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 only if a new input, SEN, is asserted. (SEN is intended to be hooked up to a pseudorandom bit-stream generator.) Both correct and incorrect button pushes 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.

9.9 Write a VHDL program for the state machine described in Exercise 7.30.

9.10 Show how to modify Table 9-13 to provide an asynchronous `RESET` input that forces the state machine to the `INIT` state. Repeat for a synchronous version that forces the state machine to the `INIT` state if `RESET` is asserted on the rising clock edge.

9.11 Write and test a VHDL program corresponding to Figure 9-8(b). Using an available synthesis tool, determine whether the circuit resulting from this version differs from the one resulting from Figure 9-8(a), and how.

9.12 Write a VHDL program for the ones-counting state machine as described by the state table in Table 7-12.

9.13 Write a VHDL program for the combination-lock state machine as described by the state table in Table 7-14.

9.14 Modify the VHDL program of Table 9-19 to include the HINT output from the original state-machine specification in Section 7.4.

9.15 Repeat Exercise 9.3 using VHDL, assuming you are targeting your design to a single CPLD or FPGA.

9.16 Modify the VHDL guessing-game program of Table 9-21 according to the idea in Exercise 9.5. Add another process to the program to provide a pseudorandom bit-stream generator according to Exercise 9.6.

9.17 Modify the behavior of the VHDL traffic-light-controller machine in Table 9-23 to have more reasonable behavior, the kind you'd like to see for traffic lights in your own home town.