Table 7-35 Test vectors for the combination-lock state machine of Table 7-31.

```
test_vectors
([CLOCK, X] -> [   S    , UNLK, HINT])
 [ .C. , 1] -> [.X.     , .X. , .X. ]; " Since no reset input, apply
 [ .C. , 1] -> [.X.     , .X. , .X. ]; "   a 'synchronizing sequence'
 [ .C. , 1] -> [.X.     , .X. , .X. ]; "   to reach a known starting
 [ .C. , 1] -> [ZIP     , .X. , .X. ]; "   state
 [  0  , 0] -> [ZIP     ,  0  ,  1  ]; " Test Mealy outputs for both
 [  0  , 1] -> [ZIP     ,  0  ,  0  ]; "   values of X
 [ .C. , 1] -> [ZIP     , .X. , .X. ]; " Test ZIP-->ZIP (X==1)
 [ .C. , 0] -> [X0      , .X. , .X. ]; "   and ZIP-->X0 (X==0)
 [  0  , 0] -> [X0      ,  0  ,  0  ]; " Test Mealy outputs for both
 [  0  , 1] -> [X0      ,  0  ,  1  ]; "   values of X
 [ .C. , 0] -> [X0      , .X. , .X. ]; " Test X0-->X0 (X==0)
 [ .C. , 1] -> [X01     , .X. , .X. ]; "   and X0-->X01 (X==1)
 [  0  , 0] -> [X01     ,  0  ,  0  ]; " Test Mealy outputs for both
 [  0  , 1] -> [X01     ,  0  ,  1  ]; "   values of X
 [ .C. , 0] -> [X0      , .X. , .X. ]; " Test X01-->X0 (X==0)
 [ .C. , 1] -> [X01     , .X. , .X. ]; " Get back to X01
 [ .C. , 1] -> [X011    , .X. , .X. ]; " Test X01-->X011 (X==1)
```

do not test (A LASTA) = 10 in state LOOKING, or (A B LASTA) = 100 in state OK. Thus, generating a complete set of test vectors for fault-detection purposes is a process best left to an automatic test-generation program. In Table 7-35, we petered out after writing vectors for the first few states; completing the test vectors is left as an exercise (7.92). Still, on the functional testing side, writing a few vectors to exercise the machine's most basic functions can weed out obvious design errors early in the process. More subtle design errors are best detected by a thorough system-level simulation.

7.12 VHDL Sequential-Circuit Design Features

Most of the VHDL features that are needed to support sequential-circuit design, in particular, processes, were already introduced in Section 4.7 and were used in the VHDL sections in Chapter 5. This section introduces just a couple more features and gives simple examples of how they are used. Larger examples appear in the VHDL sections of Chapter 8.

7.12.1 Feedback Sequential Circuits

A VHDL process and the simulator's event-list mechanism for tracking signal changes form the fundamental basis for handling feedback sequential circuits in VHDL. Remember that feedback sequential circuits may change state in response to input changes, and these state changes are manifested by changes propagating in a. feedback loop until the feedback loop stabilizes. In simulation,

Table 7-36 Dataflow VHDL for an S-R latch.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vsrlatch is
  port (S, R: in STD_LOGIC;
        Q, QN: buffer STD_LOGIC );
end Vsrlatch;

architecture Vsrlatch_arch of Vsrlatch is
begin
  QN <= S nor Q;
  Q  <= R nor QN;
end Vsrlatch_arch;
```

this is manifested by the simulator putting signal changes on the event list and scheduling processes to re-run in "delta time" and propagate these signal changes until no more signal changes are scheduled.

Table 7-36 is a VHDL program for an S-R latch. The architecture contains two concurrent assignment statements, each of which gives rise to a process as discussed in Section 4.7.9. These processes interact to create the simple latching behavior of an S-R latch.

The VHDL simulation is faithful enough to handle the case where both S and R are asserted simultaneously. The most interesting result in simulation occurs if you negate S and R simultaneously. Recall from the box on page 440 that a real S-R latch may oscillate or go into a metastable state in this situation. The simulation will potentially loop forever as each execution of one assignment statement triggers another execution of the other. After some number of repetitions, a well designed simulator will discover the problem—delta time keeps advancing while simulated time does not—and halt the simulation.

WHAT DO 'U' WANT?

It would be nice if the S-R-latch model in Table 7-36 produced a `'U'` output whenever S and R were negated simultaneously, but it's not *that* good. However, the language is powerful enough that experienced VHDL designers can easily write a model with that behavior. Such a model would make use of VHDL's time-modeling facilities, which we haven't discussed, to model the latch's "recovery time" (see box on page 441) and force a `'U'` output if a second input change occurs too soon. It's even possible to model a maximum assumed metastability resolution time in this way.

Note that if a circuit has the possibility of entering a metastable state, there's no guarantee that the simulation will detect it, especially in larger designs. The best way to avoid metastability problems in a system design is to clearly identify and protect its asynchronous boundaries, as discussed in Section 8.9.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity VposDff is
  port (CLK, CLR, D: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end VposDff;

architecture VposDff_arch of VposDff is
begin
  process (CLK, CLR)
  begin
    if CLR='1' then Q <= '0'; QN <= '1';
    elsif CLK'event and CLK='1' then Q <= D; QN <= not D;
    end if;
  end process;
end VposDff_arch;
```

Table 7-37 Behavioral VHDL for a positive-edge-triggered D flip-flop.

7.12.2 Clocked Circuits

In practice, the majority of digital designs that are modeled using VHDL are clocked, synchronous systems using edge-triggered flip-flops. In addition to what we've already learned about VHDL, there's just one more feature needed to describe edge-triggered behavior. The *`event` attribute* can be attached to a signal name to yield a value of type `boolean` that is `true` if an event on the signal caused the encompassing process to run in the current simulation cycle, and `false` otherwise.

`event` attribute

Using the `event` attribute, we can model the behavior of a positive-edge triggered D flip-flop with asynchronous clear as shown in Table 7-37. Here, the asynchronous clear input `CLR` overrides any behavior on the clock input `CLK`, and is therefore checked first, in the "`if`" clause. If `CLR` is negated, then the "`elsif`" clause is checked, and its statements are executed on the rising edge of `CLK`. Note that "`CLK'event`" is true for any change on `CLK`, and "`CLK='1'`" is checked to limit triggering to just the rising edge of `CLK`. There are many other ways to construct processes or statements with edge-triggered behavior; Table 7-38 shows two more ways to describe a D flip-flop (without a `CLR` input).

In the test bench for a clocked circuit, one of things you need to do is to generate a system clock signal. This can be done quite easily with a loop inside a process, as shown in Table 7-39 for a 100 MHz clock with a 60% duty cycle.

```
process
  wait until CLK'event and CLK='1';
  Q <= D;
end process;

Q <= D when CLK'event and CLK='1' else Q;
```

Table 7-38 Two more ways to describe a positive-edge-triggered D flip-flop.

SYNTHESIS STUFF

You may be wondering, how does a synthesis tool convert the edge-triggered behavior described in Table 7-37 or 7-38 into an efficient flip-flop realization? Most tools recognize only a few predetermined ways of expressing edge-triggered behavior, and map those into predetermined flip-flop components in the target technology.

The Synopsis synthesis engine used in the Xilinx Foundation Series 1.5 software recognizes the "`CLK'event and CLK='1'`" expression that we use in this book. Even with that as a given, VHDL has many different ways of expressing the same functionality, as we showed in Table 7-38. Peter Ashenden, author of *The Designer's Guide to VHDL* (Morgan Kaufmann, 1996), ran these statements and one other, with some modification, through several different synthesis tools. Only one of them was able to synthesize three out of the four forms; most could handle only two. So, you need to follow the method that is prescribed by the tool you use.

Table 7-39 Clock process within a test bench.

```
architecture TB_arch of TB is
signal MCLK: STD_LOGIC;
signal ... -- Declare other input and output signals

process -- Clock generator
begin
  MCLK <= 1;  -- Start at 1 at time 0
  loop
    MCLK <= 0 after 6 ns;
    MCLK <= 1 after 4 ns;
  end loop;
end process;

process -- Generate the rest of the input stimulii , check outputs
begin
  ...
end;
```

References

The problem of metastability has been around for a long time. Greek philosophers wrote about the problem of indecision thousands of years ago. A group of modern philosophers named Devo sang about metastability in the title song of their *Freedom of Choice* album. And the U.S. Congress still can't decide how to "save" Social Security.

Scan capability was first deployed in latches, not flip-flops, in IBM IC designs decades ago. Edward J. McCluskey has a very good discussion of this and other scan methods in *Logic Design Principles* (Prentice Hall, 1986).