CHAPTER

FOUR

MORE IMPLEMENTATIONS

In the last chapter, we studied several MSI circuits that also happened to be ASMs themselves. In this chapter, we'll look at the counter circuit both because of its own importance as a circuit element and because it will serve as an example to introduce new topics. These topics are JK flip-flops, ASM coupling, hang-up states, and initialization.

COUNTERS AND FLIP-FLOPS

A counter is designed exactly like any ASM. The ASM chart for a simple eight-state counter is shown in Fig. 4-1. The outputs and state variables have been made identical to eliminate the need for any output functions. The outputs are simply in each state to eliminate the need for any output functions. The outputs are simply the state variables. In a ROM implementation, only next-state ROM outputs would be required. In a gate implementation, only next-state gating would be required. This counter has no enable or other inputs. It counts continuously as long as clock signals are applied. The circuit for this counter, using D flip-flops, is shown in Fig. 4-2. Even in this simple 3-bit counter, the next-state gating is becoming complex, especially for the more significant bits. In fact, counters implemented with D flip-flops having more than 4 or 5 bits require intolerable amounts of gating. Another type of flip-flop, called a *JK flip-flop*, can be used to reduce the number of gating circuits required.

JK Flip-Flops

The symbol for a positive edge-triggered JK flip-flop is shown in Fig. 4-3. The abbreviated Next-State Table 4-1 is easily understood; so we'll use it to explain the operation of this circuit. The table heading Q^{t+1} is the Q output after the clock transition,

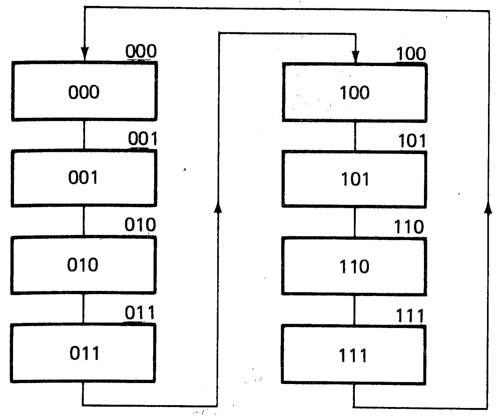


Figure 4-1 ASM chart of counter.

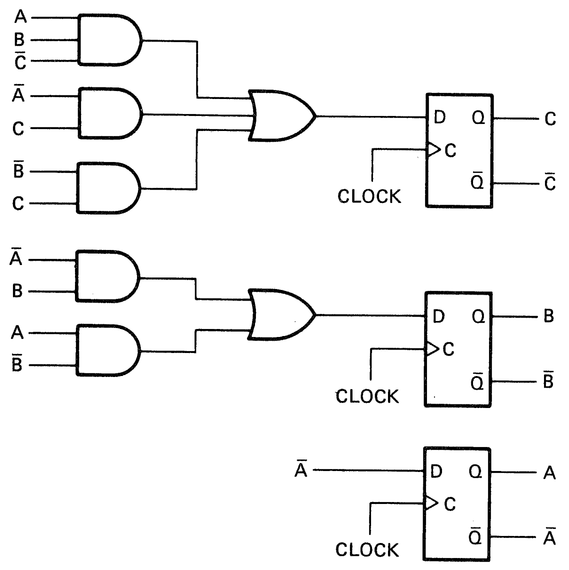


Figure 4-2 Counter with D flip-flops.

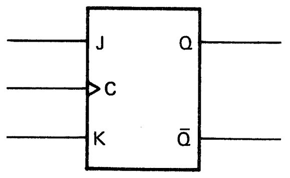


Figure 4-3 Symbol for edge-triggered JK flip-flop.

Table 4-1 Next-state table for JK flip-flop

J	K	Q^{t+1}
0	0	Q^t
1	0	1
0	1	0
1	1	$\overline{Q^t}$

while the entry Q^t represents the Q output before the clock pulse occurred. If both the J and K inputs are 0 when a clock transition occurs, the Q output remains the same, $Q^{t+1} = Q^t$. If $J = 1$ and $K = 0$, the Q output will become 1 at the next clock transition. If $J = 0$ and $K = 1$, the Q output will become 0 at the next clock transition. Finally, if both the J and K inputs are 1, the output will assume the opposite state at the clock transition, $Q^{t+1} = \overline{Q}^t$. This last behavior, changing to the opposite state regardless of the current state, is called *toggling.* A flip-flop is often required to toggle so that the connection of a JK flip-flop in Fig. 4-4 is also called a *type T* or *toggle flip-flop.* A toggle flip-flop remains unchanged as long as its T input is a 0 and toggles to alternate states on each clock transition when its T input is a 1.

You can already see that the JK flip-flop is more versatile than the D flip-flop. Only two operations are possible with the D flip-flop. These are setting the D flip-flop's output to 1 and resetting its output to 0. Four operations are possible with the JK flip-flop. Besides the operations of setting or resetting its output at each clock transition, the JK flip-flop may also toggle or remain in the same state. It is these two additional operations that we will use to our advantage.

In Table 4-2, we've tabulated the inputs necessary to cause each possible next state for each possible current state. If the Q output is 0 and we desire that it remain 0 after the clock transition, we have two choices of inputs. If we select inputs $J = 0$

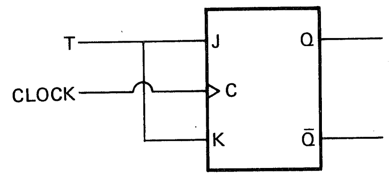


Figure 4-4 Type T flip-flop.

Table 4-2 JK inputs required for flip-flop state state changes

Original state	Next state	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

and $K = 0$, the flip-flop will remain in its original state of $Q = 0$. If we select inputs $J = 0$ and $K = 1$, the flip-flop will reset to state $Q = 0$. Either way, the state after the clock transition will be $Q = 0$. Notice that all four possible state transitions may be accomplished in two ways. These alternatives will allow us to enter don't cares on our K maps, which may allow us to simplify the next-state gating.

Designing with JK Flip-Flops

The next-state table for the simple 3-bit counter of Fig. 4-1 is shown in Table 4-3. The next-state columns are not marked D_C, D_B, and D_A because we shall use JK, not D, flip-flops to implement this counter. For simplicity, we'll consider only the next-state function of flip-flop C, as shown in Table 4-4. We've listed the current state and the C flip-flop's next state. We desire to specify the J_C and K_C inputs to flip-flop C. At state 0 on the table, flip-flop C is a 0 and must remain so. Table 4-2 indicates that the J input must be 0 and K can be a don't care in order to keep the flip-flop output at 0 if its current output is 0. We enter those inputs in the line of Table 4-4 corresponding to the transition from state 0. At state 3 in Table 4-4, a 0-to-1 transition of flip-flop C

Table 4-3 Next-state table for eight-state counter

Current state			Next state		
C	B	A	C^{t+1}	B^{t+1}	A^{t+1}
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

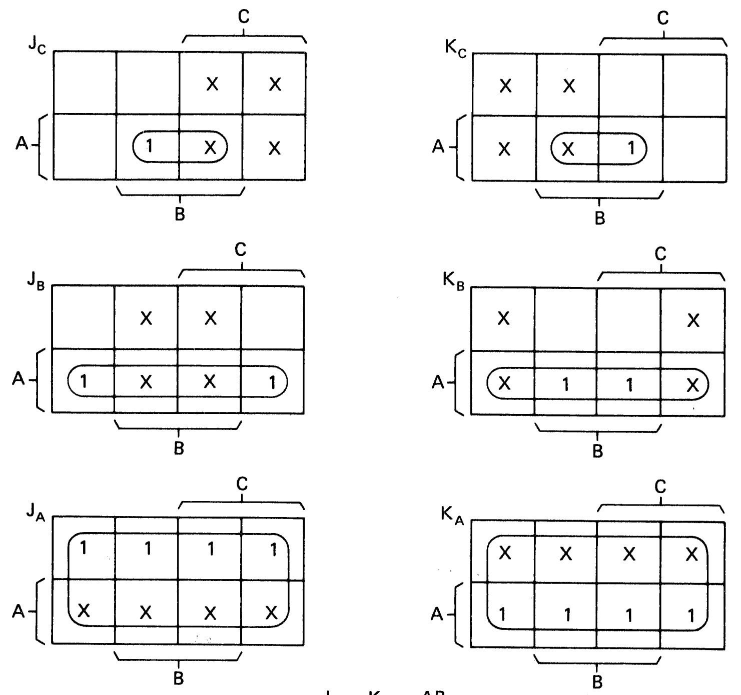
Table 4-4 Determining JK inputs for flip-flop C

Current state					
C^t	B^t	A^t	C^{t+1}	J_C	K_C
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	0	0	X
0	1	1	1	1	X
1	0	0	1	X	0
1	0	1	1	X	0
1	1	0	1	X	0
1	1	1	0	X	1

Table 4-5 Determining JK inputs for eight-state counter

Current state			Next state			Flip-flop inputs					
C^t	B^t	A^t	C^{t+1}	B^{t+1}	A^{t+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

causes us to enter $J_C = 1$ and $K_C = X$. At state 4, the C flip-flop remains a 1 and we enter $J_C = X$ and $K_C = 0$. At state 7, a 1-to-0 transition of flip-flop C requires an entry of $J_C = X$ and $K_C = 1$. In this way, we can determine the correct logic function for each of the six inputs of the three flip-flops, as shown in Table 4-5. Each of these functions is transferred to a K map in Fig. 4-5. These functions may be considerably simplified because of many don't cares. Also, the J and K inputs of each flip-flop just happen to be equal to the same boolean function! The counter of Fig. 4-6, using JK flip-flops, is considerably simpler than the equivalent counter of Fig. 4-2, using type D flip-flops.



$J_C = K_C = AB$
$J_B = K_B = A$
$J_A = K_A = 1$

Figure 4-5 Karnaugh maps for JK inputs.

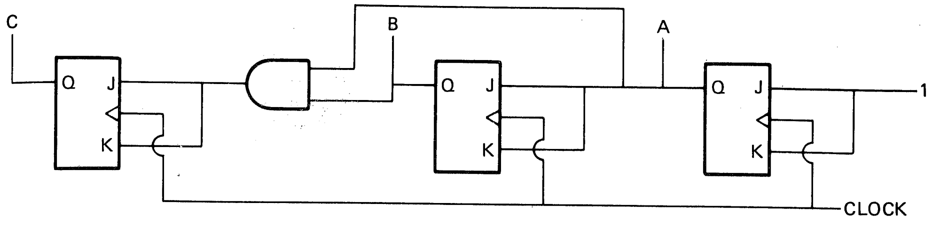


Figure 4-6 Eight-state counter using JK flip-flops.

Another JK Counter

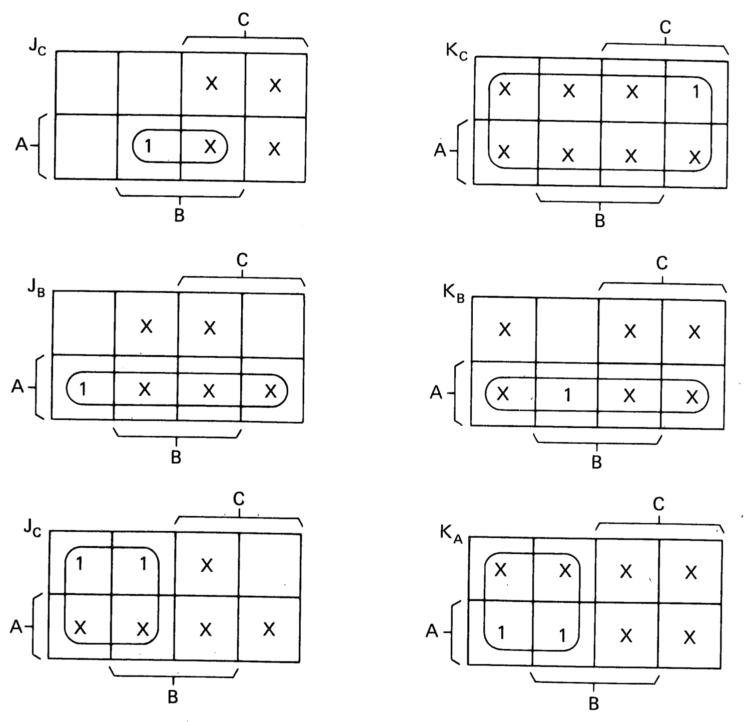
Every JK implementation is not as simple and symmetric as the example above. The above example was particularly simple because it was a binary counter and had a total number of states equal to a power of 2. However, many JK implementations are often simpler than D implementations. Let's consider a binary counter with five states. Its next states and the J and K inputs needed to establish them are tabulated in Table 4-6. Transferring these functions to the K maps of Fig. 4-7, we specify states 5, 6, and 7 as don't cares since these states are unused. The circuit of Fig. 4-8 is a five-state counter using JK flip-flops. Although not as simple as the eight-state counter, this circuit is simpler than a D flip-flop implementation.

JK Flip-Flops and the General ASM

The JK design technique described above for ASM counters applies equally well to any ASM. The use of JK flip-flops will often lead to simpler next-state gating than an equivalent D flip-flop implementation. In the previous counter examples, no gates were required to generate the output functions. When output gating is required, it will be identical in both the JK and D implementations. Finally, JK flip-flops are almost always *undesirable* when used with a ROM for generating next-state functions. The JKs

Table 4-6 Determining JK inputs for five-state counter

Current state			Next state			Flip-flop inputs					
C^t	B^t	A^t	C^{t+1}	B^{t+1}	A^{t+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	0	0	0	X	1	0	X	0	X



$J_C = AB$
$J_B = A$
$J_A = \bar{C}$

$K_C = 1$
$K_B = A$
$K_A = \bar{C}$

Figure 4-7 Karnaugh maps for five-state counter.

will require twice the number of next-state outputs from the ROM than the D flops require. The cost of the ROM is not dependent on the function implemented, but does increase as the number of output bits increases. The JK implementation's don't cares are of no use with the ROM.

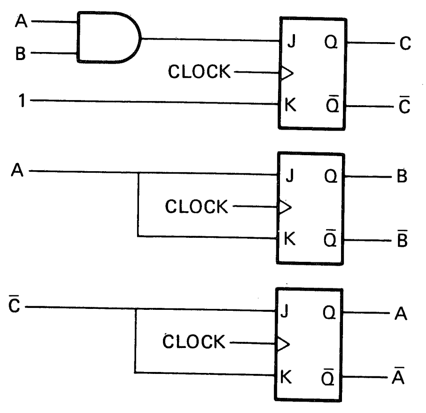


Figure 4-8 Five-state counter.

COUPLING ASMs

ASMs must often be coupled or interconnected. Designing a very large digital system is simplified by dividing the system into smaller parts, each of which is a separate ASM. The method of interconnecting ASMs that we've been using is called *synchronous coupling*. We have simply connected the output of one ASM to the input of another. Figure 4-9 shows two 4-bit counters coupled this way. The carry output of the least significant counter is connected to the enable input of the most significant counter. If these counters are binary counters, the carry output will be 1 when the least significant counter is 1111. This enables the most significant counter so the same clock pulse will reset the least significant counter to 0000 *and* make the most significant counter count up by one. *Synchronous coupling is usually the best way to couple ASMs.*

Synchronous coupling does have drawbacks. First, extra inputs are required. Each extra input doubles the size of the ROM in a ROM implementation or complicates the gating in a gate implementation. Second, cascading multiple ASMs in this way *may* slow the maximum rate at which they will operate. Consider the counters of Fig. 4-9. Each time a carry output is generated, it is delayed by the output logic of the following counter. The carry output of the last counter in a 12-bit cascade will be delayed by 3 times the propagation delay of a single counter's output logic. Since the next clock transition must not occur until the last carry is correctly generated, these propagation delays limit the speed at which this counter will operate.

Several alternatives to synchronous coupling exist. The important alternatives are *asynchronous coupling* and *gated-clock coupling.*

Asynchronous Counters

Even using JK flip-flops to advantage, synchronous counters of many bits will have complex gating structures to generate the next state. Studying the timing diagram for a

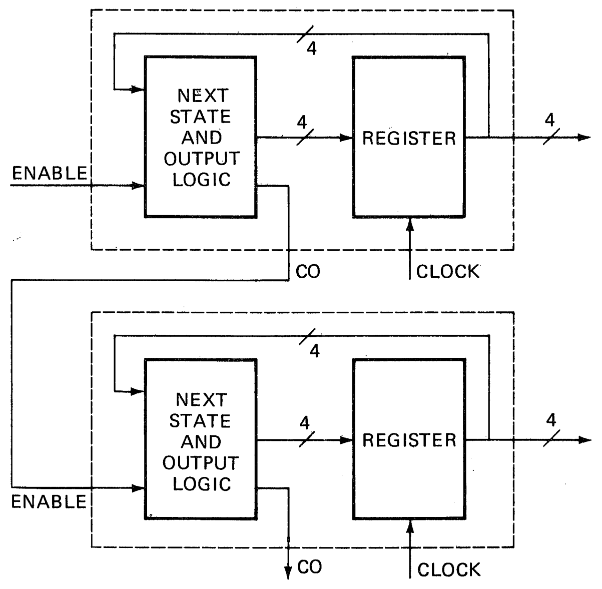


Figure 4-9 Synchronous coupling of MSI counters.

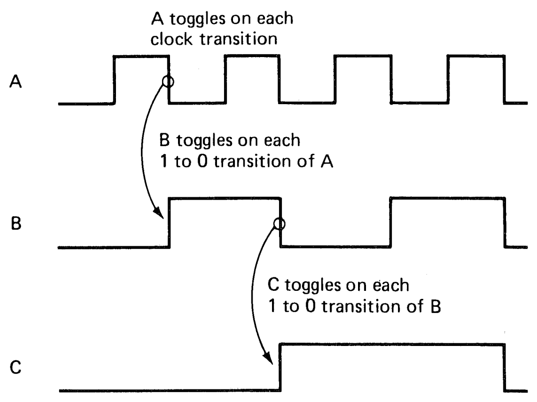


Figure 4-10 Binary counting sequence.

binary counter in Fig. 4-10, note that each bit toggles at the same time the next least significant bit makes a 1-to-0 transition. If we use a 1-to-0 transition of a bit to toggle the next most significant bit, we should be able to build a counter requiring no gating! We will use transitions of outputs to clock flip-flops, rather than a common clock signal. A circuit in which all flip-flops do not have their clock lines driven in synchronism is called an *asynchronous* circuit. A 3-bit counter using this technique is shown in Fig. 4-11. This asynchronous counter consists of three JK flip-flops connected as type T or toggle flip-flops. A 1-to-0 transition on any Q output causes the next most significant bit to toggle. The $\overline{Q}$ output is actually connected to the next most significant clock input because these positive edge-triggered flip-flops are toggled by a 0-to-1 transition. Unlike the synchronous counter, all outputs do not change simultaneously. For example, let's assume that $Q_C = Q_B = Q_A = 1$ and a clock transition occurs. This will cause the Q_A flip-flop to toggle, after a short delay, through the electronic circuitry of the flip-flop. The $\overline{Q_A}$ output will change state from 0 to 1, causing the Q_B flip-flop to change state after another short delay. This will, in turn, cause the Q_C flip-flop to change after another short delay. Because the flip-flops change sequentially, this type of counter is often called a *ripple counter.* As we'll see later, the short delay through each flip-flop can cause malfunctions of circuitry connected to the counter or, at the very least, degrade performance.

Asynchronous Coupling

Since small synchronous counters are easily implemented, it would be advantageous to connect small synchronous counters together asynchronously. A 5-bit counter may be

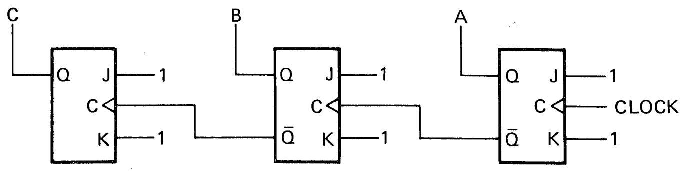


Figure 4-11 Asynchronous counter.

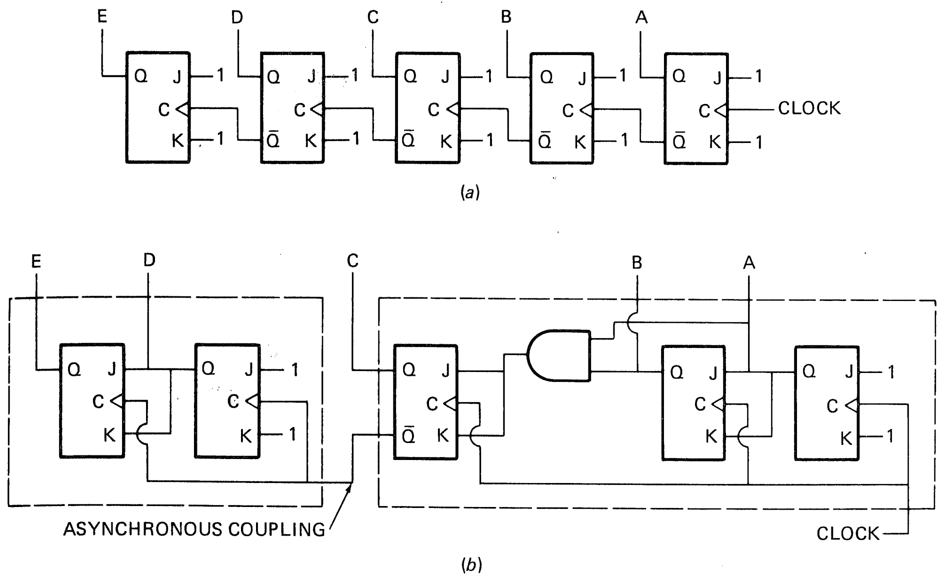


Figure 4-12a 5-bit asynchronous counter.
Figure 4-12b 5-bit counter using two asynchronously coupled synchronous counters.

implemented in several ways. As a completely synchronous ASM, it would require considerable gating. Two simpler alternatives are shown in Fig. 4-12. The 5-bit ripple counter in Fig. 4-12*a* is the simplest circuit but requires five flip-flop delays between a clock transition and a change of the most significant flip-flop Q_E. The alternate circuit shown in Fig. 4-12*b* consists of a 3-bit synchronous counter coupled asynchronously to a 2-bit synchronous counter. The most significant counter is clocked whenever the most significant bit of the least significant counter changes from a 1 to 0. Only two flip-flop delays occur between a clock transition and a change in the most significant bit Q_E. Output C changes one flip-flop delay after the clock. Output E changes one flip-flop delay after output C.

It should be obvious to you that one reason we worry about reducing delays is because we might want to operate the circuit at as high a clock rate as possible. For example, if we're building a digital computer, we'd like to solve problems as fast as possible. Using our 5-bit ripple counter would require waiting at least five flip-flop delays between clock transitions so that all five outputs have a chance to change.

Output Races

Suppose that we desire to connect gating to the outputs of the 3-bit ripple counter of Fig. 4-11. The timing diagram for this counter is shown in Fig. 4-13, along with the outputs of three AND gates we might use to decode particular counter states. Note that short undesired outputs occur on these AND gates in addition to the desired outputs. Because the three state variables change in sequence, the three flip-flop outputs momentarily assume *combinations that are not the actual next state.* For exam-

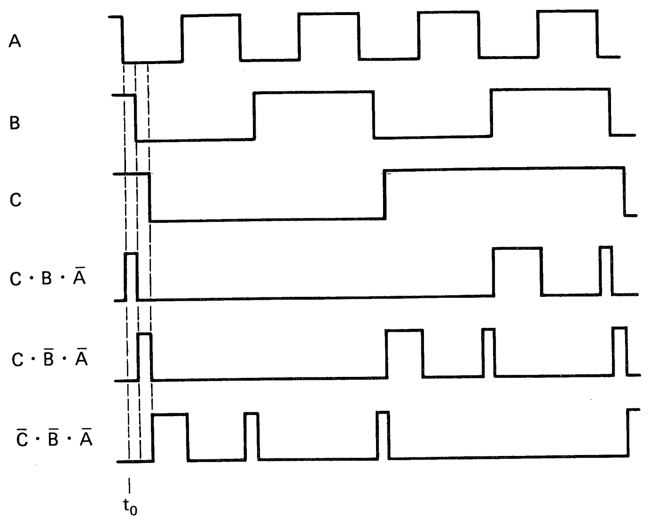


Figure 4-13 Output races when decoding an asynchronous counter.

ple, at time t_0, the circuit should make a transition from state 7 to state 0. In reality, flip-flop A changes first so that state 6 seems to momentarily occur. Flip-flop B changes next so that state 4 seems to momentarily occur. Finally, flip-flop C changes so that state 0 occurs. The decoding gates for states 6 and 4 will have momentary undesired outputs. These undesired outputs are called *output races.* Output races even occur in completely synchronous circuitry. Even though all flip-flops of a synchronous circuit are clocked simultaneously, each flip-flop will have a different propagation delay so that their outputs do not change exactly simultaneously. We will study problems such as races in detail in a later chapter. Note that even in our asynchronous ASM counter, where these output races are more pronounced, no difficulties will be encountered as long as these outputs are inputs to another ASM connected to the same clock. This clock must be slow enough that all outputs with races reach their correct value before the next clock transition. If outputs with races are used as immediate outputs or as inputs to another ASM which is connected to a different clock, the output races will be sensed as real outputs.

Gated-Clock Coupling

One more technique can be used to couple counters or other ASMs together. By selectively turning the clock of a counter on or off, we can cause it to count only when desired. Since the same clock is being used to change all the flip-flop's states, all outputs will change within one flip-flop delay. A 5-bit counter is formed by cascading a 2- and 3-bit counter, as shown in Fig. 4-14. The $A = 1$ and $B = 1$ state of the 2-bit counter is used to gate the clock to the 3-bit counter. For proper operation of this circuit, the gated clock *GCLK* must have two properties. It must be generated so that

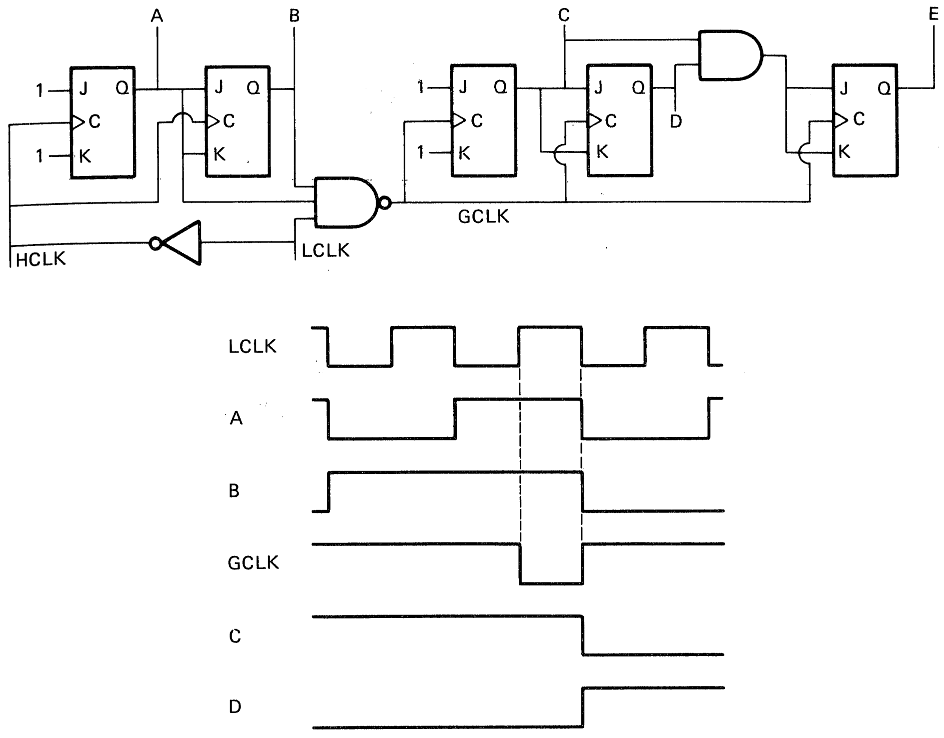


Figure 4-14 Coupling counters with a gated clock.

the circuitry it is driving is clocked at the same time as the other flip-flops in the system. Also, it must be generated in the *last half of the clock cycle* so that any state variables connected to the clock gate inputs have as much time as possible to reach their correct values before a gated-clock pulse is generated. A gated clock may be generated in four ways, as shown in Fig. 4-15. *Only one of these is correct for any given circuit.* If ever in doubt, remember the following two rules.

1. The edge of the gated clock causing a state transition must cause this transition simultaneously with other nongated state transitions in the circuit.
2. The edge described above must always be the second or trailing edge of the gated-clock pulse.

The same clock signal is not really used for the gated and nongated flip-flops of Fig. 4-14. These two signals will differ in time by the *difference* in propagation delay between the inverter generating *HCLK* and the gate generating *GCLK*. This is the reason we generated *HCLK* by inverting *LCLK*. If we had generated *LCLK* by inverting *HCLK*, the *HCLK* and *GCLK* transitions would differ by the *sum* of the inverter and gate delays.

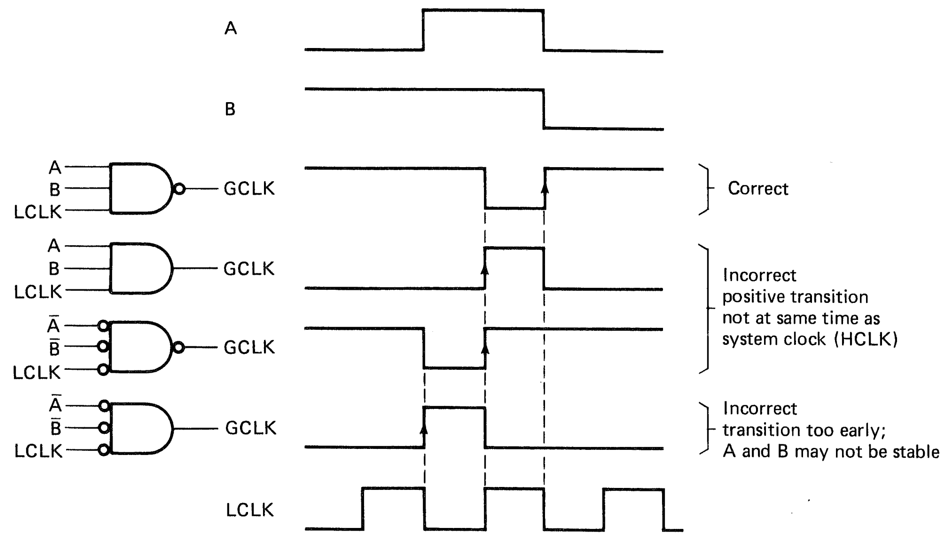


Figure 4-15 Generating gated clocks for Fig. 4-14.

Selection of Coupling Technique

These three coupling techniques apply to interconnection of any ASM, as well as counters. The following table describes the important considerations involved in selection of a coupling technique.

Asynchronous coupling

1. Used in counters with very long count sequences
2. Used when cost is paramount
3. Not used with immediate outputs, unless special provisions made to eliminate output races

Gated-clock coupling

1. Used with MSI devices that do not have an integral enable input. The clock to the MSI device must be turned off to disable it.
2. Used with complex functions made from individual gates and flip-flops that would become hopelessly complex if enabling logic were included in next-state circuitry
3. Not used when highest speed is required

Synchronous coupling

Generally recommended in all cases

HANDSHAKING

Often more than 1 bit of information must be transferred between ASMs. A common requirement, shown in Fig. 4-16, is transferring multiple bits between ASMs. These bits may represent a number or the code for a function to be performed or many other things. In a previous chapter, we have transferred data between ASMs, using a data-valid signal to indicate to the the receiving ASM that data were present. In Fig. 4-16, a data-available signal (*HDAV*) indicates to the receiving ASM that data are available. Also, a data-accepted signal (*HDAC*) indicates to the sending ASM that the data have been accepted. The operation of these signals is called a *handshake* and is shown in the timing diagram of Fig. 4-16.

At time t_0, ASM1 has data read to be sent to ASM2. ASM1 asserts *HDAV* to notify ASM2 that new data must be transferred. When ASM2 tests and recognizes that *HDAV* is asserted, ASM2 stores (or tests) the data sent to it and asserts *HDAC* at t_1 to tell ASM1 that it has accepted this data byte. ASM1 makes *HDAV* false at t_2, indicating that valid data are no longer being sent. Finally, ASM2 makes *HDAC* false at t_3 to prepare for another data transfer at t_4.

ASM1 transmits new data only after it recognizes an *HDAC* transition from true to false, indicating completion of the last data transfer. ASM1 leaves *HDAV* true until it recognizes an *HDAC* transition from false to true, indicating that the data were accepted. ASM2 accepts data only when it recognizes an *HDAV* transition from false to true. ASM2 acknowledges a transfer complete only when it recognizes an *HDAV* transition from true to false. The handshake sequence assures that ASM2 accepts each data byte before another is transmitted. A data byte cannot be missed, nor can it be read twice, as long as the handshake sequence is followed. The handshake sequence does not rely on synchronism between the clocks of the two ASMs and will work even if each ASM has a different clock.

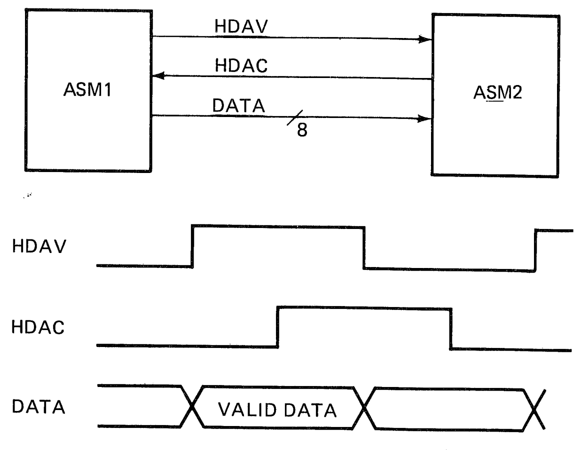
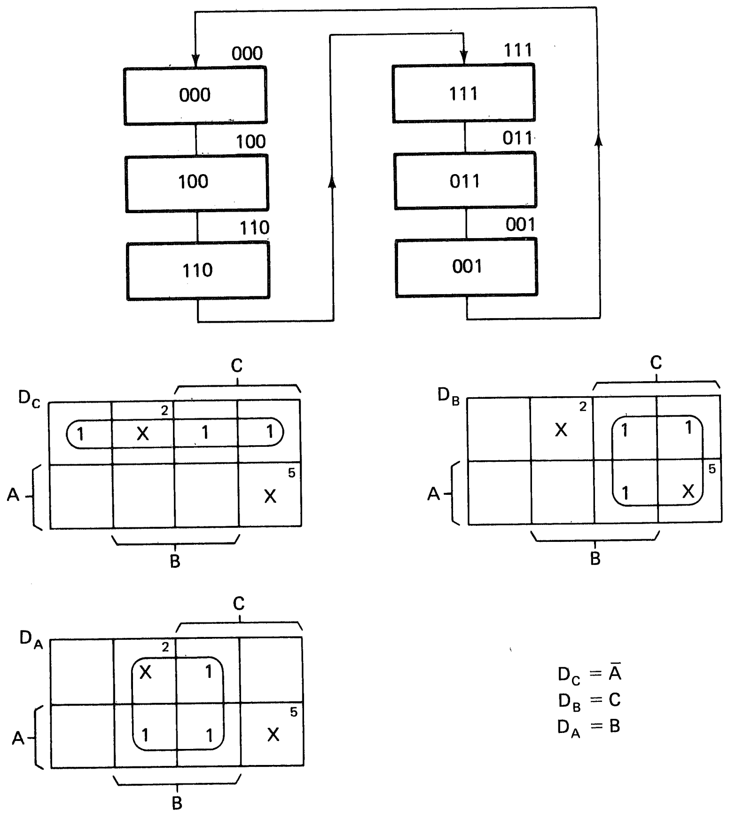


Figure 4-16 Handshaking.

HANG-UP STATES AND SYSTEM INITIALIZATION

Let's implement the six-state counter whose ASM chart and K maps are shown in Fig. 4-17. Note that the counting sequence we chose caused a particularly simple implementation, shown in Fig. 4-18. States 2 and 5 were not used; so we specified all entries corresponding to these states to be don't cares. Since we have no control over the initial state of the counter when power is first applied to the circuit, either of these unused states could occur at that time. Returning to Fig. 4-17, we can calculate the next states if the current state is either 2 or 5. If the counter is in state 2, we find the next state by looking in square 2 of the K maps of Fig. 4-17. The next state will be $D_C = 1, D_B = 0$, and $D_A = 1$ or state 5. If the counter is in state 5, its next state will be state 2. If the counter is in state 2, it will count to state 5. If the counter is in state 5, it will count to state 2. The ASM chart of Fig. 4-19 describes the counter's behavior in states 2 and 5. If the circuit starts in state 2 or 5, it will just alternate between these states. It will never count as we designed it! Any ASM that has fewer states than the maximum allowed by the number of state variables can have undesirable sequences like this one. Any unused state that doesn't eventually lead back to a desired state is called a *hang-up state.*



$$D_C = \bar{A}$$
$$D_B = C$$
$$D_A = B$$

Figure 4-17 Six-state counter.

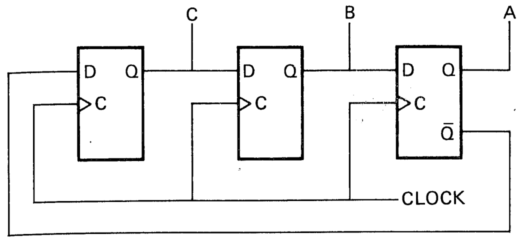


Figure 4-18 Six-state counter schematic.

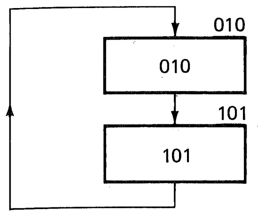


Figure 4-19 Hangup states of the six-state counter.

System Initialization

One way to make sure hang-up states are not entered when power is applied is to specify the first state explicitly, as shown in Fig. 4-20. Often, the first state must be specified anyway for functional reasons. For example, we couldn't have a computer start in the "erase disk memory state" when power is first applied. Specifying the first state on the ASM chart is done by simply notating this state with the words "power on" or "initialize," as shown in Fig. 4-20. Implementing this notation is more difficult. A signal (*LPWRON*) indicating that power has just been applied must be generated in the power supply of the logic circuit. This signal may be used in several ways to

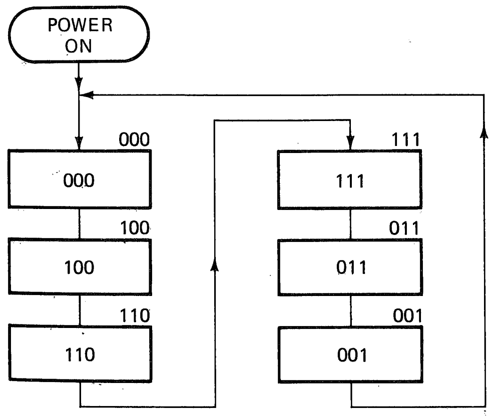


Figure 4-20 Initializing the ASM chart.

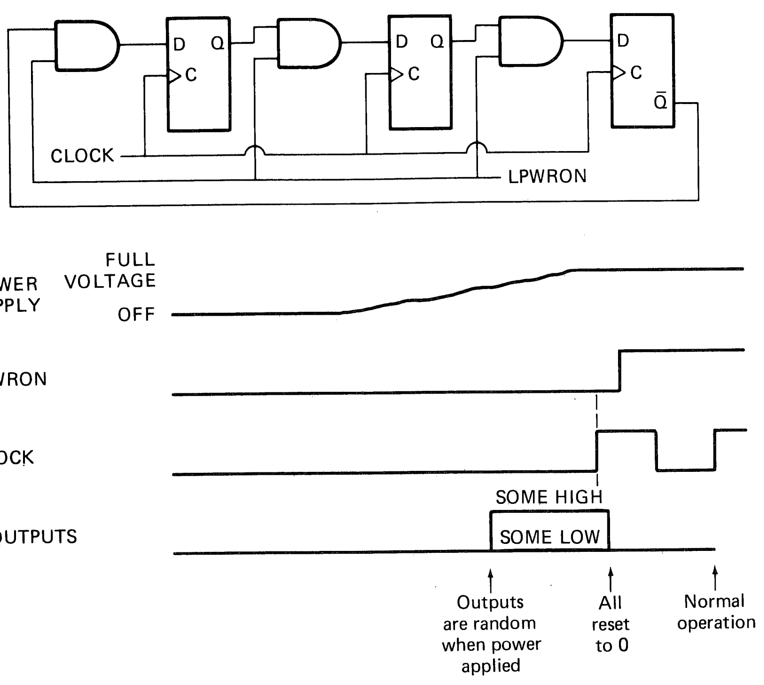


Figure 4-21 Initializing by modifying flip-flop inputs.

preset the state variables. In Fig. 4-21 each flip-flop is given additional input gating to allow *LPWRON* to set the *D* inputs to the desired next state. The first clock pulse to be generated clocks this state into the flip-flops. The simple gating shown is possible because we've picked 000 as our first state. More complex gating might be necessary for other initial states. In general, the all-0s and all-1s states are easiest to use as the first state after power is applied. In Fig. 4-22 we've saved input gating by noting that we can set only one *D* input to 0 and let three clock transitions propagate this 0 to the three flip-flop outputs. Propagating the initialization signal may be inconvenient or time-consuming with many ASMs. This technique is naturally suited to shift register implementations, such as this example. This technique could be the only one possible if these flip-flops were part of an MSI or LSI circuit and external connections were not available.

The most common initialization technique uses *direct clear* and *direct preset* inputs found on many flip-flops and MSI circuits containing flip-flops. These inputs, shown on a D flip-flop in Fig. 4-23, are usually immediate-acting. They do not depend on the clock input for operation. If the preset input is low, the Q output will immediately become 1 (and $\overline{Q}$ will become 0). If the clear input is low, the Q output will immediately become 0 (and $\overline{Q}$ will become 1). If both clear and preset are low, the operation of the circuit is unpredictable. Both clear and preset must be high just before and during a clock pulse if the clock transition is to have a predictable effect on the flip-flop's output. In Fig. 4-24, the direct clear inputs of the three flip-flops are held low by *LPWRON* until after power is applied. After the clear inputs are brought to a 1, the clock is started.

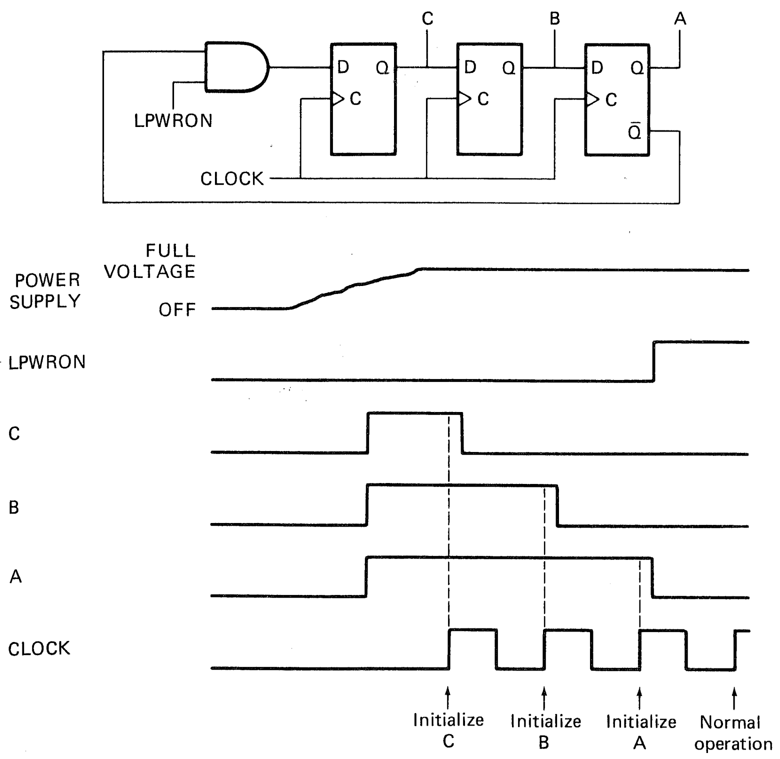


Figure 4-22 Initializing by propagating through ASM.

Self-Clearing Logic

Even though we have now made sure that our original six-state counter will start in the correct state, we have no guarantee that it will stay in that sequence. Environmental conditions such as excessive electric noise could cause the circuit to accidentally and erroneously enter state 2 or 5. It would alternate between these two states until we removed and reapplied power. To avoid this malfunction, we will design the circuit

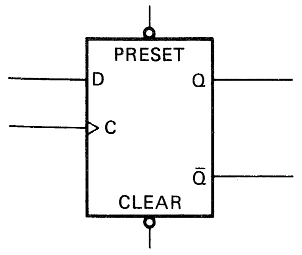


Figure 4-23 Type D flip-flop with direct preset and direct clear.

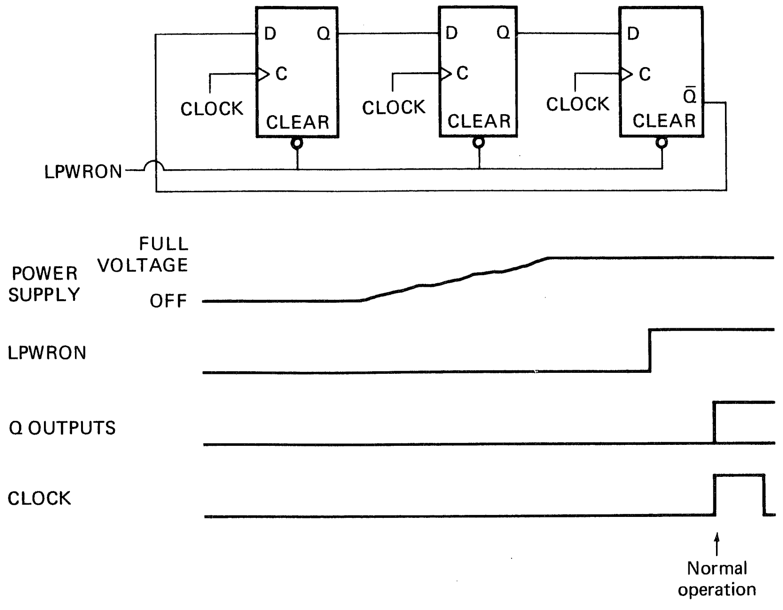


Figure 4-24 Initializing with direct inputs

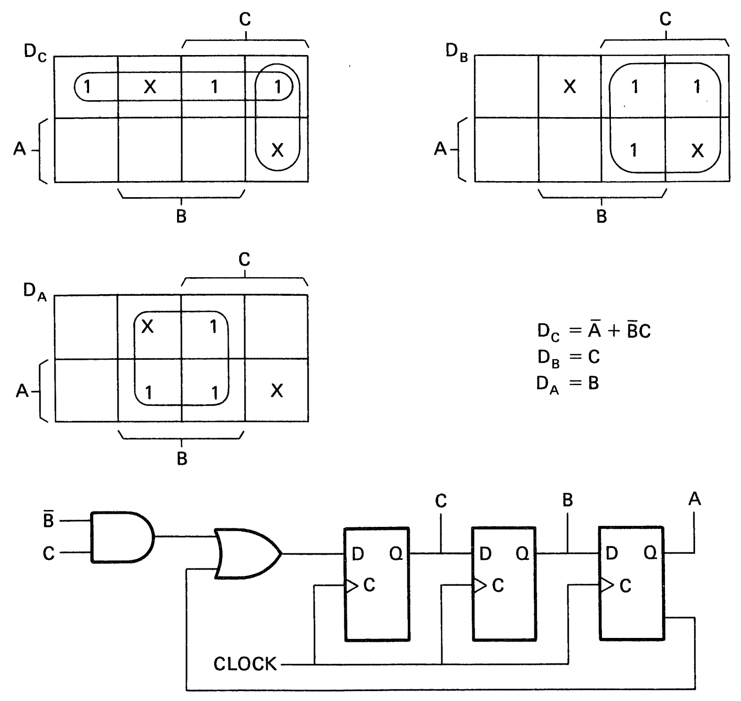


Figure 4-25 Modified K-maps and circuit to eliminate hang-up states.

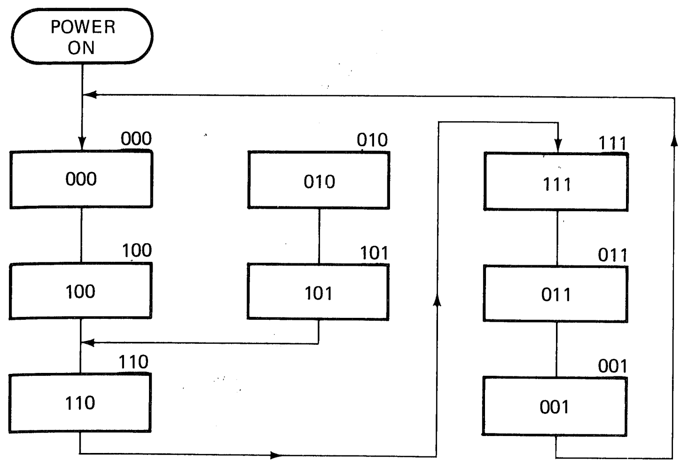


Figure 4-26 ASM chart for six-state counter with initialization and without hangup states.

so that all unused states will eventually lead back to a desired state. If electric noise should ever disturb its operation, the malfunction would last only momentarily, until the circuit sequenced back to a desired state. *Never design a logic circuit with hang-up states*! The K maps for our counter are redrawn in Fig. 4-25. Only now we would like to change the next state corresponding to a current state of 2 or 5. One simple alternative is to change the interpretation of the don't care of D_C from a 0 to a 1, as shown. This complicates the expression for D_C and adds two gates to our implementation, also shown in Fig. 4-25. However, the ASM chart for this circuit in Fig. 4-26 has no hang-up states. If the circuit is accidentally put in state 2 or 5, it will eventually return to state 6, where it will continue the correct counting sequence.

PROBLEMS

4-1 Design counters using JK flip-flops for the following sequences.
(*a*) 00, 01, 11
(*b*) 00, 11, 01
(*c*) 01, 11, 00

4-2 Design a five-state counter using D flip-flops to perform identically to the JK flip-flop counter of Fig. 4-8.

4-3 Draw a circuit to synchronously couple three 4-bit counters, like those shown in Fig. 4-9. Devise a coupling scheme that *does not* cascade delays through each counter's output logic.

Hint: You can only use the least significant counter's carry output since only this carry output occurs one propagation delay after the clock.

4-4 Add an enable input to the asynchronous counter of Fig. 4-11 in the simplest way possible. Do *not* gate the clock.

4-5 Figure P4-1 shows three other possible configurations for the counter of Fig. 4-14. Draw the correct gated-clock logic for each.

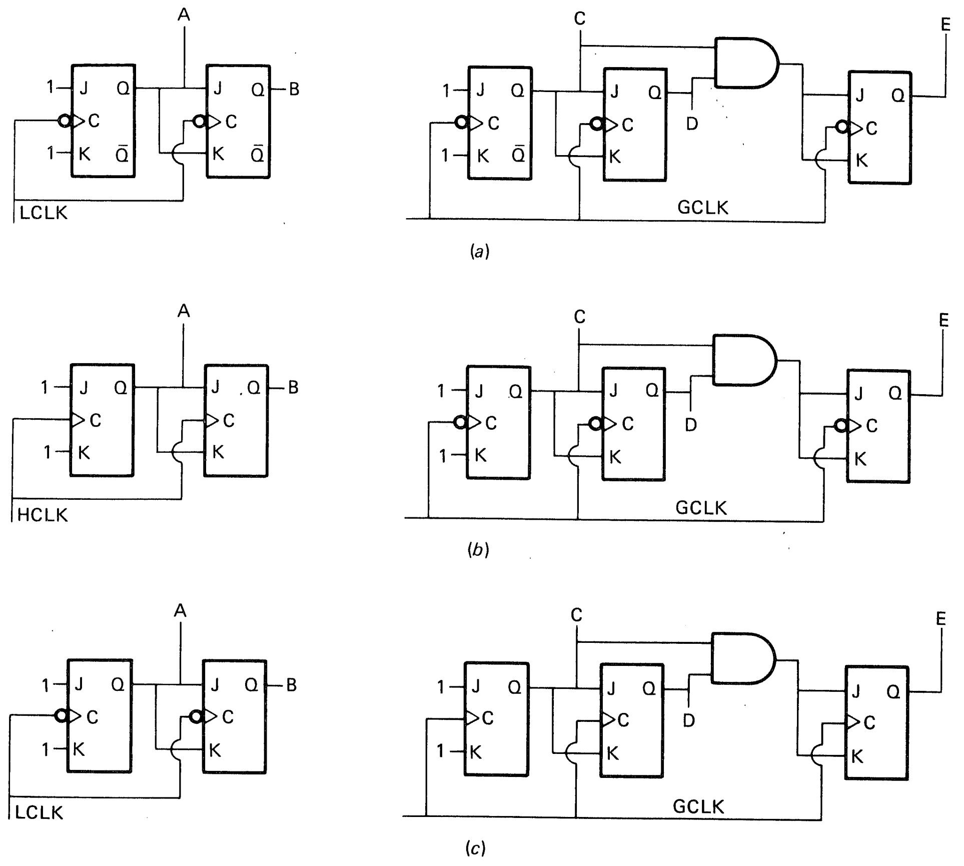


Figure P4-1 Gated clock configurations.

4-6 Design a six-state counter, using JK flip-flops for the ASM chart of Fig. 4-17.

4-7 Consider the counter of Prob. 4-6. It is called a Moebius counter and has several useful properties.

(*a*) Design a decoder of each of the six states. What do these decoders have in common? Can you find a simple rule for decoding a state without having to resort to K maps?

(*b*) What additional gating would be required to stop the counter when it reached a particular state?

Hint: It is a single gate and does not require decoding states.

(*c*) If the three flip-flops had different delays, would output races occur in the decoding of part (*a*)? Explain your answer.

CHAPTER

FIVE

CLOCKS AND SIGNALS

The clock signal of any sequential machine is very important. Up to now we have been ignoring the details of the clock signal, assuming that it had any attributes necessary for proper operation of the ASM. Actually, generating and distributing clock signals in an ASM is a complex subject.

Clock Skew

Since digital systems are often very large, the situation depicted in Fig. 5-1 usually occurs. Here, the number of flip-flops is so large that the clock circuit cannot drive all of them directly. In this example, two inverters are used to drive the clock lines. Some of the flip-flop clock inputs are connected to each inverter. The signal propagation delays through these two inverters are probably different. Usually only maximum delay times are specified for logic circuits. Any individual circuit may have a delay considerably less than the maximum. The delay through any logic circuit is dependent on the number of other circuits connected to it. If one inverter drove 10 clock inputs and the other drove 5 inputs, their propagation delays would be different, even if both inverters were electrically identical. Also, as seen in the last chapter, gating the clock also causes unequal clock delays.

Does the fact that the clocks of two flip-flops occur at different times cause problems? In the simple 2-bit shift register of Fig. 5-2, clock inputs are driven by two different sources. The clock of the first flip-flop *X* occurs later than the clock of the second flip-flop *Z*. The timing diagram in Fig. 5-2 shows data shifted through the register correctly. Because *CLKA* is later than *CLKB*, data are shifted properly from flip-flop *X* to flip-flop *Z*. In fact, whenever information is transferred from a later clock to an earlier clock, the information transfer will take place correctly. However, if information is transferred from an earlier to a later clock, the transfer could occur