The program in Table 9-21 does not specify a state assignment; a typical synthesis engine will use three bits for `Sreg` and assign the six states in order to binary combinations 000–101. For this state machine, it is also possible to use an output coded state assignment, using just the lamp and error output signals that are already required. VHDL does not provide a convenient mechanism for grouping together the entity's existing output signals and using them for state, but we can still achieve the desired effect with the changes shown in Table 9-22. Here we used a comment to document the correspondence between outputs and the bits of the new, 5-bit `Sreg`, and we changed each of the output assignment statements to pick off the appropriate bit instead of fully decoding the state.

Table 9-22 VHDL architecture for guessing game using output-coded state assignment.

```
architecture Vggameoc_arch of Vggame is
signal Sreg: STD_LOGIC_VECTOR (1 to 5);
-- bit positions of output-coded assignment: L1, L2, L3, L4, ERR
constant S1:   STD_LOGIC_VECTOR (1 to 5) := "10000";
constant S2:   STD_LOGIC_VECTOR (1 to 5) := "01000";
constant S3:   STD_LOGIC_VECTOR (1 to 5) := "00100";
constant S4:   STD_LOGIC_VECTOR (1 to 5) := "00010";
constant SERR: STD_LOGIC_VECTOR (1 to 5) := "00001";
constant SOK:  STD_LOGIC_VECTOR (1 to 5) := "00000";
begin

  process (CLOCK)
  ...              (no change to process)
  end process;

  L1  <= Sreg(1);
  L2  <= Sreg(2);
  L3  <= Sreg(3);
  L4  <= Sreg(4);
  ERR <= Sreg(5);

end Vggameoc_arch;
```

9.2.4 Reinventing Traffic-Light Controllers

If you read the ABEL example in Section 9.1.5, then you've already heard me rant about the horrible traffic light controllers in Sunnyvale, California. They really *do* seem to be carefully designed to *maximize* the waiting time of cars at intersections. In this section we'll design a traffic-light controller with distinctly Sunnyvale-like behavior.

An infrequently used intersection (one that would have no more than a "yield" sign if it were in Chicago) has the sensors and signals shown in

Table 9-23 VHDL program for Sunnyvale traffic-light controller.

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

entity Vsvale is
  port ( CLOCK, RESET, NSCAR, EWCAR, TMSHORT, TMLONG: in  STD_LOGIC;
         OVERRIDE, FLASHCLK:                          in  STD_LOGIC;
         NSRED, NSYELLOW, NSGREEN:                    out STD_LOGIC;
         EWRED, EWYELLOW, EWGREEN, TMRESET:           out STD_LOGIC );
end;

architecture Vsvale_arch of Vsvale is
type Sreg_type is (NSGO, NSWAIT, NSWAIT2, NSDELAY,
                   EWGO, EWWAIT, EWWAIT2, EWDELAY);
signal Sreg: Sreg_type;
begin

process (CLOCK)
begin
  if CLOCK'event and CLOCK = '1' then
    if RESET = '1' then Sreg <= NSDELAY; else
      case Sreg is
        when NSGO =>                                              -- North-south green.
          if    TMSHORT='0' then Sreg <= NSGO;                    -- Minimum 5 seconds.
          elsif TMLONG='1'  then Sreg <= NSWAIT;                  -- Maximum 5 minutes.
          elsif EWCAR='1' and NSCAR='0' then Sreg <= NSGO;        -- Make EW car wait.
          elsif EWCAR='1' and NSCAR='1' then Sreg <= NSWAIT;      -- Thrash if cars both ways.
          elsif EWCAR='0' and NSCAR='1' then Sreg <= NSWAIT;      -- New NS car? Make it stop!
          else                               Sreg <= NSGO;        -- No one coming, no change.
          end if;
        when NSWAIT  => Sreg <= NSWAIT2;                          -- Yellow light,
        when NSWAIT2 => Sreg <= NSDELAY;                          --   two ticks for safety.
        when NSDELAY => Sreg <= EWGO;                             -- Red both ways for safety.
        when EWGO =>                                              -- East-west green.
          if    TMSHORT='0' then Sreg <= EWGO;                    -- Same behavior as above.
          elsif TMLONG='1'  then Sreg <= EWWAIT;
          elsif NSCAR='1' and EWCAR='0' then Sreg <= EWGO;
          elsif NSCAR='1' and EWCAR='1' then Sreg <= EWWAIT;
          elsif NSCAR='0' and EWCAR='1' then Sreg <= EWWAIT;
          else                               Sreg <= EWGO;
          end if;
        when EWWAIT  => Sreg <= EWWAIT2;
        when EWWAIT2 => Sreg <= EWDELAY;
        when EWDELAY => Sreg <= NSGO;
        when others  => Sreg <= NSDELAY;                          -- "Reset" state.
      end case;
    end if;
  end if;
end process;
```

Figure 9-5 on page 809. The state machine that controls the traffic signals uses a 1 Hz clock and a timer and has four inputs:

NSCAR	Asserted when a car on the north-south road is over either sensor on either side of the intersection.
EWCAR	Asserted when a car on the east-west road is over either sensor on either side of the intersection.
TMLONG	Asserted if more than five minutes has elapsed since the timer started; remains asserted until the timer is reset.
TMSHORT	Asserted if more than five seconds has elapsed since the timer started; remains asserted until the timer is reset.

The state machine has seven outputs:

NSRED, NSYELLOW, NSGREEN	Control the north-south lights.
EWRED, EWYELLOW, EWGREEN	Control the east-west lights.
TMRESET	When asserted, resets the timer and negates TMSHORT and TMLONG. The timer starts timing when TMRESET is negated.

A typical, municipally approved algorithm for controlling the traffic lights is embedded in the VHDL program of Table 9-23. This algorithm produces two frequently seen behaviors of "smart" traffic lights. At night, when traffic is light, it holds a car stopped at the light for up to five minutes, unless a car approaches on the cross street, in which case it stops the cross traffic and lets the waiting car go. (The "early warning" sensor is far enough back to change the lights before the approaching car reaches the intersection.) During the day, when traffic is heavy and there are always cars waiting in both directions, it cycles the lights every five seconds, thus minimizing the utilization of the intersection and maximizing everyone's waiting time, thereby creating a public outcry for more taxes to fix the problem.

Table 9-23 (continued) VHDL program for Sunnyvale traffic-light controller.

```
TMRESET  <= '1' when Sreg=NSWAIT2 or Sreg=EWWAIT2 else '0';
NSRED    <= FLASHCLK when OVERRIDE='1' else
            '1' when Sreg/=NSGO and Sreg/=NSWAIT and Sreg/=NSWAIT2 else '0';
NSYELLOW <= '0' when OVERRIDE='1' else
            '1' when Sreg=NSWAIT or Sreg=NSWAIT2 else '0';
NSGREEN  <= '0' when OVERRIDE='1' else '1' when Sreg=NSGO else '0';
EWRED    <= FLASHCLK when OVERRIDE='1' else
            '1' when Sreg/=EWGO and Sreg/=EWWAIT and Sreg/=EWWAIT2 else '0';
EWYELLOW <= '0' when OVERRIDE='1' else
            '1' when Sreg=EWWAIT or Sreg=EWWAIT2 else '0';
EWGREEN  <= '0' when OVERRIDE='1' else '1' when Sreg=EWGO else '0';

end Vsvale_arch;
```

While writing the program, we took the opportunity to add two inputs that weren't in the original specification. The OVERRIDE input may be asserted by the police to disable the controller and put the signals into a flashing-red mode at a rate determined by the FLASHCLK input. This allows them to manually clear up the traffic snarls created by this wonderful invention.

Table 9-24 Definitions for Sunnyvale traffic-lights machine with output-coded state assignment.

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

entity Vsvale is
  port ( CLOCK, RESET, NSCAR, EWCAR, TMSHORT, TMLONG: in  STD_LOGIC;
         OVERRIDE, FLASHCLK:                          in  STD_LOGIC;
         NSRED, NSYELLOW, NSGREEN:                    out STD_LOGIC;
         EWRED, EWYELLOW, EWGREEN, TMRESET:           out STD_LOGIC );
end;

architecture Vsvaleoc_arch of Vsvale is
signal Sreg: STD_LOGIC_VECTOR (1 to 7);
-- bit positions of output-coded assignment:  (1) NSRED, (2) NSYELLOW, (3) NSGREEN,
--                                (4) EWRED, (5) EWYELLOW, (6) EWGREEN, (7) EXTRA
constant NSGO:    STD_LOGIC_VECTOR (1 to 7) := "0011000";
constant NSWAIT:  STD_LOGIC_VECTOR (1 to 7) := "0101000";
constant NSWAIT2: STD_LOGIC_VECTOR (1 to 7) := "0101001";
constant NSDELAY: STD_LOGIC_VECTOR (1 to 7) := "1001000";
constant EWGO:    STD_LOGIC_VECTOR (1 to 7) := "1000010";
constant EWWAIT:  STD_LOGIC_VECTOR (1 to 7) := "1000100";
constant EWWAIT2: STD_LOGIC_VECTOR (1 to 7) := "1000101";
constant EWDELAY: STD_LOGIC_VECTOR (1 to 7) := "1001001";

begin

process (CLOCK)
...             (no change to process)
end process;

TMRESET  <= '1' when Sreg=NSWAIT2 or Sreg=EWWAIT2 else '0';
NSRED    <= Sreg(1);
NSYELLOW <= Sreg(2);
NSGREEN  <= Sreg(3);
EWRED    <= Sreg(4);
EWYELLOW <= Sreg(5);
EWGREEN  <= Sreg(6);

end Vsvaleoc_arch;
```

Like most of our other examples, Table 9-23 does not give a specific state assignment. And like many of our other examples, this state machine works well with an output-coded state assignment. Many of the states can be identified by a unique combination of light-output values. But there are three pairs of states that are not distinguishable by looking at the lights alone: (NSWAIT, NSWAIT2), (EWWAIT, EWWAIT2), and (NSDELAY, EQDELAY). We can handle these by adding one more state variable, "EXTRA", that has different values for the two states in each pair. This idea is realized in the modified program in Table 9-24.

Exercises

9.1 Write an ABEL program for the state machine described in Exercise 7.30.

9.2 Modify the ABEL program of Table 9-2 to include the HINT output from the original state-machine specification in Section 7.4.

9.3 Redesign the T-bird tail-lights machine of Section 9.1.3 to include parking-light and brake-light functions. When the BRAKE input is asserted, all of the lights should go on immediately, and stay on until BRAKE is negated, independent of any other function. When the PARK input is asserted, each lamp is turned on at 50% brightness at all times when it would otherwise be off. This is achieved by driving the lamp with a 100 Hz signal DIMCLK with a 50% duty cycle. Draw a logic diagram for the circuit using one or two PLDs, write an ABEL program for each PLD, and write a short description of how your system works.

9.4 Find a 3-bit state assignment for the guessing-game machine in Table 9-5 that reduces the maximum number of product terms per output to 7. Can you do even better?

9.5 The operation of the guessing game in Section 9.1.4 is very predictable; it's easy for a player to learn the rate at which the lights change and always hit the button at the right time. The game is more fun if the rate of change is more variable.

Modify the ABEL state machine in Table 9-5 so that in states S1–S4, the machine advances occurs only if a new input, SEN, is asserted. (SEN is intended to be hooked up to a pseudo-random bit-stream generator.) Both correct and incorrect button pushs should be recognized whether or not SEN is asserted. Determine whether your modified design still fits in a 16V8.

9.6 In connection with the preceding exercise, write an ABEL program for an 8-bit LFSR using a single 16V8, such that one of its outputs can be used as a pseudo-random bit-stream generator. After how many clock ticks does the bit sequence repeat? What is the maximum number of 0s that occur in a row? of 1s?

9.7 Add an OVERRIDE input to the traffic-lights state machine of Figure 9-7, still using just a single 16V8. When OVERRIDE is asserted, the red lights should flash on and off, changing once per second. Write a complete ABEL program for your machine.

9.8 Modify the behavior of the ABEL traffic-light-controller machine in Table 9-9 to have more reasonable behavior, the kind you'd like to see for traffic lights in your own home town.