The synchronous circuits that you have studied thus far have their state variable changes synchronized by a special input signal called a clock. In these notes, we briefly study some aspects of asynchronous circuits. An asynchronous circuit has one or more state variable changes that occurs without being directly synchronized by the special clock signal. For example, every flip-flop or latch we have considered can be modeled as an asynchronous circuit if the clock is regarded as just another input rather than a special input for synchronization. Our presentation here is a bit unusual since it does not dwell on the details of asynchronous circuit design. Instead, we focus on the things that can go wrong and why, if at all possible, you should avoid using asynchronous circuits!

Our focus here will deal 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 techniques. The interfaces to be considered are shown in Figure 4-1. First, we will look at the problems of driving an asynchronous circuit with the outputs of a synchronous circuit as in Figure 4-1(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.

Next, we will treat the problem of an asynchronous circuit driving a synchronous circuit as shown in Figure 4-1(b). The asynchronous circuit can be as simple as a latch that deals with a phenomena called contact bounce generated by manually-operated push-buttons or switches. It is obvious that signals originating from a pushbutton are not synchronized with an internal electronic clock. The same problem can also come from a
A synchronous circuit having a clock signal $X$ unrelated to the clock $Y$ of the circuit being driven as in Figure 4-1(c). In such a case, the signals entering the driven circuit are asynchronous with respect to clock $Y$. Both of these cases can cause circuit malfunction, so we offer the synchronizing of such signals as a solution. Following the perverse nature of asynchronous behavior, this solution isn’t perfect, but suffers from a troublesome phenomena referred to as metastability, a topic treated briefly here.

Our final topic is “I thought this was a synchronous circuit; after all, it does have a clock.” Here we will illustrate how a circuit designer can easily fall into the trap of unknowingly producing an asynchronous design.

As indicated previously, the coverage of asynchronous circuits here does not deal with their design. If you are entering the Computer Engineering Option in ECE and actually want to know something about how to design asynchronous circuits, check out Mano’s *Digital Design*, 2nd Edition, Chapter 9. Also, it is important to note that there is a very different approach to asynchronous design that is a product of relatively recent research at Cal Tech, MIT, Stanford and University of California-Berkeley. This more recent approach has been used in industry to design entire computer plug-in boards, but it remains to be seen whether or not it enters the mainstream of digital design.
Combinational Circuit Hazards and Their Prevention

In this section, we will focus on combinational circuit hazards. But first, we present a circuit that can be used to demonstrate that a hazard can really be a bad thing. In general, a combinational hazard occurs if an output signal changes twice when it should not change at all or changes three or more times when it should change only once. An informal name for such a signal effect is a “glitch.”

An Asynchronous Binary Counter

Figure 4-2(a) shows the symbol for an asynchronous binary counter and Figure 4-2(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 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.

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 4-3, a change in input signal X causes all three responses A, B and C. In Figure 4-3(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 4-3(b) illustrates an SICS 0-hazard in which the change in X causes output signal B to momentarily go to 0 when it is supposed to remain at 0. Just to illustrate why we use the term “static,” in Figure 4-3(c), we show a dynamic hazard in Figure 4-3(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 \((1,0)\) 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 SISC 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 4-4** Example of a Static Hazard in a Multiplexer

![Example of a Static Hazard in a Multiplexer](image)

Figure 4-4(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 4-4(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 shows 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.

**Exercise**

In Figure 4-4(a), assume that signal $S$ changes from 0 to 1 instead of from 1 to 0 and repeat the simulation in Figure 4-4(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 4-4(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 $D_0 = D_1 = 1$, 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 4-5(a). The two product terms $S D_1$ and $S D_0$ used in the original design are shown. In order to determine the location of the static hazard on the map, we set $D_1 = D_0 = 1$, 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 $S D_1 D_0$ to minterm $\overline{S} D_1 D_0$. Now, suppose that we combine these two minterms:

$$\overline{S} D_1 D_0 + S D_1 D_0 = D_1 D_0$$

Product term $D_1 D_0$ 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 4-5(b) and then change $S$ from 1 to 0? Since $D_1$ and $D_0$ 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. A sum-of-products implementation is automatically free of static 0-hazards. (Note that if there are don’t cares, there may be some prime implicants that can be left out.) 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.

FIGURE 4-6  
Karnaugh Map and Implementation for a Static Hazard-Free Circuit.

A Karnaugh map for a function $F$ is given in Figure 4-6(a). All prime implicants for $F$ are used in the static hazard-free sum-of-products implementation of $F$ in Figure 4-6(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 minterms 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 opera-
tion. For example, consider the change from 0011 to 0000 in Figure 4-6(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 implementa-
tion, 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. Unfortu-
nately, delay control is difficult and tedious and there are cases where function hazards
cannot be eliminated by using delay control.

Exercise

Identify locations of function hazards that occur for two inputs changing for the multi-
plexer in Figure 4-5(a) with the SICS hazard term included in the design.

FIGURE 4-7 Location of Function Hazards

Answer: See Figure 4-7. 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 syn-
chronous 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 vari-
able 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 syn-
chronous circuit? A different approach is to make the output \( F \) come from a synchro-
nous 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 4-8 illustrates this technique assuming that \( F \) is the function realized by the multiplexer in Figure 4-4. \( F \) 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 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. Since it does cost an additional flip-flop, cost must be carefully considered.

FIGURE 4-8
Use of a Flip-flop to Eliminate Glitches

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 \( D0, D1 \) 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 4-9(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 4-8(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 generate multiple clock phases in computer control units. P0 through P5 can be used as clocks since they are “glitch-free.”

**Synchronization**

In this section, we turn our attention to asynchronous circuits or signals driving synchronous circuits. Initially, we look at the problem that occurs if an asynchronous signal is applied directly to the synchronous circuit without special treatment. Then we offer a solution but find that there is an additional problem with the solution, which we also attempt to remedy.

The circuit in Figure 4-10 can illustrate erroneous behavior due to an input signal not synchronized with the clock. The circuit is initialized by using the Reset signal which sets the state of the circuit to $S_0 \ (y_0, y_1, y_2 = 1, 0, 0)$. As long as $RDY = 1$, the circuit
cycles through the states S0 (1,0,0) and S1 (0,1,0) and S2(0,0,1). If RDY = 0, then the circuit waits in state S0 until RDY = 1 causes it to go to state S1. Also, the state can change from S1 to S2 and from S2 to S0 with RDY = 0. All other combinations of state variables are invalid during the normal operation of the circuit.

Now suppose that RDY is asynchronous with respect to Clock. This means that it can change any time during the clock period. In Figure 4-11(a), the signal RDY changes well away from the positive clock edge, so that the setup and hold times for flip-flops y0 and y1 are easily met. The circuit operates normally. When RDY goes to 0 and the circuit reaches state S0, it waits in state S0 until RDY goes to 1. At the next positive clock edge, the stage changes to S1. The circuit then proceeds to state S2 and back to S0.

In Figure 4-11(b) and Figure 4-11(c), the change in signal RDY from 0 to 1 reaches two flip-flops. The change arrives at the flip-flop inputs very near the positive clock edge within the setup time, hold time interval. This violates the specified operating conditions of the flip-flops. Instead of the flip-flops responding as if they correctly see opposite values at their D inputs, they may respond as if they see the same inputs yielding circuit states (0,0,0) or (1,1,0).

In Figure 4-11(b), y0 resets to 0, but y1 fails to set to 1, giving state (0,0,0). Since there is no 1 to circulate among the flip-flops, the state remains at (0,0,0). The circuit is locked in this state and has failed.

In Figure 4-11(c), y1 sets and y0 fails to reset, giving state (1,1,0). There are now two 1s circulating among the flip-flops, giving state sequence 110, 011, 101. These are all invalid states and give an incorrect output sequence. Thus, the circuit has again failed. Whether or not these failures occur depends upon circuit delays, the setup and hold times, and the detailed behavior of the flip-flops. Since none of these can be tightly con-
trolled, a solution is needed to prevent these failures that is independent of these parameters. Such a solution is the use of a synchronizing flip-flop.

**Synchronizing Flip-flop**

In Figure 4-12(a), a D flip-flop has been added to the example circuit. The asynchronous signal $RDY$ enters the D flip-flop and $RDY_S$, its output, is synchronous with signal Clock in the sense that $RDY_S$ changes one flip-flop delay after the positive edge. Since the asynchronous signal $RDY$ enters the circuit through this single synchronizing flip-
flop, the behavior exhibited when RDY reached two flip-flops is avoided. RDY_S cannot cause such behavior since it does not change during the setup time, hold time interval for the flip-flops.

A remaining question is how does the synchronizing flip-flop behave when RDY changes during the setup time, hold time interval. Basically, the flip-flop either sees the change or it doesn’t. If it doesn’t see it, then the change is seen at the next positive clock edge, one clock period later. Not that this can only happen if the changes in the asynchronous signal are separated by a minimum interval. It is the designer’s responsibility to insure that this minimum interval specification is met by the asynchronous input. The behavior discussed in this paragraph is illustrated in Figure 4-13. The case in which the change in RDY is immediately sensed by the flip-flop and the case in which RDY is not sensed until the next positive clock edge are shown. In the latter case, the response to the change in RDY is delayed by an extra clock period. Since RDY is asynchronous, the fact that the times at which state changes occur due to changes in RDY may vary by a clock period should be of no consequence. If it is critical, then the circuit specifications may not be realizable.

Metastability

At this point, it seems as if we have a solution that deals with the asynchronous input signal problem. Unfortunately, our solution is imperfect. Latches use to construct flip-flops actually have three potential states: stable 1, stable 0 and metastable. These states can best be described by the mechanical analogy in Figure 4-14. The state of the latch is represented by the position of a ball on a hilly surface. If the ball is in the left valley, then the state is a 0. If the ball is in the right valley, then the state is a 1. In order to move the ball between the valleys say from state 0 to state 1, it is necessary to push the ball up the hill and over the top. This requires a certain amount of energy expenditure. If the energy runs out with the ball in position M, it just stays there, halfway between 0 and 1. In fact, however, it will eventually, at some non-deterministic time, go on to 1 or back to 0 due to some mechanical “noise” such as wind, a minor earthquake, or disturbance by some creature. The analogy of this situation in a latch is as follows. When an input to the cross-coupled pair of latch gates changes in just the right timing relationship with
the clock edge, a narrow pulse can be generated. The pulse may have just enough energy to change the latch state to the metastable point where both gates have equal output values with voltages between 1 and 0. Like the mechanical system, the latch and hence the
flip-flop containing it will eventually go to either 0 or 1 due to a tiny electronic “noise”
disturbance. The length of time it remains in the metastable state is non-deterministic.
The interval during which a change in the input will cause metastable behavior is very
narrow, of the order of a few tens of picoseconds. Thus the behavior is unlikely, but it
can happen. When it does, it is unknown how long the metastable state will persist. If it
does persist for a clock period, then the two flip-flops in our example will see a value on
the synchronizing flip-flop output $RDY_S$ that is between 0 and 1. Response by the two
flip-flops to such a value is unpredictable, so there is a good chance that the circuit will
fail.

This phenomena was discovered by two electrical engineering faculty members at
Washington University in St. Louis. In the late 1960’s, I attended a presentation they
made here at Wisconsin. They had pictures of oscilloscope traces showing the metasta-
ble behavior. At about the same time, Digital Equipment Corporation was experiencing
infrequent, unexplained failures in their new, faster computers. You can probably guess
the cause! The nature of metastable behavior for a particular CMOS D flip-flop used as

\[\text{FIGURE 4-15} \quad \text{Metastable Behavior}\]

\[\text{D} \quad \text{CLOCK} \quad \text{Q} \quad 30 \text{ ns} \quad 45 \text{ ns} \quad 13 \text{ ns}\]

a synchronizing flip-flop is shown in Figure 4-15; this data was gathered over 30 min-
utes. The normal delay from the Clock to $Q$ is 13 ns as indicated by the dotted line. But
by carefully controlling the timing of the changes in $D$ and the Clock, the flip-flop is
forced into its metastable region. In that region, the best flip-flop delay seen is 30 ns and
the worst is 45 ns. Thus, if the clock period is less than 45 ns, a metastable event that can
adversely affect the behavior of two or more flip-flops within the circuit being driven by
the synchronizing flip-flop occurs many times in 30 minutes. Actually, although not
shown in the figure, the changes in $Q$ closer to 30 ns are much more frequent than those
close to 45 ns. So the shorter the clock period the worse the problem gets. If the sampling interval were 50 hours, there would be a few events appearing as late as 55 ns. The value between 1 and 0 that occurs for a time inside the flip-flop in this experiment is converted to a longer delay by the output buffer of the flip-flop and so is not visible at the output.

So what can be done about this problem? There have been many solutions proposed, some of which are worthless. A simple one is to use a series of synchronizing flip-flops, i.e., a small shift register. The likelihood of the second flip-flop in the series going metastable because the first one applies a metastable or delayed input to it is less than the first flip-flop going metastable, etc. Some commercial designs have used as many as six flip-flops in series to deal with this problem. More common is the use of two to three flip-flops in series. The more flip-flops, the more the circuit response to a change is delayed and the less likely the circuit is to fail due to metastability. But the probability never goes to zero. Some degree of uncertainty of incorrect operation always remains, however small.

**Synchronous Circuit Pitfalls**

Just because there is a clock does not mean that a circuit is synchronous. For example, in a ripple counter, the clock drives at most one flip-flop clock input directly. All other clock inputs driving the flip-flops are actually state variables. So the changes in the state variables that are the outputs of these flip-flops are not synchronous with the clock. For a 16-bit ripple counter, in the worst case where all flip-flops change state, the most significant bit changes 16 flip-flop delays after the clock edge on the first flip-flop.

In the counter realm, consider the synchronous counter in Figure 4-16. The Clear input is the asynchronous clear to all of the flip-flops of the counter and clears the flip-flops when it is 0. When the count reaches 0110 (6), the NAND output goes to 0 and clears the flip-flops giving 0. So the counter is supposed to count 0, 1, 2, 3, 4, 5, 0, ... But suppose that $A_2$ goes to 0 a bit earlier than $A_1$. Then the output of the NAND can go to a 1 before all flip-flops in the counter have been reliably reset. If flip-flop $A_1$ is slow enough and $A_2$ fast enough, the state 0010 could result instead of 0000. We have actu-

---

* FIGURE 4-16

Example of an asynchronous circuit
ally seen this type of incorrect behavior in the laboratory. Because this counter “kills itself” back to value zero, it is called a suicide counter. Unfortunately, using it is more like committing “job suicide.”

The suicide counter is just one example of a sneaky class of asynchronous circuits posing as synchronous ones. If you use the direct inputs, clear or preset, to a flip-flop for anything other than power-up reset and overall system reset, you have designed an asynchronous circuit. Further, with the complexity of flip-flops plus whatever logic you may have added, you have no idea what sort of hazard problems or other timing problems you may have.

In summary, there are certainly situations where you must use asynchronous circuits to get the desired behavior. But these situations are far fewer than the cases where someone thinks they need an asynchronous circuit. So try to avoid them whenever you can.

As for synchronizing flip-flops, their use is essential in making the transition from asynchronous signals to a synchronous circuit. Care must be taken to deal with metastability. There is a lot more to synchronization than we have presented here. For example, if the timing of a set of asynchronous signals is known relative to another particular asynchronous signal, only the latter signal may need to be synchronized. Also, just because our example suggests making an asynchronous signal enter a circuit through one flip-flop to solve the problem, this does not say in general that a circuit having only one flip-flop instead of two or more will be free of synchronization problems. A notable problem is the case where there is combinational logic containing hazards between the asynchronous signal or signals and the single flip-flop.

References


Corrections or Comments

Your feedback would be greatly appreciated. Please e-mail corrections or comments to kime@engr.wisc.edu.