial clock edge would be

Time	Q_2	Q_1	Q_0
0	0	1	1
t_p	0	1	0
$2t_p$	0	0	0
$3t_p$	1	0	0

In a typical appliation, the count code is fed into a decoder to produce indificual signal lines for each count. In this case, we would have momentary true hazards at decoder outputs 2 and 0, each lasting for a time t_p. These glitches are seldom useful and may be quite harmful. We can eliminate them by using a synchronous counter.

Figure **??** represent a 2-bit special case of synchronous counters. The rule for changing the nth bit of a binary counter is that all lower bits must be 1. Using

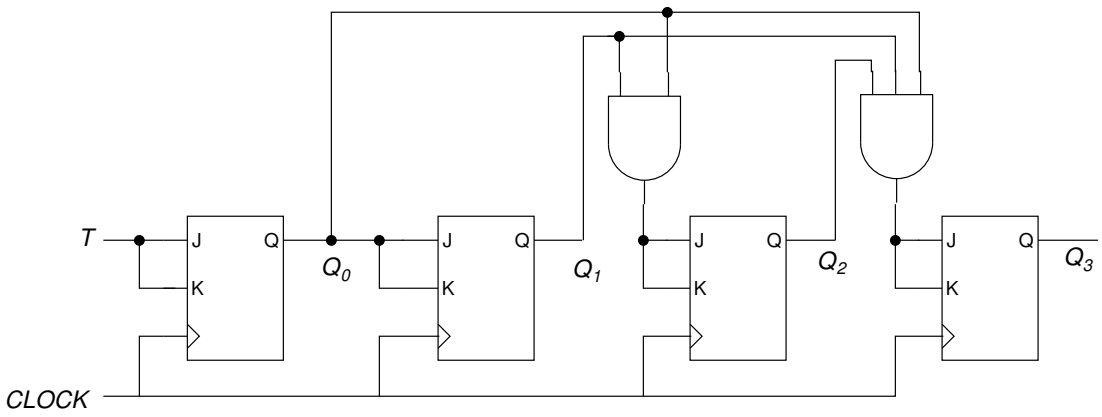


Figure 5.15: A 4-bit (modulo-16) synchronous counter.

this rule, we can consturct a modulo-16 synchronous counter for JK flip-flops, as in Fig 5.15. At the cost of the extra AND gates, we have manipulated the inputs to each flip-flop to cause the flip-flops to toggle at the proper time. SInce a common clock signal runs to each flip-flop, the outputs will occur simultaneously, without ripple.

MSI counters . Synchronous counters are soe useful that manufacturers have prepared a wide variety as MSI integrated circuits, typically modulo-10 (decade) and modulo-16 (4-bit binary) devices with provisions for cascading to form higher-modulus counters. Some synchronous counters have an asynchronous-clear input. You must be careful to supply a noise-free signal to this terminal to ensure reliable operation. Remember, it's a 1's catcher. Novices (and some experienced designers) tend to use the asynchronous clear feature too often. About the onlyu time it can be safely used is during a power-up or master reset sequence to drive crucial flip-flops to a known state.

The ideal counter would be caxcadable and have a synchronous clear input terminal, such as in the 74LS163 Four-Bit PRogrammable Binary7 COunter. WIth this device, a cascated 12-bit synchrnous counter would appear as in Fig. 5.16. Each 74LS163 chip is a synchronous counter. CLOCK.H is the system clock signal, and since it goes to each 4-bit chip, all ouput changes will be synchrnous.

TC (Terminal Count) and CET (Count Enable Trickl) are individual device controls that permit proper counting in a cascaded configuration. The counting rule is that a given bit must toggle if all lower bits are equal to 1. This rule will yield the normal binary counting sequence. We may reinterpret the binary sequence as a hexadeimal sequence by grouping the binary bits into 4-bit ;uints and giving each 4-bit unit a range of 0_{16} to F_{16}. The rule for counting a

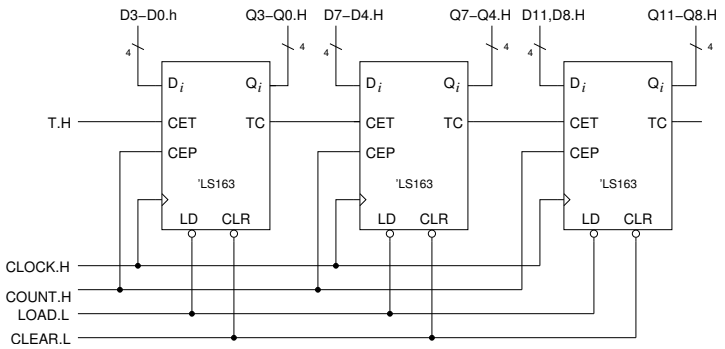


Figure 5.16: A 12-bit binary counter constructed with 74LS163 chips.

hexadecimal digit is:

all lower-order hexadecimal digits must equal F_{16}.

TC and CET implement thiss rule. The defining equation for TC is

$$TC = (COUNT = F_{16}) \cdot CET$$

The signal to the CET input comes from the TC output of the previous counting stage. When CET is true, it serves as a signal to the chip that all previous stages are at the terminal F_{16} count. TC is thus an output that notifies the next stage that all lower bits in the counter cascade are 1. You can see the proper cascading connections in Fig. 5.16.

CEP (Count Enable Parallel) is a master eount enable signal which goes to all chips. It allows the designer to specify when the circuit should engage in counting activity.

The 74LS163 has two more controls: the synchronous clear input CLR, which permits clearing of the chip to zero at the next clock pulse, and the synchronous load input LD, whicl allows loading of the counter with the 4-bit pattern appearing at the data inputs. The existence of the load feature accounts for the "programmable" in the chip's name. THe priority of operations is such that asserting CLR will override LD, which will override CEP.

It will be a useful exercise for you to derive J and K inputs to each flip-flop in the 74LS163 counter. For an input data bit D_0, the (unsimplified) result for bit 0 is

$$\begin{aligned} J_0 &= CLR \cdot F + \overline{CLR} \cdot LD \cdot D_0 + \overline{CLR} \cdot \overline{LD} \cdot CEP \cdot CET \cdot T \\ K_0 &= CLR \cdot T + \overline{CLR} \cdot LD \cdot \overline{D_0} + \overline{CLR} \cdot \overline{LD} \cdot CEP \cdot CET \cdot T \end{aligned}$$

Many other synchronous counters are available as MSI chips. We have covered the 74LS163 in detail, since it is an example of a useful and well-engineered

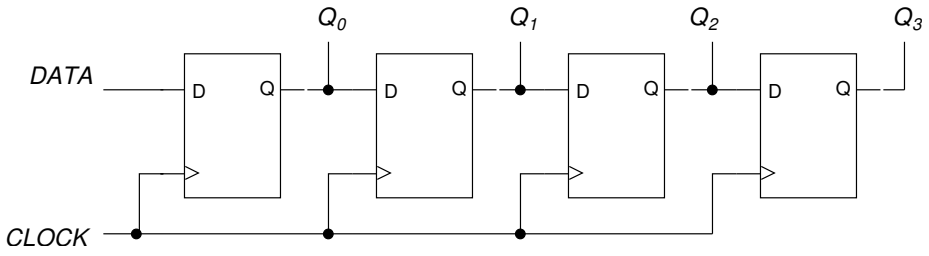


Figure 5.17: A simple shift register constructed with D flip-flops.

MSI building block. When dealing with circuits of MSI or LSI complexity, you must be cautious about adopting new designs. Frequently, a chip that appears to be exciting will have subtle features that make it less desirable or useless in good design. An example is the 74LS193 Four-Bit BInary Up-Down Counter. This chip has two serious flaws that may escapt the attention of the "chip-happy" designer. First, its data-loading feature is asynchronous., whish presentsu us with the necessity of keeping the data-load signal clean at all times, to avoid its habit of reacting to any momentary true glitch. Second, the chip has two clocks, one for counting up and the other for counting down. Using this chip requires very careful and arduous planning of the tipe that we choos to avoid enhtirely in our designs in Part II. Another up-down counter, the 74LS669, does not sufrfer from the 74LS193's deficiencies. Our point is that you must be alert to detect such deviations from good design and should choose your building blocks carefully and conservatively.

5.4.3 Shift Registers

A *shift register* performs an orderly lateral movement of data from one bit position to an adjacent position. We may constuct a sumple shift retister from D flipflops, as shown in FIg. 5.17. This circuit accepts a single bit of data DATA and shifts it sown the chain of flip-flops, on shift per clock pulse. Data enter the curcuit serially, one bit at a time, but the entire 4-bit shifted result is available in parallel. Bits shifted off the right-hand end are lost. Such a circuit is a primitive serial-in, parallel-out shift register.

For practice, we have need for four shift register configurations: serial-in, parallel-out; parallel-in, serial-out; parallel-in, parallel-out; and serial-in, serial-out. The parallel-in, parallel-out variety is the most general, subsuming the other forms. Let's design one.

Asume that we are building a 4-bit general shift register. What features do we require?

(a) We must be able to load initial data into the register, in the form of a 4-bit parallel load operation.

(b) We must be able to shift the assembly of bits right or left one bit position, accepting a new bit at one end and discarding a bit from the other end.

(c) When we are not shifting or loading, we must retain the present data unchanged.

(d) We must ba able to examin all 4 bits of the output.

Suppose we start with an assembly of four identical and independent D flip-flops, clocked by a common clock signal. Let the flip-flop inputs be D_3–D_0 and the outputs be Q_3–Q_0, from left to right. Let the external data inputs be $DATA_3$–$DATA_0$. We have four shift register operations: load, shift left, shift right, and hold. These will require at least 2 bits of control input to the circuit; let *S1* and *S0* be the names of two such control bits. Our task is to derive the proper input to each D flip-flop, based on the value of the control inputs *S1* and *S0*. In our design of an enabled D flip-flop, we encountered a related problem, actually a subset of the present problem. There we had two operations, hold and load, that we implemented with one control input, using a multiplexer. We may employ the same technique here, using a four-input multiplexor to provide input to each flip-flop. We may then define codes *S1*, *S0* for our four operations. Using *S1* and *S0* as mux selector signals, we may infer the proper inputs to the multiplexers. Here are the inputs for a typical bit i of the shift register:

Clock	*S1*	*S0*	Result desired	Selected mux position	Required mux input
↑	0	0	Hold present data	0	Q_i
↑	0	1	Shift right	1	Q_{i+1}
↑	1	0	Shift left	2	Q_{i-1}
↑	1	1	Load new data	3	$DATA_i$

In Fig 5.18, the logic for the ith bit is displayed. This circuit makes a useful parallel-in, parallel-out shift register. These functions are incorporated into the 74LS194 Four-bit BIdirectional Universal Shift Register in this way.

Providing both parallel inputs and parallel outputs requires many pines on an integrated circuit chip, so chip manufacuters make a variety of good shift registers for all four combinations of serial and parallel input and outpus with up to 8 bits per package.

5.4.4 Three-state Outputs

In Chapter 3, you learned the advantage of three-state control of the inputs to a data bus. We may provide such control with three-state buffers, such as the 74LS244. However, some register chips provide builti-in three-state control of their outputs. In bussing applications, this can be very convenient. When you are using a chip but do not need the high0impedance state, you may permanently enable the outputs by wiring the three-state output enable line to a true value.

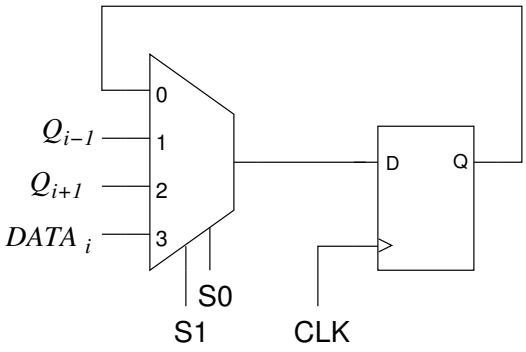


Figure 5.18: A typical bit Q_i of a general shift register.

5.4.5 Bit Slices

In Chapter 3 we discussed combinational circuits that incorporated a complex set of vunctions for a serveral-bit array of inputs. The arithmetic logic unit was the central example of this purely combinational bit-slice curcuitry. In chapter 7 you will see how to assemble registers, ALUs, multiplexers, shifters, and other components into a central processing unit for a computer. Experience has shown that the architecture of the processing units of a large class of register based computers and device controlers is quite similar, even when the widthe of the computer word varies over a wide range. Figure 5.19 is an abstraction of this common architecutre. A collection of registers, usually called a *register file*, provides inputs to an ALU. The output of the ALU passes through a shift unit before being routed back to the regtister file. The register file is a three-port memory that accepts three addresses and simultaneously reads from two addresses and writes to a third. Within this common architecture, the width of the data paths, the internal structure of the register f8ile, the operations p0erformed by the ALU, and the complexity of the shifter vary. Several manufacturers have abstracted a *processor bit slice* from this architecture, and have provided integrated circuits that capture 4 or 8 bits of the architecutre. These chips are frequently general enough to span a range of liely desings of processors, and powerful enough to eliminat much of the MSI-level "glue" usuall found in such designs. By stacking the bit slices side-by-side, it is often possible to design a high-speed, simple, and powerful processor.

On of the first chips to be designed as a bit-slice component was Advanced Micro Devices' AM2901 Four-Bit processor, which incorporated a 16-word register file, 16 arithmetic and logical operations, and rudimentary provisions for supporting the extra storage required for multiplication and division. Bit-slice architecute has evolved into quite powerful ALU slices, supported by large families of auxiliary chips for processor sequencing, input-output support, and so forth. The AM2903 and TI888 are representative of the central elements in such

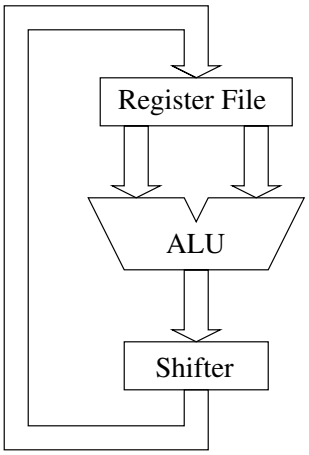


Figure 5.19: The architecture of a typical processor

bit-slice families. The TI888 is an auxiliary multiplier-quotient register to support operations, including binary multiplication and division and binary-coded decimal arighmetic. It has provisions for attaching an external register file, to enlarge the coapcity of the holding store.

To build structures based on conventional processor archiecture, you should ivestigate the use of processor bit-slice chops. But be aware that thewse are highly complex circuits whose data sheets will require considerable study before you understand the details of the architecture, se of instructions and timing.

5.5 Large Memory Arrays

If a few bits of register storage are good, would 1,024 bits be better? How about 4K, 64K, or 1M? For data storage alone, the more bits per package the better. But with such a large number of bits stored, we would have to give up the ability to gain access to all the bits simultaneously, and we would need some way of specifying which bit or group of bits we wish to lock at. Several forms of large-scale solid-state memories are avialable. Such devices have revolutionized computer techonlogy by making inexpensive, fast storage readily avialiable to the comptuer architect. Integrated circuit manufacturers are doubling the number of bits per package roughly every three years, wheras the price per bit is steadily declining. This trend will continue, and memories will be in the forefront of new electonic technology, becuase manufacturers can spread engineering and development costs over millions of identical units.

5.5.1 Random Access Memory

A large memory that requires the same time to access each data bit is called a *random access memory* (*RAM*). All RAMs share some common features. The unit of storage is the bit, which is built into the suface of a thin silicon wafer. The area of silicon devoted to a bit is a *cell*. Several sell structures are in use: some closely resemble the D flip-flop and others store a bit by the presence or absence of an electric charge on a mocroscopic apacitor embedded in the silicon surface.

Atypical RAM has so many cells that it would be impossible to connect each cell to its own integrated circuit pin. To conserve pins, RAMs contain a demultiplexer to distribute an incoming data bit along an internal bus to the correct cell. Similar, there is a multiplexer to select on cell's output and route it to the output pin. Both the demultiplexer and the multiplxer receive their control from an address supplied to the chip on a set of address lines. The address is encoded: eight pins devoted to an address can selecto on of 256 cells; 12 pins will handle 4,-96 cells.

RAMs are characterized by their total number of storage cells, for example 4K (4,096), and by the sixe of the words they contain. A 4K $\times$ 1 RAM contains 4K 1-bit words, whereas a 1K $\times$ 4 RAM sstill contains 4K cells, but the cells are organized as 1,024 four-bit words. The 1-bit-per-word organization saves integrated circuit package pins as compared to the 4-bits-per-word structure. (You should figure out why this is so and how many pins are saved.) Thus, in large memories, 1-bit-per-word RAMs are common, whereas designers of smaller memories often use the 4-bits-per-word type to keep down the number of chips devoted to memory.

Large RAMs requrie many address bits to specify the cellse. A 64K$\times$1-bit RAM requires 16 address bits; a A 1M$\times$1-bit RAM requires 20 address bits. In each case there are too many address bits to allocate each bit to a separate pin on the chip. To alleviate this problem, the address bits in large RAMSs ar split into two parts, a *row addrss* and a *column address*. At the proper time in a memory cycle, each part is fed to the rame over the same address inputs—that is, the row address and column address are *time-multiplexed*. (The row-and-column nomenclature derives from the internal structure of the RAM, which can be viewed as a large, two-dimensionall array with elements addressed by row and column indices.) A 256K RAM has 9 input pins devoted to the address; the row address and the column address each have 9 bits. The multiplexing of address inputs saves valuable pins on the chip, but at the cost of considerable additional complexity in the timing of the memory. The multiplexing of addresses permits physically smaller integrated circuit packages, which saves valuable space on printed circuit boards having many RAM chps. We discuss aspects of RAM memory timing later in this chapter.

Memory system organization. Just as RAM chops have internal bus stucture, so is each chip edesigned to be a component of a bus-organized memory system in which only one unit of memory is available at any time. Suppose

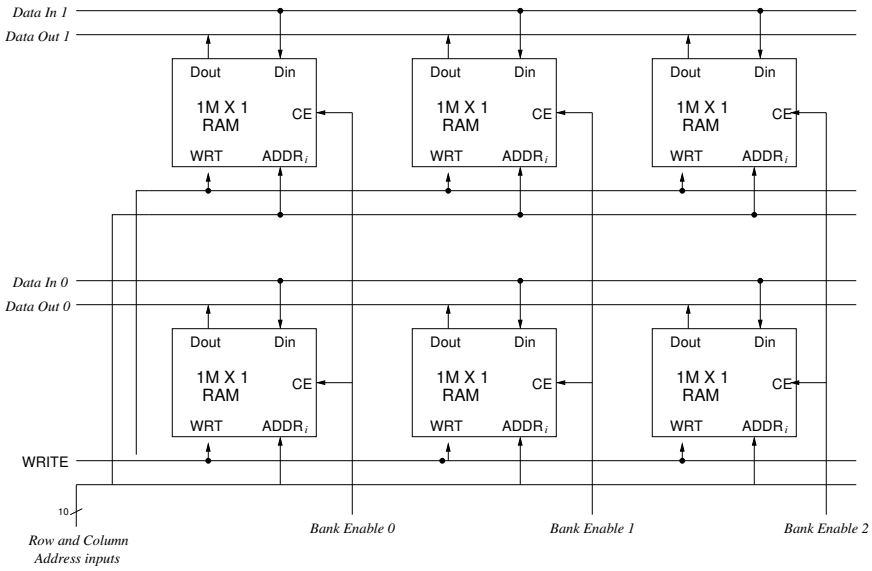


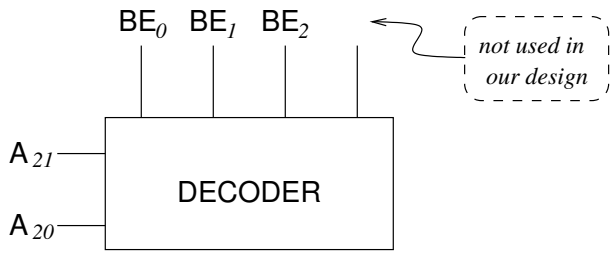
Figure 5.20: A 3M-word by 2-bit memory constructed with 1M×1 RAM chips.

we are given the task of building a 3M×2 memory (3 million words, 2 bits per word), using 1M×1 RAM chips. We would create a pair of system busses, data in and dat out, and hang the memories ont the busses as shown in in Fig. 5.20. As in all bus-organized systems, we must have some way of selecting a given element while ensuring that all other devices stay off the bus. RAMs provide a *chip enable* (CE) for this purpose. The system in Fig. 5.20 enables RAMs in pairs, since the original specification called for 2 bits per word. The input and output busses may be either open-collector or three-state, depending on the tipe of the chip. For most applications, we prefer the three-state systems because they are faster and eliminate the open-collector's pull-up resistors. Having developed and architecture for our memory system, we must now determin how to address a particular work in the memory. To derive this address, we shall back away from the chips, look at the memory system as a whole, and ask how we specify a location within the system. Our memory device will need a 22-bit address, sincd $2^{21} = 2M$ and $2^{22} = 4M$. Each 1-megabit RAM chip requires 20 bits of address; the remaining 2 bits will specify which of the three *banks* of 1M-bit RAM chips is being selected. Although the partition of the address is arbitrary, it is comon to select the bits for the bank field from the most significant positions in the address. Since 1M-bit RAM chips require multiplexed address inputs, the 20 bits of RAM chip address must be presented as a 10-bit row address followed by a 10-bit column address. The division of the 20 bits of chip address into row and column address is arbitrary, and is often decided

by the ease of layout on the printed circuit board. Here is one straighforware choice for the sys5tme memory address:

$$\begin{array}{lcl} \textit{System Memory Address}[A_{21} \cdots A_0] & = & [A_{21}\,A_{20}]\,[A_{19} \cdots A_{10}]\,[A_9 \cdots A_0] \\ & & \begin{array}{ccc} \text{Bank} & \text{Column} & \text{Row} \\ \text{Address} & \text{Address} & \text{Address} \end{array} \end{array}$$

We can now rout the 10 RAM address lines to all RAMs in parallel, and use the bank address $A_{21}\,A_{20}$ as a code for the proper bank. A decoder can produce indifidual enable signals for each bank from the 2-bit bank address code:



BE_0 enables the chips in bank 0, BE_1 enables the chips in bank 1, and so on. A bank contains two RAM chops, since there are 2 bits per word. (The row-select and column-slelct cotrol lines are not shown in the diagram.)

The last system-wide signal is the read-write selector *WRITE*. RAMs can both read and write, so we must supply a logic signal for the operation to be performed while the chip is enables. By convention, $WRITE = F$ implies reading; $WRITE = T$ means writing.

RAM timing requirements. RAMS are unclocked (asynchronous) devices that require detailed specification of the timing of their control and data signals. Timing is a function of the internal technology of the chip, and vaires widely with the type of memory. Ro achive reliable RAME operation, you mus adhere figitly to the manufacturer's requirements. In this section we will present a fue of the major timing parameters. The crucial parameters are:

(a) The period of stability of the address lines.

(b) The period when the chip enable CE is active.

(c) The value of the *WRITE* signal.

(d) During RAM writes, the period of stability of thw write-data input.

(e) During RAM reads, the period of stability of the read-data output.

There are three parameters that characterize all RAMs. The *read cycle time* is the minimum time that must elaps after the start of a read operation until another operation may begin. This usually determines the time during which

the address must be stable. Read-cycle times may range from a few nanoseconds to several hundred nanoseconds, depending on the type of the chip. The *write cycle time* is the corresponding parameter for the write operation. It is usually similar in magnitude to the read-cycle time. *Read access time* is a measure of the time that must elapse after the start of a read operation before the read data is available for used. The access time for read is equal to or less than the cyclt time for read.

For RAM write operations, the data sheed specifies when the WRITE signal may becme true relative to the address, chip enable, and data signals, and for how long WIRTE must remain stable to complete the write operation.

Besides these fundamental measures of RAM performance, the data sheets usually contain a welter of other timing figures, specifying times betwein various possible signal changes. You can simplify all this by making such reasonable design assumptions as:

(a) Chip enable (CE) becomes true at the sme time as the address becomes stable.

(b) During a RAM write operation, the data to be written stabilizes at the same time as the address lines.

With these assumptions, provided you choose the most conservative values for the timing parameters, you can usually reduce the number of relevant timing figures to a handful.

RAM timin parameters fall loosely into two groups: setup times and hold times. *Setup* implies that a signal mus be stable prior to some event; it is similar to the setu time for inputs to clocked circuits. *Hold* time means that an input must remain stable stable for a time after some event.

Read access time is an example of a setup time; it describes the required period of stability of the address prior to the apperarnce of stable data at the output. data sheets also contain setup times for chip enable. The intervals during which address, chip enable, and data must remain stable after a WRITE becomes false are examples of hold times. COnversely, in RAM reading, the data is often stable for a period after the address or CE changes; such a period is also called a hold time.

In using RAMs in digital designs, you must remember that there will also be delays in the host system. These will arise from combinational propagation delays, bus driving delays, and so on. The RAM timing computations begin with the arival of stable sighnals *at the RAM*, so you must tak into account the delays in the host when you determin how fast your system will really be.

Static RAM. The *static RAM* contains bit cells that are similar to a type D flip-flop, similar enough that you may use the D dlip-flop as a model for predicting static RAM behavior. As long as the address is constant and the WRITE lind is false, the selected cell will continu to put its read data onto the output bus. A transition from fals to true on the WRITE line serves as a clock signal to the RAM to caus input data to be written into the specified

RAM cell. As long as the power is on, the static RAM will retain its contents without the need for intervention. THe address lines of most static RAMs are not multiplexed—the entire address is presented to the chips, considerably simplifying the RAM control circuitry. These properties make static RAMs simpe to incoprorate into designs, once you have mastered the basic timing requirements. We will use a static RAM for the memory in our minicomputer design in Chapters 7 and 8.

Dynamic RAM. This device stores data on a tiny capacitory within each bit cell. The dynamic storage cell is much smaller than a cell in a static RAM, typically allowing four times as many bits per unit area of silicon. This factor of 4 becomes of overwhelming importance in large memory systems and nearly all large systems use dynamic RAM chips. The penalty is a significanly increased complexity of the control of the memory; nevertheless, you should consider dynamic RAMs for systems that would require more than, say, 32 static RAM chips.

Most dynamic RAMS have such a large storage capacity that their address inputs are multiplexed to save pins. The multiplexing complicates control of the RAM—each half of the address must be presented separately to the RAM. The RAM has addutional *row address strobe* and *column address strobe* control pins, which the circuit that controls the memory must assert at the proper time in the memory cycly to announce that the tow address or column address is stable on the address input pins.

The storage element in a dynamic RAM is a capacitor which, like all capacitors, is an imperfect holder of charge. After a period of time—a function of the temperature and geometry of the device and of circuit technology the charge will leak away. To preserve the data, the system using a dynamic RAM must periodically read and rewrite the stored contents. This periodic restoration is called *refreshing*: the entire memory must be refreshed at intervals of several milliseconds. Dynamic RAMs allow an entire column of bits to be refreshed an once. A typical RAM controller will cycle through the clumn addresses, inserting a refresh sycle at regular intervales so that each column is refreshed in a timely fashion. The control circuitry of dynamic RAMs may be contained within each RAM chip of in a separate controller chip.

5.5.2 Read-Only Memory

Read-only memory (*ROM*) is actually a write-once, read-thereafter memory. SInce the wirting takes place during the manufacture of the chip, large numbers of identical ROMs can be fabricated at low cost. Changing the write patter is very expensive, and ROMs are therefore only appropriate when we can amortize the initial cost by purchasing several thousand identical chips.

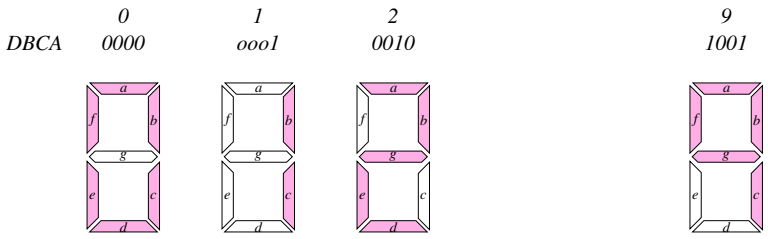
The bussing and timing requirments tend to be similar to those of the static RAM, with the omission of the now-superfluous read-write line.

The ROM has important uses in digital design as a permanent memory, a coded converter, and, surprisingly, a logic function generator.

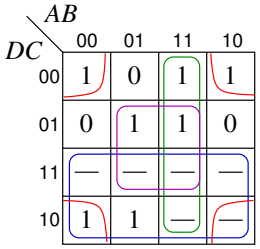
Firmware memory Consider the ubiquitous pocket calculator with transcendental functions such as quare rood and trigonometrics. A debugged square-root function need never change, and could therefore be committed to ROM and made part of the calcularot's address space. Such an application is called *firmware*. ROM firmware remains in the memory when the power is off, and is ready to use as soon as pwer is restored.

Code converstion. As you learned in Chapters 2 and 3, we may use conventional Boolean algebraic techniques to synthesize outputs from inputs. In a sense, we are using gates to *compute* the ouptuts. Consider Exersise 1-36, in which a 4-bit BVCD code generates the outputs to drive a seven-segment lamp. To aid your recollection, we will sketch to Boolean algebraic solution:

(a) List the BCD bit patterms for digits 0 through 9 and draw the corresponding segments to be lit for each digit:



(b) Plot each setment a thoufh g on a 4-bit Karnaugh map with the unused cotes 10 through 15 plotted as "don't cares." Derive equations for each of the seven segments. For example, the result for setment a is



$$SEG.a = \overline{A} \cdot \overline{C} + A \cdot B + A \cdot C + D$$

(c) Assemble gates to compute the functions a thought g. In other words, synthesize logic circuites corresponding to the equations for segments a through g.

Viewing the problem in this light, we have a large combinational logic circuit that accepts four inputs and computes seven outputs. We may package this as a building block if we wish; such a seven-segment lamp driver is available as an MSI chip.

From another viewpoint, the seven-segment lamp driver is a problem in code conversion, in which we mus convert a 4-bit BCD code into a 7-bit drive code.

	Inputs				Outputs						
Row	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0	-	-	-	-	-	-	-
11	1	0	1	1	-	-	-	-	-	-	-
12	1	1	0	0	-	-	-	-	-	-	-
13	1	1	0	1	-	-	-	-	-	-	-
14	1	1	1	0	-	-	-	-	-	-	-
15	1	1	1	1	-	-	-	-	-	-	-

Table 5.1: Truth Table for a BCD-to-Seven-Segment Code Converter

A seven-function truth table with four inputs and seven outputs is a convenient way to display the code conversions; refer to Table 5.1. We may consider the truth table as representing the contents of an addressable memory; each row of the truth table is an address, and the corresponding set of output values is the contents to the addressed memory element. Table 5.1 then represents a 16-word memory, with 7 bits in each word. The outputs (the contents of the memory) do not change, so we can realize the truth table as a 16-word by 7-bit ROM; such a ROM requires four address inputs: the BCD code.

The conversion of a 7-bit ASCII code to the corresponding 12-bit Hollerith card code can be accomplished with a ROM. The ASCII code contains 128 characte5rrs; each charcter has a Hollerith representation as a set of holes (1's) and blanks (0's) in the 12 rows of a card column. The ROM wil contain $2^7 = 128$ words, each of 12 bits. Every ASCII code represents the address of one ROM word; the content is the corresponding Hollerith code. Each word of ROM contains a unique code.

ROM may also be used to convert a valid 12-bit Hollerith code into 7-bit ASCII. This ROM has 12 address input and consists of $2^{12} = 4,096$ words of 7 bits. Of the 4,096 rows in the Hollerith-to-ASCII truth table, only 128 are relevant; the resto correspond to don't care outputs, present in the ROM but of no interest in this code conversion.

Generating logic functions. The foregoing descriptions show that a ROM provides a general way to generate an arbitray logic function. Every logic funcgtion has a truth-table repressentation. A ROM gives the function value, T or F, for every row in some canonical truth table; the ROM explicitly contains each bit of information in the full truth table. In a syntheis of logic functions using gates, we explicitly use only a portion of the truth-table's information—the product terms leading to true values of functions (or, equivalently, those leading to false values). Furthermore, we often can simplify the logic expressions to reduce the number of terms.

Such simplifications are not useful, and in fact are disadvantageous, in ROM synthesis of logic functions. Although the ROM approach is completely general, it suffers severely ecause the size of the ROM is doubled by each additional input variable. In logi syntheis, ROMs are bes used in situations similar to code conversion, where highly encode information must be transformed to a large nubmer of output functions. Other, more appropriate devices are available to support the uniform syntheis of logic functions. We will discuss these devices later in this chapter.

5.5.3 Field-Programmable Read-Only Memories

In the preceding section you learned some of the uses of the ROM in digital design. A ROM must be programmed by the manufacturer and is therefor not a suitable tool in the developmental phases of a design, nor in systems in whech the read-only material must occasionally be altered. When only a few copies of a system are required, or occasional changes in the memory are required, we need permanen or semipermanent memories that can be programmed by the designer in the laboratory. Several types of *programmable memories* are available.

PROM The *programmable read-only memory*, or *PROM*, is a ROM in which the one-0time writing process has been deferred to the end user. During manufacture, the bits of the PROM have microscopic metallic or polysilicon fuses that set all the bits to 1. The user can blow these fuses by selecting a given bit and then applying a pulse to a special programming pin, thereby creading a 0. The process is inexpensive and relatively easy, and PROM programming devices are readily available. In use, the access to a PROM is rapid, equivelent in speed to a ROM. PROMs are also physically similar to ROMs, and the layout of the circuit board is simplified in systems that will eventuall contain ROMs. Once the PROM is programmed, it cannot be reprogrammed.

EPROM The *erasable programmable read-only memory* (*EPROM*) is an even more flexible design based on the ROM. As its name implies, the EPROM allows the designer to erase the contents and start over. Bits are stored by electric charges that are trapped in the silicon of the chip. When erased, all the bits of the EPROM becom 1's. To write a zero value into a bit, the bit receives a hig voltage value that temporarily mkes the silicon a conductor, alowing charge

to accumulate at the site of the bit. After the hight charge is removed, the trapped charge remains, essentially forever. EPROMs are programmed with PROM programmers similar to, but more complex than those used to program PROMs.

A transparent quartz lid covers the silicon of the EPROM chop. Erasure occurse when high-intensity ultraviolet light makes the silicon a weak conductor, allowing the trapped charge to bleed slowly away from the bits in the memory. To be erased, the chop must be placed in an *EPROM erasor* and exposed to ultraviolet light for about a half-hour. The EPROM will withstand many erasures before it becomes unreliable.

The EPROM is convenient in system development because the designer may write the content of the memory, test the design and, if necessary, erase the old pattern and install a new one. The housing of microcomputer firmware is a common and important use of EPROMs, but we can use them with equal facility to generate digital logic functions. The device is slower than the PEOM, but its capacity for erasure makes it a potent design tool. EPROMs are physically larger than ROMs or PROMs, and have longer access times. Sizes range from 16K (2K×8) to 256K (32K×8)

EEPROM The *electronically erasable programmable read-only memory* (*EEPROM*) may be altered without removing it from the circuit in which it is used. Sequences of voltages of 5V or greater are applied to special programming pins to permit the wrwriting of selected bytes of the memory. This device and its cousins are important in designes in which in-place "writing with difficultiy" is necessary—for instance in a unit housed in a remote area that must be manupulated at a distance. The structure and the pin assignment of EEPROMs are not compatible with other read-only devices, and only a meager selection of chops is available. A sypical size is 16K, es exemplified by the 2K×8 bit 2816 EEPROM.

As you have seen, the devices for "read-only memoyr," whether permanen or reprogrammbble, are used in data and progrm storage, in code conversion, and even to generate random logic functions, but the first two uses of ROMs are by far the most important. We shall now describ programmable devices suitable for general logic applications in digital design.

5.6 Programmable Logic

Programmable logic provides a systematic way to generate complex logic functions. Programmable logic devices allow the designer to create arbitrary sum-of-produc functions of many variables, in a highly structured and regular manner. The term *programmable* means that the functions may be specified after the chp has been manufactured.

Usually the programmin is accomplished by blowing (breaking) fuses along the chip's internal data lines, as is don3e in the PROM. The fuse is a narrow ribbon of metal or other conductor deposited during the manufacture of the chip, and it serves a role simlar to the three-state gate. When the fuse is intact,

it allows an input signal to prceed unimpeded; when the guse is blown, no signal can pass and the input is disconnected from the remainder of the circuit. Unlike the three-state circuit, once the fuse is blown, its input is forever disconnected.

As you saw in Chapter 1, it is always possible to express an arbitrary combinational logic function in sum-of-products form. This is the starting pint of the application of programmable logic. To construct an arbitrary sum-of-products functions, we need to be able to fdorm the required AND (product) terms, and then form the *or* of the product terms to create the approprate sum.

5.6.1 The PLA

The most general programmable logic structure is the *programmable logic array* (*PLA*), which allows the programming of arbitrary products and arbitrary sums. Within a VLSI design, the PLA is an important tool for creating complex circuits on a single chip. The PLA may also be packaged as an LSI-level integrated circuit, to be used at the same level of design that we are now studying. But becuase of its extreme generality, requiring many inputs, many outputs and much internal circuitry, the PLA is not widely available as a discrete integrated circuit.

Consider the programmable logic implementation of a single function of three variables

$$TEST = \overline{A} \cdot \overline{B} + \overline{A} \cdot A + A \cdot B \cdot \overline{C}$$

In this example, the programmable device must have at least three inputs and one output, and must have the capability of specifying at least three proeuct tuerms within its structure. Within the device, each input passes through a non-inverting and an inverting buffer, thereby providing the true and complemented forms of each input variable. Figure 5.21 is a schematic of the portion of the PLA required for this example. Every vertical line represents a product term, and each horizontal crossline is an input to the AND gate that forms the product. The horizontal line at the bottom represents the single sum of the produc terms; each product term is an input to the OR gate that forms the sume. Before programming, each crossing has a fuse that determines if the input will contribute to its product or sum. In Fig. 5.21, each vertical line initially generates the trivial case of

$$A \cdot \overline{A} + B \cdot \overline{B} + C \cdot \overline{C}$$

wich, of course, is identically false. The act of programming the PLA destroys the unwanted fuses. In Fig. 5.21. the large dots represent the remaining fuses; all the other fuses have been blown.

Usually, the designer wishes to create several functions using the same input variables. Figure **??** is a PLA pattern for producing four functions of five input variables. This PLA is capable of receiving up to six input variables, gnerating up to twelve different product terms, and producing up to five different sums of these products. On input, three product terms, and one sum output are unused. The equations implemented by this PLA are

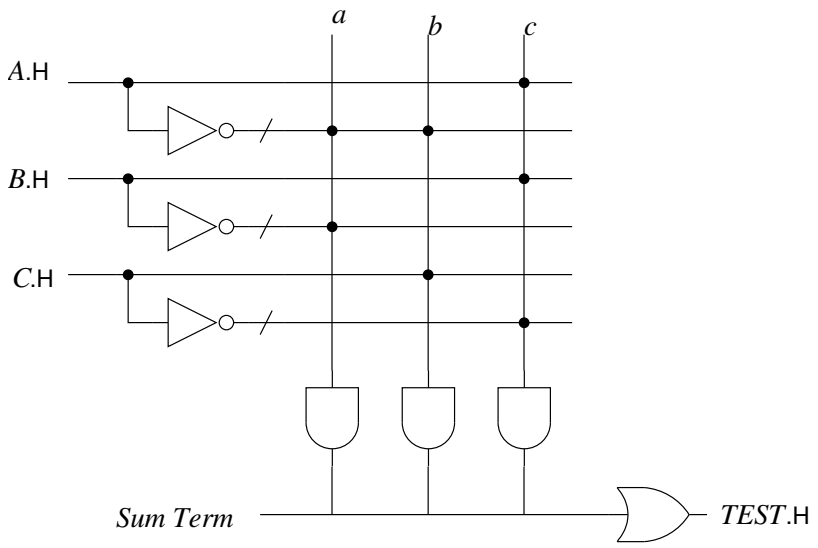


Figure 5.21: A PLA realization of $TEST = \overline{A} \cdot \overline{B} + \overline{A} \cdot A + A \cdot B \cdot \overline{C}$.

$$Y_1 = I_3 + \overline{I_1} \cdot I_2 + I_1 \cdot \overline{I_2} \quad (5.1)$$
$$Y_2 = \overline{I_3} \cdot I_4 \cdot \overline{I_5} + I_1 \cdot \overline{I_3} \cdot I_4 \quad (5.2)$$
$$Y_3 = I_3 \cdot I_4 \cdot \overline{I_5} + I_2 \cdot I_3 \cdot I_4 \cdot I_5 + \overline{I_2} \cdot \overline{I_4} \quad (5.3)$$
$$Y_4 = I_1 \cdot \overline{I_3} \cdot I_4 + \overline{I_2} \cdot \overline{I_4} + \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot I_3 \cdot \overline{I_4} \cdot I_5 \quad (5.4)$$

A typical PLA that is available as an integrated circuit is the National Semiconductor DM7575, which accepts 14 inputs, supports 96 product terms, and generates 8 logic functions as outputs.

5.6.2 The PROM as a Programmable Logic Device

We have alreadh mentiond that the programmable logic devices, exemplified by the PROM, can be used to generate logic fucntions. We usually think of the PROM as a memory device whocw n address inputs select one of 2^n words of memory. To perform this selection, the PROM use contain a complete decoding of its addrss inputs. Think of the truth table with the address lines as the inputs. For a PROM, the thruth table is canonical—it iexplcitly contains all 2^n rows. Thus, the PROM provides *all possible product terms* of its inputs, whether we like it or not. Each bit of the PROM's output can represent a logic function—a column on the output section of the truth table. To generate a logic function, we must specifiy whether each cononical product term is to be

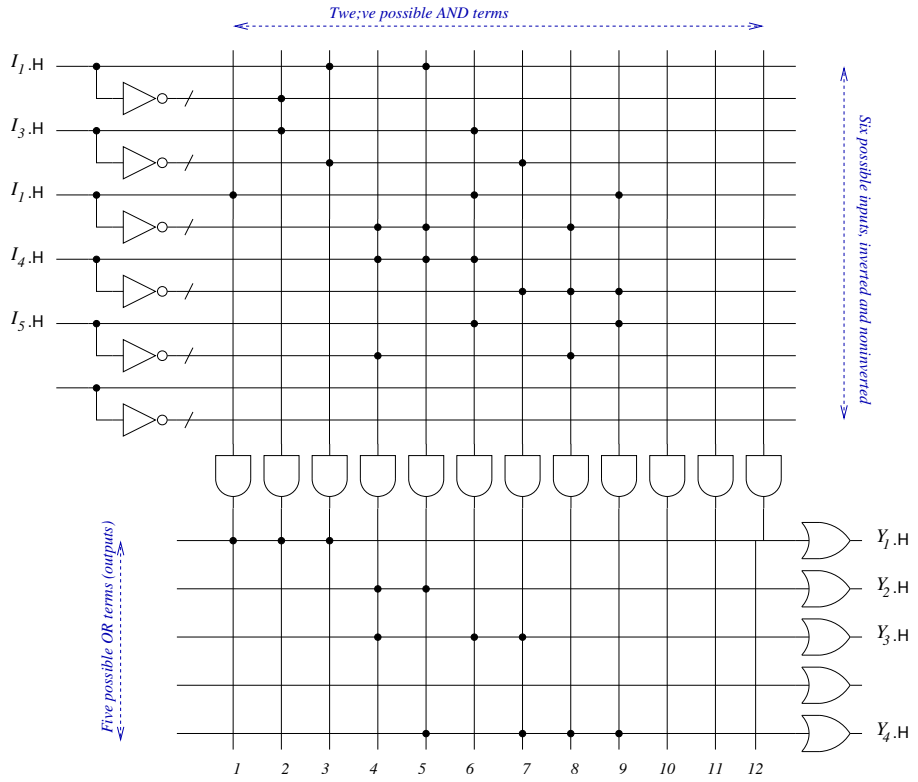


Figure 5.22: A PLA realization of Eqs. 5.1 to 5.4. The sixth input, the fourth output, and AND gates 10, 11, and 12 are not used. This PLA is smaller than commercial chips.

a contributor (the value in the memory is 1) or not (the value in the memory is 0). In such applications as code conversion, this is quite useful, since we are concerned with converting all possible patterns of input bits. For random logic, the PROM is of limited attractiveness.

Monolithic Memories, a leader in the development of programmable logic devices, has coined the term *PLE*, *programmable logic element* for such applications. Manufacturers produce a limited variety of PLEs, with a fixed array of ANDs (fixeed prduct terms) but with less than the full annonical set found in PROMs.

5.6.3 The PAL

The PLA generates a general sumg of gernal products; within the limits of the design, the dsigner may specify the detailes structure of each product term and each sum of products. However, the great generality of the PLA makes for difficultities in tis manufacture and use. The PROM (or PLE) provides a predetermined set of product terms, and the designer may specify the nature of each sum of products. This stucture us useful as a memory and for logic operations requiring the full decoding of inputs, but the PROM has limited application as a tool for progrmmable logic.

The *PAL* (*programmable array logic*) allows the designer to specify the nature of the produc terms, but the ways in which the pruducts may be formed into sums is fixed in the chip. In practice, the PAL is by far the most useful of the trio for generating random logic functions. The programmable product array, coupled with a limited summing capability, fulfills the requirements of digital designs well. The term PAL is a registered trademark of Monolithic Memories, which originated this application. PALs are available from many manufacturers in a wide variety of useful configurations. The PAL is indeed a pal of the designer of digital circuits.

The PAL18L4, shown schematically in Fig. 5.23, is typical of the PALs that generate combinational logic functions. As suggested by its number, the PAL18L4 has 18 inpuyt pins and 4 output pins. the "L" signifies that the outputs are produced in low-active ($\mathsf{T} = \mathrm{L}$) form. Two of the 4 outputs, pins 18 and 19, each provide the sum of six product terms. Within the chip, each of the 18 inputs is split into its tru and complemented form. Each of the twenty product terms available on this chip can contain any combination of the tru and complemented forms of the 18 inputs. Fuses appear at each intersection of the product-term lines. The programming task consists of blowing the unwanted fuses. Figure 5.24 shows an implementation of $Y1$, $Y2$, and $Y3$ in Eqs. 5.1 through 5.3, using the PAL18L4. Unused portions of the PAL do not appear in the figure. $Y1$.L, $Y2$.L, and $Y3$.L are formed as summs of products, drawn directly from the equations (the high-active $Y2$.H is discussed later). Each bold dot denotes a contribution to a product term, and represents a fuse that is to remain unblown. The *unused* product terms in a sum have been blacked in. Inputs of either voltage polarity aree easily handled, using standard mixed-logic notations. Within the chip, the PAL generates sums of products with $\mathsf{T} = \mathrm{H}$.

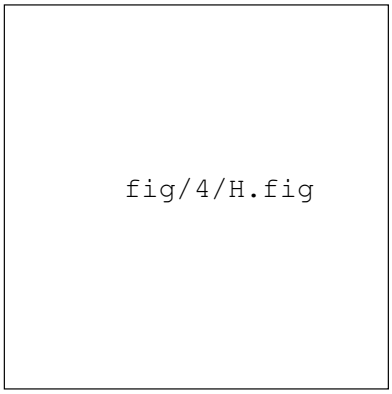


Figure 5.23: A PAL18L4. (Courtesy of Monolithic Memories, Inc.)

The manufacturer's diagram shows the input buffers with the inverted voltage form of the input emerging below the noninverted form in each buffer. If $\mathsf{T} = \mathrm{H}$ at the input, the ogic inversion occurs on the bottom of the two buffer outputs; if $\mathsf{T} = \mathrm{L}$ at the input, we redraw the input buffer to show that the logic-inverted form emerges from the upper buffer. Figure 5.24 contains illustrations of the notational changes to accommodate low-active inputs.

The PAL18L4 produces low-active sum-of-products outputs. If we desire a high-active output, we mixed logicians have several choices. We may add a voltage inverter to the output signal to change its ploarity—a reasonable but inelegant solution. If the PAL has an unused output and an unused input, we may feed the low-active form of our signal back into the PAL and produce the high-active form on the unused input. We may use a different PAL, such as a PAL18H4, that produces high-active outputs. Another approach, useful when one is striving to use minimal hardware, is to generate a sum-of-products form of the *inverse* of the desired function. The mixed logician immediately recognizes that the output represents a high-active version of the desired function

$$FUNCTION.\mathrm{H} = \overline{FUNCTION}.\mathrm{L}$$

For instance, suppose we wish to produce Y_2 of Eq. 5.2 in high-active form, using the PAAL18L4. To generate the inverse of Y_2, we may plot a Karnaugh

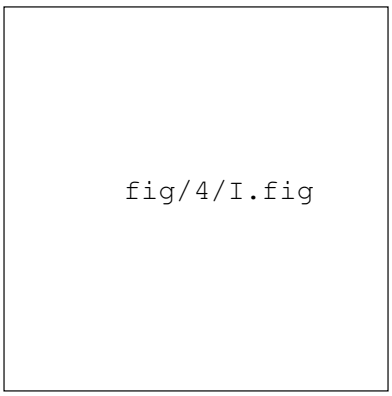
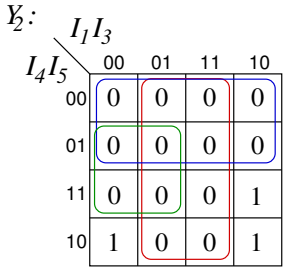


Figure 5.24: An implementation of Eqs. 5.1 through 5.3, using the PAL18L4. Note the mixed logic changes to the diagram.

map of the function, circle the 0s, and write an equation for $\overline{Y_2}$:



$$\overline{Y_2} = \overline{I_4} + \overline{I_3} + \overline{I_1} \cdot I_5$$

Figure 5.24 contains the resulting implementation of Y_2.H.

Many PALs are available with clocked flip-flops and three-state control at their outputs. An exampl is the PAL16R8, shown in Fig. 5.25. The "R8" in the chip number signifies a PAL with eight "registered" outputs. Each of the eight flip-flops in the register receives a clock signal from a single input pin, and each flip-flop's output is buffered by a three-state inverter controled by another input pin. The 16 inputs implied by the chip number arise from 8 external input pins and 8 signals fed back from the flip-flop outputs.

5.6.4 PAL, PLA, and PLE Programming

Programmable logic devices are a powerful toll for the designer, providing the equivalent of complex, tedious logic and architecture within compact and relatively inexpensive integrated circuit chips. These devices must be programmed and, with the exception of reprogrammable devices, such as EPROMs and certian PALs, the programming is a one-time, unalterable act. The programming

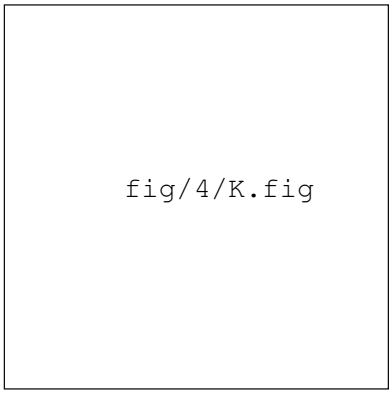


Figure 5.25: A PAL18L4. (Courtesy of Monolithic Memories, Inc.)

requires special equipment for the presentation to the chip of a complex sequence of properly timed voltages. Modern "programmers" are costly, but they provide many necessary capabilities conveniently and reliably. Most such programming devices are capable of accepting programming data from an external source—-valuable when one is transmitting computer-generated programming files. The fuse patterns become the raw data for the programmer, but developing fuse patterns from diagrams such as Fig. 5.24 is a tedious and error-prone task. Many programmers for PALs permit the designer to enter logic equations directly; these equations, expressed in a suitable notation, provide the input to a software translator that produces fuse patterns.

Programmable logic devices are powerful design tools that provide some of the desirable attributes of VLSI for the chip-level designer without the extraordinary commitment necessary to develop a custome-made VLSI chip. The size and complexity of programmable logic devices are growing rapidly–PALs with64 inputs and 30 registered outputs, and containing more than 32,000 fuses are available. You will read about several applications of PALs and PROMs in later sections of this book.

5.7 Timing Devices

Such systems as RAMs and ROMs have rigid timing requirements that a designer must obey. In this section, we discuss two nondigital devices, the single shot and the delay line, that hel the designer to develop appropriate signals for timing.

We might use a single shot or a delay line when we must derive a timing event of arbitrary duration with respect to some arbitrary starting point. For instance, in RAM reading, we must produce a timing signal for the host system

that starts when the RAM read operation begins, and ends only after the read access time has elapsed. This time is independent of the systm clock; the RAM's timing requirements remain the same whether the system clock is slow or fast.

The idea is simple, but the simplicity of the concept does not mean that the corresponding hardware is simple to use. Another way of describing the arbitrary starting time and duration of the operation is to say that the timing is *asynchronous*. Sychnronous devices derive their timing from some external source, usually the system clock. Asynchronous devices are *internally* timed. This is not inherently bad, but the result is that independent timings are spread throughout the system.. We have relinquished central control, and for this reason along we should avoid asynchrnous devices unless absolutely necessary.

Centralized control is nice, but we cannot always have it. Memory devices, for example, require internal timing that is independent of the host system. We cannot do withoug devices for memory so we are forced to gener5ate their arbitrary timing locally. The alternative is distinctly worse; we could have cntral control if we rigitdly fixed our system clock to match the local ascynchronous timing requirements. This would eliminate our most powerful hardware debugging tool, the ability to slow the central clock to zero speed, thereby frezzing our system in a given condition.

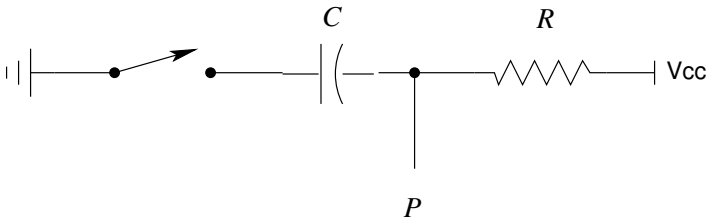
5.7.1 The Single-Shot

The *singel-shot*, or *monostable multivibrator*, is based on the electrical properties of capacitors. A capacitor stores a charge (electrons) as a function of the voltage impressed accross the capacitor. The amount of stored charg q is proportional to impressed foltage V; the proportionality constant is the *capacitance* C. Thus

$$q = CV$$

Capacitors require a period of time for the charge to build op or decay, and it is this behavior that allows the single-shot to function as a timed delay element.

A single-shot behavies like the electrical circuit below



Normally, the single-shot switch is open and no current is flowing. The voltage at point P is Vcc, which causes the the single-shot output Q to be false. When we trigger the single-shot by (digitally) closing the switch, charge begins to rush into the capacitor C. Because of this current flow through R, the Flotage at poin P becomes low, which in turn causes the singel-shot output Q (See Fig. 5.26) to assume a true value. As the flow of the current charges the capacitory, the

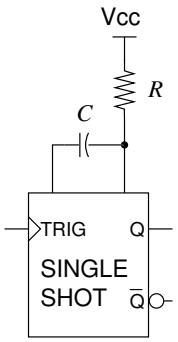


Figure 5.26: A single-shot circuit symbol.

foltage at P slowly rises back towarc Vcc. When the vlotage at P reaches a sertain level, the single-shot turns off, bringing output Q to false. The time during which Q is asserted is the single-shot delay time.

By choosing capacitor C and resistor R according to ttables or formulas in the single-shot data sneet, we may selet any desired single-shot delay time within a wide range of values. Typical delays, using the popular 96L02 Dual Single-Shot, range from about 50 naonseconds to 10 seconds.

In circuit diagrams, we deaw a single-shot as a rectangle similar to a flip-flop and add the external timing capacitor and resistor (See Fig. 5.26). Single-shots respond to a $F \rightarrow T$ transition on the trigger input, so they will start up if any glitch occurs at the input. Therefore, you must be careful to drive the single-shot with a clean trigger to avoid spurious results. The value of the singel-shot delay with drift somewhat, and 5 percent stability is about all you should expect.

5.7.2 The Delay Line

Delay lines, on the other hand, generate highly stable delays. Their stability derives from the fact that they are *passive devices*—they have no transistor amplifiers built into them, as do all gates, flip-flops, and so on. This explains their lack of power-supply pins, since with no active amplifiers they do not need operating power.

Delay lines are available in standard integrated circuit packages. Internally they are constructed from a series of stable inductors and capacitors. Whatever waveform is put into the delay line will appear, after a given delay, on the output. There is no concept of a trigger; the waveform at the input is simply reproduced after the delay.

In their most convenient form, delay lines have a seris of taps (pins), oftwn 10, that yield fractions of the nominal delay. For instance, ia 10-top delay line with delay rated at t_d, the nth tap will produce a delay of $t \times n/10$. SDuch taps are useful in generating the timing in dynamic RAMs, which require a sequence

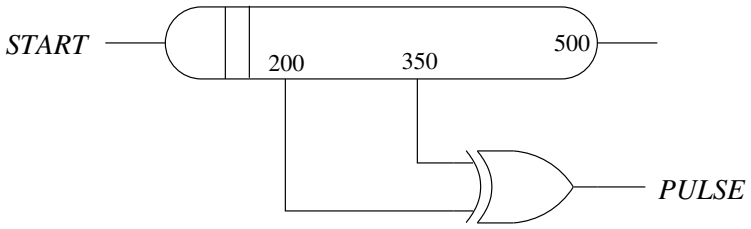


Figure 5.27: A circuit for a delay line to produce a 150-nsec pulse delayed by 200 nsec.

of accurately timed pulses of varying duration. We may obtain the pulses by sending and edge down the delay line and using an Exclusive-Or gate to detect the period during which the edge has arrived at the input but not yet reached the output. For example, suppose we need a pulse that 200 nanoseconds after *START* becomes true and lasts for 150 nanoseconds. Figure 5.27 is a diagram of the circuit, using the standard delay-line symbol.

Delay lines are moderately expensive but can generate delays from about 2 nanoseconds to greater than 1 microseccond. Since a delay line faithfully transmits the input waveform, you must be sure the signal at the input is clean. Often the operating specifications of the delay line will require that the input signal be derived from a line driver, and the the output be properly terminated (See Chapter 12).

5.8 The Metastability Problem

We began this chapter with a discussion of hazards, a nuisance created by the charcteristics of physical devices used to implement logical concepts. In Chapter 5 you will encounter other design pitfalls rooted in physical behavior—pitfals that arise through the interactions of several components of a design. In this chapter there remins to discuss the most alarming physical problem of all—metastability. We will alert you to the problem and give some advice, but you should look to Chapter 12 for a more extensive treatment of this topic.

Digital devices are fundamentally analog devices that behave digitally only when stringent rules of operation are obeyed. Sequential devices contain amplifiers (gates) and feedback loops to achieve their storage properties. In addition to establishing proper voltage levels at the inputs, to assure proper operation of a sequential device you must adhere to the setup times, hold times, and other timing specified in the data sheets. When the operation requirements are met, the device's outputs will be proper digital voltage levels, and changes in the level of the output will occur quickly and cleanly. Except during the rapid period of transition, the circuit remains in one of its stable states. You have seen that there are difficulties associates with the RS flip-flop when one tries

to move from the $R = S = T$ input configuration to the hold configuration, in which $R = S = F$. The difficulties arose from the attempt to change both inputs simultaneously. As long as no mor than one input is changing at a time, the sequential circuit performs well, but if the voltage level of mor than one input is allowed to change at nearly the same time, the circuit is being required to perform outside the framework of design for digital operation and the result may be unpleasant. For the peoper operation of clocked circuits, the setup and hold times require that certain inputs must not change too near the time that the clock signal is changing.

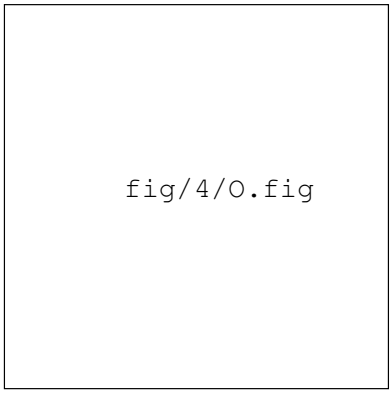
Violation of the timing requirements of a sequential circuit may throw the circuit into a *metastable state*, during which the outputs may hold improper or nondigital values for an unspecified duration. In one form of metastability, the output volgage lingers for an indefinite period in the transition region between digital voltagle levels, before it eventually resolves to a stable value. In another form of metastability, the output appears to be aproper digital value, but after an unpredictable interval switches to another value. Metastability can be disastrous. In synchronous design, we sidestep the problem by never changing the inputs in the vicinity of the clock. As you will see, this allows vast simplification of the design of complex circuits. Bot every circuit is at some point exposed to external reality—other circuits with different clocks, unclocked or nondigital devices, or human operators, for instance. Signals from such sources are not tied to our clock and may change at any time during our clock cycle. Therefore, although we can simplify our design by using good practices, no amount of digital or analog wizardry will eliminate the problem of metatability. However, by proper design or choice of components, we may lower the problability of finding the circuit in a metastable state to a satisfactory level. In Chapter 12, we discuss metastability in more detail and offer guidelines for dealing with the problem.

5.9 Conclusion

:You have completed Part I of this book, in which we have explored the fundamental tools underlying digital design. From basic combination circuites we have developed a set of building blocks that range from simple logic gates to complex ALUs, from flip-flops to large memories. Now you are ready to begin the exciting activity of digital design. Part II introduces you to theis process.

5.10 Exercises

1. (a) Show that the following combination circuit contains a hazard.

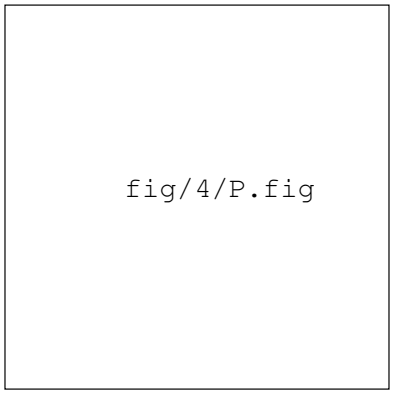


   (b) Write the logic equation corresponding to the circuit, and draw a K-map with circles corresponding to the circuit.

   (c) Most of the time our design techniques will nullify the bad effects of hazards; nevertheless, suppose that you must eliminate the abolve hazard from the circuit. Starting with the K-map you drew for part (b), produce a hazard-free map by making certain that adjacent 1s share at least one circle. Write the logic equation and draw the hazard-free circuit.

   (d) Prove, by using a timing diagram, that your new circuit is free of hazards.

2. Assume that each combination circuit element has a propagatoin delay of t_p. What is the total (worst-case) propagation delay in the following circuit?



3. In Fig. **??**, the circuit for the enabled multiplexor emposes the enablin operation on each of the initial AND gates, forcing them to have three

inputs. Suggest why, in Fig. **??**, the enablin operation was not designed as a single final AND gate with only two inputs.

4. A circuit consisting of a closed loop of an odd number of inverters (greater than 1) can function as an oscillator. Assum that the propagation delay through an inverter is 10 nanoseconds.

   (a) With a timing diagram, show the oscilatory behavior of a loop of three inverters.

   (b) The oscillator consisting of a loop with just a single inverter is not stable. Speculate about why the circuit is unsatisfactory.

5. What is feedback in digital design? Draw a gate circuit that exhibits feedback with memory.

6. Why are combination methods inadequate to deal with sequential circuits?

7. Explain "1's catching." Why is this behavior usually a disadvantage in digital design?

8. Explain the terms *asynchronous* and *synchronous*.

9. Show that the as;ynchrnous RS flip-flop has two stable states.

10. Why do we usually avoid asynchnronous flip-flops in digital design?

11. What is switch debouncing? Why can we usually not use a mechanized switch directly in digital design? Draw a switch debouncing circuit.

12. Using a timing diagram, analyze the behavior of the switch debouncer shown in Fig. **??**a or **??**b.

13. Assume that two (noisy) mechanical switches generate the *DATA* and *HOLD* signals for the latch in Fig. **??**. Is there any sequenc of switch closings and openings that would yield a clean output signal at Y?

14. The RS flip-flop exhibis anomalous output behavior if both R and S are true.

   (a) What is the anomaly?

   (b) Does the anomaly occur in outputs X and Q of Fig. **??**?

   (c) In Fig. **??**, assume that $R = S = T$. What is the value of Q if both signals become false, but R becomes false slightly befor S?

   (d) Under similar conditions, what value does Q assum after precisely simultaneous $T \rightarrow F$ transitions of R and S?

15. What is an edge-driven flip-flop? Why is it desirable? What is the deflect in the master-slave flip-flop? What is a pure edge-driven flop-flop? What kind of flip-flop do we use in digital design?

16. Consider and edge-driven JK flip-flop such as the 74LS109 with direct set input and the K input asserted (true), and the direct clear input and the J input negated (False). What will be the flip-flop's output shortly after the next active clock edge arrives?

17. Suppose your design requires a 74LS109 JK flip-flop with output *FLG*. You with to set the flip-flop with a logic variable *SETFLG* and clear the flip-flop with a variable *CLRFLG*. In the design, you find that the control inputs are available as *SETFLG*.L and *CLRFLG*.H. Draw the desired circuit with a 74LS109 flip-flop using no inverters. Draw the mixed-logic diagram.

18. Repeat Exercise ?? under the condition that both *SETFLG* and *CLRFLG* are available with $\mathsf{T} = \mathrm{L}$, and inverters are available.

19. The text describes three cases in which the JK flip-flop may be used to store a bit. Two of these cases are (a) clearing, followed by later setting if the data bit is true; (b) setting, followed by later clearing if the data bit is false. Verify the text's rules for implementing these two cases.

