5 Role of Simulation in Testing

5.1 Introduction

Design verification is necessary to check its conformity to specifications. Simulation is the most used means of verification, although a trend is starting toward formal verification. Simulators are essential programs in the CAD toolbox since they are used at different stages of the DFT cycle and at different levels of abstraction ranging from behavioral, RTL, structural, and switch level to circuit and device level. Each level is a finer process of simulation, requiring more information from the designer and more computer time. A *mixed-mode simulator* allows the simulation of different parts of the design at different levels of abstraction. For example, whereas most of the design is simulated on the functional level, a critical part is simulated on the circuit level and switch level [Bryant 1984]. Usually, we distinguish between functional and timing simulation. The first checks the correct operation of the circuit, and the second helps to determine timing violation and critical paths. For present high complexity circuits, a unified design and test platform that facilitates the interaction between high level representation and the circuit level is becoming necessary [Moundanos1998].

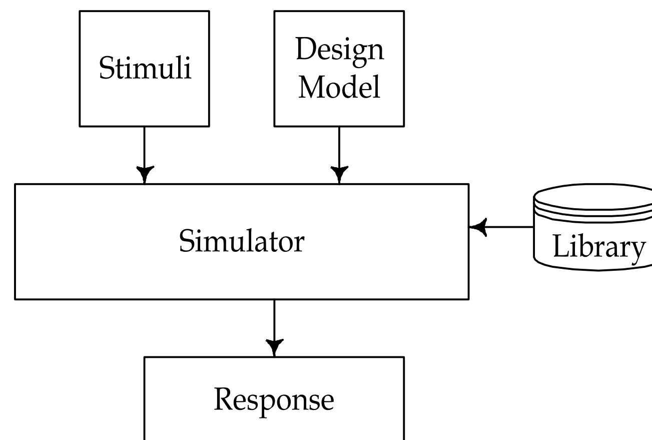

Figure 5.1 The Simulation Process

To perform its task, the simulation program requires the following components, illustrated in Fig. 5.1: (1) design models, (2) a component library, (3) stimuli to be applied to the design, and (4) expected responses. The various types of design descriptions have been presented in Chapter 3. The input stimuli can be applied using one of the following formats: logic values, waveforms, pseudorandom pattern generator, and test benches. Different libraries correspond to the level of design abstraction. At the functional level, each entry in the library is represented by its

Copyright © 2005 Samiha Mourad

logical function without details of the internals of the block. The library for the structural level consists of a set of standard cells described in terms of basic logic functions and their propagation delays. However, no information is available regarding the transistor level. Detailed simulations are then required before fabrication to determine critical paths. For a large circuit, this type of simulation is not practical. A special class of timing simulator is used, the *static timing analysis*. Here the simulator takes as input a netlist connecting transistors and metal and poly paths, and constructs a model of the circuit in terms of resistance, capacitances, and whenever applicable, inductance. Path delays are then calculated, and the largest path, the critical path, is determined without the need of generating stimuli for the circuit. Although the emphasis is on digital testing in this book, it is important to mention that many circuits used at present include analog subcircuits. In such cases an analog simulation is also expected. A mixed signal or analog – digital simulator is more appropriate.

For testing, as opposed to verification, the simulator is used at different stages of the design [Fujiwara 1986, Miczo 1986, Abramovici 1990]. After incorporation of a DFT structure, the circuit is simulated to evaluate the effectiveness of this structure. Also, after generating a test set, say, for stuck-at faults, its fault coverage is calculated using a *fault simulator*. Since the final test to be applied to the chip is usually done on the gate level, fault simulation is also performed on the gate model of the circuit. However, fault injection in circuits described at higher level of abstraction is also possible.

The first topic we address is the effect of design size on the simulation process. Then we discuss logic and timing simulation. In the remainder of the chapter we concentrate on fault simulation.

5.2 Simulation of Large Designs

When designing circuits with millions of gates, one can no longer rely on traditional gate-level simulation to complete the design on time. The only way to contain the verification time is to adopt a new verification that is more compatible with large designs.

- It is important to shift the verification effort to the RTL level instead of the gate level. The main advantage is that the simulation time is a fewfold faster.
- It is more practical to use the appropriate verification technique for different parts of the verification process. Early in the design cycle, it is more suitable to use formal verification to check the result of synthesis transformation from RTL to gate level. However, formal verification is still not as effective as simulation in verifying RTL design functionality against design specifications.

Copyright © 2005 Samiha Mourad

• *Static timing analysis* has replaced simulation for timing verification. This methodology is a more exhaustive verification that does not require the generation of a large set of data (stimuli), but which is much faster.

5.2.1 Test Benches

To simulate a design described in HDL, irrespective of its level of abstraction, a test bench is used to:

(1) apply the stimuli to the design to be simulated and, (2) collect the responses and compare them to expected values. The following test bench invokes a design called *adder* and performs the operations on this design described above. It is written in Verilog HDL.

```
' timescale 1ns/1ns                     // Time unit is 1ns
module adder;                           // Design Test bench
reg PA, PB, PCI;
wire PCO,PSUM;

FA_Behav F1(PA, PB, PCI, PSUM,PCO);    // Instantiate module under test

initial
        begin: ONLY_ONCE
        reg[3:0] Count          ;       // Count hold the count of stimuli
                                        //4 bits are needed to accommodate up to 8
        for (Count=0; Count < 8; PAL =Count +1)
                begin
                {PA, PB, PCI} =Count;   //The stimuli are the values of Count
                #5 $display ("PA, PB, PCI=%b%b%b", PA, PB, PCI,":::PCO,
                            PSUM=%b%b", PCO, PSUM);
                                        //Creation of the response display
                end
        end
endmodule
```

The output of a simulation is requested in binary form (%b) but may be expressed in a hexadecimal number, or, of course, be displayed as waveforms.

5.2.2 Cycle-Based Simulation

As the design passes through various phases in the design cycle, it is important to check that it is still performing the function intended from the onset. Thus it is important to compare the simulation results at these different phases. Early in the cycle different components are designed independently, then they are integrated, and ultimately they form a large design with a large test bench. For simulation on the structural level, it is not possible to consider the entire design; therefore, only parts of the design may be simulated at one time. To compare the results of simulation, the function of the circuit is synchronized to a master clock. That is, the design is evaluated when the clock edge

Copyright © 2005 Samiha Mourad

occurs, and all other timing internal to the circuit is ignored. In this fashion it is the functionality at the different design phases that can easily be compared.

5.3. Logic Simulation

Logic simulation is divided primarily into three types: software-based, hardware accelerators, and hardware emulators. The first type is the most widely used. We concentrate on it in this chapter. Since circuits are growing continually larger, simulation takes more time and there are attempts to use parallel algorithms to speed up the operations of software simulators [Banerjee 1994]. In the second type of simulation, the simulation algorithm is *hardwired* instead of implementing it in a software program. This improves its performance and is particularly useful for a large design with a large volume of stimuli.

Emulation is the third means of verification. It is similar to prototyping. However, while in the past it was possible to prototype by wire-wrapping SSI and MSI components, the present technology builds emulators with powerful field-programmable gate arrays (FPGAs). For example, the Pentium chip was emulated using 3500 Xilinx 3000 FPGA chips. The are two main advantages of hardware emulators. First, it is not possible to verify an entire processor using software simulator because the simulator is not fast enough to handle the enormous amount of stimuli. Second, the emulator allows the verification of the design within the system environment. An emulator speeds up the simulation by two to three orders of magnitude over software simulation.

5.4 Approaches to Simulation

Software simulators may be of the compiled or interpretive types [Breuer 1972, Miczo 1986]. The latter is known mostly as event-driven.

5.4.1 Compiled Simulation

In the compiled type, the circuit netlist is converted into a sequence of machine language instructions, which executes the various logic and arithmetic functions of the circuit. The following procedure for a compiled code simulator is a variation of that given in [Banerjee 1994]:

Procedure Compile-code simulator
    Read the circuit description
    Break feedback loops, if any
    Order the gates into levels
    Generate compiled code
    Read in the initial value of each line
    Read in next input vector and update values
    FOR each new input data DO

Copyright © 2005 Samiha Mourad

```
        FOR each level of logic DO
            For each gate in the level DO
                Execute the compile code for the gate
            END FOR
        END FOR
        IF new value of outputs of feedback lines same as old values
            THEN output results
        ELSE set new input value of inputs of feedback lines same as outputs
    END FOR
```
End procedure

The algorithm implies operating on all gates on one level before proceding on the next level. The levelization of the circuit is illustrated in Fig. 5.2. All primary inputs and feedback lines are at level 0. All gates at a given level have inputs only from lower levels. In other words, no gate can get inputs from a gate at the same or higher level than itself.

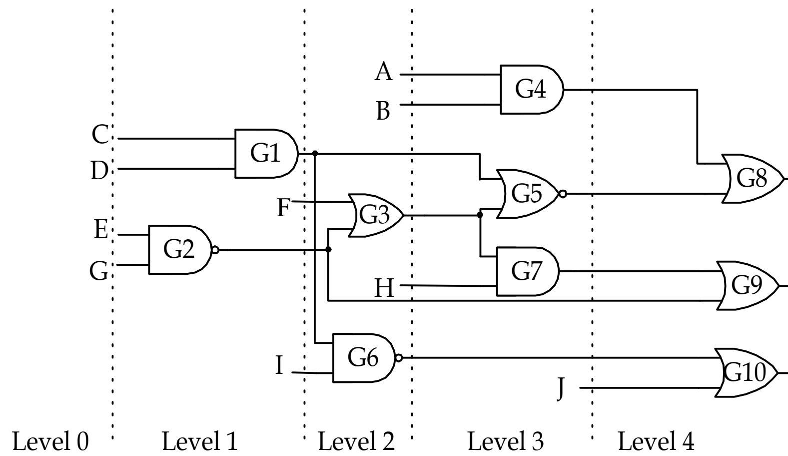

Figure 5.2 Circuit Levelization

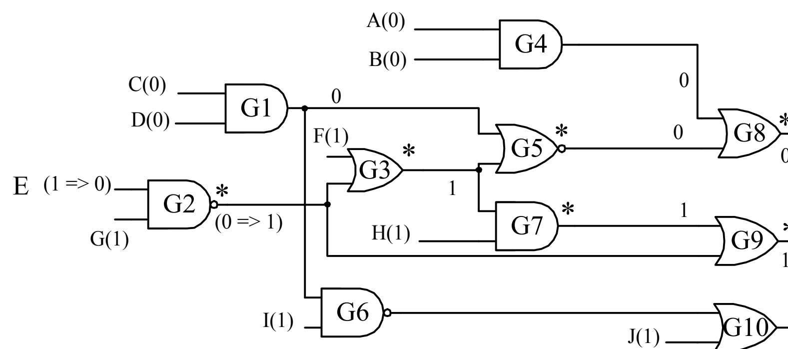

Figure 5.3 Circuit of Fig. 5.2 Illustrating Event-driven Simulation

The primary advantage of compiled code is its fast execution. However, it has a few shortcomings. The circuit needs to be compiled each time a design change is needed. In addition, for each input pattern, activities of all

Copyright © 2005 Samiha Mourad

parts of the circuit are evaluated, although many signals remain unchanged. For example, consider a multi-input AND gate with one of the inputs being 0. No matter what the changes are on the other inputs, the output will remain 0, and hence it is unnecessary to execute any instructions that evaluate the output of this gate. Another problem with this type of simulator is related to the delays, which is discussed in detail in the next section.

Compiled simulators are usually used for zero delay and are useful for combinational circuit and synchronous sequential circuits. However, they cannot handle delay-associated problems such as hazard and races. To alleviate the problems of compiled simulation, event-driven simulators are more effective, particularly for large circuits.

5.4.2 Event-Driven Simulation

In event-driven simulation, gates are evaluated only when there are *events*. An *event* is a change in the value of the signal. A gate is evaluated only if at least one of its inputs changes value. Otherwise, no evaluation is performed. We illustrate the concept using the circuit of Fig. 5.2, annotated as shown in Fig. 5.3. A change of E ($1 \rightarrow 0$) has the potential of affecting all gates indicated by an asterisk (*) in the following order: G2, G3, (G5 and G7), and finally, (G8 and G9). Since the output of G2 changes ($0 \rightarrow 1$), the output of the gate on the next level, G3, has to be evaluated. Given that this output has not changed, there is no need to evaluate G5 or G7 and, consequently, gate G8. The output of G9, though, has to be evaluated since it may be affected by the change on G2. However, being an OR gate with one input already equal to 1, G9 will not be affected by the change on G2. As we proceed from one level to another, a delay δ is assumed before evaluation. Events on the same level are evaluated at the same time interval. To allow appropriate scheduling, a time wheel is used. After the change in G2, G3 is scheduled for processing in time 2δ and G9 in time 4δ. After evaluation of G3, then G5 and G7 are scheduled for time slot 3δ, and so on. Scheduling ahead may exceed the number of the slots in the wheel, and accommodation must be made to keep the schedule correct. The process can be summarized as follows [Banerjee 1994].

```
Procedure Event-driven simulator
    Read circuit description
    Read input vectors
    FOR each input vector to be simulated DO
        Process new inputs
        Update input nodes
        Schedule connected elements on timing wheel
        WHILE elements left for evaluation
                Evaluate element
                IF change on the output
                    THEN update all fan-outs and schedule
```

Copyright © 2005 Samiha Mourad

```
                Connect element on timing wheel
        END WHILE
    END FOR
```
End procedure

Given that only a fraction of the signals in a circuit change value in response to the application of a stimulus, event-driven simulation causes a significant reduction in computation time.

5.5 Timing Models

In addition to the levels of simulation, there are several operational modes that reflect different accuracy of the results. Higher accuracy of the results require longer simulation time. The delay in the circuit can be represented as shown in Fig. 5.4*a*, where every square represents the delay of the gates it follows. In the least accurate simulation, all components are having the same *unit delay*. No actual time is associated with this delay. This is the simplest mode of simulation (Fig. 5.4*b*) because most factors, such as the loading effects of fan-in and fan-out, are ignored.

If the components of the circuit are assigned different delay values, the simulation results are more realistic, as illustrated in Fig. 5.4*c*. Notice that before the propagation of the input values through the different gates in the circuit, the outputs of these gates were undetermined. This is indicated in the figure by x, which means *undetermined* or *unknown* value. For this simulation we extend the logical values of the nodes from {0,1} to {0,1,x} [Eichelberger 1965]. The accuracy can be improved further by assigning to each component multiple delays: rise and fall times. The effect of these times is shown in Fig. 5.4*d*.

Copyright © 2005 Samiha Mourad

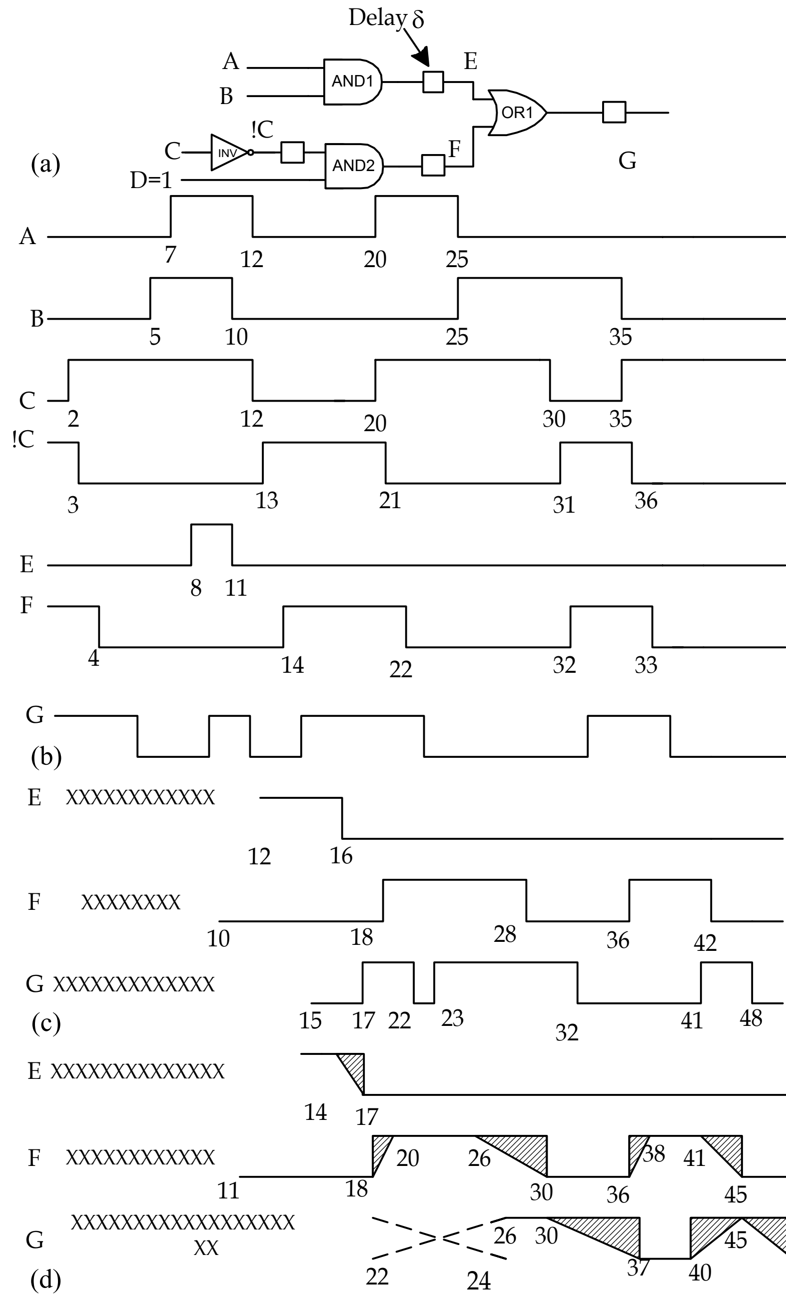

Figure 5.4 Timing Simulation a) Simulated Circuit, b) Unit-delay Simulation, c) Varied delay Simulation, d) Multi-valued Simulation

Actual timing simulation is performed taking into account all parasitic capacitances, in addition to those of the transistors. The parasitic data can either be estimated or extracted after layout, as mentioned in Chapter 4. In this fashion one observes not only the logic level of the signal but also its shape. Because the rise and fall times are not unique, they are in a range of values from t_{min} to t_{max}. Therefore, the slew rate of the signals can vary and cause these signals to remain for an appreciable amount of time in a logic range that is rising (*u*) or falling (*d*). It is possible that

Copyright © 2005 Samiha Mourad

the end result will be an erroneous signal that is undetermined (*e*). Thus for accurate simulation we extend the logic values to {0, 1, *x, u, d, e*}.

The high capacitive load of MOS technology and its bidirectional signal flow have imposed special requirements on simulators. This has led to the concept of *strength of the signal*, which may be forcing, nonforcing, or of high impedance. Forcing strength is the status of a node that is pulled up or down at a fast rate. A nonforcing strength is the status of a node that is connected to power or ground through a resistor that is allowing any charge to be added or removed at a finite rate. When signals of different strengths collide, the strongest signal always overcomes the weaker. Signals *A* and *B* may flow in either direction through the transmission gate (Fig. 5.5) whenever *S* is high. Thus either inverter 1 or 2 drives the output. The strength of the signal driving any of these inverters depends on the strengths of the output signals of 1 and 2. If the strength of G1 is greater than G2, the signal flows from *A* to *B* and vice versa. Using logic levels and strength, a nine-value logic system has been adopted by the IEEE as Standard 1164-1933. This standard may be invoked in HDL simulators.

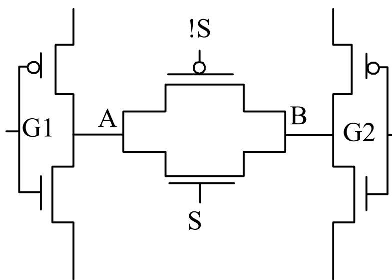

Figure 5.5 Driving Strength in MOS Technology

5.5.1 Static Timing Analysis

In static timing analysis (STA), the concern is with determining the critical path delay. The delays are calculated using the data sheets of the various circuit components under worst-case analysis conditions. The critical path delays reported by the STA are not usually the same as those provided by functional simulation. Models used in the function simulator do not generally include the loading capacitances and hence they do not yield as accurate results as those obtained by STA. Most STA programs do not consider the logic function as the circuit is analyzed, and the delay calculation may yield false *paths*. For example, consider a circuit that implements a function $f(a, b, s)$ that under some values of a and b is represented by $f = s.s'$. The path from s to f is impossible to activate, nevertheless, an STA will produce a value for it.

Copyright © 2005 Samiha Mourad

5.5.2 Mixed-Level Simulation

Mixed-level simulation mode is a useful feature since it is possible to exercise real timing simulation on the most critical parts and multidelay on other parts.

5.6 Fault Simulation

Fault simulation is performed during the design cycle to achieve the following goals:

- Testing specific faulty conditions
- Guiding the test pattern generator program
- Measuring the effectiveness of the test patterns
- Generating fault dictionaries

To perform its task, the fault simulation program requires, in addition to the circuit model, the stimuli, and the responses of a good circuit to the stimuli, a fault model and a fault list. This is illustrated in Fig. 5.6. As we learned in Chapter 2, there are different fault models, and the most widely used is the stuck-at model. Test patterns generated for this model have proven to be useful for other types of models, such as multiple stuck-at, bridging, and delay faults. The responses deduced by the fault simulator are used to determine the fault coverage.

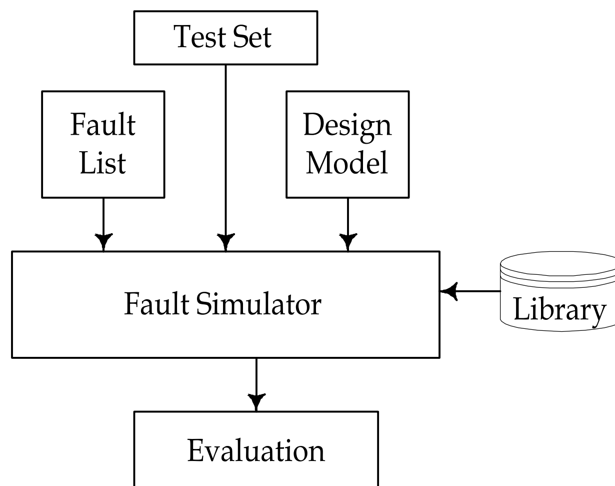

Figure 5.6 Elements of Fault Simulation

The fault simulation process is illustrated in Fig. 5.6. A fault is considered from the list and a pattern is applied to the circuit. If the fault is detected, it is dropped from the fault list and the next fault is considered. Otherwise, another pattern is applied, and if the fault is not detected when all patterns are applied, the

Copyright © 2005 Samiha Mourad

fault is then considered undetectable by the test and is removed from the fault list. The process is continued until the fault list is empty.

As for logic simulators, fault simulators are of the compiled type or the event-driven type. The compiled code is executed for each fault for as many times as there are test patterns or only as many times as it takes to detect the fault in the case where *fault dropping* is allowed. The execution time has an upper bound of $N_f N_p$, where N_f is the number of faults in the fault list and N_p is the number of test patterns. It is possible to reduce this execution time. To represent the faulty value, the simulation program uses one computer word. However, the logic value on any of the nets is either 0 or 1; it is thus sufficient to store this value in only one bit of the computer word. In this fashion, the different bits can represent several faults. For example, a 16-bit word can be used to represent 15 faults, leaving one bit for the fault-free value of the circuit. Simulators using this scheme are called *parallel fault simulators*.

There are three main approaches to event-driven simulators: (1) parallel single fault, 2) deductive fault simulation, and 3) concurrent fault simulation. We illustrate each one with an example. Any of these approaches starts with a fault list and generates the good circuit response to the stimuli. We call this the *fault-free copy* or *image* of the circuit. It is the manner in which they handle faults from the fault list that differentiates among these approaches.

5.6.1 Parallel Fault Simulation

In parallel fault simulation, only one copy of a good circuit is calculated for a given test pattern [Thomas 1975, Iyengar 1988]. By one copy we mean the collection of values of all signals in the circuit resulting from application of a test pattern. If the computer word size is N, then N - 1 copies of faulty circuit are also generated. For a total M faults in the circuit, $\lceil M/(N-1) \rceil$, simulation runs are then necessary. The symbol $\lceil K \rceil$ is equal to K if K is an integer, or it is equal to the smallest integer greater than K. If the circuit has P primary outputs, the number of runs has a lower limit given by $\lceil M/[(N-1)P] \rceil$.

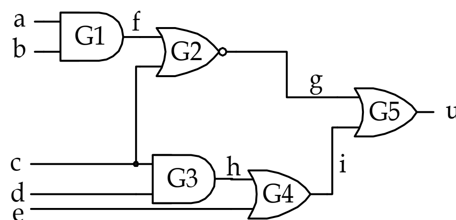

Figure 5.7 Circuit Used to Illustrate Fault Simulation

Copyright © 2005 Samiha Mourad

Next we illustrate the detection of faults taken ($N - 1$) at a time for the example shown in Fig. 5.7. First, a fault list is generated for the circuit. The circuit has 10 lines, excluding fan-outs. This results in 20 stuck-at faults. Notice, however, that this number can be reduced by collapsing the fault list. Second, we decide about the test pattern to be simulated. In this example we used the pattern *abcde* = 10010. We assume arbitrarily that $N = 16$ and we calculate the fault-free copy and up to 15 faulty copies of the circuits. All this information is summarized in Table 5.1. The faults are listed in the first row of the table.

Table 5.1 Parallel Fault simulationTable 5.1 Parallel Fault simulation

	ff	*a*/0	*b*/1	*c*/1	*d*/0	*e*/1	*f*/0	*f*/1	*g*/0	*g*/1	*h*/0	*h*/1	*i*/0	*i*/1	*u*/0	*u*/1
a=1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
b=0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
c=0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
d=1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
f=ab	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
g=(f+c)'	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	1
h=cd	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
e=0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
i=e+h	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0
u=g+i	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	1

The first column of this table gives the signal on each line in the circuit, including the primary inputs, *a*, *b*, *c*, *d,* and *e*, and the primary output, *u*. For the other lines, the signals are expressed in terms of the logical operation performed by the corresponding gate. The second column lists the fault-free (ff) circuit response at each of the lines represented in the first column. Each of the other 15 columns represents the responses under the fault indicated in the first row. Notice that for the primary inputs, the faults selected are limited by the test pattern. That is, we could have, for example, listed *a*/0 and *a*/1; however, the pattern requires that *a* = 1. Thus it is not necessary to check whether or not the fault *a*/1 is detected. If it is to detect a fault on *a*, this pattern would be *a*/0. The third column represents the copy of the circuit under the fault *a*/0. The next column is the copy for *b*/1, and so on for the rest of the columns. To facilitate examination of the table, the faulty response is shown in boldface type whenever it is different from the corresponding *ff* response at any line.

We recognize whether or not the fault is detected by comparing the fault-free value of the primary output, *u*, to the corresponding values in the column representing the copies of the faulty circuit. We then recognize that only the following faults are detected by the test pattern applied: *b*/1, *f*/1, *g*/0, *u*/0. A quick investigation of this set of faults indicates that *f*/1 and *g*/0 are equivalent; *f*/1 dominates *b*/1, and *u*/0 dominates *g*/0. Thus we could have represented these faults by one of them, *b*/1. Detection of this fault implies detection of *f*/1 and its equivalent *g*/0, as

Copyright © 2005 Samiha Mourad

well as *u*/0, which dominates *g*/0. This equivalence of dominance, was discussed in Sections 2.6.1 and 2.6.2. We could actually have reduced the fault list by applying equivalence and dominance relationships before simulating.

5.6.2 Deductive Simulation

Deductive fault simulation requires one image of the circuit, the fault-free response to the test pattern [Armstrong 1972]. Fault detection is deduced from the current image. All detectable faults are deduced in only one pass. Such a pass takes longer than a pass in parallel simulation. However, there are fewer passes than parallel simulation. It has been proven that for small circuits, fewer than 500 gates, parallel simulation is more efficient than deductive simulation [Chang 1974]. These experimental results were dependent, of course, on using much slower computers having 4 bits per word. Also most present circuits are much larger than 500 gates, deductive simulation is more realistic.

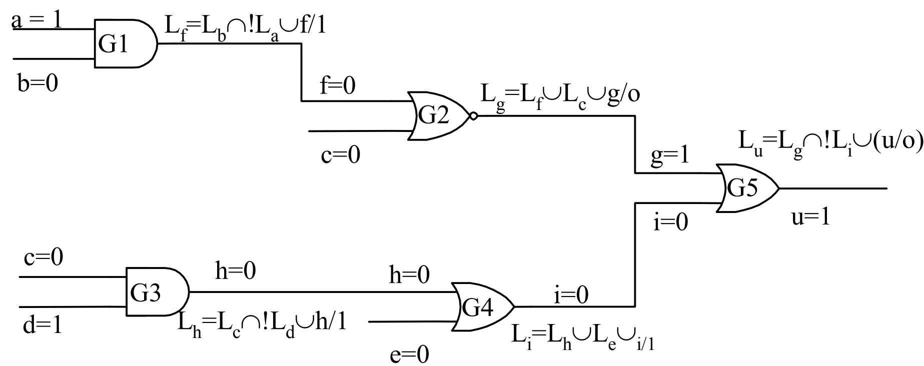

Figure 5.8 Deductive Simulation
a)Application to the Circuit in Fig. 5.7, b) Operation on Elementary Logic Gates

This type of fault simulator creates the fault-free image for the test pattern. Then a list of faults is associated with every line in the circuit. These faults are sensitizable to the output through this line. The faults are propagated from the primary inputs of the circuit to its primary outputs, one level at a time. We illustrate the process using the example in Fig. 5.7 and the same pattern as that used in the preceding section. Propagation of the faults is illustrated on the circuit shown in Fig. 5.8a. The results are also displayed, level by level, in Table 5.2. The list of *a* and *b* is passed to *f*, then from *f* and *c* to *g*, and so on. The fault list at the output of a gate is deduced from the values of the fault-free image and the type of gate. Let us refer to the lists associated with lines *a* and *b* as L_a and L_b. Each list includes only one fault, *a*/0 and *b*/1, respectively. Since $a = 1$ and $b = 0$, only L_b is sensitizable to *f*. The faults on *a* are masked by the 0 value on *b*. Thus $L_f = [L_b \cap !\ L_a] \cup \{f/1\}$, where ! indicates the complement, $\cap$ the intersection operation on sets, and $\cup$ the union operation. Also recall that $[L_b \cap !\ L_a] = L_b - L_b \cap L_a$. In other words,

Copyright © 2005 Samiha Mourad

they are the faults in L_b that are not in L_a. Therefore, $L_f = \{b/1, f/1\}$. Had $b = 1$, L_a could have been sensitized to *f*; that is, $L_f = L_a \cup L_b \cup \{f/1\}$. The operation on the sets will thus depend on the logic gate and the values on its inputs. They are all presented in Fig. 5.8*b*.

Table 5.2. Deductive Simulation Level by Level.

		Fault Lists					
		LEVEL 0	LEVEL 1	LEVEL 2	LEVEL3	LEVEL4	LEVEL 5
Node	FF	PRIMARY INPUTS					PRIMARY OUTPUT
a	**1**	*a*/0					
b	**0**	*b*/1					
c	**0**	*c*/1					
d	**1**	*d*/0					
e	**0**	*e*/1					
f	0	---------------------	$Lb \cap !La \cup f/1$	*b*/1, *f*/1			
g	1	---------------------	---------------------	$Lf \cup Lc \cup g/0$	*b*/1, *f*1, *c*/1, *g*/0		
h	0	---------------------	---------------------	$Lc \cap !Ld \cup h/1$	*c*/1,*h*/1		
i	0	---------------------	---------------------	---------------------	$Lh \cup Le \cup I/1$	*c*/1,*e*/1, *I*/1	
u	1	---------------------	---------------------	---------------------	---------------------	$Lg \cap !Li \cup u/0$	*b*/1,*f*1, c/1, *g*/0,*u*/0

$La \cap !Lb = La - La \cap Lb$

This can be written as all elements of La minus the elements in the intersection of *La* and *Lb*

So, $Lu = Lg - Lg \cap Li \cup u/0 = (b/1, f/1, c/1, g/0) - (c/1) + u/0 = b/1, f/1, g/0, u/0$ as indicated in the last column

We know from the fault-free image that since $f = 0 = c$, the fault list on *g* will be $L_g = L_f \cup L_c \cup \{g/1\} = \{b/1, f/1, c/1, g/0\}$. In a similar fashion, we can find the fault list at input *i* of G5. It is $L_i = \{c/1, h/1, e/1, i/1\}$. Since the output gate, G5, is an OR gate and its inputs *g* and *i* are 1 and 0, faults detected at the output, *u*, are given by $L_g \cap ! L_i \cup \{u/1\} = [L_g - L_g \cap L_i] \cup \{u/0\} = \{b/1, f/1, g/0, u/0\}$.

5.6.3 Concurrent Fault Simulation

Concurrent fault simulation combines features of parallel and deductive simulations [Ulrich 1974, Abramovici 1977, Rogers 1987]. Faulty gates are simulated when their output differs from that of the fault-free circuit. It is faster than other algorithms and uses dynamic storage allocations.We illustrate this simulation using the circuit in Fig. 5.7 and the same pattern as that used to illustrate the other types of simulation. The results of the various operations are shown in Table 5.3. The first row of the table includes all gates and their respective inputs. The second row shows the values of the test applied on the primary inputs, *a*, *b*, *c*, *d*, and *e*, and the fault-free values on the outputs of the five gates. In subsequent rows, the values of the faulty signals are given. From the values on the inputs and outputs of the gate, the possible detectable faults are deduced. For example, for gate G1, the values on *a*, *b,* and *f* are 1, 0, and 0. The faults that are potentially detectable are *a*/0, *b*/1, and *f*/1. The behavior of this gate for these faults is shown in the fourth through sixth rows. If any of these faults is to be detected, it has to be observable

Copyright © 2005 Samiha Mourad

on the output of the gate, *f*. That is, the faulty values of *f* have to be different from the faultfree values shown in the second row. Thus only two faults, *b*/1 and *f* /1, have the potential to be detectable and are considered in the next steps of the simulation. Fault *a*/0 is dropped immediately.

Table 5.3 Concurrent Fault Simulation for Circuit in Fig. 5.7.

G1	***a***	***B***	***f***	**G2**	***c***	***f***	***g***	***G3***	***c***	***d***	***h***	***G4***	***e***	***h***	***i***	***G5***	***g***	***I***	***u***
	1	0	0		0	0	1		0	1	0		0	0	0		1	0	1
a/0	0	0	0	b/1	0	1	**0**	c/1	1	1	**1**	c/1	0	1	**1**	b/1	0	0	**0**
b/1	1	1	**1**	c1/1	1	0	**0**	d/0	0	0	0	e/1	1	0	**1**	c/1	0	1	1
f /1	1	0	**1**	*f* /1	0	1	**0**	h/1	0	1	**1**	h/1	0	1	**1**	e/1	1	1	1
				g/0	0	0	**0**					i/1	0	0	**1**	*f* /1	0	0	**0**
																g/0	0	0	**0**
																h/1	1	1	1
																i/1	1	1	1
																u/1	1	0	**0**

Gate G2 is a NOR gate with inputs *c* and *f* (the output of G1) and output *g*. In addition to the two faults carried out from the previous gate, *b*/1 and *f* /1, the possible detectable faults are *c*/1 and *g*/0. Since for all faults the output of the faulty gate is different from the fault-free value (row 2, column 8), then all faults on gate G2 are propagated to the next gate, G5. But before exploring these faults any further, we need to examine the other gates, G3 and G4*,* since signals from these gates are required to propagate the fault to the primary output, *u*.

Gate G3 is an AND gate with inputs *c* = 0 and *d* = 1 and thus *h* = 0. The potentially detectable faults are deduced as *c*/1, *d*/0, and *h*/1 (column 12). Only the first and third faults have a chance to be detectable eventually and they are carried on in investigating the next gate, G4. The fault *d*/0 is dropped for the same reason that we dropped *a*/0 in the earlier pass. The results of the similar operations on gate G4 propagate the faults c/1, *e*/1, *h*/1, and *i*/1 to gate G5. Next we combine the faults sensitized at *g*, *b*/1, *c1*/1, *f* /1, and *g*/0, and those sensitized on line *i*, *c*/ 1, *e*/1, *h*/1, and *i*/1. The two sets of faults are detected if the faults on the inputs of G5 are detected. The faulty responses at *u* for all of these faults are given at the rightmost column of the table. Comparison of these values with the fault-free value of 1 implies that the only faults detected by the pattern are *b*/1, *f* /1, *g*/0, and *u*/0. As expected, these are the same faults detected using parallel fault simulation with the same test pattern (10010). The results can be summarized as follows.

1. The test pattern used is *abcde* = 10010.
2. Output of G1: Fault *a*/0 is dropped.

Copyright © 2005 Samiha Mourad

3. Output of G2: All faults are possibly detected.

4. Output of G3: Fault d/0 is dropped.

5. Output of G4: All faults are possibly detected.

6 Output of G5: Faults *i*/1, *e*/1, *h*/1 and *c*/1 are dropped and the remaining faults are detected: *b*/1, *f*/1, *g*/0, and *u*/0.

5.7 Fault Simulation Results

After fault simulation, the faults are either *detected* or *undetected*. The fault is detected if it has been controllable and observable by one of the patterns in the test set. In such a case, at least one of the primary outputs of the faulty circuit is different from the good circuit. Otherwise, it is not detected by any of the patterns of the test set. However, the output of a fault simulator separates faults into more than these two categories. Stuck-at faults on redundant logic are not testable and are referred to as *untestable*. That is, there are no patterns that can detect these faults. Other unstable faults occur on lines in the circuit that are *tied*, pulled up to V_{dd}, or pulled down to GND. In the circuit of Fig. 5.9, node 11 is held low, and thus SA0 on this line cannot be detected. Such tied lines may also mask or block other faults. In this case, faults on input H and node 10 are masked as well as the SA0 on node 12, the output of this AND gate. A fault on a line that is unused is not testable. This is the case of node 14 in Fig. 5.9. As a result, faults on line *B* are also undetectable because they are not observable. The other categories vary from one vendor to another. If the PO of the good circuit is either 1 or 0 but the corresponding PO of the fault circuit is x, we have a *possibly detected fault*, which is sometimes called a *potentially detected fault*. In some situations the injected fault causes the circuit to oscillate, and we have an *oscillatory fault*. The oscillation probably occurs because of the feedback in a combinational circuit with zero-delay models. An *oscillatory fault* that affects a large part of a circuit is called a *hyperactive fault*. Such faults cause a serious problem to the fault simulator since they may make simulation time unnecessarily long.

Copyright © 2005 Samiha Mourad

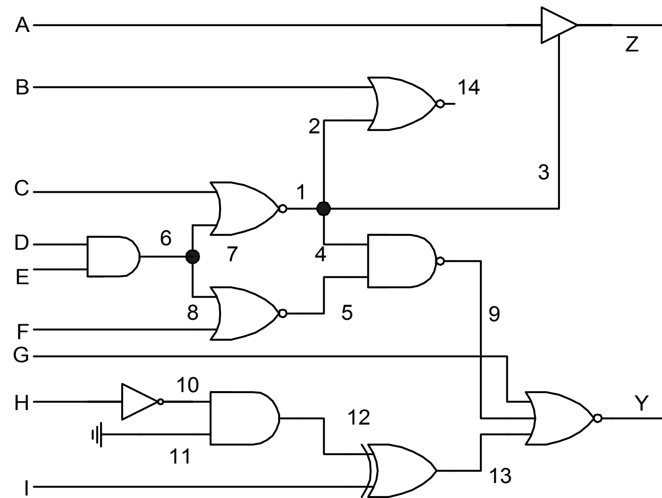

Figure 5.9 Circuit to Illustrate Undetectable Faults

Usually, faults are dropped from the fault list as soon as they are definitely detected. However, sometimes a fault simulator includes the option of dropping it after it has been detected by *n* test patterns, where *n* is usually selected by the user of the fault simulator. This terminology is not standard and every simulator vendor utilizes its own terminology and attributes its own interpretation to this terminology [Mourad 1993].

5.7.1 Fault Coverage

The effectiveness of a test set is quantifiable. It is the percentage of the faults detected by a test and is known as the fault coverage, defined as

fault coverage = faults detected / total number of faults

A more realistic expression is

fault coverage = faults detected / detectable faults, where detectable faults = all faults – untestable faults. It is also possible to report the results in terms of fault classes. That is, after collapsing faults, the entries in the fault list represent fault classes rather one single fault. The calculations of fault coverage based on individual faults, T_f, versus fault classes, T_c, may yield disparate results. Consider a circuit with 36 faults that are collapsible into 20 classes where a class may include 1 to 4 faults. Assume that a fault class is not detected. Then T_c = 19/20 = 95%. The corresponding values of T_f are 35/36 = 97% and 32/36 = 89% [Mourad 1993].

As we mentioned in Section 2.2, a single fault may represent many physical detects, but SAFs do not model all types of defects. Therefore, even if a fault coverage of 100% is reachable, this does not guarantee that the circuit is *defect-free*. The relationship between fault coverage and defect level is shown in Fig. 1.14. The defect level is

Copyright © 2005 Samiha Mourad

usually measured in defects per million (DPM). Thus a defect level of 0.1% is equivalent to 1000 DPM. Inspection of the figure indicates that high fault coverage is required to ensure low defect levels, 1 to 100 DPM. To reach such a level, a SAF coverage of 99.9% is required. To improve defect coverage, current testing is used to complement SAF voltage discussed in Chapter 7.

5.7.2 Fault Dictionary

In addition to the fault coverage, a fault simulator may provide a *fault dictionary*. The fault dictionary associates with each fault a set of test patterns that uniquely identifies the fault. For example, consider the information in Table 5.4, where an entry of $a_{jk} = 1$ indicates that fault k is detected by pattern j. Now, if we know that a circuit passes all test patterns except the fourth pattern, we can deduce that the failure was due to fault *f*8. If, however, the circuit fails both test patterns 3 and 4, the cause is either *f*5, f7, or *f*8. To distinguish the two faults, more information is needed, such as the output signal associated with each fault and pattern. This information is used at test application time and is helpful in identifying the defect that caused the fault. This is known as *fault diagnosis* and is very important in enhancing the process yield. The effectiveness of the diagnosis depends on that of the fault model in representing defects. Defects that are not modeled by the stuck-at fault cannot then be identified. Current testing help improve the diagnosis.

Table 5.4 Fault Dictionary

	Fault							
Pattern	1	2	3	4	5	6	7	8
1	1		1			1		
2		1	1	1				
3			1		1		1	
4			1				1	1

5.8 References

Abramovici, M., M. Breuer, and A. D. Friedman (1990), Digital Systems Testing and testable design, *IEEE Press,* Piscataway, NJ.

Abramovici, M. et al., M.A. Breuer, and K. Kumar (1977), Concurrent fault simulation and functional level modeling, *Proc. 14th Design Automation Conf*erence, pp. 128 - 137.

Armstrong, D.B. (1972), A deductive method for simulating logic faults in logic circuits, *IEEE Trans*. *Comput*., Vol. C-21 No.5, pp. 464 – 471.

Banerjee, P. (1994), Parallel Algorithms for VLSI Computer-Aided Design, Prentice Hall, Upper Saddle River, NJ.

Copyright © 2005 Samiha Mourad

Brayton R.K., et al. (1984), *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic, Norwell, MA.

Breuer M. A. (Ed.), (1972), Design Automation of Digital Systems, Upper Saddle River, NJ.

Bryant R.E., (1984), A switch-level model and simulator for MOS digital systems, *IEEE Trans. Comput.,* Vol. C-33, No. 2, pp. 160 – 177.

Chang H.Y., et al. (1974), Comparison of parallel and deductive fault simulation methods, *IEEE Trans. Comput.,* Vol. C-23, No.11, pp. 193 – 200.

Eichelberger, E. (1965), Hazard detection in combinational and sequential circuits," *IBM J. Res. Dev*., Vol. 9, No. 1, pp. 90 – 99.

Fujiwara, H. (1986), *Logic Testing and Design for Testability*, MIT Press, Cambridge, MA.

Iyenger V. S. and D. T. Tang (1988), On Simulating Faults in Parallel, *Digest of Papers 18th International Symposium on Fault-Tolerant Computing*, pp. 110 – 115.

Lee C.Y. (1959), Binary Decision Diagrams, *Bell System Technical Journal 38*, No.4 (July 1959), pp. 985-999.

Lee C.Y., (1961), "An algorithm for path connections and its applications," *IRE Trans. on Electronic Computers*, Vol. EC-10, No. 3, pp. 346-365.

Miczo, A., (1986), Digital Logic Testing and Simulation, Wiley & Sons, NY, 1986.

Moundanos, D., J.A. Abraham, and Y.V. Hoskote, (1998), "Abstraction Techniques for Validation Coverage Analysis and Test Generation," *IEEE Trans. on Computers*, Vol. 47 No. 1, pp. 2-14.

Petra Michel, Ulrich Lauther and Peter Duzy, Eds., (1992), "The Synthesis approach to digital system design," Kluwer Academic, Norwell, MA.

Mourad, S. (1993), Computer-aided testing systems: evaluation and benchmark circuits, *VLSI Design*, Vol.1, No. 1, pp. 87 – 97.

Rogers W. A, J. F. Guzolek and J. Abraham. (1987), Concurrent Hierarchical Fault Simulation, *IEEE Trans. Computer-Aided Design*, Vol. CAD-6, No. 9, pp. 848 – 862.

Sahni, S. and A. Bhatt. (1980) The complexity of design automation problems, *Proc Design Automation Conference,* pp 402 – 410.

Thomas, E.W. and S. A. Szygenda, (1975), Digital logic simulation in a time-based, table-driven environment, Part2, Parallel fault simulation, *Computer*, Vol. 8, No. 3, pp. 38 – 49.

Ulrich, E. G. and T. Baker (1974), "The concurrent simulation f activity in digital networks," Computer Magazine, Vol. 7, No. 1,pp. 39 – 44.

Problems

5.1. Design a transistor level for the logic gate in Fig. 5.4*a* and estimate the delays at the output. Check your results using a SPICE-like simulator.

5.2. Determine the stuck-at faults in the circuit of Fig. 5.2 that can be detected by the pattern 1110011101. Use the parallel fault simulation algorithm.

Copyright © 2005 Samiha Mourad

5.3. Repeat Problem 5.2 using deductive fault simulation.

5.4. Which, if any, of the faults detected in Problem 5.2 are equivalent?

5.5. For the circuit of Fig. P5.,1 use concurrent fault simulation to determine the faults detected by an exhaustive test set. Plot the detectability profile of the circuit (See Problem 2.2 in for a definition of the detectability profile.)

5.6. With reference to Table 5.4, determine, if possible, the most likely fault causing the failure if (a) the circuit passes the entire test except for pattern 1; (b) the circuit does not pass any of the patterns in the test.

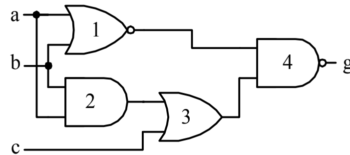

Figure P5.

Copyright © 2005 Samiha Mourad