switches through programming language–like behavioral descriptions. Finally, we have reviewed the changing technological landscape, focusing on the new approaches for building digital systems more rapidly: computer-aided design tools and user-programmable devices. We are now ready to begin a more serious study of the design and implementation techniques for digital systems.

Further Reading

A very good description of the design process can be found in Chapter 3 of S. Dasgupta's book *Computer Architecture: A Modern Synthesis*, John Wiley, New York, 1989. An excellent discussion of a variety of programmable logic technologies can be found in R. C. Alford's book, *Programmable Logic Designer's Guide*, published by Howard W. Sams & Co., Indianapolis, IN, in 1989. For those not familiar with the basic background concepts of electronics, a gentle introduction can be found in T. M. Frederiksen's work, *Intuitive Digital Computer Basics*, published by McGraw-Hill, New York, in 1988 (the entire "Intuitive" series is quite good). The classic text on digital design for very large scale integrated circuits is by Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980. The complete spectrum of computer-aided design tools is described in Steven Rubin's text *Computer Aids for VLSI Design*, Addison-Wesley, Reading, MA, 1987. Carlo Séquin covers the issues involved in managing the complexity of large complex digital designs in his paper "Managing VLSI Complexity: An Outlook," which appeared in *Proceedings of the IEEE*, 71:1, 149–166 (January 1983).

Exercises

1.1 *(Description of Digital System Behavior)* Develop flowchart-like diagrams similar to Figure 1.2 for the following variations on the basic traffic light controller described in Section 1.1.1. Make reasonable assumptions about the duration of lights if not otherwise specified. Consider each variation independently (each is independent of the original specification represented by Figure 1.2).

a. Suppose a left-turn arrow is added, but only in the direction of drivers facing North from the South (they wish to make left turns from South to West). The green arrow should be illuminated for 15 seconds, and during this time the lights are red for East-West and red for the South-facing traffic. The drivers facing North see the sequence: green arrow (15 seconds), green (30 seconds), yellow (15 seconds), red (60 seconds), and repeat. From this specification, you should be able to determine the light timings for the other three directions.

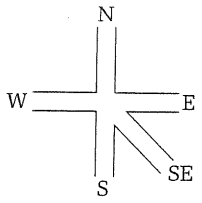

Figure Ex1.1 Intersection for Exercise 1.1(d).

b. Consider adding Walk–Don't Walk signs in all directions. These cycle through green Walk, flashing red Don't Walk, and solid red Don't Walk. The lights are green for 30 seconds, flashing red for 15 seconds, and solid red for 75 seconds.

c. In all directions, add push-buttons that have the following effect: if the light is red in the direction in which the pedestrian wishes to cross, the green time duration in the other direction is reduced from 45 to 30 seconds. Pushing the button more than once or from more than one corner has no further effect.

d. Consider the design of a traffic light for a five-way intersection. The directions are N-S, E-W, and SE (see Figure Ex1.1). No direction should be green for more than 45 seconds and yellow for more than 15. Every direction should eventually see a green light. *Warning:* Make sure you never have more than one direction green at the same time!

1.2 *(Design as Assembly)* Think of a complex system that you know well, such as the automobile you drive or the structure in which you live.

a. Describe the decomposition of this "complex object" into ever more primitive components, stopping at a reasonable level of "most primitive" component (e.g., a brick, a nail, a piece of lumber).

b. Consider some alternative representations of your complex system. Briefly explain what they are. (*Hint*: Do the electrician and the sanitation engineer refer to the same representation of your house when they need to repair something?)

1.3 *(Logical Statements)* Write logic statements for the traffic light variants of Exercise 1.1, using IF-THEN statements and AND, OR, and NOT connectives, as described in Section 1.2.1.

1.4 *(Logical Statements)* Make the following assumptions about a burglar alarm system in your home: (1) you cannot set the alarm unless all windows and doors are closed; (2) the system is "preset" if (1) is true and the secret code has been entered; (3) the system is "set" if (2) is true and 45 seconds have elapsed since presetting the alarm; (4) if the alarm is set, opening any window or a door other than the front door will cause the alarm to sound immediately; (5) if the front door is opened and the alarm is set, it will sound if the system is not disarmed within 30 seconds; (6) the system is disarmed by entering the secret code. Write logic statements for:

a. Setting the alarm

b. Disarming the alarm

c. Sounding the alarm

1.5 *(Analog vs. Digital)* Consider the inverter transfer characteristic described in Section 1.2.3. Suppose two inverter circuits are placed in series so that the output of the first inverter is the input to the second inverter. Assume initially that the input to the first stage is a logic 1 represented by 5 volts. Of course, the output of the second stage will be identical, at least initially. Describe what happens to the outputs of the first and second stages as the first stage input slowly changes from 5 volts to 0 volts. Do this by drawing a graph whose X axis is time and whose Y axis is voltage, showing two curves, one each for (a) the first stage output and (b) the second stage output.

1.6 *(Combinational vs. Sequential Circuits)* Which of the following contain circuits that are likely to be combinational and which contain sequential circuits? Explain your rationale.

a. A washing machine that sequences through the soak, wash, and spin cycles for preset periods of time.

b. A three-input majority circuit that outputs a logic 1 if any two of its inputs are 1.

c. A circuit that divides two 2-bit numbers to yield a quotient and a remainder.

d. A machine that takes a dollar bill and gives three quarters, two dimes, and a nickel in change, one at a time through a single coin change slot.

e. A digital alarm clock that generates an alarm when a preset time has been reached.

1.7 *(Switching Networks)* Draw switching networks as described in Section 1.3.1 for the three conditions of Exercise 1.4.

1.8 *(Switching Networks)* Although we concentrate mostly on gate-level designs in the following chapters, switching logic as described in Section 1.3.1 is quite useful for functions that "steer" inputs to outputs, such as *shifters* and *multiplexers/demultiplexers*. The functions of these devices are described by the following specifications. Design networks of switches for the following functions:

a. A 2-bit-wide *shifter* takes two input signals, i_0 and i_1, and shifts them to two outputs, o_0 and o_1, under the control of a shift signal. If this signal SHIFT is false, then the inputs are connected straight through to the outputs. If SHIFT is true, then i_0 is routed to o_1 and o_0 should be set to a 0.

b. A 1-bit *demultiplexer* takes an input signal IN and shifts it to one of two outputs, o_0 and o_1, under the control of a single SELECT signal. If SELECT is 0, then IN is connected through to o_0 and o_1 is connected to a 0. If SELECT is 1, then IN is connected to o_1 and o_0 should be connected to a 0.

c. A 2-bit *multiplexer* takes two input signals, i_0 and i_1, and shifts one of them to the single output OUT under the control of a 1-bit select signal. If the SELECT signal is false, then i_0 is passed to OUT. If SELECT is true, then i_1 is passed to OUT.

1.9 *(Truth Tables)* Write truth tables for the three functions of Exercise 1.8.

1.10 *(Boolean Algebra)* Write sum of products expressions for the truth tables of Exercise 1.9.

1.11 *(Gates)* Given the Boolean expressions of Exercise 1.10, draw logic schematics using AND, OR, and INVERT gates that implement those functions.

1.12 *(Design Representations)* Examine the three switching networks in Figure Ex1.12. Write a truth table for each of the networks. For each input combination, describe briefly how the network operates.

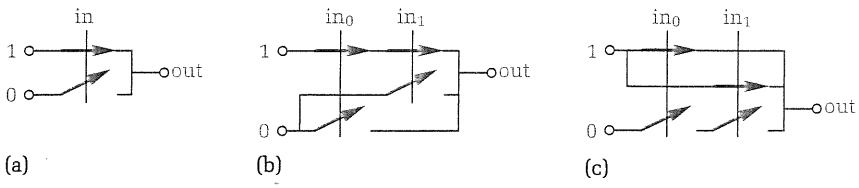

Figure Ex1.12 Switching networks for Exercise 1.12.

1.13 *(Waveforms)* Trace the propagation of 1's and 0's through the half adder of Section 1.3.4 to explain why the glitch occurs when the inputs switch from $A = 0$, $B = 1$ to $A = 1$, $B = 0$. Assume all gates have the same delay of 10 time units.

1.14 *(Block Diagrams)* Given the truth table for the half adder, show that the composition of two half adders and an OR gate as in Section 1.3.6 yields the same truth table as the full adder.

1.15 *(Waveform Verification)* This chapter has described two different gate-level implementations for a full adder circuit: direct implementation, as in Figure 1.22, and hierarchical implementation via cascaded half adders, as in Figure 1.25(a). Would you

expect the waveform behaviors of these implementations to be identical? Justify your answer.

1.16 *(Behaviors)* Write a program in your favorite programming language that mimics the behavior of (a) the basic traffic light controller of Section 1.1.1 and the four variations (b through e) described in Exercise 1.1.

1.17 *(Synthesis)* Simplify the following Boolean expressions by examining their truth tables for simpler terms that cover multiple 1's rows of the truth table:

$$F(A, B, C) = \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$

$$F(A, B, C) = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + AB\overline{C} + ABC$$

1.18 *(Programming with 1's and 0's)* Describe the contents of a memory that you would use to implement the three functions of Exercise 1.8. Identify the address inputs and signal outputs.

1.19 *(Truth Tables)* Consider a function that takes as input two 2-bit numbers and produces as output a 3-bit sum. Write the truth table for this function.

1.20 *(Truth Tables)* An increment-by-1 function takes a single-bit input and generates a *Sum* and *Carry* as follows. If the input is 0, *Sum* is 1 and *Carry* is 0. If the input is 1, *Sum* is 0 and *Carry* is 1. Using the truth table for the full adder, demonstrate that you can implement the increment-by-1 function by setting *Cin* of the full adder to 1 while the *B* input is set to 0. Can you think of any reasons why it may be advantageous to use a standard building block like the full adder rather than a special circuit?