CHAPTER

SYSTEM ON CHIP (SoC) DESIGN AND TEST

3

CONTENTS

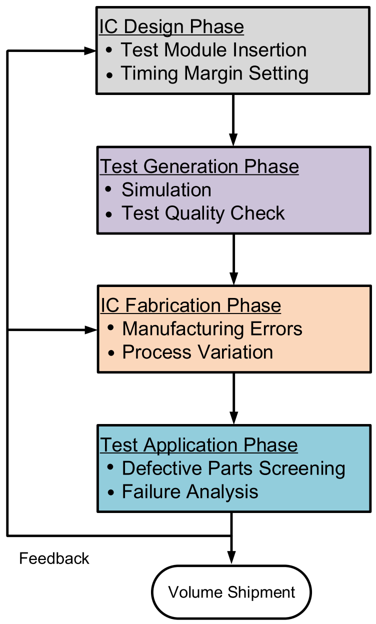

FIGURE 3.1

Simplified IC design, fabrication and test flow.

3.1 INTRODUCTION

As technology feature sizes of devices and interconnects continue to shrink at the rate predicted by Moore's law, gate density and design complexity on integrated circuits (ICs) also continue to increase in recent decades. The nanoscale fabrication process is expected to introduce more manufacturing defects. New failure mechanisms that are not covered by the current fault models are observed in designs fabricated in new technologies with new materials. At the same time, the power and signal integrity issues, which come with scaled supply voltages and higher operating frequencies, increase the number of faults that violate the pre-defined timing margins. As a result, very large scale integration (VLSI) testing has become more and more important and challenging to verify the correctness of design and manufacturing processes. The diagram shown in Fig. 3.1 illustrates a simplified IC production flow. In the design phase, the test modules are inserted in the netlist and synthesized in the layout. Designers set timing margin carefully to account for the difference between simulation and actual operation in the field, such as uncertainties introduced by process variations, temperature variation, power supply noise, and clock jitter. However, due to imperfect design and fabrication processes, there are variations and defects that make the chip violate this timing margin and cause functional failure in the field. Functional bugs, manufacturing errors, and defective packaging process could be the source of these errors. It is thus imperative to screen out the defective chips and prevent shipping them to customers to reduce field returns.

Today, the information collected from testing is used not only to screen defective products from reaching the customers but also to provide feedback to improve the design and manufacturing process (see Fig. 3.1). Therefore, VLSI testing can also improve manufacturing yield level and profitability.

3.1.1 TEST COST AND PRODUCT QUALITY

Although high test quality is preferred, it always comes at the price of high test cost. Trade-offs are necessary to reach the required test quality with minimum cost [1]. In this section, concepts such as test cost, yield, and product quality are introduced. These concepts, when applied to electronic test, lead to economic arguments that justify the need for design-for-testability (DFT) [2].

3.1.1.1 Test Cost

Test cost includes the cost of automatic test equipment (ATE) (initial and running cost), the cost of test development (computer-aided design (CAD) tools, test vector generation, test programming) [3], and the cost of DFT [4]. The scan design techniques can significantly reduce the cost of test generation, and the built-in self-test (BIST) method can lower the complexity and cost of ATE [5].

As shown in Fig. 3.2, the electronic industry tests chips at different levels. Wafer testing is performed during semiconductor device fabrication using ATE. During this step, each device that is present on the wafer is tested for faults—such as stuck-at and transition delay—by applying specifically generated test patterns to it. The wafer is then cut into rectangular blocks, each of which is called a die. Each good die is then packaged, and all packaged devices are tested through final testing again on the same or similar ATE used during wafer testing. After the chips are shipped to the customers, they typically perform PCB testing and system testing (sometime is also called acceptance testing) again because the rule of ten holds according to experience [6]. That is, it is usually ten times more expensive than chip level to repair or replace defective IC at PCB level. After chips are assembled into systems, if a board fault is not caught in PCB testing, it costs ten times as much at the system level than at the board level to find the fault. Since modern systems are much more complex than in 1982 when the empirical rule was first stated in [6], the cost increase is much more than 10 times. For airplanes, a chip fault uncaught in testing can cost thousands or millions times more. Therefore, VLSI testing and use of DFT is essential to reach the goal of high test quality for mission-critical applications, such as automotive, space, and military.

3.1.1.2 Defect, Yield and Defect Level

A manufacturing *defect* is a finite chip area with electrically malfunctioning circuitry caused by errors in the fabrication process. Defect on wafers could be caused by process variations, such as impurities in wafer materials and chemicals, dust particles on masks or in the projection system, mask misalignment, and incorrect temperature control. Typical defects are broken (open) metal wires, missing contacts, bridging between metal lines, missing transistors, incorrect doping levels, void vias, resistive open vias, and many other phenomena that can cause a circuit to fail. In short, a chip with no manufacturing defect is called a good chip. Fraction (or percentage) of good chips produced in a manufacturing process is called *yield*. Yield is commonly denoted by symbol Y in practice. For chip area A, with fault density f, where f is the average number of faults per unit area, fault clustering parameter β, and fault

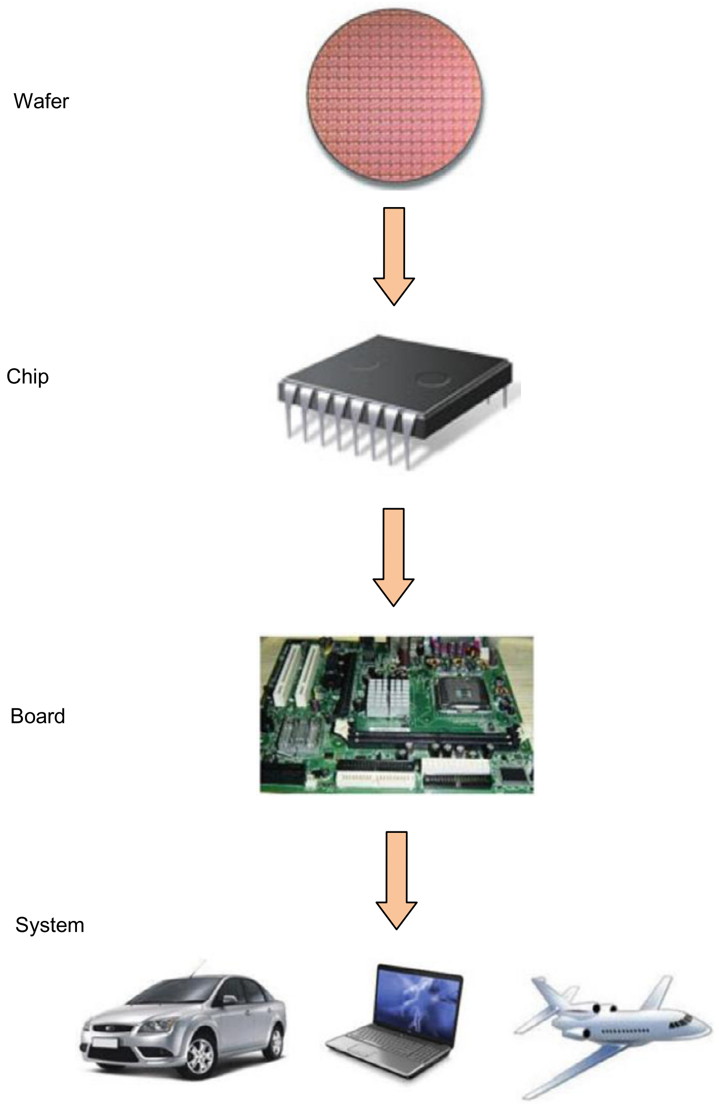

FIGURE 3.2

Test levels: wafer, packaged chip, PCB, and system in field.

coverage T, the yield equation [5] is expressed as

$$Y(T) = (1 + \frac{T \cdot A \cdot f}{\beta})^{-\beta}. \tag{3.1}$$

Assuming that tests with 100% fault coverage ($T = 1.0$) remove all faulty chips, the yield $Y(1)$ is

$$Y = Y(1) = (1 + \frac{A \cdot f}{\beta})^{-\beta}. \tag{3.2}$$

Good test process can reject most or all of the defective chips. However, even if it rejects all the faulty chips, it cannot improve the process yield by itself, unless the diagnostic information collected during test is fed back to the design and fabrication process. Briefly, there are two ways of improving the process yield [5]:

- **Diagnosis and repair.** Defective chips are diagnosed and then repaired. Although this will help improve the yield, it increases the cost of manufacturing.
- **Process diagnosis and correction.** By identifying systematic defects and their root cause, the yield can be improved once the cause is eliminated during manufacturing process. Process diagnosis is the preferred method for yield improvement.

A metric used to measure the effectiveness of tests and the manufactured product quality is defect level (DL), which is defined as the ratio of faulty chips among the chips that pass the tests. It is measured as parts per million (ppm). For commercial VLSI chips, a DL greater than 500 ppm is considered unacceptable. For critical applications such as automotive, zero DPPM is sought.

There are two methods for the determination of defect level. One is from the field return data. Chips failing in the field are returned to the manufacturer. The number of returned chips normalized to one million chips shipped is the defect level. The other is using test data. Fault coverage of tests and chip fallout rate are analyzed. A modified yield model is fitted to the fallout data to estimate the defect level, where chip fallout is the fraction of chips failing up to a vector in the test set, which is $1 - Y(T)$.

When chip tests have a fault coverage T, the defect level is given by the following equation [5]:

$$DL(T) = \frac{Y(T) - Y(1)}{Y(T)} = 1 - \frac{Y(1)}{Y(T)} = 1 - (\frac{\beta + T \cdot A \cdot f}{\beta + A \cdot f})^{\beta}, \tag{3.3}$$

where Af is the average number of faults on the chip of area A, and β is the fault clustering parameter; Af and β are determined by test data analysis. This equation represents DL as a fraction that should be multiplied by 10^6 to obtain ppm. For zero fault coverage, $DL(0) = 1 - Y(1)$, where $Y(1)$ is the process yield. For a 100% fault coverage, $DL(1) = 0$.

An alternative equation relating defect level, yield, and fault-coverage, in case of unclustered random defects is [7]

$$DL(T) = 1 - Y^{1-T}, \tag{3.4}$$

where T is the fault coverage of tests, and Y is the yield.

3.1.2 TEST GENERATION

3.1.2.1 Structural Test vs. Functional Test

In the past, functional patterns were used to verify if there were any errors at the output of a manufactured IC. A complete functional test will check each entry of the truth table. This may be possible with small input numbers for smaller circuits. However, as the exhaustive testing of all possible input

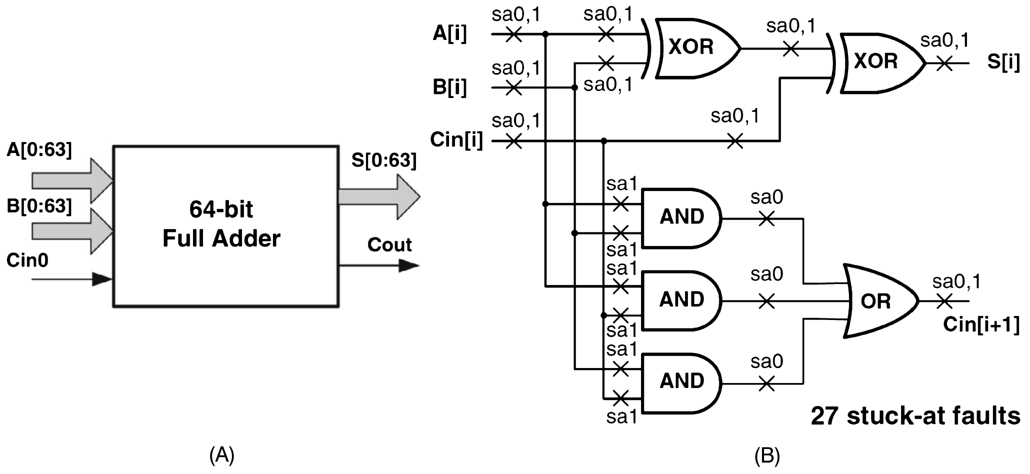

FIGURE 3.3

A 64-bit ripple-carry adder: (A) functional test; (B) structural stuck-at fault test.

combinations grows exponentially as the number of inputs and circuit size increase, such a test will be simply too long and impossible for real circuits with several hundred inputs. Eldred derived tests that would observe the state of internal signals at primary outputs of a large digital system in 1959 [8]. Such tests are called structural tests because they depend on the specific structural aspects, such as gate type, interconnect, and netlist of the circuit under test [5]. Structural test has become more attractive over the past two decades because of the controllable testing time and much lowered test cost.

Structural testing is considered a white-box testing because the knowledge of the internal logic of a system is used for test generation. It makes no direct attempt to determine if the overall functionality of the circuit is correct. Instead, it checks whether the circuit has been assembled correctly from low-level circuit elements as specified in the netlist. The stipulation is that, if the circuit elements are confirmed to be assembled correctly, then the circuit should be functioning correctly. Functional test attempts to validate that the circuit under test functions according to its functional specification. Hence, it can be viewed as black-box test. Sequential automatic test pattern generation (ATPG) programs generates complete set of tests for circuit input-output combinations to completely exercise the circuit function. Figure 3.3 shows a 64-bit ripple-carry adder and the logic circuit design for one bit slice of the adder. As can be seen from Fig. 3.3A, the adder has 129 inputs and 65 outputs. Therefore, to exhaustively test it using functional patterns, one needs $2^{129} = 6.80 \times 10^{38}$ input patterns, and to verify $2^{65} = 3.69 \times 10^{19}$ output response. Using an ATE with operating frequency of 1 GHz, it would take 2.15×10^{22} years to apply all these patterns to this adder circuit, assuming that the circuit can operate at 1 GHz too. Today, considering that most circuits size is much larger than this rather simple adder, exhaustive functional test is impractical. It is worth mentioning that small number of functional test patterns are found to be useful to screen timing defects in practice. For some applications, such as microprocessors, functional testing still plays a very important role. However, instead, structural tests are quite fast to apply to this 64-bit adder circuit. There are 27 stuck-at faults in total for one bit adder after we discard the equivalent faults in Fig. 3.3B. For a 64-bit adder, there are $27 \times 64 = 1728$ faults. It needs at most

1728 test patterns. Using 1 GHz ATE it takes only 0.000001728 s to apply these patterns. Since this pattern set covers all possible stuck-at faults in this adder, it achieves same fault coverage as the large functional test pattern set.

3.1.2.2 Fault Models

The following terminologies are commonly used to describe the incorrectness of semiconductor chips:

- **Defect:** A defect in an electronic system is the unintended difference between the implemented hardware and its intended design. Typical defects in VLSI chips are process defects, material defects, aging defects, and package defects.
- **Error:** A wrong output signal produced by a defective system is called an error. An error is an effect whose cause is some "defect."
- **Fault:** A representation of a "defect" at the abstracted function level is called a fault.

Fault model is a mathematical description of how a defect alters design behavior. A fault is said to be detected by a test pattern if, when applying the pattern to the design, any logic value observed at one or more of the circuit's primary outputs differs between the original design and the design with the fault. There are several fault models developed to describe different kinds of physical defects. The most common fault models for modern VLSI test include stuck-at fault, bridging fault, delay faults (transition delay fault and path delay fault), stuck-open faults, and stuck-short faults.

- **Stuck-at faults:** A signal, which is an input or an output of a logic gate or flip-flop is stuck at 0 or 1 value, independent of the inputs to the circuit. Single stuck-at fault is widely used, that is, two faults per line, stuck-at-1 (sa1), and stuck-at-0 (sa0). An example of stuck-at fault in a circuit is shown in Fig. 3.3.
- **Bridging faults:** Two signals are connected together when they should not be. Depending on the logic circuitry employed, this may result in a wired-OR or wired-AND logic function. As there are $O(n^2)$ potential bridging faults, they are normally restricted to signals that are physically adjacent in the design. Sketches of seven typical types of bridging faults are shown in Fig. 3.4. These types are derived from design rule check (DRC), design for manufacturability (DFM) rules, and known bridge between layout features [9]:
  - Type 1: Side-to-Side
  - Type 2: Corner-to-Corner
  - Type 3: Via-to-Via
  - Type 4: End-of-Line
  - Type 5: Side-to-Side Over Wide Metal
  - Type 6: Via Corner to Via Corner
  - Type 7: Side-to-Side with Minimum Width
- **Delay faults:** These faults make the signal propagate slower than normal, and cause the combinational delay of a circuit to exceed clock period. Specific delay faults are: transition delay faults (TDF), path delay faults (PDF), gate delay faults, line delay faults, and segment delay faults. Among them, slow-to-rise and slow-to-fall PDF and TDF are the most commonly used ones. Path delay fault model targets the cumulative delay through the entire list of gates in a path, whereas the transition fault model targets each gate output in the design.

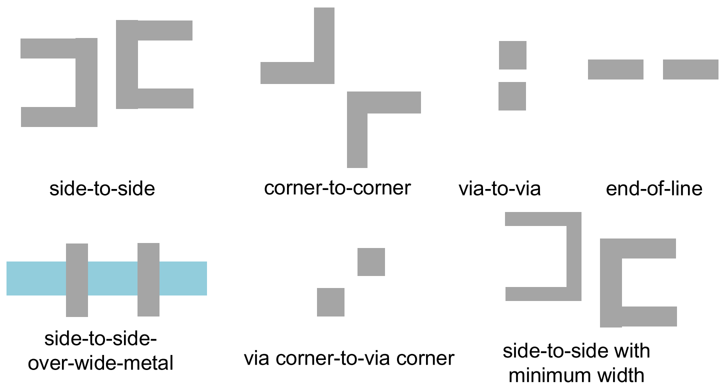

FIGURE 3.4

Seven types of bridging faults.

- **Stuck-open and Stuck-short faults:** A MOS transistor is considered as an ideal switch. Stuck-open and stuck-short faults model the switch being permanently in either the open or the short state. They assume only one transistor is in stuck-open or stuck-short mode. The effect of a stuck-open fault is a floating state at the output of the faulty logic gate. It can be detected in the similar way as detecting a stuck-at fault at the output fault on the gate's output pin. The effect of stuck-short fault is that it short-connects power line and ground line. Measuring quiescent current (IDDQ) can be used to detect such faults and applied as one effective solution.

3.1.2.3 Testability: Controllability and Observability

Testability is represented by *controllability* and *observability* measures that approximately quantify how hard it is to set and observe internal signals of a circuit. Controllability is defined as the difficulty of setting a particular logic signal to a 0 or a 1. Observability is defined as the difficulty of observing the state of a logic signal. Testability analysis can be used for analyzing the difficulty of testing internal circuit parts and—based on it—to redesign or add special test hardware (test point) in the circuit to improve its testability. It can also be used as guidance for algorithms computing test patterns to avoid using hard-to-control lines. Test generation algorithms using heuristics usually apply some kind of testability measures to their heuristic operations, which greatly speed up the test generation process. Through testability analysis, estimation of fault coverage, number of untestable faults and test vector length is also possible.

Testability analysis involves circuit topological analysis without test vectors and search algorithm; it has linear complexity. Sandia controllability observability analysis program (SCOAP) is a systematic, efficient algorithm, proposed by Goldstein [10] and widely used to compute controllability and observability measures. It consists of six numerical measures for each signal (l) in the circuit. Three combinational measures are:

- **CC0(l):** combinational 0-controllability; it represents the difficulty of setting a circuit line to logic 0.
- **CC1(l):** combinational 1-controllability; it represents the difficulty of setting a circuit line to logic 1.
- **CO(l):** combinational observability; it describes the difficulty of observing a circuit line.

Similarly, there are three sequential measures: SC0(l) as sequential 0-controllability, SC1(l) as sequential 1-controllability, and SO(l) as sequential observability. Generally, the three combinational measures are related to the number of signals that may be manipulated to control or observe signal l. The three sequential measures are related to the number of timeframes (or clock cycles) needed to control or observe [5]. The controllability range is between 1 and infinity (∞), and observability range is from 0 to ∞. The higher the measure is, the more difficult it will be to control or observe that line.

According to Goldstein's method in [10], the method to compute combinational and sequential measures is described as below:

- For all primary inputs (PIs) I, set CC0(I) = CC1(I) = 1 and SC0(I) = SC1(I) = 0; for all other nodes N, set CC0(N) = CC1(N) = SC0(N) = SC1(N) = ∞.
- Starting from PIs to primary outputs (POs), use the CC0, CC1, SC0, and SC1 equations, to map logic gate and flip-flop input controllabilities into output controllabilities. Iterate until the controllability numbers stabilize in feedback loops.
- For all POs U, set CO(U) = SO(U) = 0; for all other nodes N, set CO(N) = SO(N) = ∞. Working from POs to PIs, use the CO and SO equations and the pre-computed controllabilities to map output node observabilities of gates and flip-flops into input observabilities. For fanout stems Z with branches Z1, ..., ZN, SO(Z) = min(SO(Z1), ..., SO(ZN)) and CO(Z) = min(CO(Z1), ..., CO(ZN)).
- If any node remains with CC0/SC0 = ∞, then that node is 0-uncontrollable. If any node remains with CC1/SC1 = ∞, then that node is 1-uncontrollable. If any node remains with CO = ∞ or SO = ∞, then that node is unobservable. These are sufficient but not necessary conditions.

When computing controllability for single logic gate, if a logic gate output is produced by setting only one input to a controlling value, then

$$\textit{output controllability} = \min(\textit{input controllabilities}) + 1. \tag{3.5}$$

If a logic gate output can only be produced by setting all inputs to non-controlling value, then

$$\textit{output controllability} = \sum(\textit{input controllabilities}) + 1. \tag{3.6}$$

If an output can be controlled by multiple input sets, such as XOR gate, then

$$\textit{output controllability} = \min(\textit{controllabilities of input sets}) + 1. \tag{3.7}$$

For a logic gate with an input signal that needs to be observed,

$$\begin{aligned} &\textit{input observability} = \textit{output observability} \\ &+ \sum(\textit{controllabilities of setting all other pins to noncontrolling value}) + 1. \end{aligned} \tag{3.8}$$

Figure 3.5 presents examples of SCOAP controllability and observability calculation using AND, OR, and XOR gates.

Figure 3.6 shows a resettable negative-edge triggered D flip-flop (DFF). The combinational controllabilities CC1 or CC0 measures how many lines in the circuit must be set to make DFF output signal Q

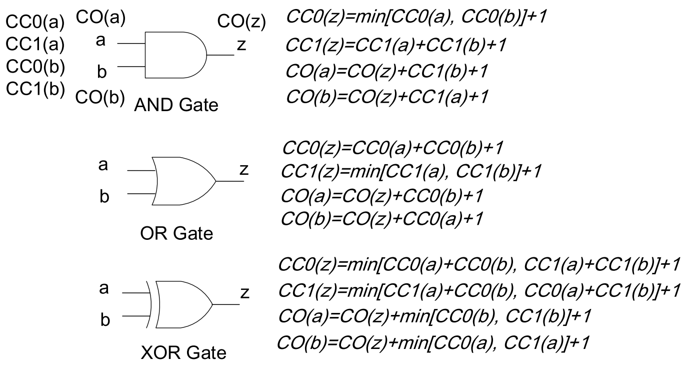
CC0(a)
CC1(a)
CC0(b)
CC1(b)
CO(a)
CO(b)
CO(z)
a
b
z
AND Gate
CC0(z)=min[CC0(a), CC0(b)]+1
CC1(z)=CC1(a)+CC1(b)+1
CO(a)=CO(z)+CC1(b)+1
CO(b)=CO(z)+CC1(a)+1
OR Gate
CC0(z)=CC0(a)+CC0(b)+1
CC1(z)=min[CC1(a), CC1(b)]+1
CO(a)=CO(z)+CC0(b)+1
CO(b)=CO(z)+CC0(a)+1
XOR Gate
CC0(z)=min[CC0(a)+CC0(b), CC1(a)+CC1(b)]+1
CC1(z)=min[CC1(a)+CC0(b), CC0(a)+CC1(b)]+1
CO(a)=CO(z)+min[CC0(b), CC1(b)]+1
CO(b)=CO(z)+min[CC0(a), CC1(a)]+1

FIGURE 3.5

SCOAP controllability and observability calculation.

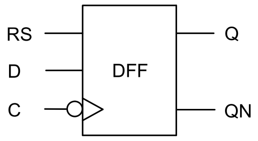

FIGURE 3.6

Resettable, negative-edge-trigged D flip-flop.

as 1 or 0, whereas sequential controllabilities SC1 or SC0 measures how many flip-flops in the circuit must be clocked to set Q to 1 or 0. To control Q line to 1, one must set input D to 1 and force a falling clock edge on C, and the reset signal line RS needs to keep as 0. Note that one needs to add 1 for the sequential measures when signals propagate from flip-flop inputs to output. Thus, CC1(Q) and SC1(Q) are calculated in the following way:

$$\begin{aligned} CC1(Q) &= CC1(D) + CC1(C) + CC0(C) + CC0(RS), \\ SC1(Q) &= SC1(D) + SC1(C) + SC0(C) + SC0(RS) + 1. \end{aligned} \tag{3.9}$$

There are two ways to set Q to 0; either through setting reset signal RS while holding clock C at 0, or clock a 0 through input D. Thus, CC0(Q) and SC0(Q) are calculated using the following equations:

$$\begin{aligned} CC0(Q) &= \min[CC1(RS) + CC0(C), CC0(D) + CC1(C) + CC0(C) + CC0(RS)], \\ SC0(Q) &= \min[SC1(RS) + SC0(C), SC0(D) + SC1(C) + SC0(C) + SC0(RS)] + 1. \end{aligned} \tag{3.10}$$

The input D can be observed at Q by holding RS low and generating a failing edge on the clock line C:

$$\begin{aligned} CO(D) &= CO(Q) + CC1(C) + CC0(C) + CC0(RS), \\ SO(D) &= SO(Q) + SC1(C) + SC0(C) + SC0(RS) + 1. \end{aligned} \tag{3.11}$$

RS can be observed by setting Q to a 1 and using RS:

$$\begin{aligned} CO(RS) &= CO(Q) + CC1(Q) + CC1(C) + CC0(C) + CC1(RS), \\ SO(RS) &= SO(Q) + SC1(Q) + SC1(C) + SC0(C) + SC1(RS) + 1. \end{aligned} \tag{3.12}$$

There are two ways to indirectly observe the clock line C: (1) set Q to 1 and clock to a 0 from D, or (2) reset the flip-flop and clock to a 1 from D. Thus,

$$\begin{aligned} CO(C) = \min[&CO(Q) + CC0(RS) + CC1(C) + CC0(C) + CC0(D) + CC1(Q), \\ &CO(Q) + CC1(RS) + CC1(C) + CC0(C) + CC1(D)] \\ SO(C) = \min[&SO(Q) + SC0(RS) + SC1(C) + SC0(C) + SC0(D) + SC1(Q), \\ &SO(Q) + SC1(RS) + SC1(C) + SC0(C) + SC1(D)] + 1. \end{aligned} \tag{3.13}$$

It is worth noting that if scan design is adopted, scan cells are controllable and observable points for testability analysis. Furthermore, controllability and observability measurement based on SCOAP provides the estimate for the testability of a circuit, which is used to guide test generation and testability improvement [11].

3.1.2.4 Automatic Test Pattern Generation (ATPG)

ATPG is an electronic design automation (EDA) method used to find an input (or test) sequence that, when applied to a digital circuit, enables testers to distinguish between the correct circuit behavior and the faulty circuit behavior caused by defects. These algorithms usually operate with a fault generator program, which creates the minimal collapsed fault list, so that the designer needs not be concerned with fault generation [5]. Controllability and observability measures are used in all major ATPG algorithms. The effectiveness of ATPG is measured by the amount of modeled defects, or fault models, that are detected and the number of generated patterns. These metrics generally indicate test quality (higher with more fault detection) and test application time (higher with more patterns). ATPG efficiency is another important consideration. It is influenced by the fault model under consideration, the type of circuit under test (combinational, synchronous sequential, or asynchronous sequential), the level of abstraction used to represent the circuit under test (register, gate, transistor), and the required test quality [12].

Today, because of the very large circuits' size and shortened time-to-market requirement, all the ATPG algorithms are performed by commercially available EDA tools. Figure 3.7 illustrates the basic ATPG running flow. The tool first reads in the design netlist and library models, then, after building the model, it checks test design rules that are specified in the test protocol file. If any violations occur in this step, the tool reports the violation rule as warning or errors, depending on the severity. Using the ATPG constraints specified by the users, the tool performs ATPG analysis and generates test pattern set. If the test coverage meets the users' needs, test patterns are saved in files with a specific format. Otherwise, the users can modify the ATPG settings and constraints, and rerun ATPG.

It is worth noting that there are two coverage metrics: *test coverage* and *fault coverage*. Test Coverage is the percentage of detected faults among those detectable and gives the most meaningful measure of test pattern quality. Fault Coverage is the percent detected of all faults. It gives no indication of undetectable faults. Usually, test coverage is used in practice as an effectiveness measure of the test patterns generated by the ATPG tool.

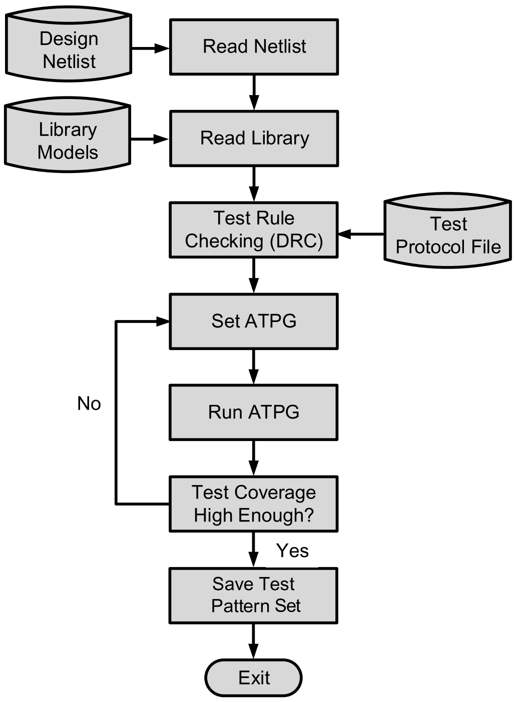

FIGURE 3.7

Basic ATPG flow.

3.2 THE IP-BASED SoC LIFE-CYCLE

SoC is an integrated circuit with all the necessary components for a given system. SoC typically includes analog, digital, and mixed-signal intellectual property (IP) cores. As an evolution of ASIC methodology, it is widely used in various applications due to its high functional density and low power consumption.

Over the past two decades, IP reuse has been extensively studied and developed in industry. IP reuse refers to the inclusion of previously designed and tested components. Since the reused IP has already been designed, verified, and tested, integrators/IP users can reuse the component in a variety of applications. More importantly, IP reuse makes it much cheaper and faster to design and develop a product. Therefore, it becomes a powerful engine to develop modern SoCs in industry to outpace the competition in performance, cost, and time-to-market.

The IP-based SoC life cycle, as shown in Fig. 3.8, refers to the process of design, fabrication, assembly, distribution, usage in the system, and finally end of life. Each step is discussed below in more detail [13]:

Design: Typically, the SoC design cycle involves the following stages. First, design specification is created by a SoC integrator, then the integrator identifies a list of IPs to implement the given specification. Next, all the IP cores, either developed in-house or procured from third party IP vendors, are

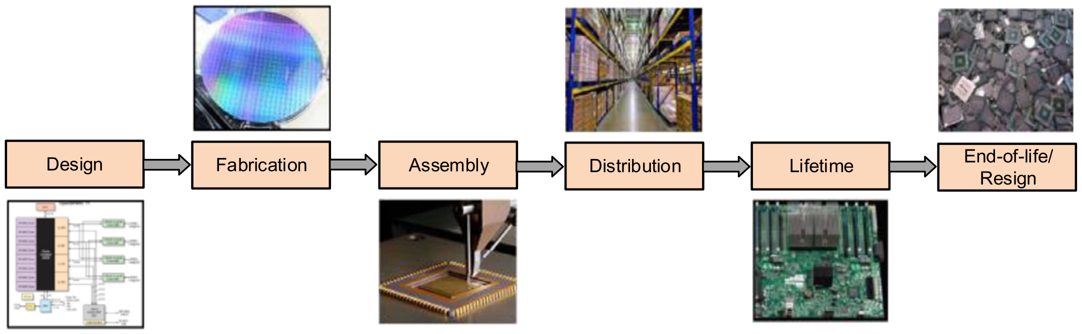

FIGURE 3.8

The IP-based SoC life cycle.

integrated to generate the register transfer level (RTL) SoC design. Following this, the SoC integrator synthesizes the RTL design to a gate-level netlist based on the target technology node. The DFT structures are also inserted to enhance the SoC testability. Then, the netlist is translated into physical layout based on the physical library. Once the timing and power closure is achieved, the final layout in GDSII format is generated. Finally, the chip is sent to foundry for fabrication and test.

The design of today's complex integrated circuits has evolved to a stage, where it is extremely challenging to complete the entire design in-house. In fact, it is common to see the design flow from RTL to GDSII being performed in many different places (even in different countries or continents), mainly to reduce the development cost and time-to-market. Today, design reuse has become an integral part of SoC design. Hard IPs, firm IPs, and soft IPs can be used for this purpose.

Fabrication: Semiconductor device fabrication is the process to make the integrated circuits. The circuits are progressively generated on a wafer made of pure semiconductor materials. The steps for creating the circuits include a sequence of photo lithographic and chemical processing. After that, the manufacturing test is performed to screen out the defective chips and prevent shipping them to next stage.

Today's integrated circuits are manufactured in fabrication facilities (fabs) located all around the world primarily to reduce the manufacturing cost. The design house contracts a foundry to fabricate their designs, discloses the details of their IPs, and also pays for mask-building costs based on their designs. The contract agreement between the foundry and design house is protected by IP rights [14].

Assembly: After fabrication, the foundry sends tested wafers to assembly to cut them into dies. To be specific, here the integrated circuit packaging is called assembly, in which the dies are encapsulated to provide electrical connections, to protect the chip from physical damage and corrosion, and to provide thermal path for dissipating the heat of the chip. Then, assembly performs final tests for those packaged dies before volume shipment.

Distribution: The tested ICs are sent either to the distributors or system integrators (that is, original equipment manufacturers).

System Integration/Lifetime: System integration is the process of assembling together all the components and subsystems into one system to enable them cooperate and act as a whole system.

End-of-life/Resign: When electronics age or become outdated, they are typically retired/resigned and subsequently replaced. Proper disposal techniques are highly advised to extract precious metals and to prevent hazardous materials, such as lead, chromium, and mercury from harming the environment [15].

3.3 SoC DESIGN FLOW

An SoC design flow is illustrated in Fig. 3.9. In Step 1, the design specification is formulated by a SoC integrator, then the integrator identifies a list of IPs to implement the given specification. These IP

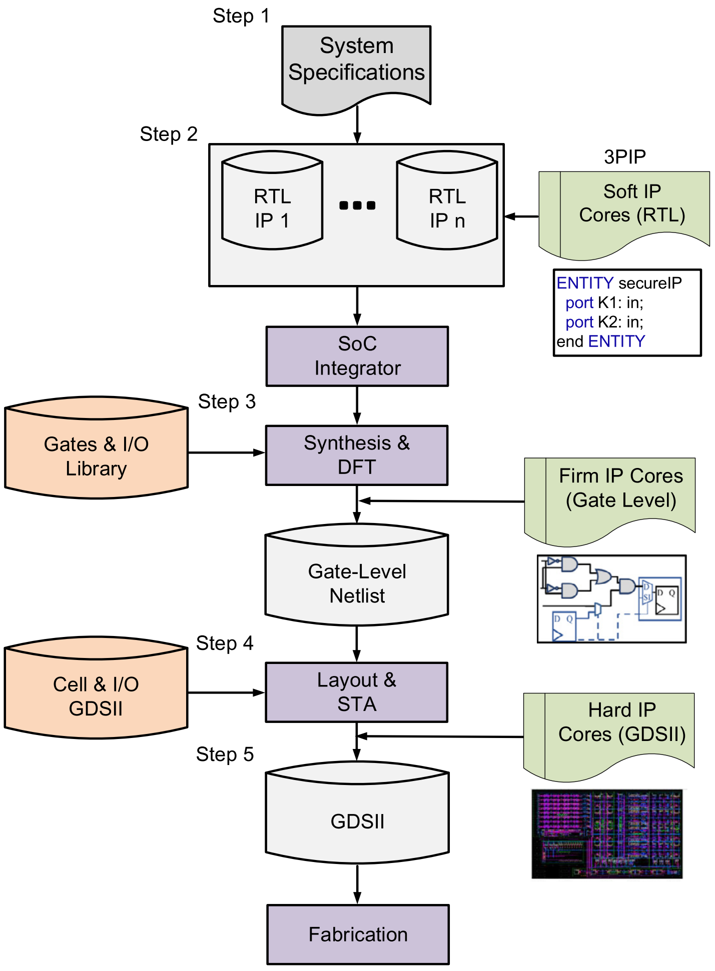

FIGURE 3.9

SoC design flow.

cores are either developed in-house or purchased from third party IP (3PIP) vendors. The 3PIP cores can be procured from the vendors in one of the following three ways [16]:

- Soft IP cores are delivered as synthesizable register transfer level (RTL) hardware description language (HDL);
- Hard IP cores are delivered as GDSII representations of a fully placed and routed core design;
- Firm IP cores are optimized in structure and topology for performance and area, possibly using a generic library.

In Step 2, after developing/procuring all the soft IPs, the SoC design house integrates them to generate the RTL description of the whole system.

In Step 3, the SoC integrator synthesizes the RTL description into a gate-level netlist, based on the logic cells and I/Os of a target technology library. Then, the integrator may integrate gate-level IP cores from a vendor into the netlist. Furthermore, design-for-test (DFT) structures are inserted into the netlist to improve testability.

In Step 4, the gate-level netlist is translated into a physical layout based on logic cells and I/O geometries. It is also possible to import IP cores from vendors in GDSII layout file format at this step.

In Step 5, once static timing analysis (STA) and power closure are complete, developers generate the final layout in GDSII format, and send it out for fabrication.

In Step 6, the chip is fabricated and tested in the foundry/assembly.

3.4 SoC VERIFICATION FLOW

With the increasing complexity and functional density in a single SoC, chip verification becomes more challenging and critical. Verification, also known as pre-silicon validation, mainly refers to the process to ensure the functional correctness and correct transformation from design specifications to the netlist before tape-out, which is shown in Fig. 3.10.

Figure 3.11 outlines the SoC verification flow used in industry. This flow [17] starts with the creation of system specification, which—to some extent—determines and drives the verification strategy. Chip verification planning is concurrently considered with the creation of design specification.

In Step 1, all IPs in the system need to be verified before integration into the system. Specifically, an IP is verified by an IP vendor before sending out to an SoC integrator or an IP user. Since IPs can be delivered in different formats from various IP vendors, the integrator is required to reverify the IP through translating design files and test benches to his/her own application environment [18].

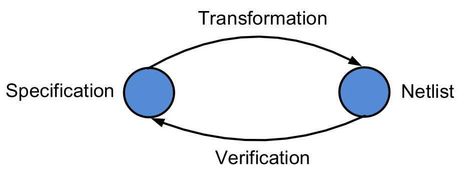

FIGURE 3.10

SoC verification.

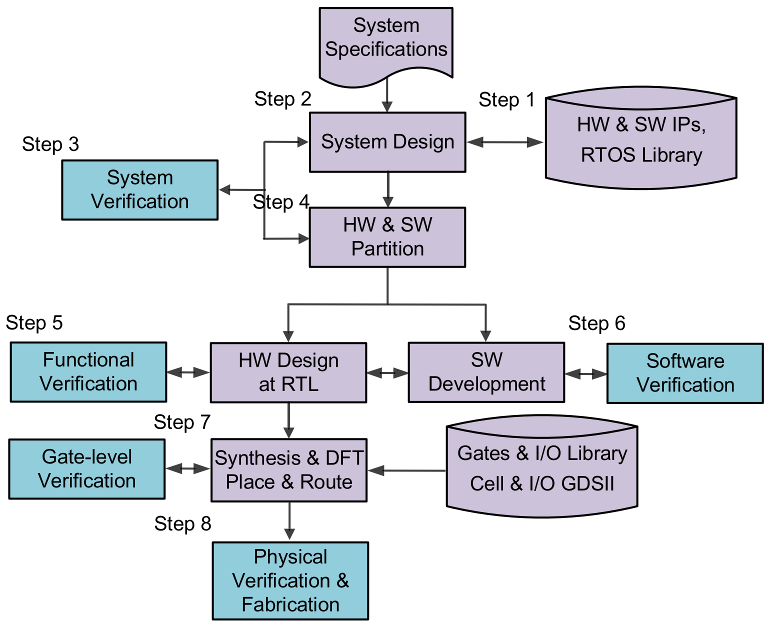

FIGURE 3.11

SoC verification flow.

After individual IP verification, the acquired IP needs to be wrapped with extra logic to communicate with the existing IPs. Then, it is ready to be integrated into the SoC [18].

In Step 2, based on interface protocols, interface verification between blocks in a chip is performed to reduce the final integration efforts, and enable early detection of errors in the system.

In Step 3, SoC level verification is carried out. To be exact, the SoC behavior is modeled based on its specification, and is verified through the behavioral simulation test bench. The test bench could be described using various languages, for example, Verilog/VHDL, C/C++. Furthermore, the test bench is required to be converted to the specified format, which is suitable for both hardware and software verification in the following steps.

In Step 4, once system level verification is complete, the SoC design is partitioned into software and hardware parts based on the software and hardware IP library. Then, the architecture, including software and hardware parts, is verified using the test bench created in the last step.

In Step 5, functional verification is performed on the hardware design at RTL obtained from the last step. Hardware verification uses the test bench created during the process of system behavior verification. Verification at RTL mainly involves line checking, logic simulation, formal verification (that is, equivalence checking and model checking), transaction-based verification, and code coverage analysis.

In Step 6, software verification is carried out based on the system specifications. Software verification and hardware/software (HW/SW) integration can be executed with different methods, including soft prototype, rapid prototype, emulation, and HW/SW co-verification. Take HW/SW co-verification as an example, it is required for SoCs with processor type cores. During this process, HW/SW integration and verification occurs simultaneously. To be specific, co-simulation is performed to couple the current hardware simulators with software emulators/debuggers, which enables the software to be executed on the target hardware design. In turn, the hardware design is provided and stimulated with the actual stimulus. Hence, this co-verification reduces the efforts to create the hardware test bench, and allows earlier hardware and software integration. In addition, it provides a significant performance improvement for system verification.

In Step 7, RTL design is synthesized with the target technology library to generate the gate-level netlist. The netlist is typically verified through formal equivalence checking tool to ensure the RTL design is logically equivalent with the netlist. Since the netlist is usually inserted with DFT components (for example, scan chains) and clock tree in order to meet the testability and the timing requirements, the updated netlist has to be reverified using formal equivalence checking tool to ensure the functional correctness of the updated design. Timing verification is then performed from the gate-level stage to the physical layout stage to avoid any timing violations, thereby to meet timing budget/requirements.

In Step 8, verification is carried out on the integrated circuit physical layout to ensure the design meets certain criteria. It involves design rule check (DRC), layout versus schematic (LVS), electrical rule check (ERC), antenna effect analysis, and signal integrity (SI) analysis, including high current, IR-drop, crosstalk noise, and electromigration. Any violations must be resolved before the chip is manufactured. Once the physical layout verification is complete, the design is ready for sign-off and tape-out.

3.5 SoC TEST FLOW

Compared with SoC verification, SoC test, also known as manufacturing test or production test, is the process to verify whether the design was manufactured correctly. This is mainly due to the fact that the fabrication process is unfortunately imperfect, thus giving rise to defects in chips. It involves different levels of test [19]. The concept comparison between verification and test is shown in Fig. 3.12.

SoC test is the process for screening the manufactured chips; it involves wafer sort, burn-in test, structural test, characterization, and functional test.

The first step during manufacturing test is wafer test. During this step, all dies on the wafer are tested to detect faults through applying test patterns to dies while they are on the wafer. Typically, wafer test employs a wafer prober to supply the necessary electrical excitation to dies on the wafer. It

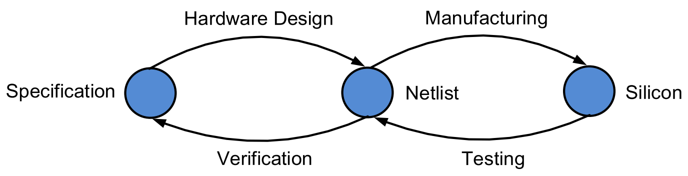

FIGURE 3.12

SoC verification vs. SoC test.

is performed through the application of evaluation methods, including wafer final test (WFT), circuit probe (CP), and electronic die sort (EDS). Once wafer test is complete, it is ready for packaging.

The second step is performed to identify manufacturing defects or faults. This process uses automatic test pattern generation (ATPG) tool to generate test patterns, which enables the test engineer to distinguish between the correct circuit behavior and the faulty circuit behavior. Test equipment such as automatic test equipment (ATE) is utilized to apply test patterns and check responses automatically.

The third step is targeted to characterize and screen chips before shipping to customers. Characterization is the process to find the ideal operating parameters of the chip, such as frequency and voltage. In modern SoCs, high speed I/Os, for example, PCI express, DDR, and Ethernet are also required for characterization through applying various electrical parameters in order to achieve the optimal transmission and error rates.

The fourth step is functional test used to identify functional defects in a chip through applying functional test patterns. Functional test patterns, which run at actual speeds, are utilized to exercise the different parts of the chip to achieve the specified coverage.

The last step is burn-in stress test for the packaged die. It is a temperature/bias reliability stress test used to detect and screen out early life failures.

3.6 DESIGN-FOR-DEBUG

As technology continued to scale further in the last few decades, functional density in a system, either on a single die (as in SoC) or on a single package (system in package (SiP)), has also increased significantly. The number of programmable cores in a system will continue to increase in the foreseeable future [20,21]. Moreover, each core can execute various functions, such as embedded software, hardware accelerators, and dedicated peripheral functions [22], or is integrated with different types of sensors [23].

For a large and complex system, techniques, such as verification, static timing analysis, simulation, and emulation methods cannot guarantee that all errors in hardware and software parts are detected and cleared before the first tape-out [22]. The challenge design and debug engineers face is that some system errors remain undetected until the first silicon is available. These errors typically include—but not limited to—functional errors, timing violations, and design rule check violations.

The reason some errors remain undetected before the first tape-out is that system verification methods in use today can only be applied to the chip model, not to the actual silicon. With the increased complexity of the model, most verification methods are hard to be applied exhaustively due to their computational cost. Therefore, in order to reduce time-to-market and cost, those errors have to be detected as early as possible once the first silicon is available [24]. Hence, design-for-debug (DFD) techniques are required to reduce the time and development cost for locating and fixing those errors, thereby improving the overall system development process.

SoC debug, also referred to as post-silicon validation, happens after the first silicon is available. It is a process to validate all test cases of the silicon/chip for practical deployment and to qualify the design for those test models. During this process, the manufactured design is tested for the functional correctness in a lab setup similar to the deployment in the real application. SoC debug typically involves validating the chip in a system-level environment with software/applications running on the hardware to test all the features and interfaces of the design [19].

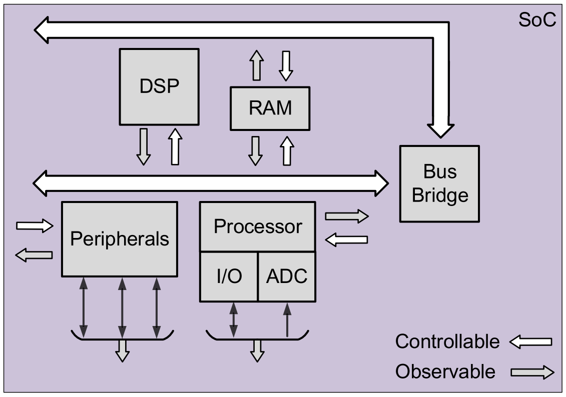

FIGURE 3.13

Debug support for SoC.

Debug support is composed of on-chip DFD architecture and software. As shown in Fig. 3.13, the strategy of implementing debug support is to place access points within a system, hence the controllability and observability of internal signals are available from outside during silicon execution. As chip complexity increases, much attention has been paid to application software besides on-chip debug architecture. The infrastructure provided by debug support forms the basis of development tools and activities such as debuggers, profilers and calibration [25].

3.6.1 DEBUG REQUIREMENTS

System debugging necessitates observing the internal behavior of the chip from outside by applying the appropriate stimuli. Through controlling the system state and repeatedly applying stimuli, a design and debug engineer is able to identify and locate the errors that remained in the system. Following is a list of the fundamental requirements for an effective and efficient debugging system [24,25]:

- External access to the internal system state and critical signals;
- External access to control system operation and facilities, including various peripherals;
- Limited impact on system behavior and overhead in terms of area and number of pins.

External access to the internal system state and critical signals: Debugging an IC requires tracing critical signals of the chip, and extracting the contents of registers and embedded memories, hence, facilitating debug engineers to diagnose and derive the root cause of the chip failure. Due to the large amount of data modern SoCs produce in a few clock cycles, it is unrealistic and unnecessary to trace and collect all that information, especially by a real-time debugging system. Therefore, only the critical signals and the contents of relevant registers and memory arrays are required to be extracted.
External access to control system operation and facilities, including various peripherals: In order to debug the system, an engineer has to control the system operation to create customized triggers and interrupts, and to perform a sequence of operations such as reset, configuration, activation, stop, dumping, single-step, and resume.

Limited impact on system behavior and overhead in terms of area and number of pins: Since debugging infrastructure is aimed at deriving the causes for system execution failure, on-chip DFD architecture needs to be easily plugged into the chip without having a noticeable impact on the chip's external and internal behavior. Otherwise, the errors might go undetected or be manifested in an improper way. In addition, DFD architecture is expected to have limited impact on area overhead and number of pins.

3.6.2 ON-CHIP DEBUG ARCHITECTURE

It is a common practice to reuse DFT infrastructure developed for manufacturing test in integrated circuits for silicon debug [26,27]. As the most popular DFT technique, scan design enables read/write access to all or subset of storage elements in a design. To be exact, it offers direct control of storage elements to a specified value and direct observation of storage elements and, hence the internal states of a circuit. Scan design is realized by replacing flip-flops by scan flip-flops and connecting them to form one or more shift registers in the test mode.

There are three primary advantages to reuse DFT infrastructure for debugging system. First, DFT structure is easily adaptable to various system architectures. Second, since DFT techniques have been well developed and widely practiced over the past few decades, there is limited impact on a design in terms of testability, area overhead, and power consumption, and, hence manufacturing and debug cost. Third, design reuse is essential and critical for modern SoC designs to maintain low cost and reduce time-to-market.

Apart from placing scan chains through critical control paths and data paths to facilitate chip controllability and observability, it is necessary for debug software to interact with on-chip DFD architecture to make debug features available from a workstation. In a nutshell, scan-based on-chip debug architecture, software-based debug support, and run-control features form the foundation of a debugging system.

3.6.3 EXAMPLES OF ON-CHIP DEBUG ARCHITECTURES

A scan design only uses a single scan path to reduce routing overhead, which limits its performance. A generic implementation for on-chip DFD architecture is to multiplex multiple scan chains onto the limited digital pins available for debugging in the application. Thereafter, the critical internal signals can be observed through these pins [24,25].

In [28], besides the scan chains, DFD modules are added into the system to provide silicon debug functionality. Furthermore, debug software is developed and used in design validation phase and silicon validation phase, and interacts with on-chip DFD structure to enable the simulator to communicate with the actual silicon execution.

In [29], the authors developed a debugger tool by leveraging the JTAG boundary scan architecture, which is typically used for testing purpose. They also designed a hardware module for more critical real-time debugging. The boundary scan architecture is responsible for controlling scan chains through JTAG interface and embedded hardware module.

The debugging system [27] developed based on IEEE Standard 1149.1 was successfully used in system debug, test, and manufacturing of Ultra-SPARC-III system. The characteristics for debug and testability were achieved through introducing a few user-defined instructions (example, Shadow and Mask) and extra logic to the chip core, and I/Os (such as, core shadow chain and I/O shadow chain). Based on these features, the critical internal states of the Ultra-SPARC-III can be accessed and controlled during system operation. Furthermore, extra logic was inserted into boundary-scan structure to maintain support for test and debug features. With the combination of shadow chains and on-chip trigger circuits, the captured data can be extracted without interrupting chip operation, thereby addressing the latency and runtime problem caused by pure scan-based control architecture.

At present, besides the infrastructures for tracing signals, dumping storage elements, and triggering customized interrupts, the complexity of SoCs requires standardizing on-chip DFD architecture to enable EDA vendors to create software APIs for accessing and controlling hardware architecture for system-level debug [31]. For example, the facilities provided by ARM Coresight^{TM} architecture [30] is used for tracing, synchronization, time-stamping hardware and software events, the trigger logic, and standardized DFD access and trace transport [31].

3.7 STRUCTURED DFT TECHNIQUES OVERVIEW

3.7.1 DESIGN-FOR-TESTABILITY

DFT techniques are widely used in modern integrated circuits. DFT is a general term applied to design methods that lead to more thorough and less costly testing. In general, DFT is achieved by employing extra hardware circuits for test purpose. The extra test circuits provide improved access to internal circuit elements. Through these test circuits, the local internal state can be controlled and/or observed more easily. It adds more controllability and observability to the internal circuits. DFT plays an important role in the development of test programs and as an interface for test application and diagnostics. With appropriate DFT rules implemented, many system design benefits ensue giving rise to easy detection and location of failures. Generally, integrating DFT in the development cycle can help:

- Improve fault coverage
- Reduce test generation time
- Potentially shorten test length and reduce test memory
- Reduce test application time
- Support hierarchical test
- Realize concurrent engineering
- Reduce life cycle costs

These benefits come at the price of extra cost from pin overhead, more area, and thus low yield, performance degradation, and longer design time. However, DFT reduces the overall costs of the chip, as it is a cost-effective methodology and, hence widely used in IC industry.

Three types of components need test in electronic systems: digital logic, memory blocks, and analog or mix-signal circuits. There are specific DFT methods for each type of components. DFT methods for digital circuits include ad hoc methods and structured methods. Ad hoc DFT method relies on good design experience and experienced designers to find the problem area, such as low coverage area. Sometimes circuit modification or test-point insertion may be required to improve the testability for these areas. The ad hoc DFT techniques are usually too labor-intensive and do not guarantee good results from ATPG. For these reasons, for large circuits it is discouraged to use ad hoc DFT. The common structured methods include: scan, partial scan, BIST, and boundary scan; from among them, BIST is commonly used for memory block testing. The following provides a short introduction to each of these structured DFT techniques.

3.7.2 SCAN DESIGN: SCAN FLIP-FLOP, SCAN CHAIN AND SCAN TEST COMPRESSION

Scan is the most popular DFT technique. Scan design offers simple read/write access to all or subset of storage elements in a design. It also enables direct control of storage elements to an arbitrary value (0 or 1) and direct observation of the states of storage elements and, hence, the internal states of the circuit. In short, it gives the circuit enhanced controllability and observability.

3.7.2.1 Scan Flip-Flop

Scan design is realized by replacing flip-flops by scan flip-flops (SFFs) and connecting them to form one or more shift registers in the test mode. Figure 3.14 illustrates an SFF design based on D-type flip-flop (DFF). A multiplexer is added in front of the DFF to construct a scan D-type flip-flop (SDFF). The test enable (TE) signal controls the working mode of the SDFF. When it is high, it selects the test mode and the scan-in (SI) bits are taken as the input of the DFF. When the TE signal is low, the SDFF works as in functional mode. It acts as a normal DFF and takes value D from the combination circuits as the input to the DFF.

SFF is generally used for clock edge-trigged scan design, whereas level-sensitive scan design (LSSD) cell is used for level-sensitive, latch-based designs. Figure 3.15 shows a polarity-hold shift register latch design that can be used as an LSSD scan cell. The scan cell consists of two latches, a master two-port D-latch L1 and a slave D-latch L2. D is the normal data line and CK is the normal clock line. Line +L1 is the normal output. Lines SI, A, B, and L2 form the shift portion of the latch. SI is the shift data in and +L2 is the shift data out. A and B are the two phase, nonoverlapping shift clocks. The major advantage of using an LSSD scan cell is that it can be used for latch-based design. In addition, it avoids performance degradation introduced by the MUX in shift-register modification. As LSSD scan cells are level-sensitive, designs using LSSD are guaranteed to be race-free. However, this technique requires routing for the additional clocks, which increases routing complexity. It can only be used for slow test application; normal speed testing is impossible.

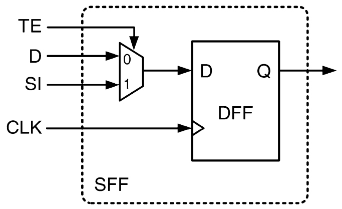

FIGURE 3.14

A scan flip-flop (SFF) constructed with D-type flip-flop and multiplexer.

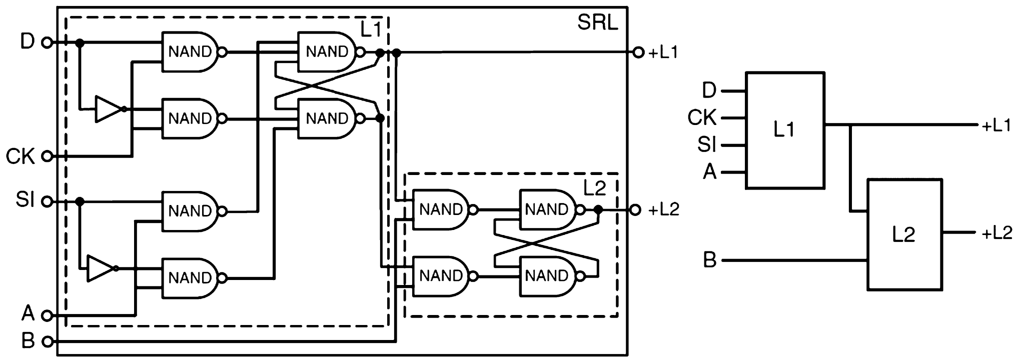

FIGURE 3.15

Level-sensitive scan design (LSSD) cell.

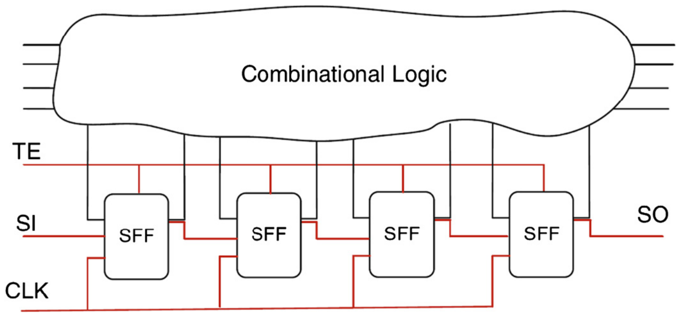

FIGURE 3.16

Scan chain in a design.

3.7.2.2 Scan Chain

Figure 3.16 shows a scan chain in a sequential circuit design. The SFFs are stitched together to form a scan chain. When test enable signal TE is high, the circuit works in test (shift) mode. The inputs from scan-in (SI) are shifted through the scan chain; the scan chain states can be shifted out through scan chain and observed at the scan-out (SO) pin. The test program compares the SO values with expected values to verify the chips performance.

Multiple scan chains are often used to reduce the time to load and observe. SFFs can be distributed among any number of scan chains, each having a separate scan-in (SI) and scan-out (SO) pin. The integrity of scan chains must be tested prior to application of scan test sequences. A shift sequence 00110011... of length $n + 4$ in scan mode ($TC = 0$) produces 00, 01, 11, and 10 transitions in all flip-flops and observes the result at scan chain output SO, where n is the number of SFFs in the longest scan chain.

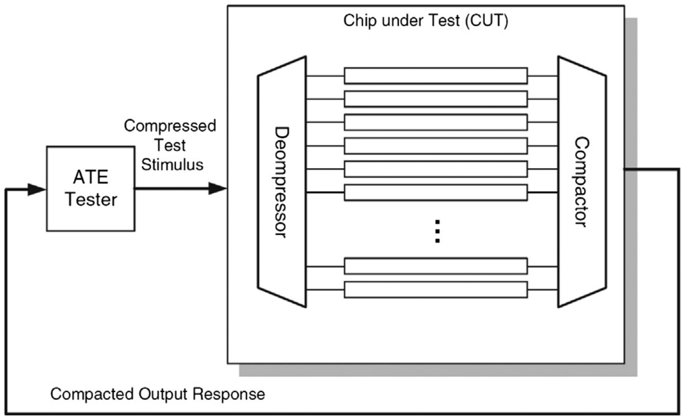

FIGURE 3.17

Scan test compression.

3.7.2.3 Scan Test Compression

As chips become larger and more complex, the growing test-data volume causes a significant increase in test cost because of much longer test time and large tester memory requirements. For a scan-based test, the test data volume is proportional to the number of test cycles, whereas the number of test cycles and test time are related to the number of scan cells, scan chains, and scan patterns as shown in Eq. (3.14). Shift frequency is at a fraction of functional clock frequency (because of high percentage of switching activity). Although, theoretically, increasing shift frequency reduces test time, in practice shift frequency cannot be increased too much due to power dissipation and design constraints.

$$\begin{aligned} Test\ Cycles &\approx \frac{Scan\ Cells \times Scan\ Patterns}{Scan\ \ Chains}, \\ Test\ Time &\approx \frac{Scan\ Cells \times Scan\ Patterns}{Scan\ Chains \times Shift\ Frequency}. \end{aligned} \tag{3.14}$$

As manufacturing test cost depends very strongly on the volume of test data and test time, one of the key requirements is to reduce them dramatically. Test Compression was developed to help address this problem. There are large portion of don't-care bits left when an ATPG tool generates a test for a set of faults. Test compression takes advantage of the small number of significant values to reduce test data and test time. Generally, the idea is to modify the design to increase the number of internal scan chains and shorten maximum scan-chain lengths. As illustrated in Fig. 3.17, these chains are then driven by an on-chip decompressor, usually designed to allow continuous flow decompression, where the internal scan chains are loaded as data are delivered to the decompressor. Many different decompression methods can be used [33]. One common choice is a linear finite state machine, where the compressed stimuli are computed by solving linear equations. For industrial circuits with test vectors' care-bits ratio ranging from 3% to 0.2%, the test compression based on this method often results in compression ratios of 30–500 times [32].

A compactor is required to compact all the internal scan-chain outputs to the output pins. As can be seen from Fig. 3.17, it is inserted between the internal scan-chain outputs and the tester scan channel

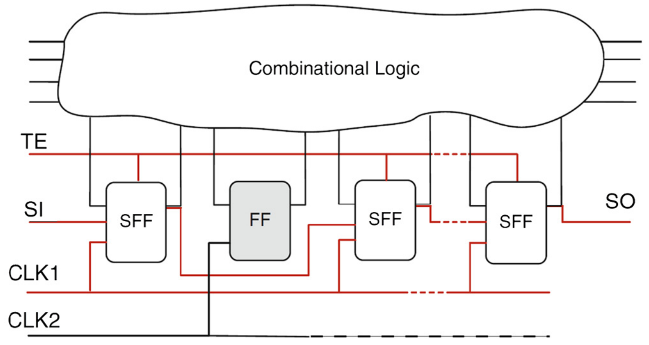

FIGURE 3.18

Partial scan design.

outputs. The compactor must be synchronized with the data decompressor, and must be capable of handling unknown (X) states, which may come from false and multicycle paths, or other unexpected reasons.

3.7.3 PARTIAL SCAN DESIGN

Whereas full scan design replaces all flip-flops with SFFs, partial scan design only selects a subset of flip-flops to be scanned, which provides a wide spectrum of design solutions that tradeoff testability for the overheads (that is, area and power overheads) incurred by the scan design.

Figure 3.18 illustrates the concept of partial scan. Being different from the full scan design shown in Fig. 3.16, not all the flip-flops are SFFs. Two separate clocks are used for scan operation and functional operation.

Selection of flip-flops that can provide the best improvements in testability is a critical part of the partial scan design process. Most SFF selection methods are based on one or several of the following techniques: testability analysis, structural analysis, and test generation [34]. Testability-based methods analyze the testability of the circuit using SCOAP measures and improve the testability by partial scan. However, for circuits with complex structure, the fault coverage achieved may not be adequate using these techniques. Partial scan selection by structural analysis aims to remove all feedback loops from a circuit, and thus simplify the circuit structure for the test generation algorithm. The problem for such techniques is that, for many circuits, it may be infeasible or unnecessary to break all feedback loops to achieve desirable fault coverage. Test generation-based methods exploit information from the test generator to drive the scan selection process. The main advantage of using test generation-based techniques is that it is possible to target specific fault detection objectives rather than simplify the circuit or improve testability of specific regions in the circuit. However, the procedure typically results in expensive computational and storage requirements [34].

As a separate clock is used for the scan operation, the states of the non-SFFs can be frozen during the scan operation and any state can be scanned into the scan register without affecting the states of the non-SFFs. In this way, test vectors can be efficiently generated by a sequential circuit test generator. However, it poses problem in the need for multiple clock trees and tight constraint on clock skew when routing of clock signals.

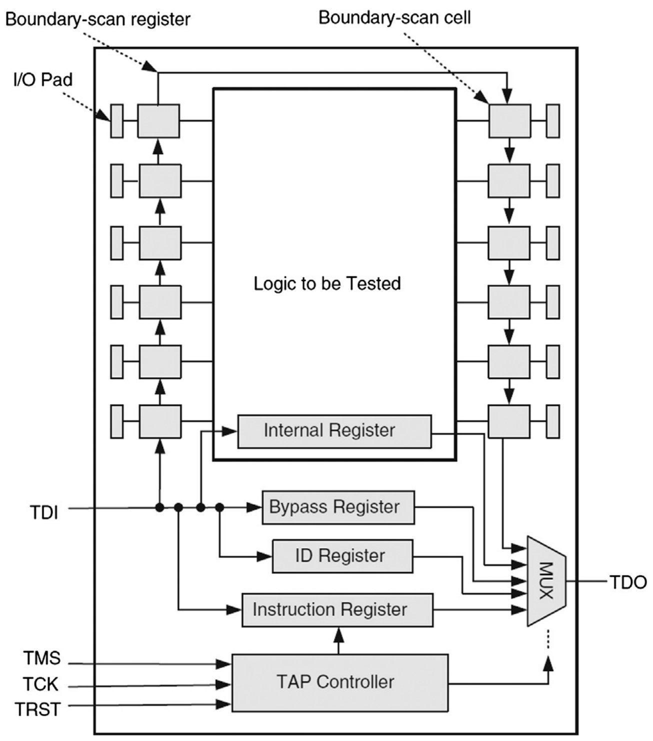

FIGURE 3.19

Boundary scan architecture.

3.7.4 BOUNDARY SCAN

The boundary-scan technique uses a shift-register stage to test factors such as interconnects and clusters of logic and memories. The boundary-scan register consists of boundary scan cells, which are inserted adjacent to each component pin, so that signals at component boundaries can be controlled and observed using scan testing principles. The boundary scan controller has also emerged as the standard mechanism on SoC designs for initiating and controlling the multiple internal memory BIST controllers. Boundary scan is now a well-known and documented IEEE standard, and some test software vendors offer automated solutions. IEEE 1149.1, also known as JTAG or boundary scan, was introduced in 1990 [35]. This standard endeavors to solve test and diagnostic problems arising from loss of physical access caused by the increasing use of high pin count and BGA devices, multi-layer PCBs, and densely packed circuit board assemblies. The standard outlines predefined protocols for testing and diagnosing manufacturing faults. It also provides a means for onboard programming of nonvolatile memory devices such as Flash, or in-system programming of devices like PLDs and CPLDs.

Figure 3.19 illustrates the essential boundary-scan architecture. The block of logic circuits to be tested is connected to multiple boundary-scan cells. The cells are created along with the IC circuitry when the chip is fabricated. Each cell can monitor or stimulate one point in the circuitry. The cells are then connected serially to form a long shift register, whose serial input, designated Test Data Input

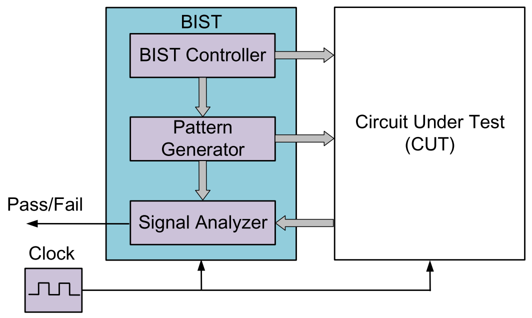

FIGURE 3.20

Built-in self-test architecture.

(TDI), and serial output ports, and designated Test Data Output (TDO) become the basic I/O of a JTAG interface. The shift register is clocked through external clock signal (TCK). In addition to the serial in, serial-out, and clock signals, a Test Mode Select (TMS) input is provided, as well as an optional Test Reset pin (TRST). The TMS, TCK, and TRST signals are applied to a finite state machine called the test access port (TAP) controller. Along with external binary instructions, it controls all of the possible boundary-scan functions. To stimulate the circuit, test bits are shifted in; this is called a test vector.

The primary advantage of boundary-scan technology is the ability to observe and control data independently of the application logic. It also reduces the number of overall test points required for device access, which can help lower board fabrication costs and increase package density. Simple tests using boundary scan on testers can find manufacturing defects, such as unconnected pins, a missing device, and even a failed or dead device. In addition, boundary scan provides better diagnostics. With boundary scan, the boundary-scan cells observe device responses by monitoring the input pins of the device. This enables easy isolation of various classes of test failures. Boundary scan can be used for functional testing and debugging at various levels, from IC tests to board-level tests. The technology is also useful for hardware/software integration testing and provides system-level debug capability [36].

3.7.5 BIST METHODS

Built-in self-test, or BIST, is a DFT methodology involving the insertion of additional hardware and software features into integrated circuits to allow them to perform self-testing, thereby reducing dependence on an external ATE and, thus, reducing testing cost. The BIST concept is applicable to about any kind of circuit. BIST is also a solution to the testing of circuits that have no direct connections to external pins, such as embedded memories used internally by the devices. Figure 3.20 shows BIST architecture. In BIST, a test pattern generator generates test patterns and a signature analyzer (SA) compares test responses. The entire process is controlled by BIST controller.

The two most common categories of BIST are the Logic BIST (LBIST) and the Memory BIST (MBIST). LBIST, which is designed for testing random logic, typically employs a pseudorandom pattern generator to generate input patterns that are applied to the device's internal scan chain, and a multiple input signature register (MISR) for obtaining the response of the device to these input test patterns. An incorrect MISR output indicates a defect in the device. MBIST is used specifically for testing memories. It typically consists of test circuits that apply a collection of write-read-write sequences for memories. Complex write-read sequences are called algorithms, such as MarchC, Walking 1/0, GalPat, and Butterfly. The cost and benefit models for MBIST and LBIST are presented in [37]. It analyzes the economic effects of built-in self-test for logic and memory cores.

Advantages of implementing BIST include:

- low test cost, since it reduces or eliminates the need for external electrical testing using an ATE
- improved testability and fault coverage
- support of concurrent testing
- shorter test time if the BIST can be designed to test more structures in parallel
- at-speed testing

Disadvantages of implementing BIST include:

- silicon area, pin counts, and power overhead for the BIST circuit
- performance degradation, timing issues
- possible issues with the correctness of BIST results, since the on-chip testing hardware itself can fail

3.8 AT-SPEED DELAY TEST

At-speed delay test is widely used to test timing-related failures. It has become a common practice for modern semiconductor industry to include at-speed test in its test flow. This section briefly introduces the basics of at-speed delay test, including its application, fault models used, test clock configuration, and some challenging issues when applying delay test on nanometer designs.

3.8.1 WHY AT-SPEED DELAY TEST?

As technology scales, feature size of devices and interconnects shrink and silicon chip behavior becomes more sensitive to on-chip noise, process and environmental variations, and uncertainties. The spectrum of defects now includes more problems, such as high-impedance shorts, in-line resistance, power supply noises, and crosstalk between signals, which are not always detected with the traditional stuck-at fault model. The number of defects that cause timing failure (setup/hold time violation) is on the rise. This leads to increased yield loss and escape, and reduced reliability. Thus, structured delay test, using transition delay fault model and path delay fault model, are widely adopted because of their low implementation cost and high test coverage. Transition fault testing models delay defects, such as large gate delay faults, for detecting timing-related defects. These faults can affect the circuit's performance through any sensitized path passing through the fault site. However, there are many paths passing through the fault site, and TDFs are usually detected through the short paths. Small delay defects can only be detected through long paths. Therefore, path delay fault testing for a number of selected critical (long) paths has become a necessity. In addition, small delay defects may escape when testing speed is slower than functional speed. Therefore, at-speed test is preferred to increase the realistic delay fault coverage. In [38], it is reported that the defects per million rates are reduced by 30–70% when at-speed testing is added to the traditional stuck-at tests.

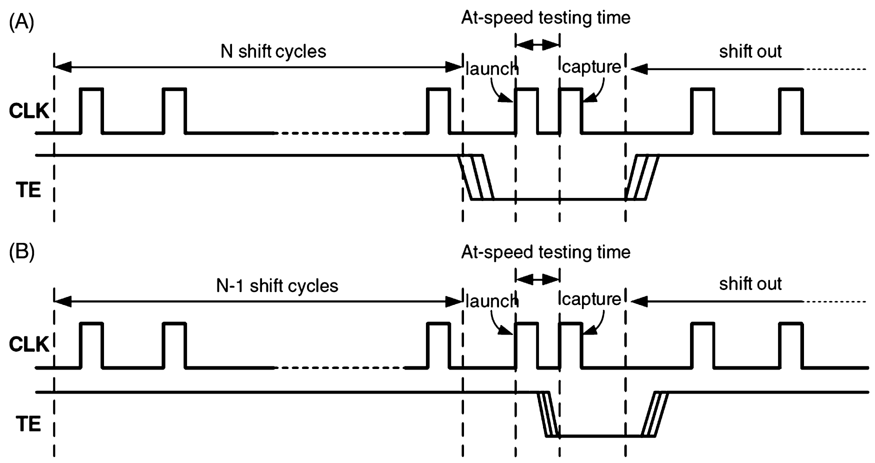

FIGURE 3.21

Clock and Test Enable waveforms for LOC and LOS tests.

3.8.2 BASICS ON AT-SPEED TEST: LAUNCH-OFF-CAPTURE (LOC) AND LAUNCH-OFF-SHIFT (LOS)

The transition fault and path delay fault are the two most widely used fault models for at-speed delay test. Path delay model targets the cumulative delay through the entire list of gates in a predefined path, whereas the transition fault model targets each gate output in the design for a slow-to-rise and slow-to-fall delay fault [5]. The transition fault model is more widely used than path delay because it tests for at-speed failures at all nets in the design, and the total fault list is equal to twice the number of nets. On the other hand, there are billions of paths in a modern design to be tested for path delay fault, leading to high analysis effort. This makes path delay fault model very cost-intensive compared to transition fault model.

Compared to static testing with the stuck-at fault model, testing logic at-speed requires a test pattern with two vectors. The first vector launches a logic transition value along a path, and the second part captures the response at a specified time, determined by the system clock speed. If the captured response indicates that the logic involved did not transition as expected during the cycle time, the path fails the test and is considered to contain a defect.

Scan based at-speed delay testing is implemented using launch-off-capture (LOC) (also referred to as broadside [39]) and Launch-off-shift (LOS) delay tests. LOS tests are generally more effective, achieving higher fault coverage with significantly fewer test vectors, but require a fast scan enable, which is not supported by most designs. For this reason, LOC-based delay test is more attractive and used by more industry designs. Figure 3.21 shows the clock and test enable waveforms for LOC and LOS at-speed delay tests. From this figure, one can see that LOS has a high requirement of TE signal timing. An at-speed test clock is required to deliver timing for at-speed tests. There are two main sources for the at-speed test clocks; one is the external ATE and the other is on-chip clocks. As the clocking speed and accuracy requirements rise, since the complexity and cost of the tester increase, more and more designs include a phase-locked loop or other on-chip clock generating circuitry to

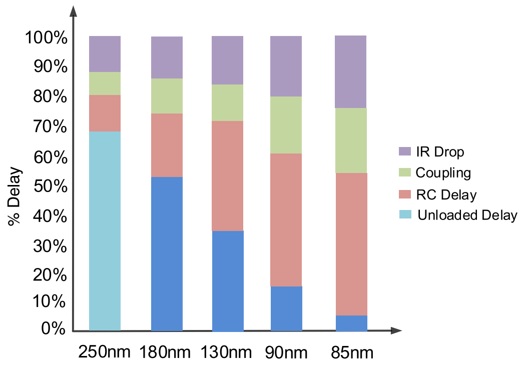

FIGURE 3.22

Parasitic effects with respect to process node.

supply internal clock source. Using these functional clocks for test purposes can provide several advantages over using the ATE clocks. First, test timing is more accurate when the test clocks exactly match the functional clocks. Secondly, the high-speed on-chip clocks reduce the ATE requirements, enabling use of a less expensive tester [38].

3.8.3 AT-SPEED DELAY TEST CHALLENGES

As circuit complexity and functional frequency increase, power integrity and timing integrity are becoming more and more important to circuit design and test. The test power consumption, supply voltage noise, and crosstalk noise caused by signal coupling effect, and hot spots caused by nonuniform on-chip temperature will significantly impact yield and reliability. As shown in Fig. 3.22, with technology node shrinking, the percentage of delay caused by coupling effect between signal lines (crosstalk noise), and IR-drop on power and ground lines (power supply noise), is responsible for a larger portion. Power supply noise and crosstalk noise are becoming two important noises that impact circuit's timing integrity. The lower supply rails in today's ICs mean much less immunity from signal integrity problems that tie directly into power integrity [40]. Supply voltages on many high-end ICs are now down to 1 V and below, leading to decreasing margins for voltage fluctuation. Simultaneous switching noise can cause ground to fluctuate, leading to difficult-to-isolate signal-integrity problems and timing issues. What makes it more challenging is that power, timing, and signal integrity (SI) effects are all interdependent at 90-nm and below.

Timing failures are often the result of a combination of weak points in a design and silicon abnormalities, which reduce the noise immunity of the design and expose it to SI issues. For example, a poor power planning or missing power vias can incur on-chip power droop for some test vectors. The power droop can impact a gate(s) on a critical path and it may cause timing failure. This failure may only be recreated with certain test vectors as inputs. If the corresponding test vector is not included in the test pattern set, the failure becomes an escape and cannot be reproduced during diagnosis with the current test pattern set. Current automatic test pattern generation tools are not aware of the switching distribution on the layout and the pattern-induced noises. There are escapes and "No Problem Found" parts returned by customers, which have passed the tests using the layout-unaware test patterns generated by ATPG tools. Thus, high-quality test patterns are imperative to capture noise-induced delay problems during production test and identify noise-related failures during diagnosis [41,42].

3.9 EXERCISES

3.9.1 TRUE/FALSE QUESTIONS

1. Full scan design is the only option in terms of scan chain insertion.
2. In terms of controllability and observability provided by test point insertion, control points have no impact on observability, and observation points have no impact on controllability.
3. SoC debug can be considered as pre-silicon validation, while SoC verification can be considered as post-silicon validation.
4. Small delay defects may escape when testing speed is higher than functional speed.
5. LBIST and MBIST are the most two common classifications of BIST.

3.9.2 SHORT-ANSWER TYPE QUESTIONS

1. The yield of a manufacturing process is defined as the fraction of acceptable parts among all parts fabricated [5]. If the number of acceptable parts is 5000, and the number of total parts that are fabricated is 7000; calculate the yield.
2. Defect level, also known as reject rate, is defined as the fraction of faulty chips among all chips passing the test, which can be expressed as parts per million (PPM) [5]. If the number of faulty chips passing final test is 10, the total number of chips passing final test is 50,000; calculate the defect level.
3. If each chip has 92% fault coverage and 70% yield, calculate the defect level.
4. List the types of common fault models.
5. What are delay faults? What are the differences between transition delay faults (TDF) and path delay faults (PDF)?
6. Explain ATPG. How can one measure the effectiveness of ATPG?
7. How do you consider that your SoC verification has a good coverage? Explain.
8. Considering an IP core, the first silicon success rate for unverified IP core is 90 percent, whereas the first silicon success rate for verified IP core is 98 percent.
   (a) If a SoC consists of 10 such unverified IP cores, what is the first silicon success rate for the SoC?
   (b) If a SoC consists of 8 such verified IP cores and 2 unverified IP cores, what is the first silicon success rate for the SoC?
   (c) If a SoC consists of 10 such verified IP cores, what should be the success rate for each verified IP core in order to achieve 90 percent of the first silicon success rate for the SoC?
   Note: You do not need to consider the interconnection issues and other IPs within the chip.
9. (a) If a scan-based design has 6400 scan flip-flops that can be constructed as 100 scan chains each with equal length, the number of test patterns is 5000, and testing clock cycle is 10 ns; what would be the test cycle and test time?

(b) What are the main factors impacting manufacturing cost? How can one reduce the manufacturing cost?

10. What are the differences between full scan design and partial scan design?

3.9.3 LONG-ANSWER TYPE QUESTIONS

1. What are the differences between functional test and structural test?
2. (a) When doing manufacturing test for a design, consider the number of detected faults in the design is 81,506, the number of detectable faults in the design is 87,122, and the number of undetectable faults in the design is 103. What are the test coverage and fault coverage, respectively? Round your answer to 2 decimal places.
   (b) What types of faults are contained in – detected faults, possibly detected faults, undetectable faults, ATPG untestable faults, and not detected faults?
3. What are the differences between verification, debug, and test for a SoC?

REFERENCES

[1] H.B. Druckerman, M.P. Kusko, S. Pateras, P. Shephard, Cost trade-offs of various design for test techniques, in: Economics of Design, Test, and Manufacturing, 1994. Proceedings, Third International Conference on the IEEE, p. 45.

[2] V.D. Agrawal, A tale of two designs: the cheapest and the most economic, Journal of Electronic Testing 5 (1994) 131–135.

[3] I. Dear, C. Dislis, A.P. Ambler, J. Dick, Economic effects in design and test, IEEE Design & Test of Computers 8 (1991) 64–77.

[4] J. Pittman, W. Bruce, Test logic economic considerations in a commercial VLSI chip environment, in: Proceedings of the 1984 International Test Conference on the Three Faces of Test: Design, Characterization, Production, IEEE Computer Society, pp. 31–39.

[5] M. Bushnell, V. Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, vol. 17, Springer Science & Business Media, 2004.

[6] B. Davis, The Economics of Automatic Testing, BookBaby, 2013.

[7] T.W. Williams, N. Brown, Defect level as a function of fault coverage, IEEE Transactions on Computers 30 (1981) 987–988.

[8] R.D. Eldred, Test routines based on symbolic logical statements, Journal of the ACM (JACM) 6 (1959) 33–37.

[9] M. Keim, N. Tamarapalli, H. Tang, M. Sharma, J. Rajski, C. Schuermyer, B. Benware, A rapid yield learning flow based on production integrated layout-aware diagnosis, in: Test Conference, 2006. ITC'06. IEEE International, IEEE, pp. 1–10.

[10] L. Goldstein, Controllability/observability analysis of digital circuits, IEEE Transactions on Circuits and Systems 26 (1979) 685–693.

[11] L.-T. Wang, C.-W. Wu, X. Wen, VLSI Test Principles and Architectures: Design for Testability, Academic Press, 2006.

[12] L. Lavagno, G. Martin, L. Scheffer, Electronic Design Automation for Integrated Circuits Handbook-2 Volume Set, CRC Press, Inc., 2006.

[13] U. Guin, D. Forte, M. Tehranipoor, Anti-counterfeit techniques: from design to resign, in: Microprocessor Test and Verification (MTV), 2013 14th International Workshop on, IEEE, pp. 89–94.

[14] T. Force, High performance microchip supply, Annual Report, Defense Technical Information Center (DTIC), USA, 2005.

[15] H. Levin, Electronic waste (e-waste) recycling and disposal-facts, statistics & solutions, Money Crashers (2011), https://www.moneycrashers.com/electronic-e-waste-recycling-disposal-facts/.

[16] V. Alliance, VSI alliance architecture document: Version 1.0, VSI Alliance, vol. 1, 1997.

[17] P. Rashinkar, P. Paterson, L. Singh, System-on-a-Chip Verification: Methodology and Techniques, Springer Science & Business Media, 2007.

[18] F. Nekoogar, From ASICs to SOCs: A Practical Approach, Prentice Hall Professional, 2003.

[19] Verification, validation, testing of asic/soc designs – what are the differences, anysilicon, http://anysilicon.com/verification-validation-testing-asicsoc-designs-differences/, 2016.

[20] D. Patterson, et al., The parallel computing landscape: a Berkeley view, in: International Symposium on Low Power Electronics and Design: Proceedings of the 2007 International Symposium on Low Power Electronics and Design, vol. 27, pp. 231.

[21] D. Yeh, L.-S. Peh, S. Borkar, J. Darringer, A. Agarwal, W.-M. Hwu, Roundtable-thousand-core chips, IEEE Design & Test of Computers 25 (2008) 272.

[22] B. Vermeulen, Design-for-debug to address next-generation SoC debug concerns, in: Test Conference, 2007. ITC 2007. IEEE International, IEEE, pp. 1.

[23] M.T. He, M. Tehranipoor, Sam: A comprehensive mechanism for accessing embedded sensors in modern SoCs, in: Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014 IEEE International Symposium on, IEEE, pp. 240–245.

[24] B. Vermeulen, S.K. Goel, Design for debug: catching design errors in digital chips, IEEE Design & Test 19 (2002) 37–45.

[25] A.B. Hopkins, K.D. McDonald-Maier, Debug support for complex systems on-chip: a review, IEE Proceedings, Computers and Digital Techniques 153 (2006) 197–207.

[26] Y. Zorian, E.J. Marinissen, S. Dey, Testing embedded-core based system chips, in: Test Conference, 1998. Proceedings. International, IEEE, pp. 130–143.

[27] F. Golshan, Test and on-line debug capabilities of IEEE Standard 1149.1 in UltraSPARC/sup TM/-III microprocessor, in: Test Conference, 2000. Proceedings. International, IEEE, pp. 141–150.

[28] G.-J. Van Rootselaar, B. Vermeulen, Silicon debug: scan chains alone are not enough, in: Test Conference, 1999. Proceedings. International, IEEE, pp. 892–902.

[29] D.-Y. Jung, S.-H. Kwak, M.-K. Lee, Reusable embedded debugger for 32-bit RISC processor using the JTAG boundary scan architecture, in: ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conference on, IEEE, pp. 209–212.

[30] Coresight on-chip trace and debug architecture, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.coresight/index.html, 2010.

[31] A. Basak, S. Bhunia, S. Ray, Exploiting design-for-debug for flexible SoC security architecture, in: Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE, IEEE, pp. 1–6.

[32] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, Embedded deterministic test, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23 (2004) 776–792.

[33] N.A. Touba, Survey of test vector compression techniques, IEEE Design & Test of Computers 23 (2006) 294–303.

[34] V. Boppana, W.K. Fuchs, Partial scan design based on state transition modeling, in: Test Conference, 1996. Proceedings. International, IEEE, pp. 538–547.

[35] C. Maunder, Standard test access port and boundary-scan architecture, IEEE Std 1149.1-1993a, 1993.

[36] R. Oshana, Introduction to JTAG, Embedded Systems Programming, 2002.

[37] J.-M. Lu, C.-W. Wu, Cost and benefit models for logic and memory BIST, in: Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings, IEEE, pp. 710–714.

[38] B. Swanson, M. Lange, At-speed testing made easy, EE Times 3 (2004), http://www.eedesign.com/article/showArticle.jhtml?articleId=21401421.

[39] J. Savir, S. Patil, Broad-side delay test, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13 (1994) 1057–1064.

[40] D. Maliniak, Power integrity comes home to roost at 90 nm, EE Times 3 (2005).

[41] J. Ma, J. Lee, M. Tehranipoor, Layout-aware pattern generation for maximizing supply noise effects on critical paths, in: VLSI Test Symposium, 2009. VTS'09. 27th IEEE, IEEE, pp. 221–226.

[42] J. Ma, N. Ahmed, M. Tehranipoor, Low-cost diagnostic pattern generation and evaluation procedures for noise-related failures, in: VLSI Test Symposium (VTS), 2011 IEEE 29th, IEEE, pp. 309–314.