4.13 Show the computations that a concurrent fault simulation would perform on the circuit of Figure 4.5(a). Assume S-A-1 faults on the inputs of gates 1 and 2, initially set all nets to x, then use the sequence $\overline{\text{Set}}, \overline{\text{Reset}} = \{(0, 1), (1, 1), (1, 0), (1, 1)\}$. Assume the elements have identical rise and fall times of 1 unit.

THE SIMULATOR: A Description

The remaining problems for this chapter are based on the simulator which follows. It is a three-valued $(0, 1, x)$, event-driven simulator written in Microsoft BASIC for a Radio Shack Model II computer with 64K memory. The simulator is barely usable as it exists and some of the problems will request elementary enhancements intended to make it more usable and to make its output readable.

The simulator recognizes seven element types. When coding a circuit description for input to the simulator, the function is specified by means of a one-letter code. The functions and their corresponding one-letter codes are: Primary Input (I), AND (A), OR (O), Invert (N), Primary Output (Y), Delay (D), Fan-out (F). When entering a circuit description into the program, the one-letter function code is always the first item on a line. It is followed by the function name, which is a decimal number between 1 and MAX, where MAX is simply an arbitrary upper limit on the number of elements in the circuit. The remaining items in a line of input are nominal delay values, the names of the inputs to this element, and the name of the output(s) driven by the named element.

The general format for the circuit description statements is as follows:

Input: I, name, output

Output: Y, name, input

Invert: N, name, delay, input, output

Delay: D, name, delay, input, output

Fan-out: F, name, output1, output2, output3, output4

OR: O, name, delay, input1, input2, input3, input4, output

AND: A, name, delay, input1, input2, input3, input4, output

The third entry in a statement depends on the function. For a primary input, the third entry is the name of the gate to which the input is connected; for a primary output it is the name of the input which drives the output; and for the logic gates and the Delay element, the third entry specifies the amount of delay inherent in the element. The Fan-out element is used when an element drives more than one element. For instance, if an AND gate drives two or more other gates, then the AND gate in the circuit model drives a Fan-out element, which in turn drives the elements normally driven by the AND gate in the actual circuit. This is merely an artifice to simplify the simulation program. The third through sixth elements in the Fan-out entry list the gates driven by the original device which are now driven by the Fan-out element.

The remaining entries in the logic gate formats identify one or more inputs, and then an output. If an AND gate or an OR gate has fewer than four inputs, the unused positions must be indicated by consecutive commas in the input statement. If an AND (OR) used in a circuit has more than four inputs, then two or more four-input AND (OR) gates can be ANDed (ORed) together to model the gate. The last statement in a circuit description is ENDCKT.

After the circuit description has been processed, the remaining statements are used to specify simulation times, the inputs that are scheduled to change, and the new value they are to assume. The first number specifies the time at which the values are to be scheduled. Then the remaining entries on a statement are, alternately, the names of inputs and the logic value assigned. The entries are separated by commas and the last entry is an asterisk. The end of input stimuli is indicated by a statement which begins with the number 9999.

Simulation of an element is performed when a change occurs on any of its inputs. However, the output value is not updated immediately; rather, the element is scheduled for update at time $t + d$, where t is present time and d is the nominal delay of the element. At time $t + d$ the computed value is compared to the existing value on the output of the gate and, if it differs from the existing value, then the output value is updated and elements in the fan-out of the gate are simulated. The three values 0, 1, and x are encoded using two binary bits as follows:

$$0,0 — 0$$
$$0,1 — x$$
$$1,1 — 1$$

The timing wheel, in the fashion of a clock, advances time by use of a counter, analogous to the minute hand on a clock. One complete rotation of the wheel represents W1 time units, which, in the simulator, are referred to as minutes (MI). After each complete rotation, MI is reset to zero and hours (HR) is incremented. The wheel is represented by two arrays, the usage bit (WH) and the link pointer (LP). If one or more elements must be processed at time i, then the usage bit WH(i) is turned on and link pointer LP(i) points to an entry in a queue.

The queue contains six items for each entry. The name of the element to be updated is placed in the queue array (QU). If additional elements are to be simulated at this time, then another link pointer (LK) points to the next element in the queue that is linked to this time slot. The link flag (LF) is on ($= 1$) if another element is linked to the element presently being processed, and it is off ($= 0$) if the present element is the last one in the linked list. The usage bit US(i) simply indicates whether position i in the queue is available or is presently in use. The scheduler searches through the queue to find an empty slot whenever an event must be scheduled. When an empty slot is found, the element is placed in the queue and the usage bit is turned on. After the element is processed the usage bit is turned off and the slot is free to be used again.

The value array (VL) stores the computed value on the element. This value is compared to the existing value stored in the (VA) array. The future time array (FT) is used to schedule remote events. These are events which occur too far in the future to permit scheduling within the next W1 time units. For example, if an element has a delay of 250 units, then it is scheduled for $t + [250(\text{mod W1})]$ but FT(i) is set equal to 250/W1. When this element is encountered in the queue, if $\text{FT}(i) > 0$, then it is decremented but the element is left in the queue. If $\text{FT}(i) = 0$, then the element is processed and removed from the queue.

The program first reads in the circuit description and creates a circuit model. Then the input stimuli are read. When inputs and their logic values are read in, they are scheduled on the timing wheel. The number of lines of input stimuli read in at a given time is determined by the length of the timing wheel. Rather than read in all input stimuli and schedule them for future time via the FT array, the read process is temporarily suspended when a stimulus time exceeds present time plus maximum wheel time.

Simulation begins at statement 790. Each timing wheel slot is examined to determine if any entries are scheduled for that time. If not, then time is incremented. If the slot is nonempty, then processing is performed as described earlier. Two major subroutines exist

at the end of the program. The first one puts an entry in the queue and either links it to the timing wheel slot, if the slot is empty, or links it to the last entry in the list presently linked to the wheel. When all processing at time i is complete, the second routine removes entries from the wheel. If queue position i is linked to the wheel, and if FT(i) is zero, then the entry is deleted. If FT(i) is nonzero, the entry is retained and FT(i) is decremented. When the end of the timing wheel is reached, the hours variable (HR) is incremented and the software which reads the input stimuli is revisited to again load the timing wheel. This continues until there are no additional input stimuli and the circuit is stable, that is, there are no more entries on any timing wheel slots.

Some of the later problems suggest program changes designed to make the simulator easier to use. You may prefer to implement some of those suggestions before working problems which make use of the simulator, in particular, those problems which simplify the input format and add some additional primitives such as the NAND, NOR, and exclusive-OR. Additional problems based on the simulator can be found at the end of Chap. 5.

Simulator Listing

```
10 DEFINT A-Z
20 CLS : PRINT @(2,0), "EVENT DRIVEN SIMULATOR":PRINT
22 PRINT "HR: MIN","VALUE":PRINT "-------","-----"
30 MAX = 400                          'MAXIMUM NUMBER OF LOGIC GATES
40 W1 = 100                           'MAX TIMING WHEEL SIZE
50 W2 = 200                           'MAX QUEUE SIZE
60 DIM F(MAX)                         'GATE FUNCTION
70 DIM NA(MAX)                        'GATE NAME
80 DIM D(MAX)                         'GATE DELAY
90 DIM IP(3,MAX)                      'GATE INPUTS (MAX OF 4)
100 DIM DS(MAX)                       'DESTINATION OF SIGNAL
110 DIM VA(1,MAX)                     'GATE VALUE (2 BITS)
120 DIM WH(W1)                        'TIMING WHEEL USAGE BIT
130 DIM LP(W1)                        'LINK POINTER TO QUEUE
140 DIM US(W2)                        'USAGE > 0 IF ENTRY IN THIS SLOT
150 DIM QU(W2)                        'QUEUE - CONTAINS NAME OF ENTRY
170 DIM FT(W2)                        'FUTURE TIME
180 DIM LK(W2)                        'LINK POINTER TO NEXT ENTRY IN STRING
190 DIM LF(W2)                        'LINK FLAG
200 DIM VL(1,W2)                      'NEW SIMULATION VALUE
210 ZX = 0
230 FOR I=0 TO MAX    '-------- READ CIRCUIT DESCRIPTION --------
240 READ F$:FC=9
250 IF F$="I" THEN FC=1 ELSE IF F$="A" THEN FC=2 ELSE IF F$="O" THEN FC=3
260 IF F$="N" THEN FC=4 ELSE IF F$="Y" THEN FC=5 ELSE IF F$="D" THEN FC=6
270 IF F$="F" THEN FC=7 ELSE IF F$="ENDCKT" THEN FC=8
280 ON FC GOTO 290,330,330,310,300,310,320,400,340
290 GOSUB 370:READ DS(NM):GOTO 350
300 GOSUB 370:READ IP(0,NM):GOTO 350
310 GOSUB 370:READ D(NM),IP(0,NM),DS(NM):GOTO 350
320 GOSUB 370:READ IP(0,NM),IP(1,NM),IP(2,NM),IP(3,NM):GOTO 350
330 GOSUB 370:READ D(NM),IP(0,NM),IP(1,NM),IP(2,NM),IP(3,NM),DS(NM):GOTO 350
340 PRINT "ILLEGAL FUNCTION CODE",F$:STOP
350 NEXT I
360 PRINT "CIRCUIT SIZE EXCEEDS LIMIT":STOP
370 READ NM:IF NM > MAX THEN 390
375 VA(0,NM)=0:VA(1,NM)=1     'INITIAL VALUE OF ALL ELEMENTS SET TO "X"
380 F(NM)=FC:NA(NM)=NM:RETURN
390 PRINT "INVALID NAME",NM:STOP
400 ES=0                              'END OF SIMULATION FLAG ES = 0
405 FF=0                              'SET FANOUT FLAG=0
440 MI=0:HR=0
450 SZ=0:SR=0:SM=0                    'INITIAL STIMULUS TIME / HOUR=0 / MINUTES=0
455 GOTO 490
460 MI=MI+1                           'UPDATE SIMULATION TIME
```

```
465 IF MI >= W1 THEN 470
467 IF WH(MI)=0 THEN 460 ELSE 790     'IF NO ACTIVITY -- UPDATE TIME
470 MI=0:HR=HR+1
480 IF SZ=9999 THEN STOP ELSE 510
490 READ SZ:IF SZ=9999 THEN 467
510 SM=SZ:SR=0
520 IF SM < W1 THEN 540
530 SM=SM-W1:SR=SR+1:GOTO 520
540 IF HR < SR THEN 467                'DONT READ STIMULUS
610 READ NM$:IF NM$="*" THEN 490
620 NM=VAL(NM$)
630 READ VA$:Z2(0)=0:Z2(1)=1           'INITIALIZE VALUE TO X
640 IF VA$="0" THEN Z2(1)=0 ELSE IF VA$="1" THEN Z2(0)=1
670 Z1=NM:Z3=SR:Z4=SM
680 GOSUB 1710
690 GOTO 610
715 REM ------------------------- BEGIN SIMULATION ----------------------
716 REM
785 REM          <<<<<< GET FANOUT FOR CHANGING GATES  >>>>>>
786 REM          <<<<<< PUT ON QUEUE AT FUTURE TIME    >>>>>>
787 REM
790 QP=LP(MI)                    'GET POINTER TO FIRST QUEUE ENTRY TO BE UPDATED
800 IF HR < FT(QP) THEN 879
810 NM=QU(QP)                              'GET NAME OF ELEMENT IN QUEUE
812 IF VA(0,NM)=VL(0,QP) AND VA(1,NM)=VL(1,QP) THEN GOTO 879
814 VA(0,NM)=VL(0,QP)                      'UPDATE VALUE ON THE OUTPUT
816 VA(1,NM)=VL(1,QP)                      ' -- OF ELEMENT IN QUEUE
820 FX=F(NM)                               'GET FUNCTION OF THIS ELEMENT
830 IF FX=7 THEN 1315                      'FANOUT FUNCTION SHOULD NOT BE IN QUEUE
840 ND=DS(NM)                              'GET NAME OF DESTINATION GATE
850 FY=F(ND)                               'GET FUNCTION OF DESTINATION GATE
860 ON FY GOSUB 1000,1010,1100,1200,1290,1260,920   'COMPUTE ITS NEW VALUE
865 IF FY=5 OR FY=7 THEN 879                'DO NOT PUT PRIMARY OUTPUT ON WHEEL
870 Z1=ND
871 Z2(0)=NV(0):Z2(1)=NV(1)
873 Z3=HR
874 Z4=MI+D(ND)
876 IF Z4 < W1 THEN 878
877 Z4=Z4-W1:Z3=Z3+1:GOTO 876
878 GOSUB 1710
879 IF FF=1 THEN RETURN                    'IF FF=1, CONTINUE FANOUT PROCESSING
890 IF LF(QP)=0 THEN GOSUB 1880:GOTO 460   'IF ZERO, GOTO PURGE ROUTINE
900 QP=LK(QP):GOTO 800                     'ELSE, GET NEXT ELEMENT IN QUEUE LINK LIST
920 REM ------------------ COME HERE TO PROCESS FANOUT ELEMENT -------------
922 IF FF=1 THEN 1320
930 VA(0,ND)=VA(0,NM)
935 VA(1,ND)=VA(1,NM)
940 FF=1:NF=ND
943 FOR FO=0 TO 3
946 ND=IP(FO,NF)
947 IF ND=0 THEN 952
949 GOSUB 850
952 NEXT FO
955 FF=0
958 GOTO 890
960 REM ------------------------- COMPUTE NEW VALUES ON LOGIC ELEMENTS ---
1000 PRINT "CANNOT HAVE FANOUT TO PRIMARY INPUT":STOP
1010 NV(0)=1:NV(1)=1                       '*** AND ***
1030 FOR IX=0 TO 3
1040 TM=IP(IX,ND)                          'IF INPUT NOT CONNECTED
1050 IF TM=0 THEN 1080                     ' -- THEN SKIP COMPUTATION
1060 NV(0)=NV(0) AND VA(0,TM)
1070 NV(1)=NV(1) AND VA(1,TM)
1080 NEXT IX
1090 RETURN
1100 NV(0)=0:NV(1)=0                       '*** OR ***
1120 FOR IX=0 TO 3
1130 TM=IP(IX,ND)
1140 IF TM=0 THEN 1170
1150 NV(0)=NV(0) OR VA(0,TM)
1160 NV(1)=NV(1) OR VA(1,TM)
1170 NEXT IX
```

```
1180 RETURN
1200 REM                                                '*** INVERT ***
1210 TM=IP(O,ND)
1230 NV(O)=1-VA(1,TM) :NV(1)=1-VA(O,TM)
1250 RETURN
1260 TM=IP(O,ND)                                        '*** DELAY ***
1270 NV(O)=VA(O,TM) :NV(1)=VA(1,TM)
1280 RETURN
1289                                                    '*** PRIMARY OUTPUT ***
1290 IF VA(O,ND)=VA(O,NM) AND VA(1,ND)=VA(1,NM) THEN RETURN
1300 VA(O,ND)=VA(O,NM) :VA(1,ND)=VA(1,NM)
1303 IF VA(O,ND)=1 THEN LO$="1" ELSE IF VA(1,ND)=O THEN LO$="O" ELSE LO$="X"
1305 PRINT HR;MI,LO$
1310 RETURN
1315 PRINT "FANOUT ELEMENT SHOULD NOT BE IN QUEUE":STOP
1320 PRINT "FANOUT NOT PERMITTED TO DRIVE FANOUT":STOP
1330 REM
1700 REM ------------------ PUT NEW ENTRY AT END OF LINKED LIST -----------
1701 REM          CALL WITH
1702 REM                Z1 = GATE NAME
1703 REM                Z2 = NEW VALUE
1704 REM                Z3 = STIMULUS HOUR
1705 REM                Z4 = STIMULUS MINUTE
1710 FOR Z5=ZX TO W2                              'FIND EMPTY SLOT IN QUEUE
1720 IF US(Z5)=O THEN 1735 ELSE NEXT Z5
1722 FOR Z5=O TO ZX-1
1724 IF US(Z5)=O THEN 1735 ELSE NEXT Z5
1730 PRINT "NO MORE ROOM IN EVENT QUEUE":STOP
1735 ZX=Z5
1740 IF WH(Z4) > O THEN 1780
1750 WH(Z4)=1:LP(Z4)=Z5                           'SET LINK BIT AND LINK POINTER
1760 QU(Z5)=Z1:VL(O,Z5)=Z2(O) :VL(1,Z5)=Z2(1) :FT(Z5)=Z3  'PUT ENTRY IN QUEUE
1770 US(Z5)=1:LF(Z5)=O:RETURN                     'SET USAGE BIT, ZERO LINK BIT
1780 SK=LP(Z4)                                    'GET LINK POINTER FROM WHEEL
1790 FOR K1=1 TO W1
1810 IF LF(SK)=O THEN 1840          'IF LINK FLAG=O, REACHED END OF LINKED LIST
1815 SK=LK(SK)                      'GET LINK POINTER
1820 NEXT K1
1830 PRINT "CANNOT FIND END OF LINKED LIST":STOP
1840 LK(SK)=Z5                      'INSERT POINTER TO NEXT STRING ENTRY
1850 US(Z5)=1                       'SET USAGE BIT
1860 LF(SK)=1                       'LINK FLAG = 1
1870 GOTO 1760
1880 REM ------------------------ PURGE WHEEL ------------------------------
1890 QP=LP(MI)
1900 IF HR=FT(QP) AND LF(QP)=O THEN WH(MI)=O:US(QP)=O:RETURN
1910 IF HR<FT(QP) AND LF(QP)=O THEN RETURN
1920 IF HR<FT(QP) THEN UX=QP:QP=LK(QP) :GOTO 1950
1930 US(QP)=O:QP=LK(QP)
1940 LP(MI)=QP:GOTO 1900
1950 IF HR=FT(QP) AND LF(QP)=O THEN QP(UX)=O:RETURN
1960 IF HR<FT(QP) AND LF(QP)=O THEN RETURN
1970 IF HR<FT(QP) THEN UX=QP:QP=LK(QP) :GOTO 1950
1980 US(QP)=O:LK(UX)=LK(QP) :QP=LK(QP) :GOTO 1950
1990 REM ------------------------ END SIMULATOR ----------------------------
3000 DATA I,1,7
3010 DATA I,2,5
3020 DATA I,3,8
3030 DATA I,4,9
3040 DATA F,5,6,7,,
3050 DATA N,6,2,5,8
3060 DATA A,7,2,1,5,,,9
3070 DATA A,8,2,6,3,,,9
3071 DATA O,9,2,7,8,4,,10
3072 DATA Y,10,9
3080 DATA ENDCKT
3100 DATA O,4,1,*
3101 DATA 3,2,O,4,O,*
3102 DATA 7,3,1,*
3103 DATA 14,2,1,*
3104 DATA 20,1,1,*
3106 DATA 30,2,O,*
3108 DATA 40,1,O,*
```

```
3110 DATA 50,4,1,*
3112 DATA 60,4,0,*
3118 DATA 90,2,1,*
3120 DATA 100,4,1,*
3122 DATA 110,1,1,*
3124 DATA 120,1,0,*
3126 DATA 130,4,0,*
3128 DATA 140,3,0,*
3130 DATA 150,1,1,*
3133 DATA 160,2,0,*
3135 DATA 398,4,1,*
3137 DATA 416,4,1,*
3139 DATA 440,4,0,*
3150 DATA 9999
```

Problems Based on the Simulator

4.14 The circuit encoded at the end of the simulator is a 2-to-1 multiplexer with data input ports connected to primary inputs 1 and 3. At time 34 the output goes to 0 even though both inputs are 1. Explain that. Redesign the circuit to prevent it from happening. Resimulate to insure that your design is correct.

4.15 Enter the following two-input AND gate to the simulator:

```
3000 DATA I, 1, 3
3010 DATA I, 2, 3
3020 DATA A, 3, 4, 1, 2,,,4
3030 DATA Y, 4, 3
3040 DATA ENDCKT
3100 DATA 0, 1, 0, 2, 1, *
3110 DATA 4, 2, 0, 1, 1, *
3200 DATA 9999
```

After running the circuit, replace statement 3110 with

```
3110 DATA 4, 1, 1, 2, 0, *
```

and rerun the circuit. If you encoded the simulator and data correctly, the output data will differ. Explain why. Alter the simulator to prevent this from happening.

4.16 If an input change occurs at time t on an element with delay d, the element is simulated immediately. However, the output is not checked until time $t + d$. Why not check immediately after simulation and skip the scheduling process if the output has not changed? Alter the simulator to try this approach. Delete line number 812 and insert a check at 872 before going to the scheduler, then repeat the experiment in the previous problem.

4.17 Add new primitive elements to the simulator. Include NAND, NOR, exclusive-OR.

4.18 Simulate all 32 input combinations on the circuit of Figure 4.20 and try to determine what functions are performed by the circuit. Which inputs appear to be the data lines and which appear to be the control lines?

4.19 In the exercises following Chap. 2, you were asked to redesign the circuit in Figure 2.5 in terms of NAND gates. Simulate your NAND gate model to verify that it is correct.

4.20 The simulator terminates at statement 480. It is possible that the simulator may terminate while residual activity remains scheduled on the timing wheel. Insert a check for activity on the wheel before allowing the program to terminate.

4.37 An effective simulator must feature ease of use, data checking to catch and report user errors, and output formats that permit the user to easily identify relevant information. The number of data entry errors is reduced if the user is not required to enter fan-out elements and destination gates in the circuit description. Write a data input routine that does not require that information. After reading in the circuit, the routine will examine the inputs of each element and use that information to create the destination array DS and the fan-out elements.

4.38 Ease of use is further enhanced if arguments in the input data are not required to be separated by commas. Write a data entry routine that recognizes one or more consecutive blanks as delimiters. If a logic gate has fewer than four inputs, this will be detected by the subroutine; the user will not be required to indicate unused inputs via successive commas.

4.39 Create a *display formatter* routine as follows: read in a list of elements whose values are to be displayed during simulation. The elements may be any combination of primary inputs, primary outputs, and internal elements. Also read in a start time at which simulation results are to be displayed, and an interval n between each display update. Then create a display with a time tape at the top, the element names on the left, and the results of simulation printed horizontally across the screen. If the user requests the values on elements 4, 11, and 37 to be displayed every 4 time units, starting at time 150, then the display may appear as follows (this is sometimes referred to as a logic analyzer format):

```
        150  170  190  210  230  250  270  290  310  330
        V....V....V....V....V....V....V....V....V....V

4       ....11111...1111111111..1111.......11.11111111
11      xxx...111111...1111..11111.................111...
37      x11.......1111..1111111111.....1111...111111.
```

When implementing the display formatter, the display data must be updated every n units. Implement this by creating *bulletins*. The bulletins will be scheduled on the timing wheel like logic events. However, they will be assigned a unique function code so that, when a bulletin is encountered on the timing wheel, the simulator will update the display rather than simulate an element.

4.40 Note the bulletins may fall in the middle of a linked list, causing the display to be updated before all events are evaluated. To avoid this, expand the wheel to 2 * W1 units. Schedule logic events at even positions and bulletins in the immediately following odd-numbered slot so that the display is updated only after all elements linked to a wheel slot have been processed.

4.41 Bulletins can also be used to schedule the read operation for input stimuli. Rewrite the input stimulus read routine to be initiated by a bulletin rather than when the end of the wheel is encountered.

4.42 A commercial quality simulator permits users to enter signal names alphanumerically rather than as numbers. Implement such a capability in the routine that reads the circuit description. Permit alphanumeric names up to six characters in length. When a name is read in, convert it to a number for internal processing but save the name in a table so it can be used with the display formatter.

4.43 Some signals, such as clock signals, switch periodically. Implement a periodic function in which the keyword PERIODIC appears at the beginning of the line, followed by

the name of the signal. This is followed by a positive integer >0 which specifies the duration of each signal and a binary number specifying the initial value of the signal.

4.44 Successive input changes on an element may cause a pulse on the output whose duration is less than the inertial delay of the element. Incorporate the ability to detect inertial delays. The user must be able to specify inertial delay of elements when entering a circuit description.

4.45 Write a serial fault simulator based on the following outline and run the fault simulator on the circuit of Figure 4.20:

```
Create test vectors
Create stuck-at fault list for logic gate inputs
FOR J=1 TO LAST_TEST_VECTOR
  Simulate Good Machine with VECTOR(J):save output response
  FOR I=1 TO LAST_FAULT
    IF FAULT(I) = DETECTED THEN NEXT I
    INSERT FAULT(I):perform simulation of VECTOR(J)
    IF steady-state output ≠ good machine response
    THEN SET FAULT(I) = DETECTED
  NEXT I
NEXT J
```

4.46 When performing serial fault simulation, the simulation time can be reduced by prescreening faults. If an *n*-input AND (OR) gate has two or more 0s (1s) on its inputs after good machine simulation of a test vector, then the test vector cannot possibly detect any of the input stuck-at faults on that gate (why?). Write a routine that prescreens the circuit after each good machine simulation and marks the testable faults so that serial fault simulation only occurs on *potentially detectable* faults.

4.47 Many functions made up of logic gates, such as latches and flip-flops, are used repeatedly in a circuit design. It is tedious and error-prone to encode them every time they are used. A simulator, like an assembler, can use macros. In the simulator, the macro is a description of the function in terms of simulator primitives, using dummy variables. Write a function expander that accepts a gate-level macro description of a function. This macro is replaced by its expanded version when the circuit model is being created. We illustrate with a cross-coupled NAND latch. It uses the freeform input developed in earlier exercises.

```
FUNCTION
LATCH &1,&2,&3,&4,&5,&6
NAND &5 &3 &6 &1
NAND &6 &4 &5 &2
END
```

The latch, when used in a circuit, would appear as follows:

```
LATCH SET RESET DELAY1 DELAY2 OUT1 OUT2
```

The function expander, upon encountering function LATCH, replaces it with the expanded version and replaces dummy variables with arguments present in the function call. The macro functions can be greatly simplified if some of the other simulator capabilities have been first programmed.

4.48 Recode the simulator in a compiled language and compare the amount of run time required for typical circuits of 50 to 100 elements and some fixed number of input stimuli.

4.49 When using descriptor cells, a signal change on the output of a gate is stored in the descriptor cell of the destination gate as one of several contiguous signals. Then, for evaluation, all input values are accessed as a single entity to index into the zoom table and provide a rapid calculation. Furthermore, the manner in which inputs and outputs are linked together in descriptor cells is rather straightforward to implement and does not require special fan-out tables. Implement these concepts in your compiled simulator.

REFERENCES

1. Druian, R. L., "Functional Models for VLSI Design," *Proc. 20th Design Automation Conf.*, 1983, pp. 506–514.
2. Rappaport, A., "Capable Digital-Circuit Simulators Promise Breadboard Obsolescence," *EDN*, March 17, 1983, pp. 105–126.
3. Johnson, W. A., "Behavioral-Level Test Development," *Proc. 16th Design Automation Conf.*, 1979, pp. 171–179.
4. Bryant, R. E., "A Switch-level Model and Simulator for MOS Digital Systems," *IEEE Trans. Comput.*, vol. C-33, no. 2, Feb. 1984, pp. 160–177.
5. Ulrich, E., and D. Hebert, "Speed and Accuracy in Digital Network Simulation Based on Structural Modeling," *Proc. 19th Design Automation Conf.*, 1982, pp. 587–593.
6. Eichelberger, E. B., "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM J. Res. Dev.*, vol. 9, no. 2, March 1965, pp. 90–99.
7. Hardie, F. H. and R. J. Suhocki, "Design and Use of Fault Simulation for Saturn Computer Design," *IEEE Trans. on Elec. Comput.*, vol. EC-16, no. 4, Aug. 1967, pp. 412–429.
8. Fantauzzi, G., "An Algebraic Model for the Analysis of Logical Circuits," *IEEE Trans. Comput.*, vol. C-23, no. 6, June 1974, pp. 576–581.
9. Phillips, N. D., and J. G. Tellier, "Efficient Event Manipulation: The Key to Large Scale Simulation," *Proc. 1978 IEEE Int. Test Conf.*, pp. 266–273.
10. Ulrich, E. G., "Exclusive Simulation of Activity in Digital Networks," *Commun. ACM*, vol. 12, no. 2, Feb. 1969, pp. 102–110.
11. Ulrich, E. G., "Non-Integral Event Timing for Digital Logic Simulation, "*Proc. 14th Design Automation Conf.*, 1976, pp. 61–67.
12. Bowden, K. R., "Design Goals and Implementation Techniques for Time-Based Digital Simulation and Hazard Detection," *Proc. 1982 Int. Test Conf.*, pp. 147–152.
13. Hitchcock, R. B., "Timing Verification and the Timing Analysis Program," *Proc. 19th Design Automation Conf.*, 1982, pp. 594–604.
14. Wold, M. A., "Design Verification and Performance Analysis," *Proc. 15th Design Automation Conf.*, 1978, pp. 264–270.
15. Ng, P., et al., "A Timing Verification System Based on Extracted MOS/VLSI Circuit Parameters," *Proc. 18th Design Automation Conf.*, 1981, pp. 288–292.
16. Levin, H., "Enhanced Simulator Takes on Bus-Structured Logic." Electron. Des., Oct. 29, 1981.
17. Holt, D., and D. Hutchings, "A MOS/LSI Oriented Logic Simulator," *Proc. 18th Design Automation Conf.*, 1981, pp. 280–287.

18. McDermott, R. M., "Transmission Gate Modelling in an Existing Three-Value Simulator," *Proc. 19th Design Automation Conf.*, 1982, pp. 678–681.
19. Roth, J. P., et al., "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits," *IEEE Trans. Comput.*, vol. EC-16, no. 5, Oct. 1967, pp. 567–580.
20. Roth, J. P., *Computer Logic, Testing, and Verification*, chap. 3, Computer Science Press, Rockville, Md., 1980.
21. Armstrong, D. B., "A Deductive Method for Simulating Faults in Logic Circuits," *IEEE Trans. Comput.*, vol. C-21, no. 5, May 1972, pp. 464–471.
22. Schuler, D. M., and R. K. Cleghorn, "An Efficient Method of Fault Simulation for Digital Circuits Modeled from Boolean Gates and Memories," *Proc. 14th Design Automation Conf.*, 1977, pp. 230–238.
23. Snethen, T. J., "Simulator-Oriented Fault Test Generator," *Proc. 14th Design Automation Conf.*, 1977, pp. 88–93.
24. Chang, H. Y., et al., "Comparison of Parallel and Deductive Fault Simulation Methods," *IEEE Trans. Comput.*, vol. 23, no. 11, Nov. 1974, pp. 1132–1138.
25. Ozguner, F., et al., "On Fault Simulation Techniques," *J. Des. Auto. and Fault Tol. Computing*, vol. 3, no. 2, pp. 83–92.
26. Case, G. R., "SALOGS-IV, A Program to Perform Logic Simulation and Fault Diagnosis," *Proc. 15th Design Automation Conf.*, 1978, pp. 392–397.
27. Case, G. R., "A Statistical Method for Test Sequence Generation," *Proc. 12th Design Automation Conf.*, 1975, pp. 257–260.
28. Ulrich, E., "High Speed Concurrent Fault Simulation with Vectors and Scalars," *Proc. 17th Design Automation Conf.*, 1980, pp. 374–380.
29. Moorby, P. R., "Fault Simulation Using Parallel Value Lists," *Proc. ICCAD*, 1983, pp. 101–102.
30. Krohn, H. E., "Vector Coding Techniques for High Speed Digital Simulation," *Proc. 18th Design Automation Conf.*, 1981, pp. 525–529.
31. Hayes, J. P., "A Unified Switching Theory with Applications to VLSI Design," *Proc. IEEE*, vol. 70, Oct. 1982, pp. 1140–1151.