

1. Overview of Testing

1.1 Reliability and Testing

The reliability of electronic systems is no longer a concern limited to the military, aerospace, and banking industries, where failure consequences may have catastrophic consequences. Reliability and testing techniques have become of increasing interest to all other applications, such as computers, telecommunications, consumer products, and automotive industry because of the following factors: (1) At present, electronic systems are becoming ubiquitous in the workplace and they are being used in harsher environments; (2), Because of the proliferation of their use, users of electronic systems are not necessarily experienced people and may misuse the machines inadvertently; and (3), The continuous decrease in technology feature size, accompanied by an increase in systems size and speed, has resulted in newer failure modes.

A key requirement for obtaining reliable electronic systems is the ability to determine that the systems are error-free [Breuer 1976]. Electronic systems consist of hardware and software. In this book we investigate only hardware testing. The majority of the hardware used today consists of digital circuits, and this book concentrates on *digital testing*. A circuit must be tested to guarantee that it is working and continues to work according to specifications. Such testing detects failures due to manufacturing defects. It can also detect many field failures due to aging, environmental changes, power supply fluctuations, and so on. Test pattern generation is a complex problem. Often, simulation patterns developed for design verification are augmented with patterns that are generated manually or by an automatic test pattern generator (ATPG) to obtain a complete test set, capable of detecting all faults in the circuit. This test also verifies the functionality of its logic [Abadir 1989]. The patterns are then applied to the circuit using automatic test equipment (ATE). Because of the complexity of testing processes, a design approach aimed at making digital circuits more easily testable has been formulated. This approach to design, is known as *Design for Test* (DFT) or *Design for Testability* is discussed in subsequent chapters. Its aim is to make these circuits more controllable and observable by embedding test constructs into the design.

Another aspect of reliability is the system's ability to run dependently on demand. This requires that the system be fault tolerant. Fault tolerance is a vast and rich field that involves digital testing, but it is not possible in

this book to give it the justice it deserves. To explore this topic further, it is recommended that you consult such references as [Siewiorick 1982] and [Johnson 1989].

In this chapter we first give a brief description of the digital design process to better understand the role and scope of verification and testing and the relationship between the two processes. In balance of the chapter we give a synopsis of what is discussed in the remainder of the book.

1.2 Design Process

As a result of the continuous decrease in the minimum feature size of transistors, both device density and design complexity have steadily increased. Densities of hundreds of million transistors are now a reality, and to manage such complexity, it is only natural to design on a hierarchical basis. In addition to their complexity, digital systems have a life cycle that sometimes becomes shorter than their design cycle. To stay competitive in the electronics field, vendors need to increase designers' efficiency and to reduce time-to-market.

The design process has been changing continually to reflect the status of the technology and of the Computer Aided Design (CAD) tools available to designers. We can view the process as divided into three main phases: system design, logic design, and physical design. The time taken to accomplish any of the three phases has changed due primarily to the availability of automation. This is illustrated in Fig. 1.1. Each of the design phases comprises a means to enter the design and verify it, then transform it to the next phase. Usually, simulation is used for verification, although more recently, formal verification has been gaining in importance.

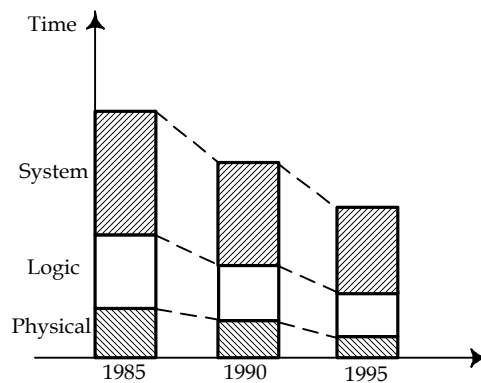


Figure 1.1 Design Phases: Design Time for Each of the Phases

The shift toward submicron technologies has allowed Integrated-circuit (IC) designers to increase the complexity of their designs to the extent that an entire system can now be implemented on a chip. This new paradigm of *system on a chip* (SOC) has changed the approach to design and testing. To increase the design

productivity, and hence to decrease time-to-market, the reuse of previously designed modules is becoming common practice in SOC design. This is known as *core-based design*. The reuse approach is not limited to in-house designs, but is extended to modules that have been designed by others. Such modules are referred to as *embedded cores*.

An example of an SOC is shown in Fig. 1.2. It consists of different cores (rectangular blocks) and User Defined Logic (UDL). The cores may be processors, DSP, RAMs, etc. The UDL components are "gluing" the various cores for the intended system. The cores may have been designed previously. They may also be described in different modes, from specifications to hardware description to layout. Because of the complexity of designing an SOC, it is likely that the system design phase will be longer than the other design phases as illustrated in Fig. 1.1. We defer the discussion of the design and testing of SOC's to Chapter 15. In the rest of the section, we discuss how a core or a UDL component is designed from specifications.

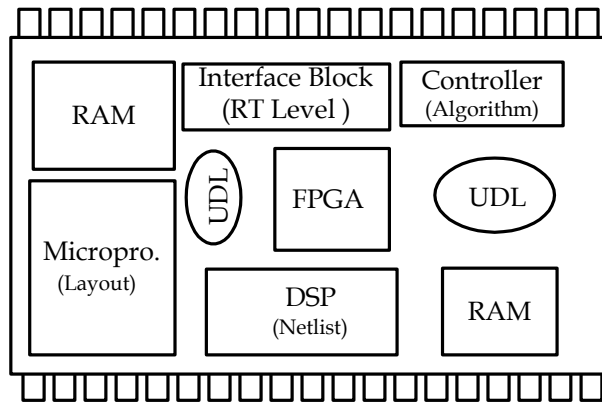


Figure 1.2 System on a Chip (SOC).

Whereas in the past hardware design was primarily done on paper, now it is described in hardware description language (HDL). Expressing the design in HDL has several advantages. The most important advantages are management of complexity and shortening the design cycle. At present, the design may be entered on a behavioral level or immediately in Register Transfer Level (RTL), then mapped into logic level as illustrated in Fig. 1.3. Finally, the design is transformed into layout masks. These automated transformations are known as *design synthesis*. Both synthesis and simulation tools utilize libraries of components. For example, consider the logic description of a design. Assume that it consists of basic logic gates such as NAND, NOR, and so on. These components form the library on the logic level. Every logic gate has a layout view that is process dependent. The layouts of the various gates constitute the physical library. Designers have the option of using a standard cell

library available from vendors. They may augment these libraries or even develop their own libraries. The transformation from the gate level to the physical level is known as *technology mapping*.

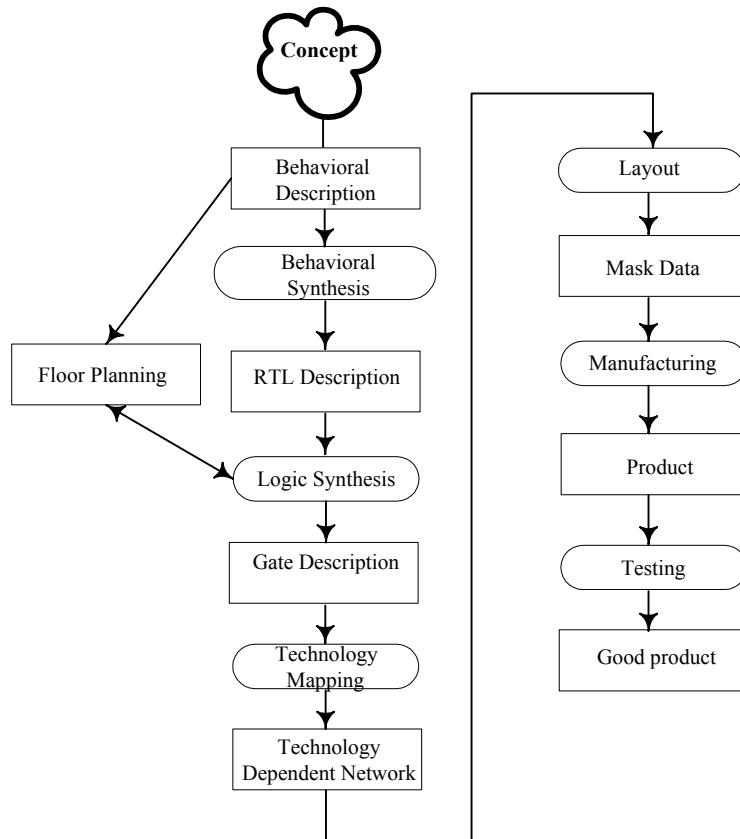


Figure 1.3 The Design Cycle

In stepping from one level to another, simulation, is repeated at each level, until the design is satisfactory. Upon completion of the layout, it is possible to extract parameters such as the load resistance and capacitance. With this information, it is thus possible to verify timing of the circuit. Parameter extraction is becoming significantly more important in present deep submicron technology.

The artwork, the layout masks, are then used to fabricate the design in the form of a die on a wafer. After fabrication, the wafer is tested and defective dies are masked. This process is known as *wafer sort*. The good dies are then packaged and tested again using ATE equipment.

Because of the close relationship of the design process to testing, two chapters in this book are devoted to this topic and to design representation.

1.3 Verification

Testing a circuit prior to its implementation is known as *design verification*. No engineer would take a design to foundry prior to verifying it. The question is not *whether one should verify* but *how well to verify* in order to have confidence that the device will comply with its specifications. During its design cycle, a circuit has several descriptions as illustrated in Fig. 1.3. The four representations of the design - behavioral, RTL, gate-level, and layout - are different perspectives of the same circuit. In mapping the design from one phase to another, it is likely that some errors are produced. Errors in mapping may be due to the CAD tools or to human mishandling of the tools. At each stage, the design is verified to assert that it is the same design from the previous phase as well as that it adheres to the specifications. At present, simulation is the most popular tool for hardware verification. Although to date, extensive research has been done in formal verification to date, there are few practical tools to prove correctness of design based on the formal verification. Exhaustive simulation is equivalent to formal verification. However such a simulation is not feasible for present complex circuits. Two types of simulations are used to verify the design: *functional simulation* and *timing simulation*.

1.3.1 Functional Simulation

In functional simulation, the designer may verify the correctness of the design but not necessarily at operational speed; that is, no delays (or, at most, constant delay) of the functional units are included. This is usually known as zero (or unit) delay. The primary concerns are: (1), To check if each block performs its intended function and (2), To modify the design to evaluate alternatives before finalizing the design.

The simulator translates the circuit description into an appropriate internal data structure that facilitates its interpretation. For example, a schematic is mapped into a netlist, which is a data structure listing of all the gates and their connectivities. In Chapter 3, we describe the various circuit representations and their influence on the performance of CAD tools. The designer prepares a set of inputs, verification patterns, applies them to the circuit, and then examines the output for correctness. Then, if the designer is satisfied with the outcome, the translation should be performed. It is a good practice to use the same set of test vectors on the actual device to confirm that the circuit is a correct implementation of the design entered into the simulator.

1.3.2 Timing Simulation

Simulation results build sufficient confidence in the design to invest in the production of a prototype. Actual verification of the prototype gives more assurance, since it embodies the process - dependent parameters. Fortunately, present CAD tools include parameter extraction that permits simulation under actual process conditions. Timing verification can be performed in conjunction with functional verification or independently. The delays associated with the various gates are assigned. Usually, nominal delays are assigned to the gates (or functional units in the case of an HDL model). The delays are part of the library used in the design, or they may be modified. In this type of simulation, the net delays are not included, and thus the timing verification is not complete. It is only when the design is placed and routed that actual delays can be assigned to the gates and to their interconnects. Also, loading can be evaluated and assigned to the output pads.

1.4 Testing

Once the design is implemented on silicon, it can be verified by applying the appropriate stimuli and checking the responses. However, testing is not the design verification. It is meant to verify the manufacturing correctness. There are two principal categories of testing : *parametric testing* and *functional testing*. The first category is concerned with the parameters of the circuit, such as current and voltage measurements. Unless the nominal voltages and currents are assured, no further testing of the circuits is really needed. Functional testing is the subject of this book.

As we mentioned earlier, the purpose of testing is to demonstrate that the manufactured IC is error free. One needs first to define an error. Early techniques of testing digital circuits were mostly concerned primarily with functional verification. To switch to a testing method that takes into consideration the structure of the circuit was suggested in a paper presented by R. Eldred at the ACM meeting in August 1959 [Eldred 1959]. The opening sentence of this paper is: “In order for the successful operation of a test routine to guarantee that a computing system has no faulty components, the test conditions imposed by the routine should be devised at the level of the components themselves, rather than at the level of programmed orders.”

To make testing more manageable, it is thus important to characterize the defects in the circuit as logical or electrical value on the nodes connecting the various components of the circuit; that is, to represent the failure modes

by a logical value. This amounts to representing physical defects by models on the logic level. If this mapping is one-to-one, there will be a large number of models to represent. As a model, the fault does not have to be an exact representation of the defects, but rather be useful in detecting the defects. For example, the most common fault model assumes *single stuck-at (SSA) lines* even though it is very clear that this model does not accurately represent all actual physical failures. However, despite its popularity, the stuck-at fault is no longer sufficient for present circuits and technologies. With the advent of MOS technology, it has become evident that additional fault models are needed to represent more comprehensively the failure modes in this technology [El-Ziq 1981]. Fault modeling is the topic of Chapter 2.

To this point we have asserted that digital testing is performed on the actual IC using test patterns that are generated to demonstrate that the product is fault-free. This view of testing implies that test generation is done at the gate-level design. This used to be the case. Nowadays, there is a testability cycle that parallels the design cycle, as illustrated in Fig. 1.4. It is realistic, therefore, to associate with every design verification step a counterpart in the testability cycle. For example, at the logic level, an automatic test pattern generation is used to develop the patterns and they are verified using *fault simulators*. As the trend is moving toward designing on higher levels of abstraction, testability is also being performed on the same levels. Attempts for test pattern generation on the RTL and behavior levels have been reported. However, at these levels of design abstraction, there are other means of assessing the testability of the circuit without actually developing the patterns. Usually known as *testability measures*, they are described in a later section and discussed in more detail in Chapter 8.

Another fundamental part of testing that is more intimately related to the design is the practice of Design for Testability (DFT). This approach advocates introducing embedded test constructs into the circuit to make testing easier. The ease here refers to test pattern generation and test application processes.

Testing is also becoming a factor in design optimization. Designers customarily strive for an optimal design: -- a high-speed, low-power design occupying the smallest possible area. However, depending on the use of the product, the designers often optimize one of the three attributes: speed (or delay), area, and power. At present, a fourth attribute is considered : testability.

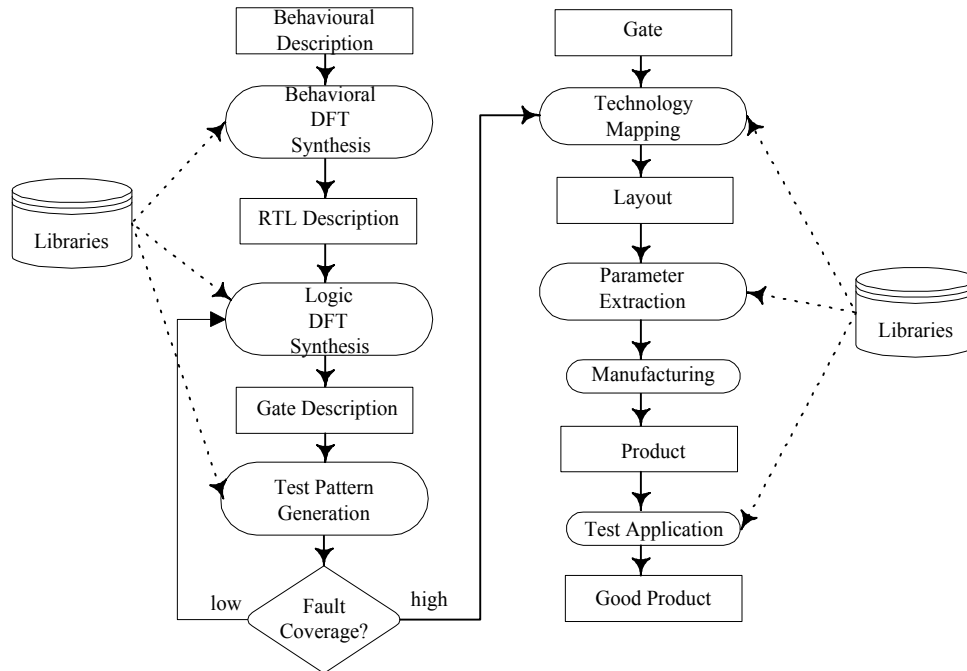


Figure 1.4 The DFT Cycle

1.5 Faults and Their Detection

In this section we describe what a fault is and how a test pattern detects it. We use the single stuck-at fault model as an example to illustrate the concept, but any other model would work as well. A *single stuck-at* (SSA) *fault* represents a line in the circuit that is fixed to logic value 0 or 1. One may think of it as representing a short between the faulty line and the ground or V_{DD} rail. Examples of failure modes that manifest themselves as stuck-at faults are shown in Fig. 1.5. The first example shows a short in a bipolar inverter that forced the input to be held to ground, thus causing a stuck-at-0 fault. The other example of a stuck-at-0 (SA0) fault on the input of a CMOS inverter is due to oxide breakdown between the gate and the source of the pull-down transistor. In these two examples, the SA0 models represent the failure mechanism. The short in the resistance, R_2 , actually resulted in a short between the base of the bipolar transistor, Q_1 , and ground. Whatever signal is imposed on the input of the XOR gate will be sensed by this gate as a zero logic and will change the functionality of the gate from an XOR, $Z = A \oplus B$, to an inverter, $Z = B'$. For the CMOS inverter, the input being stuck-at-0 is equivalent to holding the output at logic one. Similarly, had the input been stuck-at-1 (SA1), the output would appear as stuck-at-0. Thus the SA faults (SSA) on the inputs on an inverter are equivalent to those at its output.

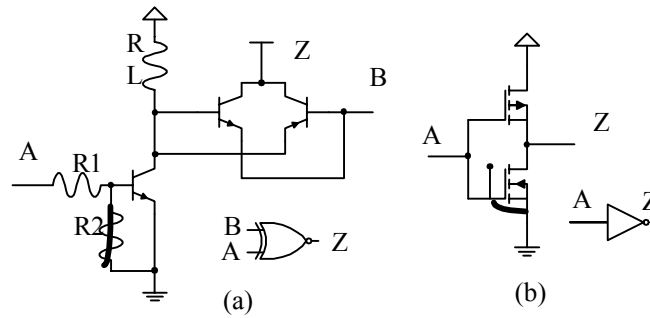
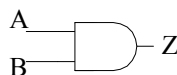


Figure 1.5 Stuck-at Faults: (a) A bipolar XNOR Gate, (b) A CMOS Inverter

Next we examine the behavior, on the logic level, of a two-input AND gate when stuck-at faults are injected, one at a time, on all input and output leads. This is illustrated in Fig. 1.6. All input combinations are given in the first column of the table. The fault-free and faulty circuit's responses, R and R_f respectively, are listed in the other columns of the table for each stuck-at fault. The fault is detected whenever there is an input combination such that $R \oplus R_f = 1$. Stuck-at faults on line A are indicated by $A/0$ for stuck-at-0 and $A/1$ for stuck-at-1. Similar notations are used for the other lines. A careful observation of the table indicates that a faulty response of the circuit is not always observable. For example, with $A/0$, it is expected that the output will always be zero irrespective of the input combination. Thus the faulty response differs from the fault free response only when the input combination $AB = 11$ is applied on the circuit. This combination is considered as a *test pattern* that detects the fault $A/0$. In a similar fashion, we can determine that this pattern also detects $B/0$ and $Z/0$. This pattern detects any of the faults but does not help diagnose which fault actually occurred. We notice also that a fault may be detected by more than one pattern. This is the case with $Z/1$. Any of the three test patterns 10, 01, and 00 should be sufficient to detect the fault. The latter pattern (00) is the only one that determines that the failure is due to $Z/1$. If the main aim is to detect the failures rather than diagnose them, only three patterns are necessary to accomplish the task. They are 01, 10, and 11 and they form a *test set of length 3*.



Inputs		Faulty Response					
AB	FF Response	A/0	B/0	Z/0	A/1	B/1	Z/1
00	0	0	0	0	0	0	1
01	0	0	0	0	1	0	1
10	0	0	0	0	0	1	1
11	1	0	0	0	1	1	1

Figure 1.6 Stuck-at Faults on a 2-input AND Gate and Their Detection

Summing up the SSA faults for the two-input AND gate, we have shown that: (1), Three patterns are sufficient to detect all faults; (2), The three faults, $A/0$, $B/0$, and $Z/0$, are equivalent; and (3), Detecting stuck-at-1 faults on the inputs guarantees detection of the same fault on the output.

1.6 Test Pattern Generation

The process followed in generating test patterns for the two-input AND gate is relatively straightforward. Conceptually, it can be repeated for any circuit. However, this is an oversimplification of the process. For any meaningful circuit size where there are several hundreds of thousands of gates and lines, the process becomes tedious and time consuming. It has actually been proven mathematically that the process is NP-complete [Ibarra 1975]. Informally speaking, this means that no solution can be obtained within a time that can be expressed as a polynomial in n where n is the size of the circuit. Therefore, heuristics are used in commercial ATPGs. A *heuristic* is a commonsense rule (or set of rules) intended to increase the probability of solving some problem without guaranteeing an optimal solution. Algorithms and heuristics are discussed in more detail in Chapter 4. The oldest technique is the D-algorithm [Roth 1966]. Other algorithms include PODEM [Goel 1981] and SOCRATES [Schultz 1988]. These algorithms are discussed at greater length in Chapter 6. Developing test patterns for sequential circuits is even more complex [Miczo 1983], while for a combinational circuit, one test pattern is sufficient to detect a fault; however, a sequence of patterns is needed for a sequential circuit. It is necessary to put the circuit in a known state before sensitizing the fault to the output.

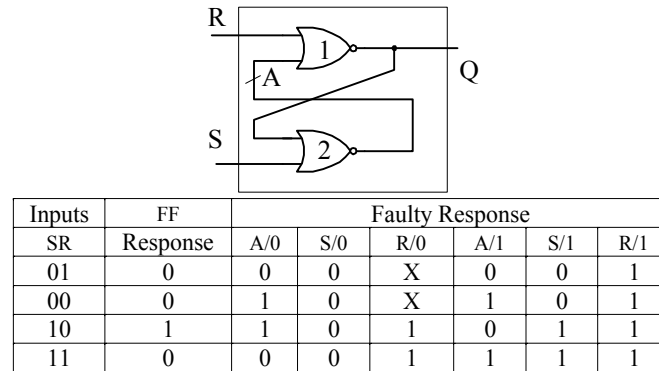


Figure 1.7 Testing a Sequential Circuit, an SR Latch

We illustrate this complexity with a very simple example. Consider the SR latch as shown in Fig. 1.7 and assume that we would like to detect the stuck-at-0 fault on line A . To provoke (or excite) the fault, we need to control A to logic 1 ($A = 1$). This implies that both inputs of gate 2 have to be low, $SQ = (00)$. But, for $Q = 0$, then $R = 1$. Thus $SR = (01)$ is needed to control A to 1. To sensitize the fault to the primary output Q through gate 1, we must have $R = 0$. To detect the fault $A/0$, it is necessary first to apply the pattern $SR = (01)$, followed by the pattern $SR = (00)$. The patterns have to be in this sequence. A sequential circuit needs first to be placed in a certain state before sensitizing (propagating) the fault to the output. In most cases more than two patterns are needed. For example, if we need to detect a SA0 fault on the ripple carry-out of a 32-bit counter, it is necessary to apply 2^{31} clock cycles!

1.7 Fault Coverage

The effectiveness of the test sets is usually measured by the *fault coverage*. This is the percentage of detectable faults in the circuit under test (CUT) that are detected by the test set. The set is complete if its fault coverage is 100%. This level of fault coverage is desirable but rarely attainable in most practical circuits. Moreover, 100% fault coverage does not guarantee that the circuit is fault-free. The test checks only for failures that can be represented by the used model, such as a stuck-at-fault model. Other failures are not necessarily detected. The fault coverage is calculated using a *fault simulator*. This is a logic simulator in which faults are injected at the appropriate nets of the circuits, usually one at a time. The response of the circuits to test pattern applications is compared with the good response of the circuit. The fault is considered detected if at least one of the test patterns has a response different from the good circuit response. Fault simulation is a topic of Chapter 5.

1.8 Types of Tests

In this section we distinguish between various types of tests according to the test generation method. Tests are categorized as off-chip or on-chip and can also be categorized according to the test application method, discussed in Section 1.9.

1.8.1 Exhaustive Tests

Since a test pattern is a combination of the values applied on the primary inputs of a circuit under test, it is conceivable to use all possible combinations (an exhaustive test set) and apply them to the circuit. This exhaustive approach to testing has the advantage of being easy to generate and of yielding 100% fault coverage. However, such a testing method is efficient only for purely combinational small circuits. Applying an exhaustive test to a 20-primary inputs circuit using a 1 MHz tester would take an entire second [Fujiwara 1986].

It should be clear from the preceding section why exhaustive testing is not applicable for sequential circuits since in these circuits the sequence in which patterns are applied is crucial for fault detection.

1.8.2 Pseudoexhaustive Tests

An alternative to exhaustive testing was proposed [McCluskey 1981]. The approach is to test the components of a circuit exhaustively without having to apply an exhaustive test on the entire circuit. We illustrate the approach with the circuit in Fig. 1.8. The circuit has eight primary inputs. The length of an exhaustive test is 256 patterns. Instead, the circuit is partitioned into three sub-circuits: α , β , and γ . It is possible to test the first component exhaustively using four patterns and observing through the primary output, Z_1 . For this, the other input to the OR gate should be controlled to zero. The other two components each depend on three of the inputs. They can be tested exhaustively with eight patterns each. The total length of the test is the sum of the three individual tests, 20 patterns instead of 256 patterns. To test sub-circuit γ , we need to sensitize the response to the test through Z_2 . This implies keeping $W = 1$. As for component β , the response may be sensitized through Z_1 or Z_2 and hence we need to keep $R = 0$ or $V = 1$. This type of testing requires an efficient way of partitioning the circuit. It is considered as an approach to design in order to ease testability and will be revisited in Chapter 9.

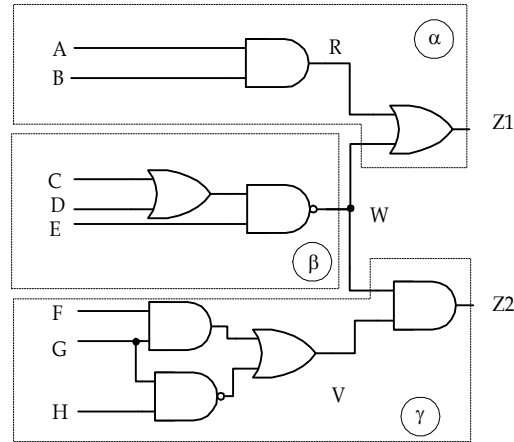


Figure 1.8 Verification Testing

Another special case of exhaustive testing is known as *verification testing* [McCluskey 1984]. It is applicable to circuits where each primary output is a function of only a subset of primary inputs. The circuit we used for pseudoexhaustive testing has two primary outputs, Z_1 and Z_2 . Each output is dependent on a subset of the primary inputs, 5 and 6, respectively, as can be determined from Fig. 1.8. The corresponding exhaustive test sets are of lengths 32 and 64. The length of the test for the circuit is then the sum of both test sets and is equal to 96 patterns. This is definitely a much shorter test set than the exhaustive test of length 256 patterns.

1.8.3 Pseudorandom Tests

Test patterns may also be generated in random order. The cost of generating the test is minimal, but a fault simulator is needed to grade the test and assess the fault coverage. The advantage of random testing is that it has been shown to detect a large percentage (possibly 85%) of stuck-at faults. Consequently, many commercial ATPGs use random testing as a first stage of the test pattern generation and then apply heuristics to deal with the still undetected faults, which are called *Random Pattern Resistant (RPR) faults*. In purely random testing, a test pattern may be generated more than once. However, pseudorandom (PR) test generation is more appropriate to ensure that there is no repetition of patterns. PR test sets may be generated by a software program or by a Linear Feedback Shift Register (LFSR) that is built on the chip. This is called *on-chip* test pattern generation. Figure 1.9 shows a three-stage LFSR that can generate all 3-bit combinations. The number of feedback leads and their positions determine the length of the test. We discuss this topic at greater length in conjunction with Logic Built-In Self-Test (BIST) in Chapter Eleven and memory BIST in Chapter 12.

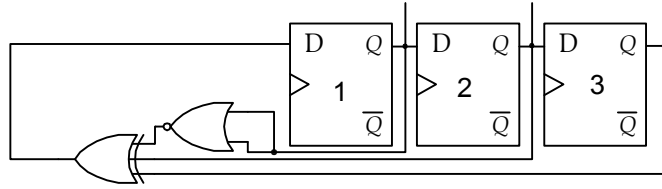


Figure 1.9 Pseudo-Random Testing using a 3-bit Autonomous Feedback Shift Register (ALSFR)

1.8.4 Deterministic Tests

Deterministic tests are fault-oriented tests. In this case, patterns are generated targeting a specific fault model, as we described in Section 1.6. Generating these patterns is an NP-complete problem and requires heuristics to speed up the process. Deterministic test pattern generation is the topic of Chapter 6. In contrast to pseudorandom testing, deterministic tests have to be generated *off-chip*.

1.9 Test Application

The value of a test set is in detecting faults when it is applied to a manufactured IC. In addition, a test set serves in defect analysis during diagnosis. Testing may also be used for debugging, repair of subassemblies, and reliability purposes while the circuit is in normal operation. Test application may be performed on the wafer before it is diced and on the packaged ICs. In this book we concentrate on testing for the purpose of *screening ICs*. However, we first distinguish between on-line and off-line testing. Then we describe the use of automatic test equipment (ATE). Finally, we examine on-line versus off-chip testing.

1.9.1 On-Line versus Off-Line Tests

Depending on the time at which test sets are applied, testing is categorized as *off-line* or *on-line*. *Off-line testing* is performed when the circuit is not in use. On the other hand, *on-line testing* can be performed while the circuit is working in normal mode. This testing requires special codes and checkers and duplication techniques. It is illustrated symbolically in Fig. 1.10. One of the main advantages of on-line testing is the capability to detect faults that are transient or intermittent. These types of faults are described in Chapter 2. In this book, however, we use the term ‘testing’ in the sense of off-line only. Test application methods are different for patterns generated off-chip and patterns generated on-chip. A test manager controls test generation and application for on-chip testing. This is described in Chapters 11, 12, and 13 for logic BIST, memory BIST, and FPGA BIST, respectively.

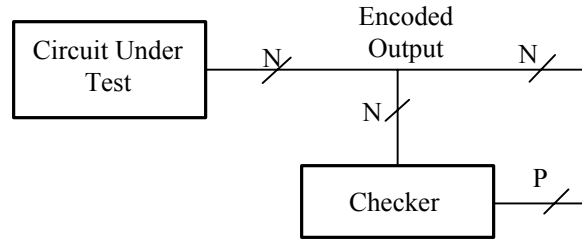


Figure 1.10 On-line Testing

1.9.2 Automatic Test Equipment

For off-chip Testing Automatic Equipment (ATE) is required to apply the test. It consists of a fixture, hardware, and software. The fixture is used to hold the IC under test. Each IC pin is supported by pin electronics that either drives it, measures its signal, and/or acts as a load. The pin electronics are connected to the measurement instrumentation by means of a complex cabling system. The hardware includes a computer system with sufficient storage for test patterns and the expected responses. According to the function of the parts that are being tested, testers are categorized as digital, memory, or analog. Logic and memory, digital ICs, are handled differently. Memory ICs are generally tested in parallel with several parts being given the same input and their outputs being ORed together, while logic parts have to be tested one at a time. Even the generation of input data is different. The data for a memory IC can be produced more easily by an algorithmic pattern generator because of the regularity of these devices' architecture, as we describe in Chapter 12. However, memory ICs are much denser than logic ICs, and for this reason, they have unique problems, such as crosstalk. *Crosstalk* is a noise created by coupling between signals on the chip that may cause the memory element to change values unexpectedly. Digital testers consist, then, of memory testers and logic testers.

Although in this book we concentrate on digital circuits, we compare digital testers to analog circuit testers. The reason is that most circuits in the near future will include both digital and analog parts. This type of circuit is known as a *mixed-signal circuit*. The biggest difference between analog and digital testers is manifested in the configuration of the individual pins. Whereas every digital pin is interchangeable with any other and is therefore supported by the same circuitry, analog pins usually have a large variation in their purpose, so the support circuitry is more unique. Combining digital and analog circuitry on the same chip requires a great deal of care to coordinate and properly isolate the two types of signals and grounds. This requirement, in addition to the various pin types, makes mixed-signal testers much more expensive per pin than other testers, which are already capital-intensive.

The cost of digital testers increases with the number of pins. The high cost per pin is primarily because they are testing leading-edge technology using existing technology - a paradox? Scheduling their use efficiently can offset this high cost. The tester is more economical when its use is amortized over a large production quantity. That is, cost is kept low by making testing time minimal. Therefore, it is important to keep the test set as short as possible. Another way of minimizing test application time and cost is to combine off-chip with on-chip tests.

1.9.3 On-Chip versus Off-Chip Testing

Shortening test application time is difficult for several reasons. First, as the device density increases, the volume of test data is becoming extremely large. Second, the growing disparity between the internal clock frequencies and the output capability of the I/Os makes at-speed testing of ICs extremely difficult, if not impossible. Third, the number of transistors per IC pin keeps increasing. All these factors make the *external bandwidth* much lower than the internal bandwidth. The external bandwidth is defined here as the product of the number of I/Os by the switching speed of the I/Os, while the internal bandwidth is the product of the number of switching transistors by the internal frequency.

To minimize interaction with the external world while keeping the on-chip overhead reasonable, it is possible to partition these functions into on-chip and off-chip resources. In this fashion, testing can be achieved at an optimal rate. This approach will be particularly beneficial for present SOCs that include analog components and RAMs in addition to the digital cores as illustrated in Fig. 1.11. From the presentation on ATE above, it is evident that off-chip testing will require insertion of the IC in three different testers : one for logic, one for memory, and one for analog test. Embedding test pattern generation on-chip as shown to the right in each part of the figure makes it possible to use a single low-bandwidth external tester [Zorian 1999].

1.10 Design for Test

The complexity of test pattern generation motivated the development of design approaches that facilitate testing. The Design For Test (DFT) discipline started formally only in the mid - 1970's after publication of the scan-path technique [Williams 1973] and its adoption by IBM [Eichelberger 1977]. Kobayashi developed the same technique earlier in Japan [Kobayashi 1968]. Also, this design style was practiced informally due to its serviceability and maintainability at IBM [Carter 1964].

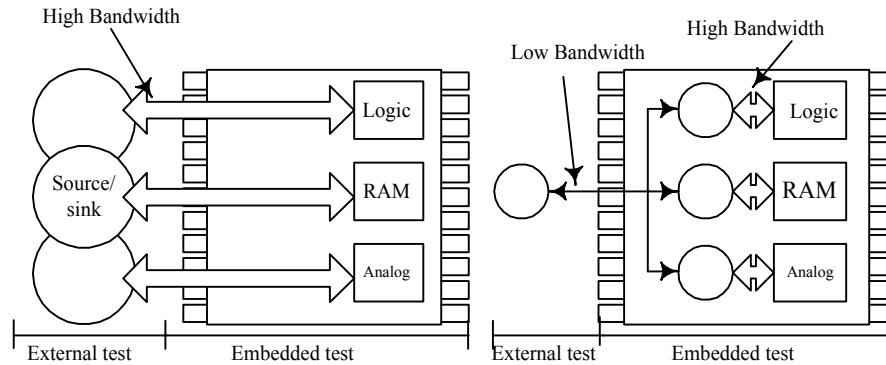


Figure 1.11 Off-chip versus On-chip Testing

Ad hoc approaches to design for testability as well as other techniques such as *Scan Path* design, Built-in Self-Test (BIST), and boundary scan (JTAG) are detailed in subsequent chapters. They are all approaches to enhance *testability*. There is no formal definition for testability. An interesting attempt was given as: “A digital IC is testable if test patterns can be generated, applied, and evaluated in such a way as to satisfy pre-defined levels of performance (e.g., detection, location, application) within a pre-defined cost budget and time scale” [Bennetts 1984]. This definition is still vague and it is interpreted differently by IC designers. One of the key words is “cost.” It is probably the cost of testing that deters the semiconductor manufacturers from doing as much testing as is really needed for reliability. An attempt to quantify testability, in the sense of fault detection, was proposed by [Chang 1974]. Two testability measures (TM) were then defined as controllability and observability.

1.10.1 Controllability

Controllability reports the cost of placing a node in the circuit at a predetermined logic value. Placing this value on a primary input is “free.” However, it is possible to assign a minimal cost, say, 1, to any primary input. The cost increases as the node depth in the circuit increases. This cost will also depend on the type of gate, the logic value to be imposed on the line, and whether the circuit is combinational or sequential. For example, controlling the output of a multi-input AND gate to 1 requires the control of all of its inputs to 1, while for an OR, it is sufficient to control only one of the inputs to 1. Also, controlling a node at a second level, such as node *G* in Fig. 1.12, requires controlling the input at level 1 (*E*) as well as those at level 0 (the primary inputs, *A*, *B*, and *C*). There is a cost, then, for stepping from one level to another.

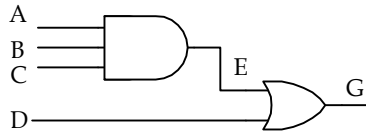


Figure 1.12 An Example to Illustrate TM Calculations

1.10.2 Observability

The *observability* of a node indicates the effort needed to observe the logic value on the node at a primary output. Again, there is no cost associated with observing a primary output. To observe an internal node, for example, node E of the circuit in Fig. 1.12, it is necessary to control D to 0 in order to sensitize through the OR gate to the primary output G. Similarly, the observability of D at G requires that E be controllable to 0. The other primary inputs A, B, and C, have equal observability. The most popular testability measures (TM) program is SCOAP [Goldstein 1980]. These measures are quick estimates of the degree of difficulty generating test patterns without actually generating them. Calculating TM is not as time consuming as test pattern generation. The complexity is only $O(n)$, where n is the circuit size (number of lines). There are several variations of testability measures. Also, TMs have been generalized to design described on RTL and functional levels. They have not been really successful in making a design easier to test, but they still serve as an aid in test pattern generation and in decisions about selecting points of observation for nodes with low observability. Examples of TM calculations are given in Chapter 8.

1.11 Testing Economics

We started this chapter discussing testing and reliability, indicating that the question was not whether to test, but rather, how thorough the test must be to ensure a highly reliable product. Achieving high quality requires a large investment in time and money. As product lifetime is becoming shorter than its development time, it is difficult to justify testing if it delays introduction of the product on time. The loss due to delay in reaching the market at the appropriate moment results in a loss of market share as indicated in Fig. 1.13. The solid line indicates that revenues increase to a peak and then decrease to the end of the product cycle. A delay Δt in arrival of the product to market with respect to its original introduction results in a shorter peak and lower revenue. The hashed area shows this loss of revenues. Thus, if testing causes delays and also incurs cost, it is often hard to justify it on a purely financial basis using this time-to-market as a basis. However, this model does not mention the loss of revenues due to rejects because of problems with the product. It is thus important to show how a slight delay may be compensated for a

better quality. Also, as DFT constructs are introduced from the onset in the design, product introduction would not be delayed. For this we first define two major quality controls, yield and defect level.

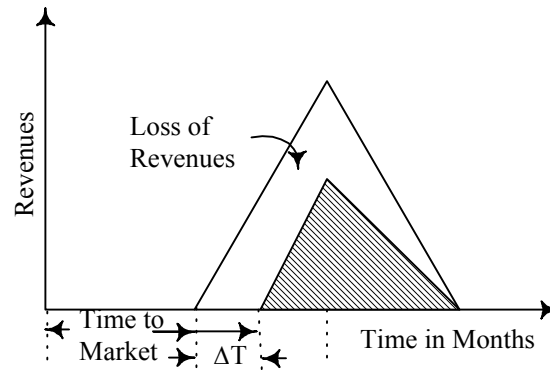


Figure 1.13 Time-to-Market Model

1.11.1 Yield and Defect Level

The yield Y of IC manufacturing is defined as the fraction of the parts that are defect-free. Thus it is determined by $Y = G / (G + B)$, where G is the number of good parts that pass all tests and B is the number of parts that failed some tests. It is difficult to find the exact value of this fraction, for three good reasons. First, there is a lack of data for parts once they are sold. Second, it is not possible to test all chips. Only sampling is used and hence a projected yield is calculated. Third, testing may result in passing, as good, bad chips. Usually, a test will target fault models, and thus they may miss a defect that is not represented by the fault model used.

Many factors affect the yield, including the die area of the wafer, process maturity, and number of process steps. Efforts to understand the defects and their causes -- fault analysis -- has become an important factor for manufacturers. This is witnessed by a preponderance of papers on the topic at testing conferences [ITC 1997]. There are several mathematical models to determine the yield. The first model developed by [Murphy 1964] is given as: $Y = [(1 - e^{-AD}) / AD]^2$, where A and D are the area and the defect density, respectively. The wafer area is increasing constantly. To improve the yield, fabricators are interested in understanding the types of defects in order to minimize them.

The good die are packaged and tested before placing them on the market. Of those that passed the test, some are actually bad. The *defect level* ΔL , the fraction of these bad chips that pass the test, is usually measured in defects per million (DPM). Thus a defect level of 0.1% is equivalent to 1000 DPM.

1.11.2 Fault Coverage and Defect Level

Among the several theoretical formulae relating DL to Y is the following expression due to [Williams 1981]:

$$DL = 1 - Y(1 - T),$$

where T is the fault coverage of the functional test used. For small values of DL, say, less than 1000 DPM

[McCluskey 1989], this expression can be written as

$$DL = TT [-\ln(Y)],$$

where $TT = 1 - T$ is the *testing transparency*, which is the fraction of defects that are not detected. It is often

approximated by $1 - C$, where C is the single-stuck-at-fault coverage. But actually, $TT \geq 1 - C$. Accordingly, TT

can be considered as the percentage of ICs that did not pass the test.

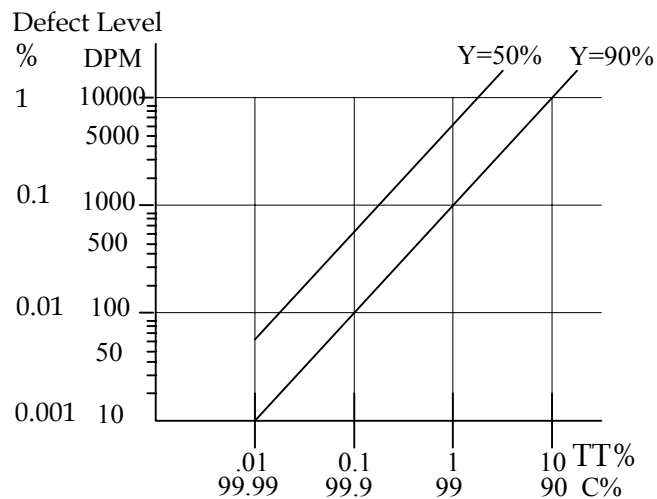


Figure 1.14: Relationship of Defect Level versus Fault Coverage for Given Yield Values

The relationship between DL and T is shown in Fig. 1.14 for two values of the yield, 50% and 90%. Inspection of the graphs indicates that high fault coverage is required to ensure low defect levels, 1 to 100 DPM. Interpretation of experimental data by [Maxwell 1991] indicates that Williams formula is rather pessimistic and that less fault coverage is needed for low values of DL. The dashed line in Fig. 1.15 represents the experimental result using functional testing, and the linear relation is a plot of Williams's expression. The relationship between the two graphs can be interpreted as follows: Although the theoretical 90% fault coverage is required to guarantee a DPM level of 5, it is actually sufficient to reach 50% fault coverage.

Initial publications on testing cost are used to concentrate on the automatic test equipment and their deployment in manufacturing testing. Pioneer papers in DFT cost appeared in the 1980s [Pittman 1984] and [Varma 1984]; Only recently has interest in this area peaked and culminating in a special workshop in 1994 on

economics of design, test, and manufacturing [D&T 1997].

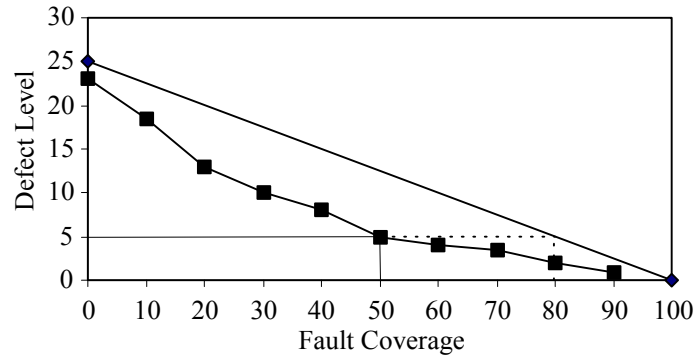


Figure 1.15: Defect Level, Fault Coverage and Yield Theoretical versus Experimental [Maxwell 1990]

1.12 To Explore Further

For those who wish to investigate in depth some of the topics covered in this chapter, the best approach to further exploration is finding resources to aid in your study of testing. For example, surveying testing publications listed in Appendix A and exploring the Web sites of the Test Technology Technical Council (TTTC) can be helpful.

1.13 References

- Abadir, M. et al. (1989), Logic Design Verification via Test Pattern Generation, *IEEE Trans. on Computer - Aided Design of Integrated Circuits and Systems*, Vol. CAD-7, No. 1, pp. 138-149.
- Bennetts, R. G. (1984), *Design of Testable Logic Circuits*, Addison – Wesley, Reading, MA, p. 164.
- Breuer, M. A. and A. D. Friedman (1976), *Diagnostics and Reliable Design of Digital Systems*, Computer Science Press., New York.
- Carter, W. C. et al. (1964), Design of Serviceability Features of the IBM System/360, *IBM*, Vol. 8, No. 4, pp. 115-126.
- Chang, H. Y. et al. (1974), *Fault Diagnosis of Digital Systems*, Krieger Publishing, Melbourne, FL.
- D&T (1997), *IEEE Design and Test of Computer*, Vol. 14, No. 4.
- Eichelberger, E. B. and T. W. Williams, (1977), A Logic Design Structure for LSI Testability, *Proc. Design Automation. Conference*, pp. 462-468.
- Eldred, R. D. (1959), Test Routines Based on Symbolic Logic Systems, *Journal of ACM*, Vol. 6, No. 1, pp. 33 – 36.
- El-Ziq, Y. M. and R. J. Cloutier, (1981), Functional Level Test Generation for Stuck Open Faults in CMOS VLSI, *Proc. IEEE International Test Conf.*, pp. 536-546.
- Fujiwara, H. (1986), *Logic Testing and Design for Testability*, MIT Press, Cambridge, MA.
- Goel, P., (1981), An Implicit Enumeration Algorithm Tests for Combinational Circuits, *IEEE Trans. Computers*, Vol C-30 No. 3, pp. 215-222.

- Goldstein, L. H and E. L. Thigpen (1980), SCOAP : Sandia Controllability/Observability Analysis Program, *Proc. Design Automation Conference.*, pp. 190-194.
- Ibarra, G. H. and S. K. Sahni (1975), Polynomially Complete Fault Detection Problems, *IEEE Trans. Computers.* Vol. C-24 No. 3, pp. 242-249.
- ITC (1997), *Proc. IEEE International Test Conf.*, pp. 633-653.
- Johnson, B. W. (1989), *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading, MA
- Kobayashi, A. et al. (1968), A Flip - Flop Circuit Suitable for FLT, (in Japanese), *Annual Meeting of the Institute of Electronics, Information and Communications Engineers*, Manuscript 892, p. 962.
- Maxwell, P. C. et al., (1991), The Effect of Different Test Sets on Quality Level Prediction: When is 80% better than 90%? *Proc. IEEE International Test Conference*, pp. 358-364.
- McCluskey, E. J. and S. Bozorgui Nesbat (1981), Design for Autonomous Test, *IEEE Trans. Computers*, Vol. C-30, No. 11, pp. 866-875.
- McCluskey, E. J. (1984), Verification Testing : A Pseudo - exhaustive Test Technique, *IEEE Trans. Computers*, Vol. C-33, No. 6, pp. 541-546.
- McCluskey, E. J. and F. Buelow (1989), IC Quality and Test Transparency, *IEEE Trans. Industrial Electronics*, Vol.36 No. 2, pp. 197-202.
- Miczko, A. (1983), The Sequential ATPG: A Theoretical Limit, *Proc. IEEE International Test Conferencen*, pp. 143-147.
- Murphy, B. T. (1964), Cost-Size Optima of Monolithic Integrated Circuits, *IEEE Proc.*, Vol. 52, No. 12, pp. 1537-1545.
- Pittman, J. S. and W. C. Bruce (1984), Test Logic Economic Considerations in a Commercial VLSI Chip Environment, *Proc. IEEE International Test Conference*.
- Roth, J. P. (1966), Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal of Research and Development*, Vol. 10, No. 7, pp. 278-291.
- Schulz, M.H. et al. (1988), SOCRATES: A Highly Efficient Automatic Test Pattern Generation System, *IEEE Trans. on Computer Aided Design*, Vol. CAD-8 No. 1, pp. 126-137.
- Siewiorick, D. P. and R. S. Swarz (1982), *The Theory and Practice of Reliable System Design*, Digital Equipment Corporation Press, Bedford, MA.
- Varma, P. et al. (1984), An Analysis of the Economics of Self Test, *Proc. IEEE International Test Conference.*, pp 20-30.
- Williams, M. J. Y. and J. B. Angel (1973), "Enhancing Testability of Large Scale Integrated Circuits via Test Points and Additional Logic, *IEEE Trans. on Computers*, Vol. C-22, No. 1, pp. 46-60.
- Williams, T. W. and N. C. Brown (1981), Defect Level as a Function of Fault Coverage, *IEEE Trans. on Computers*, Vol. C-30, No. 12, pp. 987-988.
- Zorian, Y. (1999), Testing the monster chip, *IEEE Spectrum*, Vol. 36, No. 7, pp. 54-60.

1.14. Problems

- 1.1. Use the method shown in Section 1.5 to find test patterns to detect stuck-at faults in (a) two-input OR gate, (b) two-input XOR gate, and (c) three-input NAND gate.
- 1.2. For each of the circuits shown in Fig. P1.2:
 - a) Determine the total number of stuck-at faults.
 - b) Develop a test to detect all stuck-at faults on the primary inputs, then find the corresponding fault coverage.
- 1.3. Plot the *detectability profile* of the circuit in Fig. P1.2a. This profile is defined as the frequency of faults detected by 1, 2, 3, ... patterns. (*Hint*: First develop the exhaustive test set, and then determine the number of the patterns that detect each fault.)
- 1.4. For the test set developed in Problem 1.3, determine the number of faults detected by each pattern, then plot the cumulative frequency of faults detected.
- 1.5. How many test patterns detect the fault stuck at 0 on line *E* in Fig. P1.5?
- 1.6. For the circuit in Fig. P1.6, develop a pseudoexhaustive test and compare its length to that of the exhaustive test.
- 1.7. If the yield of a process is 90% and you wish to decrease the DPM from 1000 to 100, by how much do you need to increase the fault coverage? Use the graph in Fig. 1.14.
- 1.8. Verify the authenticity of the following statement: A complete test set for a NAND/NAND implementation of the circuit detects all SSFs of the NOR/NOR implementation of the same circuit.
- 1.9. Assume that the relative cost of repairing defects, C_d , expressed as a function of the percentage, t , of faults tested, is $Cd = (100 - 0.7t)m$, where m is the number of units to be manufactured. Further, assume that the cost C_p of achieving a particular test percentage t is $C_p = t/(100 - t)$. What value of t will minimize the total cost? [Miczo 1986]

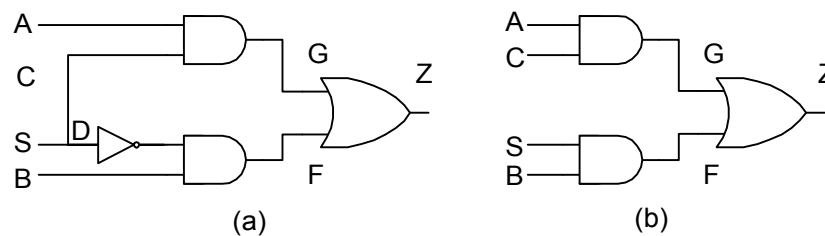


Figure P1.2

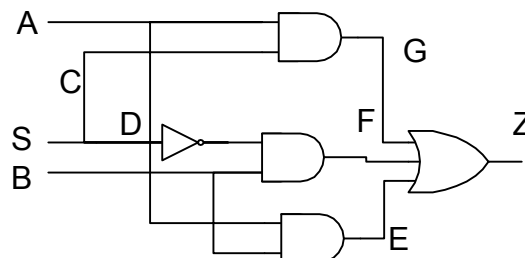


Figure P1.5

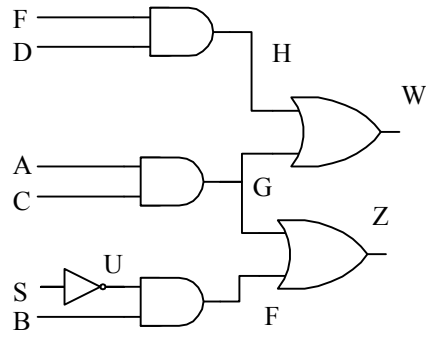


Figure P1.6