

# COMBINATIONAL HAZARDS

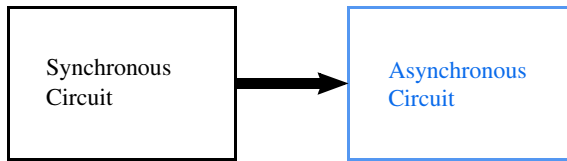
**S** elected topics not covered in the fourth edition of *Logic and Computer Design Fundamentals* are provided here for optional coverage and for self-study. This material fits well with the desired coverage in some programs but may not fit within others due to time constraints or local preferences. This supplement is referenced in Section 6-5, Asynchronous Interactions. While it is often associated with the design of asynchronous sequential circuits, it is included here primarily for dealing with problems that arise when synchronous circuit outputs drive asynchronous circuit and system inputs, the situation presented in Figure 6-10(a).

Our focus here deals mostly with situations encountered at the interfaces between circuits. Similar problems plague asynchronous designs internally as well, but to study those directly, would require getting deeply into asynchronous circuit analysis and design techniques. The interfaces to be considered are shown in Figure 1, which is a copy of Figure 6-10 from the text. We will look at the problems of driving an asynchronous circuit with the outputs of a synchronous circuit as in 1(a). We begin by introducing a simple asynchronous circuit, an asynchronous counter. We do not dwell on the internal structure of this circuit, but consider only its behavior. We then use this circuit to illustrate the effects of combinational circuit hazards. Hazards cause problems if they appear in the inputs of an asynchronous circuit or on combinational logic outputs realizing the state variables within the asynchronous circuit. We will look at some techniques for eliminating hazards and touch upon hazards that are very difficult to handle. The other interface cases in Figure 1(b) and (c) are covered in the text in Chapter 6.

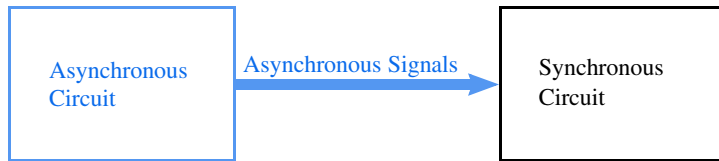
## An Asynchronous Binary Counter

Figure 2(a) shows the symbol for an asynchronous binary counter and Figure 2(b) shows the circuit behavior. The circuit has two inputs, *CNT* and *CLEAR*, and two outputs, *C0* and *C1*. If *CLEAR* is 1, then both *C0* and *C1* go to 0. When *CLEAR* returns to 0, *C0* and *C1* remain at 0 until an input appears on *CNT* that causes the circuit to count up. The circuit counts up every time it sees a value 1 on the input

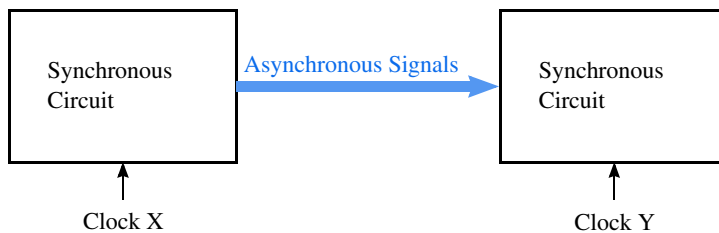
<sup>1</sup>© Pearson Education 2008. All rights reserved.



(a) Synchronous to Asynchronous



(b) Asynchronous to Synchronous

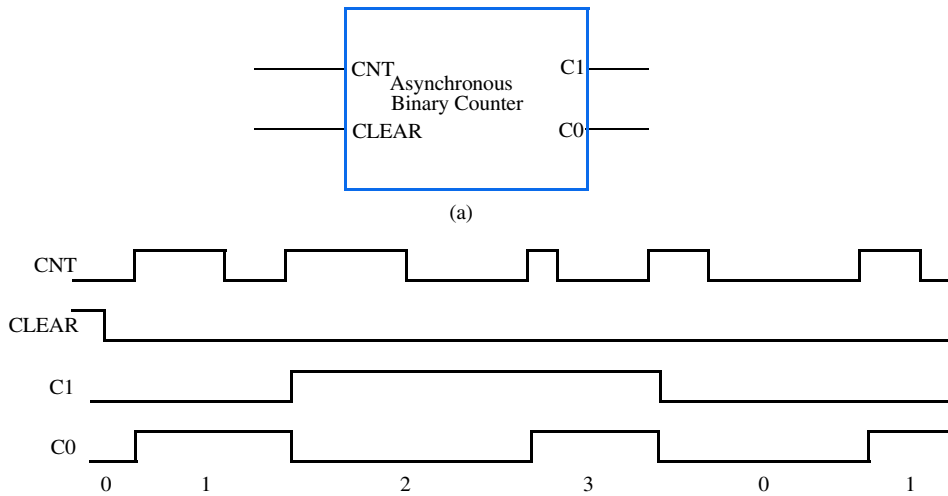


(c) Synchronous Circuits with Unrelated Clocks

□ **FIGURE 1**  
Examples of Synchronous/Asynchronous Interfaces

*CNT*. Note that since there is no clock, the circuit does not see multiple 1s in sequence, so a *new* value 1 can be identified only after a value 0 has appeared. The result of this is that the circuit appears to count positive edges (0-to-1 transitions). If the count on  $(C1,C0)$  is 3 (11), then on the next positive edge on *CNT*, the count become 0 (00). So, the counter counts up, 0, 1, 2, 3, 0, 1, 2, ... in binary on the output  $(C1,C0)$  and is thus a modulo 4 counter. Have you seen this circuit before? How about a two-bit ripple counter with a *CLEAR* input added? The ripple counter is one structure that implements this functional behavior. Note that the input *CNT* is not a clock in the sense that it is not required to be a periodic synchronizing signal. If this were a synchronous circuit, the counter would have a clock in addition to *CNT* or *CNT* itself would become a clock input.

We will break down combinational hazards into two major categories, logic hazards and function hazards. We will deal with logic hazards first and then proceed briefly to the more troublesome function hazards. A *logic hazard* is characterized by the fact that it can be eliminated by proper combinational logic design methods. *Function hazards* come with the function being implemented and cannot be dealt with by basic combinational design techniques.

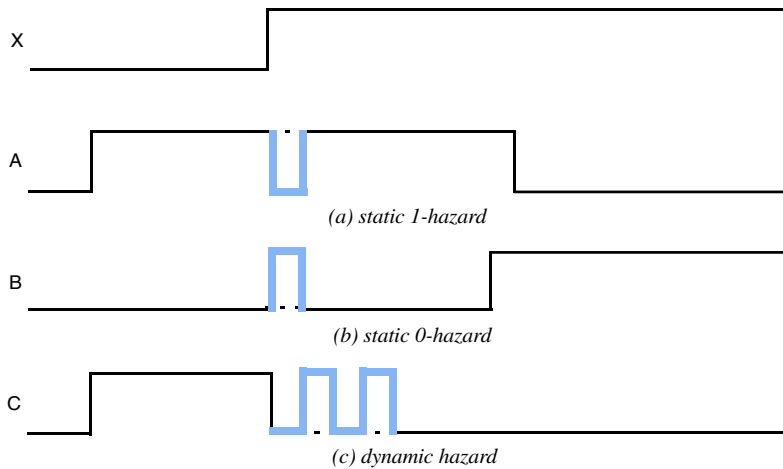


□ **FIGURE 2**  
An Asynchronous Binary Counter

## Logic Hazards

Suppose that input variable changes are spaced such that the effects of a change in one variable is permitted to propagate throughout the circuit before another variable is allowed to change. This is the single-input change case, since input signal patterns can change in only one variable at a time. For example, 00 can be followed by 01 or 10, but not 11. A *single-input change static hazard (SICS hazard)* is a momentary change in an output that occurs as the result of the change of a single input variable when the value of the output variable is to remain fixed. In Figure 3, a change in input signal  $X$  causes all three responses  $A$ ,  $B$  and  $C$ . In Figure 3(a), the output signal  $A$  is to remain at 1 until the change to 0 in the right portion of the waveform. But instead, it momentarily goes to 0 and then returns to 1 in response to the change in  $X$ . In this case, where the signal is supposed to remain at 1, the hazard is referred to as an *SICS 1-hazard*. Figure 3(b) illustrates an *SICS 0-hazard* in which the change in  $X$  causes output signal  $B$  to momentarily go to 1 when it is supposed to remain at 0. Just to illustrate why we use the term “static,” in Figure 3(c), we show a *dynamic hazard* in Figure 3(c) due to the change  $X$ . It is distinguished from the static hazard in the sense that the correct signal is “dynamic,” i.e. changing, in this case from 1 to 0. But instead of changing once, it changes some odd number of times depending on the circuit structure; five changes are illustrated.

Now, let’s look at the effect of these signals, with and without the respective hazard, assuming that each appears as the  $CNT$  input on the asynchronous binary counter. In all cases,  $(C1, C0)$  is assumed to be  $(0, 0)$  initially. If  $CNT = A$  is applied without the SICS 1-hazard, then the resulting value of  $(C1, C0)$  is  $(0, 1)$ . But if  $CNT = A$  is applied with the SICS 1-hazard, then the resulting value of  $(C1, C0)$  is  $(1, 0)$

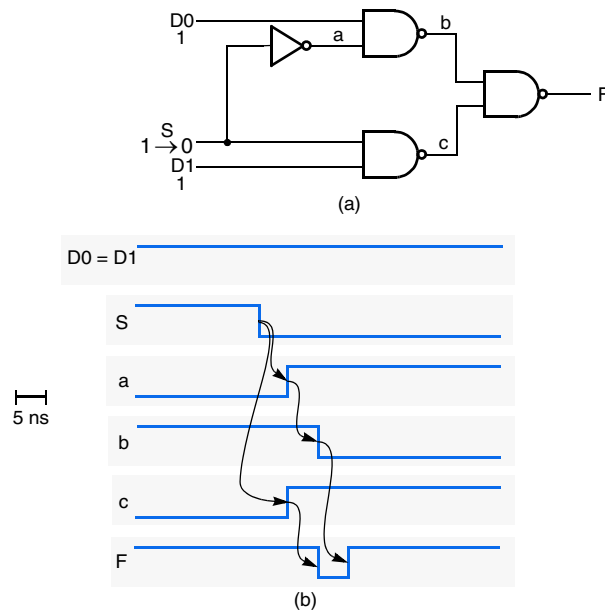


□ **FIGURE 3**  
Logic Hazards

since there are two positive edges in  $A$ . The same result occurs for the SICS 0-hazard present in signal  $B$ . Signal  $C$  applied without the dynamic hazard gives  $(C1,C0)$  as  $(0,1)$ . But with the dynamic hazard present,  $(C1,C0)$  is  $(1,1)$  since there are now three positive edges instead of one. In all three cases, the count value resulting in the presence of the hazards is not the correct value  $(01)$ ! Another way of saying this is that, in the presence of the SICS hazards and the dynamic hazard, the asynchronous counter assumes an *incorrect* state.

In fact, the occurrence of an incorrect state due to a presence of a hazard in an input signal does not apply just to the asynchronous binary counter but to a broad range of asynchronous circuits. Such hazards in the state variables of an asynchronous circuit can also produce a wrong state. Thus, hazards are to be avoided in any situation in which they can result in an incorrect state variable or output variable value for an asynchronous circuit. In order to avoid SICS hazards, we first examine their source.

Figure 4(a) shows the implementation of a multiplexer using NAND gates with  $D0$  and  $D1$  equal to 1 and  $S$  changing from 1 to 0. If we assume that all of the gates have a delay of 5 ns, then a simulation of the circuit yields the result shown in Figure 4(b). The curved arrows represent a causal relation between signal transitions. The transition at the tail of the arrow causes the transition at the head of the arrow one gate delay later. When  $S$  changes from 1 to 0,  $a$  and  $c$  change from 0 to 1 after 5 ns. At this time, both  $b$  and  $c$  are 1, causing  $F$  to change to 0 after 5 ns. The condition on  $b$  and  $c$  remains for 5 ns, causing  $F$  to remain at 0 for 5 ns. In response to the change of  $a$  to 1,  $b$  goes to 0 after 5 ns causing  $F$  to return to 1 after 10 ns. Careful examination of the waveforms show that the difference in the circuit delay along path  $S - a - b - F$  from the circuit delay along path  $S - c - F$  causes the SICS 1-hazard to occur.



□ **FIGURE 4**  
Example of a Static Hazard in a Multiplexer

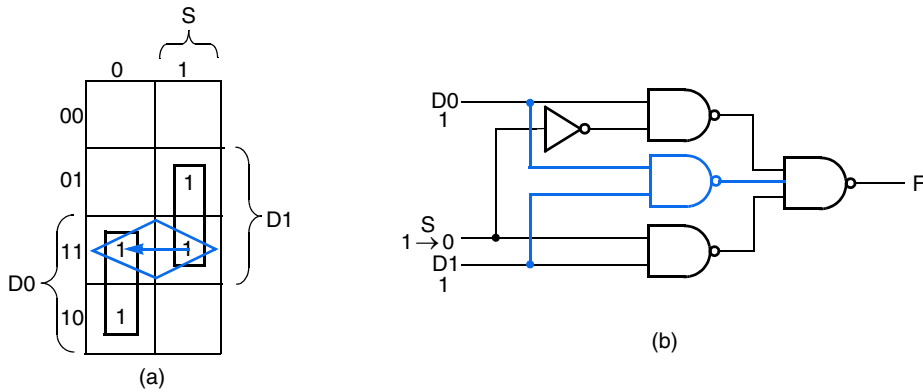
● **EXAMPLE 1 Static Hazard Occurrence**

In Figure 4(a), assume that signal  $S$  changes from 0 to 1 instead of from 1 to 0 and repeat the simulation in Figure 4(b). Does an SICS 1-hazard actually occur? Answer: No static 1-hazard occurs. While  $b$  and  $c$  are both 1 for 5 ns in Figure 4(b),  $b$  or  $c$  is 0 at all times in this new simulation, keeping  $F$  at 1.

Thus we find that the *occurrence* of a static hazard can depend on the direction of a signal change. It can also depend on very minor changes in the circuit structure that affect path delays. ●

What happens if the output of the multiplexer is the  $CNT$  input to the asynchronous binary counter? Every time that  $S$  changes from 1 to 0 with  $D0 = D1 = 1$ , a static 1-hazard appears on  $CNT$  giving an extra positive edge. This edge causes an extra count up to occur, giving an erroneous counter state.

Can we prevent the static hazard from occurring? Suppose we examine the Karnaugh map of the multiplexer given in Figure 5(a). The two product terms  $\bar{S}D0$  and  $SD1$  used in the original design are shown. In order to determine the location of the static hazard on the map, we set  $D1 = D0 = 1$ , which indicates that the hazard is in the third row of the map. The hazard actually occurs when  $S$  goes from 1 to 0. In the figure, we denote the location of this hazard by a diamond containing an arrow indicating the direction of the change of  $S$ . The hazard occurs as the circuit goes from minterm  $SD1D0$  to minterm  $\bar{S}D1D0$ . Now, suppose that we combine these two minterms:



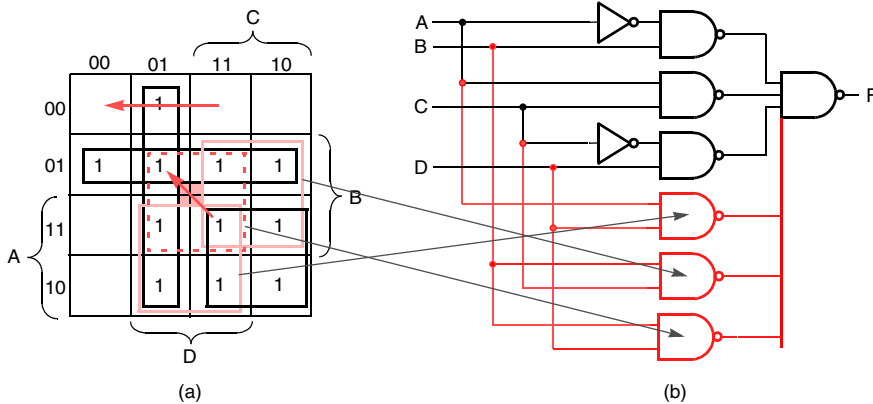
□ **FIGURE 5**  
Karnaugh Map of Multiplexer and Static Hazard Prevention Logic

$$\bar{S} D1 D0 + S D1 D0 = D1 D0$$

Product term  $D1 D0$  is exactly the location of the diamond on the map. What if we add this product term to the multiplexer as shown in gray in Figure 5(b) and then change  $S$  from 1 to 0? Since  $D1$  and  $D0$  are both fixed at 1, the output of the new NAND gate is 0 keeping  $F$  fixed at 1. The addition of this new product term has prevented the hazard from occurring! Further, we can note that after the addition of this product term, all of the prime implicants of the multiplexer have been used in the implementation.

In general, for a sum-of-products implementation, the potential for a SICS hazard exists wherever there are two adjacent 1's in the Karnaugh map that are not included within a product term of the implementation. To remove all potential for static 1-hazards from a sum-of-products implementation of  $F$ , *all* prime implicants of  $F$  must be included in the circuit implementation. (Note that if there are don't cares, there may be some prime implicants that can be left out.) A sum-of-products implementation is automatically free of static 0-hazards. For a product-of-sums implementation of function  $F$ , all of the complements of the prime implicants of  $\bar{F}$  must be included. A product-of-sums implementation is automatically free of static 1-hazards. Finally, any sum-of-products or product-of-sums implementation free of static 1-hazards and static 0-hazards is free of dynamic hazards.

A Karnaugh map for a function  $F$  is given in Figure 6(a). All prime implicants for  $F$  are used in the static hazard-free sum-of-products implementation of  $F$  in Figure 6(b). The normal minimum solution for  $F$  is represented by the prime implicants in black. There are three other prime implicants available. The two outlined in gray,  $A D$  and  $B C$ , are present to deal with single-input change static hazards. The one outlined with dashes,  $B D$ , is present to deal with a *multiple-input-change static hazard (MICS hazard)*. This type of hazard occurs when two or more inputs change very close in time to each other causing movement between a pair of min-



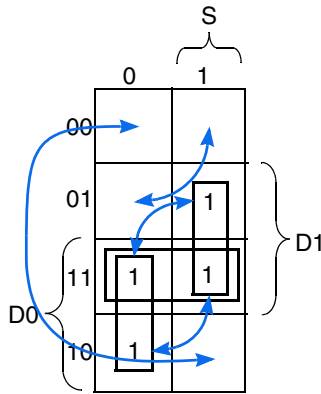
□ **FIGURE 6**  
Karnaugh Map and Implementation for a Static Hazard-Free Circuit

terms not contained in a single prime implicant. Such a pair is illustrated by the arrow over the shaded area (which represents a 0 output value) between the two minterms. All three of the redundant prime implicants of  $F$  appear in the static hazard-free implementation as shown by the thin arrows from the implicants to the gates. This implementation can be attached to  $CNT$  of the asynchronous binary counter with no concern for static hazards occurring.

Unfortunately, if multiple variable values can change in an interval so short that the effect of the first change has not propagated throughout the circuit before the second change occurs, elimination of static hazards is not sufficient to guarantee correct operation. For example, consider the change from 0011 to 0000 in Figure 6(a). If  $C$  changes before  $D$ , the combination 0001 will momentarily appear. For 0001, function  $F$  has value 1, causing  $F$  to change from 0 to 1 and then back to 0, generating a 0-hazard. This is not a logic hazard, since there is no way that we can avoid it by changing the logic implementation! Since this hazard is built into the function  $F$  regardless of implementation, it is called a *function hazard*. The only way that it can be controlled is by changing relative path delays in the circuit. For this example function hazard with inputs going from 0011 to 0000, delay must be inserted in the circuit to effectively delay  $C$  so that it changes after  $D$ . The sequence of input values during the change would then be 0011, 0010, 0000. Since all of these have output value 0, no function hazard occurs. Unfortunately, delay control is difficult and tedious and there are cases where function hazards cannot be eliminated by using delay control.

● **EXAMPLE 2 Finding Function Hazards**

Identify locations of function hazards that occur for two inputs changing for the multiplexer in Figure 5(a) with the SICS hazard term included in the design.

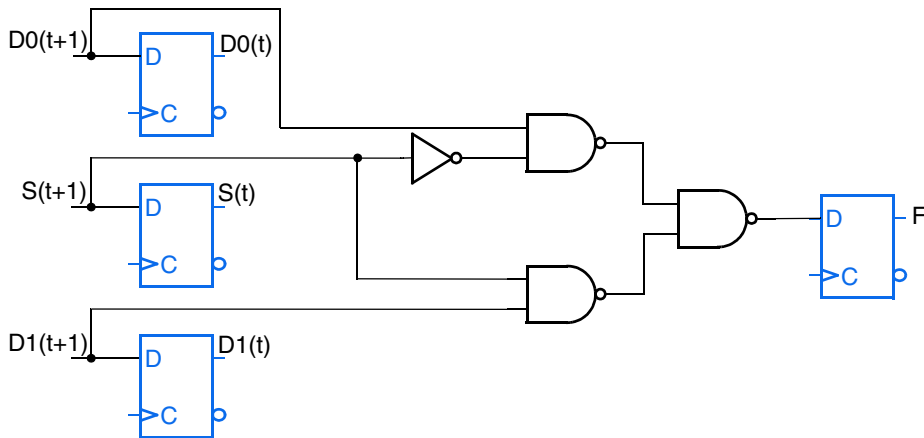


□ **FIGURE 7**  
Location of Function Hazards

Answer: See Figure 7. 1-function hazards occur between  $(D0, D1, S) = (0,1,1)$  and  $(1,1,0)$  and between  $(D0, D1, S) = (1,1,1)$  and  $(1,0,0)$ . 0-function hazards occur between  $(0,0,1)$  and  $(0,1,0)$  and between  $(1,0,1)$  and  $(0,0,0)$ . Suppose that  $D0$ ,  $D1$  and  $S$  are synchronous circuit flip-flop outputs that can change one, two or three at a time (but cannot be guaranteed to change at exactly the same time) and that the asynchronous counter input  $CNT$  is  $F$ . It is tedious, if not impossible, to manipulate the circuit delays in the multiplexer to avoid all of these function hazards including those involving three variable changes. But if this is not done, the counter will count incorrectly. ●

How else can we produce a “glitch-free” input to an asynchronous circuit from a synchronous circuit? A different approach is to make the output  $F$  come from a synchronous D flip-flop such that all glitches at the flip-flop input occur before the synchronizing clock edge. The flip-flop will change zero or one times according to its fixed input value in the setup-hold time interval. A glitch on  $F$  cannot occur since it requires *two* closely-spaced flip-flop output changes.

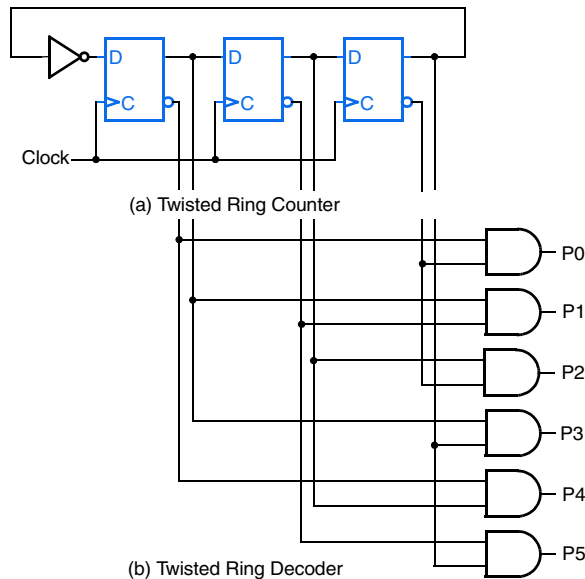
How can we design such a circuit while preserving the logic function on  $F$ ? If we simply add a D flip-flop at  $F$ , the change will be delayed by one clock cycle. If this is acceptable, fine. But if not, what can we do? We need to have flip-flop  $F$  assume its state at the same time as the state variables driving the multiplexer. This means that  $F(t+1)$  must be implemented by using the *inputs* to the D flip-flops  $D0(t+1)$ ,  $D1(t+1)$  and  $S(t+1)$ , instead of their *outputs*  $D0$ ,  $D1$  and  $S$ . This combinational circuit driving flip-flop  $F$  can be full of glitches, i.e., no hazard prevention required, and the circuit will function correctly since the hazards are eliminated by making  $F$  a synchronous state variable. The circuit in Figure 8 illustrates this technique assuming that  $F$  is the function realized by the multiplexer in Figure 4.  $F$  now comes from a flip-flop which delays  $F$  by one clock cycle. In the figure, this delay is prevented by making the input to the flip-flop occur one clock cycle earlier. This earlier production of the flip-flop input is obtained by moving the inputs of the



□ **FIGURE 8**  
Use of a Flip-flop to Eliminate Glitches

multiplexer from the outputs of the flip-flops driving the multiplexer to their inputs. This is not a technique that can always be used, but can be useful when applicable. For example, it may not be possible to use this approach if the output is a Mealy output, i.e., is a function of an input to the circuit, not just state variables. Also, since it does cost an additional flip-flop, its cost compared to eliminating hazards by other means must be considered.

Suppose we consider one more possible solution to the combinational hazard problem. We were able to get rid of all SICS static hazards and found that MICS static hazards and function hazards cannot happen if the single input change (SIC) restriction on the inputs to the multiplexer can be enforced. This can be done in some cases by using a single input change state assignment for which any state transition in the synchronous circuit involving a change in  $D_0$ ,  $D_1$  and  $S$  has at most one of them changing. It is, however, impossible to do this for every state diagram. An example of the use of this technique is the twisted ring counter in Figure 9(a). This counter is a rotating shift register with the bit fed back from the output of the shift register to its input inverted. The counter produces repetitions of the state sequence 000, 100, 110, 111, 011, 001 (assuming the counter is initialized to 000). In general, for an  $n$ -bit twisted ring counter, a sequence of  $2n$  states is produced. Note that only one bit changes for each state transition. This means that there will be no MICS hazards and no function hazards. Suppose that we attach the counter outputs to the inputs of the special decoder in Figure 9(b). This combined circuit produces the following sequence on its outputs: 100000, 010000, 001000, 000100, 000010, 000001. This is the same sequence that a properly initialized ring counter would produce. But the ring counter has twice as many flip-flops and costs more than the twisted ring counter. The outputs from both are hazard-free, the ring counter since all of its outputs are from flip-flops and the twisted ring counter plus decoder because the twisted ring counter has only single input changes and the attached decoder logic is SICS hazard-free. This structure has been used to



□ **FIGURE 9**  
Twisted Ring Counter and Decoder

generate multiple clock phases in computer control units. P0 through P5 can be used as clocks since they are “glitch-free.”

## References

1. MANO, M. M. AND C. R. KIME. *Logic and Computer Design Fundamentals*, 4th ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2008.
2. McCLUSKEY, E. J., *Logic Design Principles with Emphasis on Testable Custom Circuits*, Englewood Cliffs, NJ: Prentice-Hall, 1986.
3. MANO, M. M. and M. D. CILLETTI, *Digital Design*, 4th Ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2007.
4. Wakerly, J. F., *Digital Design – Principles and Practices*, 4th ed. Upper Saddle River, NJ: Pearson Prentice-Hall, 2006.

## Problems

1. Analyze the following functions for combinational hazards:  

$$F(A, B, C, D) = ABC\bar{C} + ACD + \bar{A}BC + \bar{A}\bar{C}D$$
  - (a) finding all single-input-change static hazards and identifying their type (1-hazard or 0-hazard),
  - (b) finding all multiple-input-change static hazards and identifying their type,
  - (c) finding all two-input change function hazards,

- (d) finding all three-input change function hazards, and  
(e) finding all four-input change function hazards.
2. Repeat Problem 1 for the product-of-sums expression for function  $F(A, B, C, D)$  in Problem 1.
  3. Add product (or sum) terms to eliminate both single-input-change and multiple-input-static hazards from the functions in (a) Problem 1 and (b) Problem 2.
  4. By using the following next state equations for A, B, C, and D, find an optimum 2-level function  $G(t + 1)$  that allows the signal F in Problem 1 to be a D flip-flop output G that is identical (implies not delayed) to F in Problem 1.  
$$A(t + 1) = A\bar{B} + C\bar{D}$$
$$B(t + 1) = A$$
$$C(t + 1) = BD$$
$$D(t + 1) = AC + \bar{B}$$