CHAPTER 1

Introduction

Charles E. Stroud
Auburn University, Auburn, Alabama

Laung-Terng (L.-T.) Wang
SynTest Technologies, Inc., Sunnyvale, California

Yao-Wen Chang
National Taiwan University, Taipei, Taiwan

ABOUT THIS CHAPTER

Electronic design automation (EDA) is at the center of technology advances in improving human life and use every day. Given an electronic system modeled at the ***electronic system level*** (ESL), EDA automates the design and test processes of verifying the correctness of the ESL design against the specifications of the electronic system, taking the ESL design through various synthesis and verification steps, and finally testing the manufactured electronic system to ensure that it meets the specifications and quality requirements of the electronic system. The electronic system can also be a ***printed circuit board*** (PCB) or simply an ***integrated circuit*** (IC). The integrated circuit can be a ***system-on-chip*** (SOC), ***application-specific integrated circuit*** (ASIC), or a ***field programmable gate array*** (FPGA).

On one hand, EDA comprises a set of hardware and software co-design, synthesis, verification, and test tools that check the ESL design, translate the corrected ESL design to a ***register-transfer level*** (RTL), and then takes the RTL design through the synthesis and verification stages at the gate level and switch level to eventually produce a physical design described in ***graphics data system II*** (GDSII) format that is ready to signoff for fabrication and manufacturing test (commonly referred to as **RTL to GDSII** design flow). On the other hand, EDA can be viewed as a collection of design automation and test automation tools that automate the design and test tasks, respectively. The design automation tools deal with the correctness aspects of the electronic system across all levels, be it ESL, RTL, gate level, switch level, or physical level. The test automation tools manage the quality aspects of the electronic system, be it defect level, test cost, or ease of self-test and diagnosis.

This chapter gives a more detailed introduction to the various types and uses of EDA. We begin with an overview of EDA, including some historical perspectives, followed by a more detailed discussion of various aspects of logic design, synthesis, verification, and test. Next, we discuss the important and essential process of physical design automation. The intent is to orient the reader for the remaining chapters of this book, which cover related topics from ESL design modeling and synthesis (including high-level synthesis, logic synthesis, and physical synthesis) to verification and test.

1.1 OVERVIEW OF ELECTRONIC DESIGN AUTOMATION

EDA has had an extraordinary effect on everyday human life with the development of conveniences such as cell phones, ***global positioning systems*** (GPS), navigation systems, music players, and ***personal data assistants*** (PDAs). In fact, almost everything and every daily task have been influenced by, and in some cases are a direct result of, EDA. As engineers, perhaps the most noteworthy inventions have been the microprocessor and the ***personal computer*** (PC), their progression in terms of performance and features, and the subsequent development of smaller, portable implementations such as the notebook computer. As a result, the computer has become an essential tool and part of everyday life—to the extent that current automobiles, including safety features in particular, are controlled by multiple microprocessors. In this section, we give a brief overview of the history of EDA in its early years.

1.1.1 Historical perspective

The history of *electronic design automation* (EDA) began in the early 1960s after the introduction of *integrated circuits* (ICs) [Kilby 1958]. At this very early stage, **logic design** and **physical design** of these ICs were mainly created by hand in parallel. Logic design constructed out of wired circuit boards that mimic the physical design of the IC was built to simulate and verify whether the IC will function as intended before fabrication. The ACM and IEEE cosponsored the **Design Automation Conference** (DAC) debut in 1964 in a joint effort to automate and speed up the design process [DAC 2008]. However, it was not until the mid-1970s when mainframe computers and minicomputers were, respectively, introduced by IBM and Digital Equipment Corporation (DEC) that design automation became more feasible.

During this period, EDA research and development was typically internal to large corporations such as Bell Labs, Hewlett Packard, IBM, Intel, and Tektronix. The first critical milestones in EDA came in the form of programs for **circuit simulation** and **layout verification**. Various proprietary simulation languages and device models were proposed. The SPICE models were used in circuit simulation (*commonly referred* to as **SPICE simulation** now) to verify whether the then so-called logic design specified at the transistor level (called **transistor-level**

design) will behave the same as the functional specifications. This removes the need to build wired circuit boards. At the same time, layout verification tools that took SPICE models as inputs were developed to check whether the physical design would meet **layout design rules** and then tape out the physical design in the ***graphics data system II*** (GDSII) format introduced by Calma in the mid-1970s.

Although circuit simulation and layout verification ensure that the logic design and physical design will function correctly as expected, they are merely **verification tools**; **design automation tools** are needed to speed up the design process. This requires **logic simulation** tools for logic design at the gate level (rather than at the transistor level) and ***place and route*** (P&R) tools that operate at the physical level to automatically generate the physical design. The **Tegas logic simulator** that uses the ***Tegas description language*** (TDL) was the first logic simulator that came to widespread use until the mid-1990s, when industry began adopting the two IEEE developed ***hardware description language*** (HDL) standards: **Verilog** [IEEE 1463-2001] and **VHDL** [IEEE 1076-2002]. The first graphical software to assist in the physical design of the IC in the late 1960s and early 1970s was introduced by companies like Calma and Applicon. The first automatic *place and route* tools were subsequently introduced in the mid-1970s. Proprietary schematic capture and waveform display software to assist in the logic design of the IC was also spurring the marketplace.

Although much of the early EDA research and development was done in corporations in the 1960s and 1970s, top universities including Stanford, the University of California at Berkeley, Carnegie Mellon, and California Institute of Technology had quietly established large ***computer-aided design*** (CAD) groups to conduct research spreading from process/device simulation and modeling [Dutton 1993; Plummer 2000] to logic synthesis [Brayton 1984; De Micheli 1994; Devadas 1994] and ***analog and mixed signal*** (AMS) design and synthesis [Ochetta 1994] to silicon compilation and physical design automation [Mead 1980]. This also marks the timeframe in which EDA began as an industry with companies like Daisy Systems, Mentor Graphics [Mentor 2008], and Valid Logic Systems (acquired by Cadence Design Systems [Cadence 2008]) in the early 1980s. Another major milestone for academic-based EDA research and development was the formation of the ***Metal Oxide Semiconductor Implementation Service*** (MOSIS) in the early 1980s [MOSIS 2008].

Since those early years, EDA has continued to not only provide support and new capabilities for electronic system design but also solve many problems faced in both design and testing of electronic systems. For example, how does one test an IC with more than one billion transistors to ensure with a high probability that all transistors are fault-free? ***Design for testability*** (DFT) and ***automatic test pattern generation*** (ATPG) tools have provided EDA solutions. Another example is illustrated in Figure 1.1 for mask making during deep-submicron IC fabrication. In this example, the lithographic process is used to create rectangular patterns to form the various components of transistors and their interconnections. However, sub-wavelength components in lithography cause problems in that the intended shapes become irregular as shown in

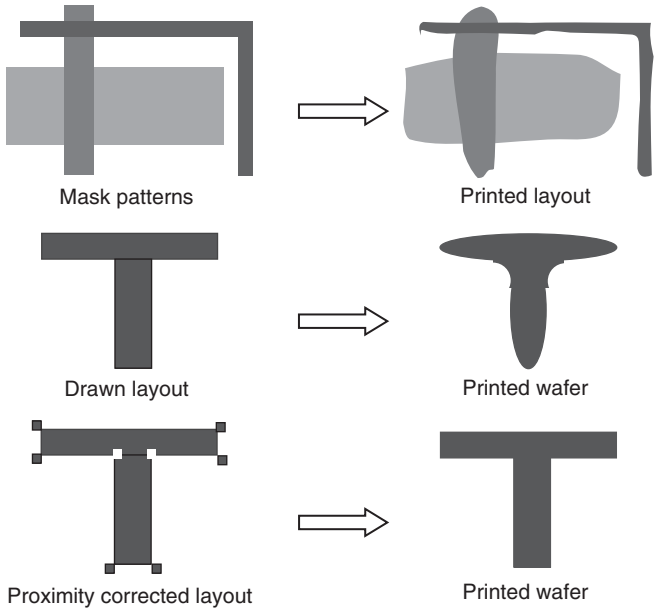

FIGURE 1.1

Sub-wavelength lithography problem and EDA solution.

Figure 1.1. This problem posed a serious obstacle to advances in technology in terms of reducing feature size, also referred to as shrinking design rules, which in turn increases the number of transistors that can be incorporated in an IC. However, EDA has provided the solution through ***optical proximity correction*** (OPC) of the layout to compensate for rounding off feature corners.

1.1.2 VLSI design flow and typical EDA flow

When we think of current EDA features and capabilities, we generally think of synthesis of *hardware description languages* (HDLs) to standard cell–based ASICs or to the configuration data to be downloaded into FPGAs. As part of the synthesis process, EDA also encompasses design audits, technology mapping, and physical design (including floorplanning, placement, routing, and design rule checking) in the intended implementation medium, be that ASIC, FPGA, PCB, or any other media used to implement electronic systems. In addition, EDA comprises logic and timing simulation and timing analysis programs for design verification of both pre-synthesis and post-synthesis designs. Finally, there is also a wealth of EDA software targeting manufacturing test, including testability analysis, *automatic test pattern generation* (ATPG), fault simulation, *design for testability* (DFT), logic/memory ***built-in self-test*** (BIST), and test compression.

In general, EDA algorithms, techniques, and software can be partitioned into three distinct but broad categories that include logic design automation, verification and test, and physical design automation. Although logic and physical design automation are somewhat disjointed in that logic design automation is performed before physical design automation, the various components and aspects of the verification and test category are dispersed within both logic and physical design automation processes. Furthermore, verification software is usually the first EDA tool used in the overall design method for simulation of the initial design developed for the intended circuit or system.

The two principal HDLs currently used include ***very high-speed integrated circuits*** (VHSIC) ***hardware description language*** (VHDL) [IEEE 1076-2002] and Verilog ***hardware description languages*** [IEEE 1463-2001]. VHDL originally targeted gate level through system-level design and verification. Verilog, on the other hand, originally targeted the design of ASICs down to the transistor level of design, but not the physical design. Since their introduction in the late 1980s, these two HDLs have expanded to cover a larger portion of the design hierarchy illustrated in Figure 1.2 to the extent that they both cover approximately the same range of the design hierarchy. These HDLs owe their success in current design methods to the introduction of synthesis approaches and software in the mid- to late 1980s. As a result, synthesis capabilities enabled VHDL and Verilog to become the design capture medium as opposed to "an additional step" in the design process.

There are many benefits of high-level HDLs such as VHDL and Verilog when used in conjunction with synthesis capabilities. They facilitate early design verification through high-level simulation, as well as the evaluation of alternate architectures for optimizing system cost and performance. These high-level simulations in turn provide baseline testing of lower level design representations such as gate-level implementations. With synthesis, top-down design methods are realized, with the high-level HDLs being the design capture medium independent of the implementation media (for example, ASIC *versus* FPGA).

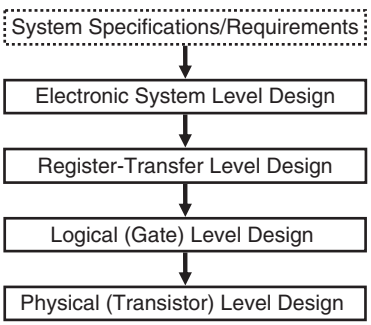

FIGURE 1.2

Design hierarchy.

This not only reduces design development time and cost but also reduces the risk to a project because of design errors. In addition, this provides the ability to manage and develop complex electronic systems and provides the basis for hardware/software co-design. As a result, ***electronic system level*** (ESL) design includes partitioning the system into hardware and software and the co-design and co-simulation of the hardware and software components. The ESL design also includes cost estimation and design-space exploration for the target system to make informed design decisions early in the design process.

The basic domains of most high-level HDLs include structural, behavioral, and RTL hierarchical descriptions of a circuit or system. The structural domain is a description of components and their interconnections and is often referred to as a **netlist**. The behavioral domain includes high-level algorithmic descriptions of the behavior of a circuit or system from the standpoint of output responses to a sequence of input stimuli. Behavioral descriptions are typically at such a high level that they cannot be directly synthesized by EDA software. The RTL domain, on the other hand, represents the clock cycle by clock cycle data flow of the circuit at the register level and can be synthesized by EDA software. Therefore, the design process often implies a manual translation step from behavioral to RTL with baseline testing to verify proper operation of the RTL design as illustrated in the example design flow for an *integrated circuit* (IC) in Figure 1.3. It should be noted that the behavioral domain is contained in the ESL design blocks in Figures 1.2 and 1.3. If the behavior of the ESL design is described in C, C++, **SystemC** [SystemC 2008], **SystemVerilog** [SystemVerilog 2008], or a mixture of these languages, modern verification

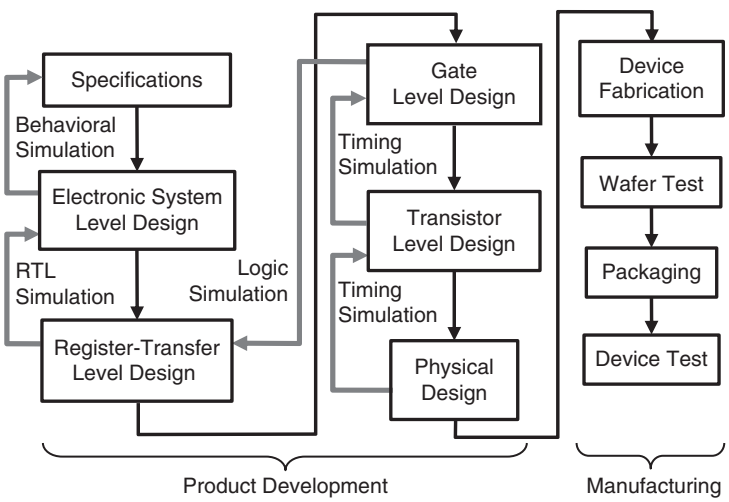

FIGURE 1.3

IC design and verification flow.

and simulation tools can either convert the language to VHDL or Verilog or directly accept the language constructs.

Although the simulation of behavioral and RTL descriptions is an essential EDA feature, most EDA encompasses the design and verification flow from the point of RTL design onward. This includes synthesis to a technology-independent gate-level implementation of the circuit followed by technology mapping to a specific implementation media such as a standard–cell–based ASIC design which in turn represents the transistor-level design in the IC design flow of Figure 1.3. Physical design then completes technology-specific partitioning, floorplanning, placement, and routing for the design. As the design flow progresses through various stages in the synthesis and physical design processes, regression testing is performed to ensure that the synthesized implementation performs the correct functionality of the intended design at the required system clock frequency. This requires additional simulation steps, as indicated in Figure 1.3, with each simulation step providing a more accurate representation of the manufactured implementation of the final circuit or system.

A number of points in the design process impact the testability and, ultimately, the manufacturing cost of an electronic component or system. These include consideration of DFT and BIST, as well as the development of test stimuli and expected good-circuit output responses used to test each manufactured product [Bushnell 2000; Stroud 2002; Jha 2003; Wang 2006, 2007]. For example, the actual manufacturing test steps are illustrated in the IC design flow of Figure 1.3 as wafer test and packaged-device test.

Physical design is one of the most important design steps because of its critical impact on area, power, and performance of the final electronic circuit or system. This is because layout (component placement) and routing are integral parts of any implementation media such as ICs, FPGAs, and PCBs. Therefore, physical design was one of the first areas of focus on EDA research and development. The result has been numerous approaches and algorithms for physical design automation [Preas 1988; Gerez 1998; Sait 1999; Sherwani 1999; Scheffer 2006a, 2006b]. The basic flow of the physical design process is illustrated in Figure 1.4.

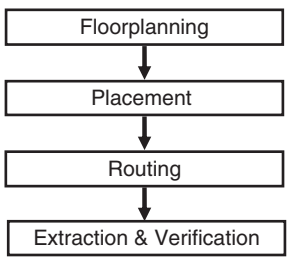

FIGURE 1.4

Typical physical design flow.

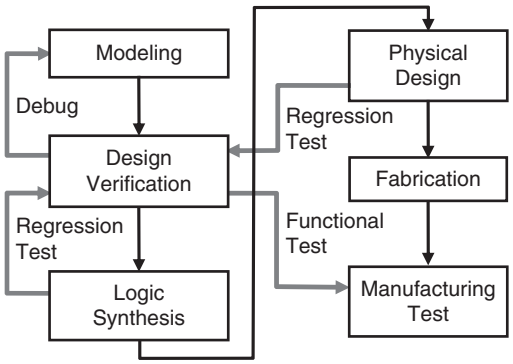

FIGURE 1.5

Typical EDA flow.

An alternate view of a typical EDA flow is illustrated in Figure 1.5, which begins with modeling and design verification. This implies a recursive process with debugging until the resultant models reach a level of detail that can be processed by logic synthesis. As a result of current EDA capabilities, the design process is highly automated from this point. This is particularly true for physical design but to a lesser extent for manufacture test and test development. Therefore, the functional stimuli developed for the device and output responses obtained from simulations during design verification typically form the basis for functional tests used during manufacturing test.

Many design space issues may be critical to a given project, and many of these issues can require tradeoffs. For example, area and performance are two of the most frequently addressed tradeoffs to be considered in the design space. Area considerations include chip area, how many ICs on a PCB, and how much board space will be required for a given implementation. Performance considerations, on the other hand, often require additional area to meet the speed requirements for the system. For example, the much faster carry-look-ahead adder requires significantly more area than the simple but slow ripple-carry adder. Therefore, EDA synthesis options include features and capabilities to select and control area and performance optimization for the final design. However, additional design space issues such as power consumption and power integrity must be considered. Inevitably, cost and anticipated volume of the final product are also key ingredients in making design decisions. Another is the design time to meet the market window and development cost goals. The potential risk to the project in obtaining a working, cost-effective product on schedule is an extremely important design issue that also hinges on reuse of resources (using the same core in different modes of operation, for example) and the target implementation media and its associated technology limits. Less frequently addressed, but equally important, design considerations include designer experience and EDA software availability and capabilities.

1.1.3 **Typical EDA implementation examples**

To better appreciate the current state of EDA in modern electronic system design, it is worth taking a brief look at the state and progression of EDA since the mid-1970s and some of the subsequent milestones. At that time, ASICs were typically hand-designed by creating a hand-edited netlist of standard cells and their interconnections. This netlist was usually debugged and verified using unit-delay logic simulation. Once functional verification was complete, the netlist was used as input to ***computer-aided design*** (CAD) tools for placement of the standard cells and routing of their interconnections. At that time, physical design was semiautomated with considerable intervention by physical design engineers to integrate ***input/output*** (I/O) buffers and networks for clocks, power, and ground connections. Timing simulation CAD tools were available for verification of the design for both pre-physical and post-physical design using estimated and extracted capacitance values, respectively. It is interesting to note that resistance was not considered in timing simulations until the mid-1980s, when design rules reached the point that sheet resistance became a dominant delay factor.

Graphical schematic entry CAD tools were available for PCBs for design capture, layout, and routing. However, schematic capture tools for ASIC design were generally not available until the early 1980s and did not significantly improve the design process other than providing a nicely drawn schematic of the design from which the standard-cell netlist was automatically generated. The actual digital logic continued to be hand-designed by use of state diagrams, state tables, Karnaugh maps, and a few simple CAD tools for logic minimization. This limited the complexity of ASICs in terms of the number of gates or transistors that could be correctly designed and verified by a typical designer. In the early 1980s, an ASIC with more than 100,000 transistors was considered to be near the upper limit for a single designer. By the late 1980s, the limit was significantly increased as a result of multi-designer teams working on a single IC and as a result of advances in EDA capabilities, particularly in the area of logic synthesis. Currently, the largest ICs exceed 1 billion transistors [Naffziger 2006; Stackhouse 2008].

One of the earliest approaches to EDA in terms of combinational logic synthesis was for implementing ***programmable logic arrays*** (PLAs) in ***very large-scale integrated*** (VLSI) circuits in the late 1970s [Mead 1980]. Any combinational logic function can be expressed as Boolean logic equations, ***sum-of-products*** (SOP) or ***product-of-sums*** (POS) expressions, and truth tables or Karnaugh maps. There are other representations, but these three are illustrated for the example circuit in Figure 1.6 and are important for understanding the implementation of PLAs and other programmable logic. We can program the truth table onto a ***read-only memory*** (ROM) with eight words and two bits/word and then use the ROM address lines as the three input signals (A, B, C) and the ROM outputs as the output signals (X, Y). Similarly, we can

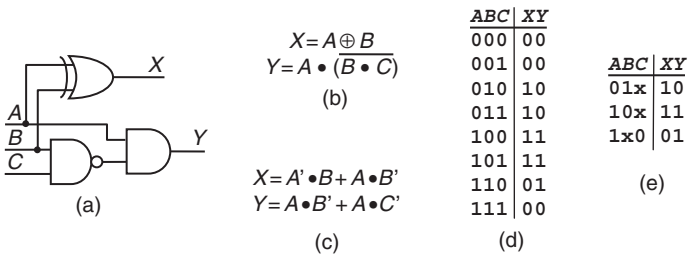

FIGURE 1.6

Combinational logic implementation example: (a) Logic diagram. (b) Logic equations. (c) SOP expressions. (d) Truth table. (e) Connection array.

also write the truth table into a ***random-access memory*** (RAM) with eight words and two bits/word and then disable the write enable to the RAM and use the address lines as the inputs. Note that this is the same thing as the ROM, except that we can reprogram the logic function by rewriting the RAM; this also forms the basis for combinational logic implementations in FPGAs.

Another option for implementing a truth table is the PLA. In the connection array in Figure 1.6e, only three product terms produce logic 1 at the output signals. The PLA allows implementing only those three product terms and not the other five, which is much smaller than either the ROM or RAM implementation. Any SOP can be implemented as a 2-level AND-OR or NAND-NAND logic function. Any SOP can also be implemented as a 2-level NOR-NOR logic function if we invert the inputs and the output as illustrated in Figure 1.7a. Note that AB' is a shared product term and allows us to share an AND gate in the gate-level implementation. PLAs take advantage of this NOR-NOR implementation of logic equations and the large fan-in limit of ***N-channel metal oxide semiconductor*** (NMOS) NOR gates, as illustrated in Figure 1.7b, for PLA implementation of the example circuit. Note that there is a direct relationship between the crosspoints in the PLA and the AND-OR connection array in Figure 1.6e. A logic 1 (0) in the input columns of the connection array corresponds to a crosspoint between the bit (bit-bar) line and the AND line, also called the product term line. A logic 1 in the output columns corresponds to a crosspoint between the AND line and the OR line, also called the output line. Therefore, the physical design of the PLA is obtained directly from the connection array. It is also important to note that a connection array is derived from a minimized truth table but is not equivalent to a truth table as can be seen by considering the X output for the last two entries in the connection array.

PLAs are of historical importance because they not only led to the development of ***programmable logic devices*** (PLDs) including FPGAs, but they also led to the further development of CAD tools for logic minimization and automated physical design, because the physical design could be obtained directly

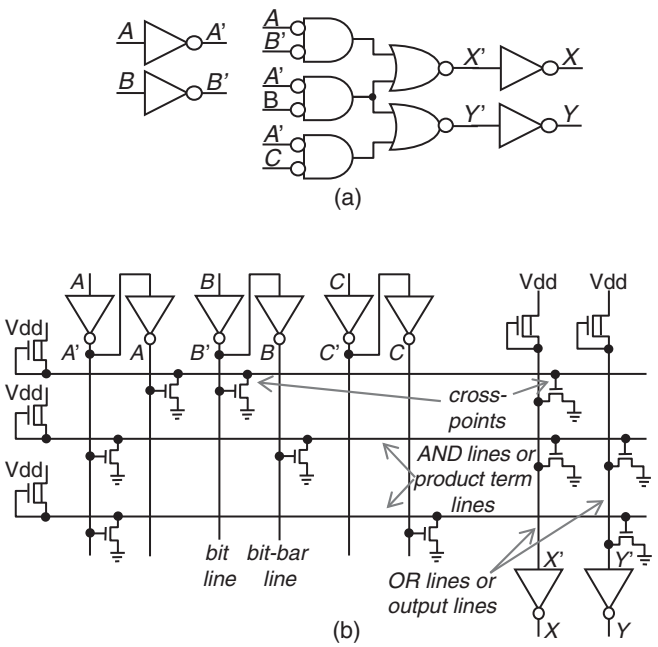

FIGURE 1.7

PLA implementation example: (a) 2-level NOR-NOR implementation. (b) PLA implementation.

from the connection array. For example, the outputs of early logic minimization tools, like **Espresso**, were usually in terms of a connection array for PLA implementations. These minimization CAD tools were the predecessors to high-level synthesis and many of the current physical design tools.

The quest for the ability to synthesize high-level descriptions of hardware began in earnest in the mid-1980s. One of the first successful synthesis tools, called **CONES**, was capable of synthesizing RTL models written in C to either standard–cell–based ASICs or to PLD-based PCBs and was used extensively internal to Bell Labs [Stroud 1986]. This timeframe also corresponds to the formation of EDA companies dedicated to synthesis, such as Synopsys [Synopsys 2008], as well as the introduction of VHDL and Verilog, which have been used extensively throughout industry and academia since that time.

The successful introduction of functional modeling into the VLSI design community was due, in part, to the development of logic synthesis tools and systems. Modeling a system at the functional level and simulating the resultant models had been previously used with simulation languages such as ADA [Ledgard 1983] to obtain a simulation environment that emulates the system to be designed. These simulation environments provided a platform from which

the design of the various modules required for implementation of the system could proceed independently with the ability to regression test the detailed designs at various points in the design process. In addition, this model-based simulation environment could ensure a degree of coherence in the system long before hardware components were available for integration and testing in the final system. Despite the advantages of this approach, it did not receive widespread attention until logic synthesis tools and systems were developed to synthesize the detailed gate-level or transistor-level design from the functional description of the circuit. As a result, the design entry point for the designer became the functional model rather than gate-level or transistor-level descriptions of the VLSI device. Once removed from time-consuming and often error-prone gate-level or transistor-level design, designers had the ability to manage higher levels of design complexity. In addition, the speed at which the logic synthesis systems can implement the gate-level or transistor-level design significantly reduced the overall design interval.

1.1.4 Problems and challenges

With exponentially increasing transistor counts in ICs brought on by smaller feature sizes, there are also demands for increased bandwidth and functionality with lower cost and shorter time-to-market. The main challenges in EDA are well documented in the ***International Technology Roadmap for Semiconductors*** (ITRS) [SIA 2005, 2006]. One of the major challenges is that of design productivity in the face of large design teams and diversity in terms of heterogeneous components in system-level SOC integration. This includes design specification and verification at the system level and embedded system software co-design and ***analog/mixed-signal*** (AMS) circuitry in the hierarchy along with other system objectives such as fault or defect tolerance. To accurately verify the design before fabrication, the challenges include the ability to accurately extract physical design information to efficiently model and simulate full-chip interconnect delay, noise, and power consumption.

A primary trend in testing is an emphasis to provide information for ***failure mode analysis*** (FMA) to obtain yield enhancement. Another trend is **reliability screening** in which testing targets weak transistors and the location of non-uniformities in addition to hard defects; this includes detecting the symptoms and effects of line width variations, finite dopant distributions, and systemic process defects. Finally, there is a need to avoid potential yield losses as a result of tester inaccuracies, power droop, overly aggressive statistical postprocessing, defects occurring in test circuitry such as BIST, overtesting delay faults on non-functional paths, mechanical damages resulting from the testing process, and faulty repairs of repairable circuits, to name a few.

In the remaining sections of this chapter, we give a more detailed overview of the three fundamental components of EDA: logic design automation, testing, and physical design automation.

1.2 LOGIC DESIGN AUTOMATION

Logic design automation refers to all modeling, synthesis, and verification steps that model a design specification of an electronic system at an ***electronic system level*** (ESL), verify the ESL design, and then compile or translate the ESL representation of the design into an RTL or gate-level representation. The design hierarchy illustrated in Figure 1.2 has described all the levels of abstraction down to physical design, a step solely responsible for **physical design automation**. In the design hierarchy, a higher level description has fewer implementation details but more explicit functional information than a lower level description. The design process illustrated in Figure 1.3 and the EDA process flow illustrated in Figure 1.5 essentially represent the transforming of a higher level description of a design to a lower level description. The following subsections discuss the various steps associated with logic design automation, which include modeling, design verification, and logic synthesis.

1.2.1 Modeling

Starting from a design specification, a behavioral description of a system is developed in ESL languages, such as SystemC, SystemVerilog, VHDL, Verilog, and C/C++, and simulated to determine whether it meets the system requirements and specifications. The objective is to describe the behavior of the intended system in a number of behavioral models that can be simulated for design verification and then translated to RTL for logic synthesis. In addition, behavioral models representing existing or new hardware that interfaces with the system may be developed to create a simulation environment in which the behavioral models for the system to be designed can be verified in the presence of existing hardware. Alternately, such hardware can be directly embedded in an emulator for design verification [Scheffer 2006a, 2006b]. During design verification, a number of iterations of modeling and simulation steps are usually required to obtain a working behavioral description for the intended system to be implemented.

Once the requirements for the system to be designed have been defined, the designer faces the task of describing the functionality in models. The goal is to write models such that they can be simulated to verify the correct operation of the design and be synthesized to obtain the logic to implement the function. For a complex system, SOC, or VLSI device, this usually requires that the functionality be partitioned into multiple blocks that are more easily managed in terms of complexity. One or more functional models may represent each of these blocks. There are different ways to proceed to achieve the goal of functional models being synthesizable. One approach is to ignore the requirement that the models be synthesizable and to describe the function at as high a level as can be handled by the designer and by the simulation tools. These high-level

descriptions can then be verified in the simulation environment to obtain correct functionality with respect to the system requirements and specifications. At that point, the high-level models can be partitioned and written as functional models of a form suitable for synthesis. In this case, the simulation environment is first used to verify the high-level models and later used as a baseline for **regression testing** of the synthesizable models to ensure that correct functionality has been maintained such that the synthesized design still meets the system requirements and specifications. At the other end of the spectrum, an alternate approach is to perform the partitioning and generation of the functional models at a level of detail compatible with the synthesis tools. Once sufficient design verification has been achieved, the design can move directly to the logic synthesis step.

Modeling the circuit to be simulated and synthesized is, in some respects, simply a matter of translating the system requirements and specifications to the ESL or HDL description. The requirements and specifications for the system or circuit to be modeled are sometimes quite specific. On the other hand, on the basis of inputs and outputs from other blocks necessary to construct the complete system, arbitrary values may be chosen by the designer.

1.2.2 Design verification

Design verification is the most important aspect of the product development process illustrated in Figures 1.3 and 1.5, consuming as much as 80% of the total product development time. The intent is to verify that the design meets the system requirements and specifications. Approaches to design verification consist of (1) **logic simulation/emulation and circuit simulation**, in which detailed functionality and timing of the design are checked by means of simulation or emulation; (2) **functional verification**, in which functional models describing the functionality of the design are developed to check against the behavioral specification of the design without detailed timing simulation; and (3) **formal verification**, in which the functionality is checked against a "golden" model. Formal verification further includes **property checking** (or **model checking**), in which the property of the design is checked against some presumed "properties" specified in the functional or behavioral model (*e.g.*, a finite-state machine should not enter a certain state), and **equivalence checking**, in which the functionality is checked against a "golden" model [Wile 2005]. Although equivalence checking can be used to verify the synthesis results in the lower levels of the EDA flow (denoted "regression test" in Figure 1.5), the original design capture requires property checking.

Simulation-based techniques are the most popular approach to verification, even though these are time-consuming and may be incomplete in finding design errors. Logic simulation is used throughout every stage of logic design automation, whereas circuit simulation is used after physical design. The most commonly used logic simulation techniques are compiled-code simulation and

event-driven simulation [Wang 2006]. The former is most effective for cycle-based two-valued simulation; the latter is capable of handling various gate and wire delay models. Although versatile and low in cost, logic simulation is too slow for complex SOC designs or hardware/software co-simulation applications. For more accurate timing information and dynamic behavior analysis, device-level circuit simulation is used. However, limited by the computation complexity, circuit simulation is, in general, only applied to critical paths, cell library components, and memory analysis.

For simulation, usually, a number of different simulation techniques are used, including high-level simulation through a combination of behavioral modeling and testbenches. Testbenches are behavioral models that emulate the surrounding system environment to provide input stimuli to the design under test and process the output responses during simulation. RTL models of the detailed design are then developed and verified with the same testbenches that were used for verification of the architectural design, in addition to testbenches that target design errors in the RTL description of the design. With sufficient design verification at this point in the design process, functional vectors can be captured in the RTL simulation and then used for subsequent simulations (**regression testing**) of the more detailed levels of design, including synthesized gate-level design, transistor-level design, and physical design. These latter levels of design abstraction (gate, transistor, and physical design) provide the ability to perform additional design verification through logic, switch-level, and timing simulations. These three levels of design abstraction also provide the basis for fault models that can be used to evaluate the effectiveness of manufacturing tests.

The design verification step establishes the quality of the design and ensures the success of the project by uncovering potential errors in both the design and the architecture of the system. The objective of design verification is to simulate all functions as exhaustively as possible while carefully investigating any possibly erroneous behavior. From a designer's standpoint, this step deserves the most time and attention. One of the benefits of high-level HDLs and logic synthesis is to allow the designer to devote more time and concentration to design verification. Because much less effort is required to obtain models that can be simulated but not synthesized, design verification can begin earlier in the design process, which also allows more time for considering optimal solutions to problems found in the design or system. Furthermore, debugging a high-level model is much easier and faster than debugging a lower level description, such as a gate-level netlist.

An attractive attribute of the use of functional models for design verification (often called **functional verification**) is that HDL simulation of a collection of models is much faster than simulations of the gate-level descriptions that would correspond to those models. Although functional verification only verifies cycle accuracy (rather than timing accuracy), the time required to perform the design verification process is reduced with faster simulation. In addition, a more

thorough verification of the design can be performed, which in turn improves the quality of the design and the probability of the success of the project as a whole. Furthermore, because these models are smaller and more functional than netlists describing the gate-level design, the detection, location, and correction of design errors are easier and faster. The reduced memory requirements and increased speed of simulation with functional models enable simulation of much larger circuits, making it practical to simulate and verify a complete hardware system to be constructed. As a result, the reduced probability of design changes resulting from errors found during system integration can be factored into the overall design schedule to meet shorter market windows. Therefore, design verification is economically significant, because it has a definite impact on time-to-market. Many tools are available to assist in the design verification process, including simulation tools, hardware emulation, and formal verification methods. It is interesting to note that many design verification techniques are borrowed from test technology, because verifying a design is similar to testing a physical product. Furthermore, the test stimuli developed for design verification of the RTL, logical, and physical levels of abstraction are often used, in conjunction with the associated output responses obtained from simulation, for functional tests during the manufacturing process.

Changes in system requirements or specifications late in the design cycle jeopardize the schedule and the quality of the design. Late changes to a design represent one of the two most significant risks to the overall project, the other being insufficient design verification. The quality of the design verification process depends on the ability of the testbenches, functional vectors, and the designers who analyze the simulated responses to detect design errors. Therefore, any inconsistency observed during the simulations at the various levels of design abstraction should be carefully studied to determine whether potential design errors to be corrected exist before design verification continues.

Emulation-based verification by use of FPGAs provides an attractive alternative to simulation-based verification as the gap between logic simulation capacity and design complexity continues growing. Before the introduction of FPGAs in the 1980s, ASICs were often verified by construction of a breadboard by use of ***small-scale integration*** (SSI) and ***medium-scale integration*** (MSI) devices on a wire-wrap board. This became impractical as the complexity and scale of ASICs moved into the VLSI realm. As a result, FPGAs became the primary hardware for emulation-based verification. Although these approaches are costly and may not be easy to use, they improve verification time by two to three orders of magnitude compared with software simulation. Alternately, a **reconfigurable emulation system** (or **reconfigurable emulator**) that automatically partitions and maps a design onto multiple FPGAs can be used to avoid building a prototype board and can be reused for various designs [Scheffer 2006a, 2006b].

Formal verification techniques are a relatively new paradigm for equivalence checking. Instead of input stimuli, these techniques perform exhaustive

proof through rigorous logical reasoning. The primary approaches used for formal verification include ***binary decision diagrams*** (BDDs) and **Boolean *satisfiability*** (SAT) [Velev 2001]. These approaches, along with other algorithms specific to EDA applications, are extensively discussed in Chapter 4. The BDD approach successively applies **Shannon expansion** on all variables of a combinational logic function until either the constant function "0" or "1" is reached. This is applied to both the captured design and the synthesized implementation and compared to determine their equivalence. Although BDDs give a compact representation for Boolean functions in polynomial time for many Boolean operations, the size of BDD grows exponentially with input size, which is usually limited to 100 to 200 inputs. On the other hand, SAT techniques have been very successful in recent years in the verification area with the ability to handle million-gate designs and both combinational and sequential designs.

1.2.3 Logic synthesis

The principal goal of logic synthesis is to translate designs from the behavioral domain to the structural domain. This includes **high-level synthesis**, in which system behavior and/or algorithms are transformed into functional blocks such as processors, RAMs, ***arithmetic logic units*** (ALUs), etc. Another type of synthesis takes place at the ***register-transfer level*** (RTL), where Boolean expressions or RTL descriptions in VHDL or Verilog are transformed to logic gate networks.

Logic synthesis is initially technology independent where RTL descriptions are parsed for control/data flow analysis. Initial gate-level implementations are in terms of generic gate implementations (such as AND, OR, and NOT) with no relationship to any specific technology. As a result, the structure at this point is technology independent and can be ultimately implemented in any technology by means of **technology mapping** into specific libraries of cells as illustrated in Figure 1.8. Before technology mapping, however, a number of technology-independent optimizations can be made to the gate-level implementation by basic logic restructuring with techniques such as the **Quine-McCluskey method** for two-level logic optimization [McCluskey 1986] or methods for multilevel logic optimization that may be more appropriate for standard cell–based designs [Brayton 1984; De Michele 1994; Devadas 1994]. Once technology mapping has been performed, additional optimizations are performed such as for timing and power. This may be followed by insertion of logic to support ***design for testability*** (DFT) features and capabilities. However, it should be noted that once technology mapping is performed, most subsequent synthesis and optimizations fall into the domain of physical design automation.

Regression testing of the synthesized gate-level description ensures that there are no problems in the design that are not apparent from the functional model simulation, such as feedback loops that cannot be initialized.

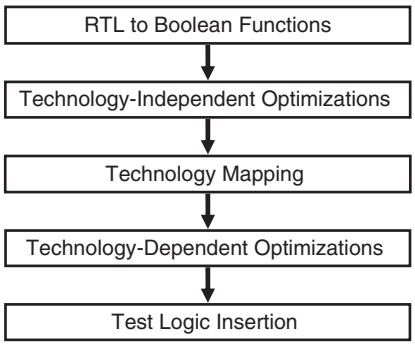

FIGURE 1.8

Logic synthesis flow.

This additional effort may seem to be avoidable with proper consideration given to undefined logic values in the function model. However, developing a functional model that initializes the same as a gate-level description requires considerable effort and knowledge of the gate-level structure of a circuit. Hence, the functional model may not behave exactly the same way as the synthesized circuit. Designers must be careful to avoid constructs in HDLs that allow the model to self-initialize but cannot be reproduced in the final circuit by the synthesis system. Therefore, regression testing is necessary and, fortunately, undefined logic values are relatively easy to trace to their source to determine the root cause. Good coding and reusability styles, as well as user-defined coding style rules, play an important role in avoiding many of the synthesis errors [Keating 1999].

1.3 TEST AUTOMATION

Advances in manufacturing process technology have also led to very complex designs. As a result, it has become a requirement that *design-for-testability* (DFT) features be incorporated in the ***register-transfer level*** (RTL) or gate-level design before physical design to ensure the quality of the fabricated devices. In fact, the traditional VLSI development process illustrated in Figure 1.3 involves some form of testing at each stage, including design verification. Once verified, the VLSI design then goes to fabrication and, at the same time, test engineers develop a test procedure based on the design specification and **fault models** associated with the implementation technology. Because the resulting product quality is in general unsatisfactory, modern VLSI test development planning tends to start when the RTL design is near completion. This test development plan defines what **test requirements** the product must meet, often in terms of **defect level** and **manufacturing yield**, test cost, and whether it is necessary to perform self-test and diagnosis. Because the test

requirements mostly target manufacturing defects rather than **soft errors**, which would require **online fault detection and correction** [Wang 2007], one need is to decide what fault models should be considered.

The test development process now consists of (1) defining the targeted fault models for defect level and manufacturing yield considerations, (2) deciding what types of DFT features should be incorporated in the RTL design to meet the test requirements, (3) generating and fault-grading test patterns to calculate the final fault coverage, and (4) conducting manufacturing test to screen bad chips from shipping to customers and performing ***failure mode analysis*** (FMA) when the chips do not achieve desired defect level or yield requirements.

1.3.1 Fault models

A **defect** is a manufacturing flaw or physical imperfection that may lead to a **fault**, a fault can cause a circuit **error**, and a circuit error can result in a **failure** of the device or system. Because of the diversity of defects, it is difficult to generate tests for real defects. Fault models are necessary for generating and evaluating test patterns. Generally, a good fault model should satisfy two criteria: (1) it should accurately reflect the behavior of defects and (2) it should be computationally efficient in terms of time required for fault simulation and test generation. Many fault models have been proposed but, unfortunately, no single fault model accurately reflects the behavior of all possible defects that can occur. As a result, a combination of different fault models is often used in the generation and evaluation of test patterns. Some well-known and commonly used fault models for general sequential logic [Bushnell 2000; Wang 2006] include the following:

1. **Gate-level stuck-at fault model**: The stuck-at fault is a logical fault model that has been used successfully for decades. A stuck-at fault transforms the correct value on the faulty signal line to appear to be stuck-at a constant logic value, either logic 0 or 1, referred to as ***stuck-at-0*** (SA0) or ***stuck-at-1*** (SA1), respectively. This model is commonly referred to as the **line stuck-at fault model** where any line can be SA0 or SA1, and also referred to as the gate-level stuck-at fault model where any input or output of any gate can be SA0 or SA1.
2. **Transistor-level stuck fault model**: At the switch level, a transistor can be **stuck-off** or **stuck-on**, also referred to as **stuck-open** or **stuck-short**, respectively. The line stuck-at fault model cannot accurately reflect the behavior of stuck-off and stuck-on transistor faults in ***complementary metal oxide semiconductor*** (CMOS) logic circuits because of the multiple transistors used to construct CMOS logic gates. A stuck-open transistor fault in a CMOS combinational logic gate can cause the gate to behave like a level-sensitive latch. Thus, a stuck-open fault in a CMOS combinational circuit requires a sequence of two vectors for

detection instead of a single test vector for a stuck-at fault. Stuck-short faults, on the other hand, can produce a conducting path between power (V_{DD}) and ground (V_{SS}) and may be detected by monitoring the power supply current during steady state, referred to as I_{DDQ}. This technique of monitoring the steady state power supply current to detect transistor stuck-short faults is called **I_{DDQ} testing** [Bushnell 2000; Wang 2007].

3. **Bridging fault models**: Defects can also include opens and shorts in the wires that interconnect the transistors that form the circuit. Opens tend to behave like line stuck-at faults. However, a **resistive open** does not behave the same as a transistor or line stuck-at fault, but instead affects the propagation delay of the signal path. A short between two wires is commonly referred to as a **bridging fault**. The case of a wire being shorted to V_{DD} or V_{SS} is equivalent to the line stuck-at fault model. However, when two signal wires are shorted together, bridging fault models are needed; the three most commonly used bridging fault models are illustrated in Figure 1.9. The first bridging fault model proposed was the **wired-AND/wired-OR** bridging fault model, which was originally developed for bipolar technology and does not accurately reflect the behavior of bridging faults typically found in CMOS devices. Therefore, the **dominant bridging fault** model was proposed for CMOS where one driver is assumed to dominate the logic value on the two shorted nets. However, the dominant bridging fault model does not accurately reflect the behavior of a resistive short in some cases. The most recent bridging fault model, called the **4-way** bridging fault model and also known as the **dominant-AND/dominant-OR** bridging fault model, assumes that one driver dominates the logic value of the shorted nets for one logic value only [Stroud 2002].

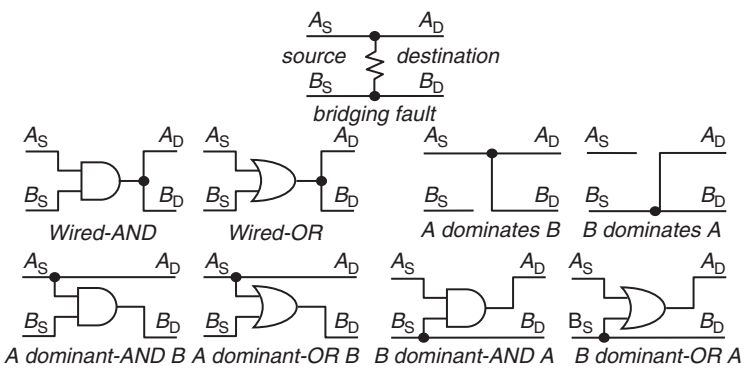

FIGURE 1.9

Bridging fault models.

4. **Delay fault models**: Resistive opens and shorts in wires and parameter variations in transistors can cause excessive delay such that the total propagation delay falls outside the specified limit. **Delay faults** have become more prevalent with decreasing feature sizes, and there are different delay fault models. In **gate-delay fault** and **transition fault** models, a delay fault occurs when the time interval taken for a transition through a single gate exceeds its specified range. The **path-delay fault** model, on the other hand, considers the cumulative propagation delay along any signal path through the circuit. The **small delay defect** model takes timing delay associated with the fault sites and propagation paths from the layout into consideration [Sato 2005; Wang 2007].

1.3.2 Design for testability

To test a given circuit, we need to control and observe logic values of internal nodes. Unfortunately, some nodes in sequential circuits can be difficult to control and observe. DFT techniques have been proposed to improve the controllability and observability of internal nodes and generally fall into one of the following three categories: (1) **ad-hoc DFT** methods, (2) **scan design**, and (3) ***built-in self-test*** (BIST). Ad-hoc methods were the first DFT technique introduced in the 1970s [Abramovici 1994]. The goal was to target only portions of the circuit that were difficult to test and to add circuitry (typically **test point** insertion) to improve the controllability and/or observability of internal nodes [Wang 2006].

Scan design was the most significant DFT technique proposed [Williams 1983]. This is because the scan design implementation process was easily automated and incorporated in the EDA flow. A scan design can be flip-flop based or latch based. The latch-based scan design is commonly referred to as ***level-sensitive scan design*** (LSSD) [Eichelberger 1978]. The basic idea to create a scan design is to reconfigure each flip-flop (*FF*) or latch in the sequential circuit to become a ***scan flip-flop*** (*SFF*) or **scan latch** (often called **scan cell**), respectively. These scan cells, as illustrated in Figure 1.10, are then connected in series to form a shift register, or **scan chain**, with direct access to a primary input (*Scan Data In*) and a primary output (*Scan Data Out*). During the shift operation (when *Scan Mode* is set to 1), the scan chain is used to shift in a test pattern from *Scan Data In* to be applied to the combinational logic. During one clock cycle of the normal system operation (when *Scan Mode* is set to 0), the test pattern is applied to the combinational logic and the output response is clocked back or captured into the scan cells. The scan chain is then used in scan mode to shift out the combinational logic output response while shifting in the next test pattern to be applied. As a result, scan design reduces the problem of testing sequential logic to that of testing combinational logic and, thereby, facilitates the use of *automatic test pattern generation* (ATPG) techniques and software developed for combinational logic.

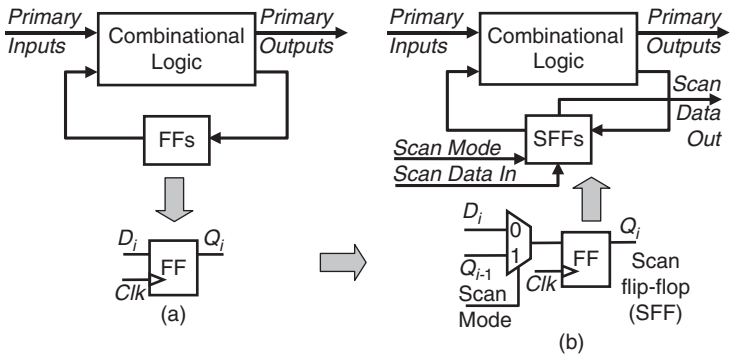

FIGURE 1.10

Transforming a sequential circuit to flip-flop-based scan design: (a) Example of a sequential circuit. (b) Example of a scan design.

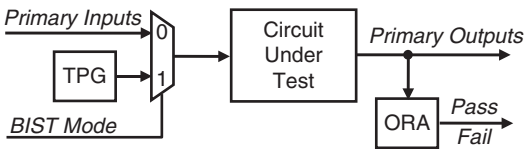

FIGURE 1.11

Simple BIST architecture.

BIST was proposed around 1980 to embed test circuitry in the device or system to perform self-test internally. As illustrated in Figure 1.11, a ***test pattern generator*** (TPG) is used to automatically supply the internally generated test patterns to the ***circuit under test*** (CUT), and an ***output response analyzer*** (ORA) is used to compact the output responses from the CUT [Stroud 2002]. Because the test circuitry resides with the CUT, BIST can be used at all levels of testing from wafer through system level testing. BIST is typically applied on the basis of the type of circuit under test. For example, scan-based BIST approaches are commonly used for general sequential logic (often called **logic BIST**); more algorithmic BIST approaches are used for regular structures such as memories (often called **memory BIST**). Because of the complexity of current VLSI devices that can include ***analog and mixed-signal*** (AMS) circuits, as well as hundreds of memories, BIST implementations are becoming an essential part of both system and test requirements [Wang 2006, 2007].

Test compression can be considered as a supplement to scan design and is commonly used to reduce the amount of test data (both input stimuli and output responses) that must be stored on the ***automatic test equipment*** (ATE) [Touba 2006]. Reduction in test data volume and test application time by 10× or more can be achieved. This is typically done by including a decompressor before the m scan chain inputs of the CUT to decompress the compressed input

stimuli and a compactor after the m scan chain outputs of the CUT to compact output responses, as illustrated in Figure 1.12. The compressed input stimulus and compacted output response are each connected to n tester channels on the ATE, where $n < m$ and n is typically at least $10\times$ smaller than m. Modern test synthesis tools can now directly incorporate these test compression features into either an RTL design or a gate-level design as will be discussed in more detail in Chapter 3.

1.3.3 **Fault simulation and test generation**

The mechanics of testing for fault simulation, as illustrated in Figure 1.13, are similar at all levels of testing, including design verification. First, a set of target faults (fault list) based on the CUT is enumerated. Often, **fault collapsing** is applied to the enumerated fault set to produce a collapsed fault set to reduce fault simulation or fault grading time. Then, input stimuli are applied to the CUT, and the output responses are compared with the expected fault-free responses to determine whether the circuit is faulty. For fault simulation, the CUT is typically synthesized down to a gate-level design (or circuit netlist).

Ensuring that sufficient design verification has been obtained is a difficult step for the designer. Although the ultimate determination is whether or not the design works in the system, fault simulation, illustrated in Figure 1.13, can provide a rough quantitative measure of the level of design verification much earlier in the design process. Fault simulation also provides valuable information

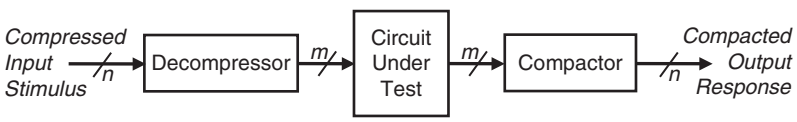

FIGURE 1.12

Test compression architecture.

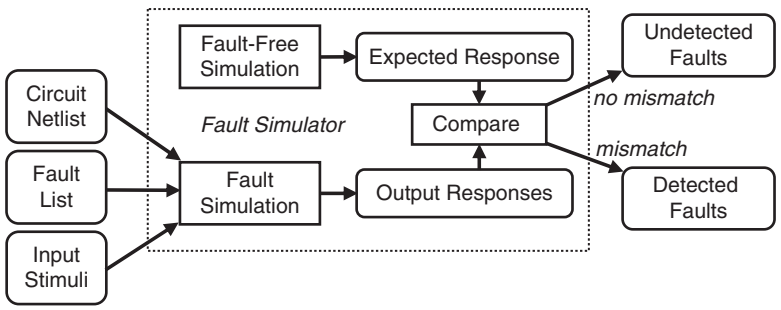

FIGURE 1.13

Fault simulation.

on portions of the design that need further design verification, because design verification vectors are often used as functional vectors (called **functional testing**) during manufacturing test.

Test development consists of selecting specific test patterns on the basis of circuit structural information and a set of **fault models**. This approach, called **structural testing**, saves test time and improves test efficiency, because the total number of test patterns is largely decreased since the test patterns target specific faults that would result from defects in the manufactured circuit. Structural testing cannot guarantee detection of all possible manufacturing defects, because the test patterns are generated on the basis of specific fault models. However, fault models provide a quantitative measure of the fault detection capabilities for a given set of test patterns for the targeted fault model; this measure is called **fault coverage** and is defined as:

$$fault\ coverage = \frac{number\ of\ detected\ faults}{total\ number\ of\ faults}$$

Any input pattern, or sequence of input patterns, that produces a different output response in a faulty circuit from that of the fault-free circuit is a test pattern, or sequence of test patterns, which will detect the fault. Therefore, the goal of ***automatic test pattern generation*** (ATPG) is to find a set of test patterns that detects all faults considered for that circuit. Because a given set of test patterns is usually capable of detecting many faults in a circuit, **fault simulation** is typically used to evaluate the fault coverage obtained by that set of test patterns. As a result, fault models are needed for fault simulation and for ATPG.

1.3.4 Manufacturing test

The tester, also referred to as the ***automatic test equipment*** (ATE), applies the functional test vectors and structural test patterns to the fabricated circuit and compares the output responses with the expected responses obtained from the design verification simulation environment for the fault-free (and hopefully, design error-free) circuit. A "faulty" circuit is now considered to be a circuit with manufacturing defects.

Some percentage of the manufactured devices, boards, and systems is expected to be faulty because of manufacturing defects. As a result, testing is required during the manufacturing process in an effort to find and eliminate those defective parts. The **yield** of a manufacturing process is defined as the percentage of acceptable parts among all parts that are fabricated:

$$yield = \frac{number\ of\ acceptable\ parts}{total\ number\ of\ parts\ fabricated}$$

A **fault** is a representation of a defect reflecting a physical condition that causes a circuit to fail to perform in a required manner. When devices or electronic systems are tested, the following two undesirable situations may occur: (1) a faulty circuit appears to be a good part passing the test, or (2) a good circuit fails the test and appears as faulty. These two outcomes are often due to a poorly designed test or the lack of DFT. As a result of the first case, even if all products pass the manufacturing test, some faulty devices will still be found in the manufactured electronic system. When these faulty circuits are returned to the manufacturer, they undergo *failure mode analysis* (FMA) or fault diagnosis for possible improvements to the manufacturing process [Wang 2006]. The ratio of field-rejected parts to all parts passing quality assurance testing is referred to as the **reject rate**, also called the **defect level**:

$$reject\ rate = \frac{number\ of\ faulty\ parts\ passing\ final\ test}{total\ number\ of\ parts\ passing\ final\ test}$$

Because of unavoidable statistical flaws in the materials and masks used to fabricate the devices, it is impossible for 100% of any particular kind of device to be defect free. Thus, the first testing performed during the manufacturing process is to test the devices fabricated on the wafer to determine which devices are defective. The chips that pass the wafer-level test are extracted and packaged. The packaged devices are retested to eliminate those devices that may have been damaged during the packaging process or put into defective packages. Additional testing is used to ensure the final quality before shipping to customers. This final testing includes measurement of parameters such as input/output timing specifications, voltage, and current. In addition, **burn-in** or **stress testing** is often performed when chips are subject to high temperature and supply voltage. The purpose of burn-in testing is to accelerate the effect of defects that could lead to failures in the early stages of operation of the device. FMA is typically used at all stages of the manufacturing test to identify improvements to processes that will result in an increase in the number of defect-free electronic devices and systems produced.

In the case of a VLSI device, the chip may be discarded or it may be investigated by FMA for yield enhancement. In the case of a PCB, FMA may be performed for yield enhancement or the board may undergo further testing for fault location and repair. A "good" circuit is assumed to be defect free, but this assumption is only as good as the quality of the tests being applied to the manufactured design. Once again, fault simulation provides a quantitative measure of the quality of a given set of tests.

1.4 PHYSICAL DESIGN AUTOMATION

Physical design refers to all synthesis steps that convert a circuit representation (in terms of gates and transistors) into a geometric representation (in terms of polygons and their shapes) [Sherwani 1999; Chang 2007]. An example is illustrated in

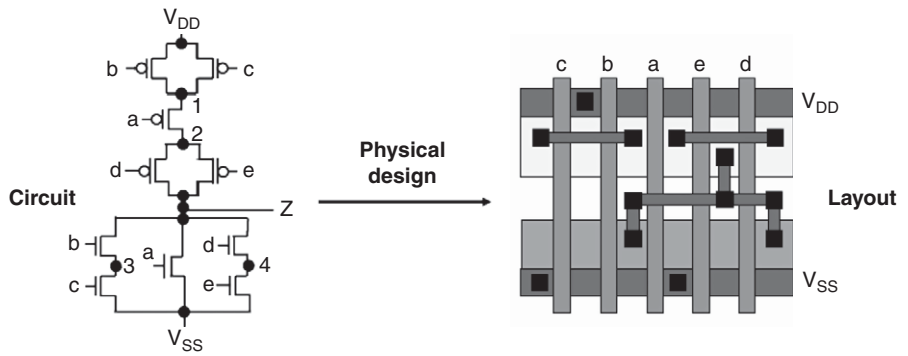

FIGURE 1.14

The function of physical design.

Figure 1.14. The geometric representation, also called **layout**, is used to design masks and then manufacture a chip. Because the design process is fairly complicated in nature, modern physical design typically is divided into three major steps: (1) floorplanning, (2) placement, and (3) routing. **Floorplanning** is an essential design step for a hierarchical, building block design method. It assembles circuit blocks into a rectangle (chip) to optimize a predefined cost metric such as area and wire length. The circuit blocks could be flexible or rigid in their shapes. **Placement** is the process of assigning the circuit components into a chip region. It can be considered as a restricted floorplanning problem for rigid blocks with some dimension similarity. After placement, the **routing** process defines the precise paths for conductors that carry electrical signals on the chip layout to interconnect all pins that are electrically equivalent. After routing, some **physical verification** processes (such as ***design rule checking*** [DRC]), **performance checking**, and **reliability checking**) are performed to verify whether all geometric patterns, circuit timing, and electrical effects satisfy the design rules and specifications.

As design and process technologies advance at a breathtaking speed, feature size and voltage levels associated with modern VLSI designs are decreasing drastically while at the same time die size, operating frequency, design complexity, and packing density keep increasing. Physical design for such a system must consider the integration of large-scale digital and ***analog and mixed-signal*** (AMS) circuit blocks, the design of system interconnections/buses, and the optimization of circuit performance, area, power consumption, and signal and power integrity. On one hand, designs with more than a billion transistors are already in production, and functional blocks are widely reused in nanometer circuit design, which all drive the need for a modern physical design tool to handle large-scale designs. On the other hand, the highly competitive IC market requires faster design convergence, faster incremental design turnaround, and better silicon area utilization. Efficient and effective design methods and tools capable of optimizing large-scale circuits are essential for modern VLSI physical designs.

1.4.1 **Floorplanning**

Floorplanning is typically considered the first stage of VLSI physical design. Given a set of **hard blocks** (whose shapes cannot be changed) and/or **soft blocks** (whose shapes can be adjusted) and a netlist, floorplanning determines the shapes of soft blocks and assembles the blocks into a rectangle (chip) so a predefined cost metric (such as the chip area, wire length, wire congestion) is optimized [Sait 1999; Chen 2006]. See Figure 1.15 for the floorplan of the Intel Pentium 4 microprocessor.

Floorplanning gives early feedback that suggests architectural modifications, estimates the chip area, and estimates delay and congestion caused by wiring [Gerez 1998]. As technology advances, designs with more than a billion transistors are already in production. To cope with the increasing design complexity, hierarchical design and functional blocks are widely used. This trend makes floorplanning much more critical to the quality of a VLSI design than ever. Therefore, efficient and effective floorplanning methods and tools are desirable for modern circuit designs.

1.4.2 **Placement**

Placement is the process of assigning the circuit components into a chip region. Given a set of fixed cells/macros, a netlist, and a chip outline, placement assigns the predesigned cells/macros to positions on the chip so that no two cells/macros overlap with each other (*i.e.*, legalization) and some cost functions (*e.g.*, wire length, congestion, and timing) are optimized [Nam 2007; Chen 2008].

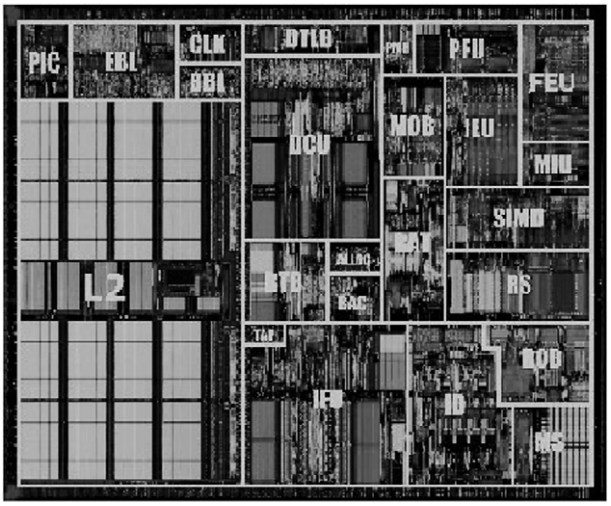

FIGURE 1.15

Floorplan of the Intel Pentium 4 microprocessor. (Courtesy of Intel Corporation.)

The traditional placement problem seeks to minimize wire length under the constraint that cells/macros do not overlap with each other. Two major challenges arise because of this high complexity for modern circuit design. First, the predesigned macro blocks (such as embedded memories, analog blocks, predesigned data paths) are often reused, and thus many designs contain hundreds of macro blocks and millions of cells. See Figure 1.16 for two example placements with large-scale cells and macros of very different sizes. Second, timing and routability (congestion) optimization become more challenging because of the design complexity and the scaling of devices and interconnects. As a result, modern design challenges have reshaped the placement problem. The modern placement problem becomes very hard, because we need to handle large-scale designs with millions of objects. Furthermore, the objects could be very different in their sizes. In addition to wire length, we also need to consider many placement constraints such as timing, routability (congestion), and thermal issues.

1.4.3 **Routing**

After placement, routing defines the precise paths for conductors that carry electrical signals on the chip layout to interconnect all pins that are electrically equivalent. See Figure 1.17 for a two-layer routing example [Chang 2004]. After routing, some physical verification processes (such as design rule checking, performance checking, and reliability checking) are performed to verify whether all geometric patterns, circuit timing, and electrical effects satisfy the design rules and specifications.

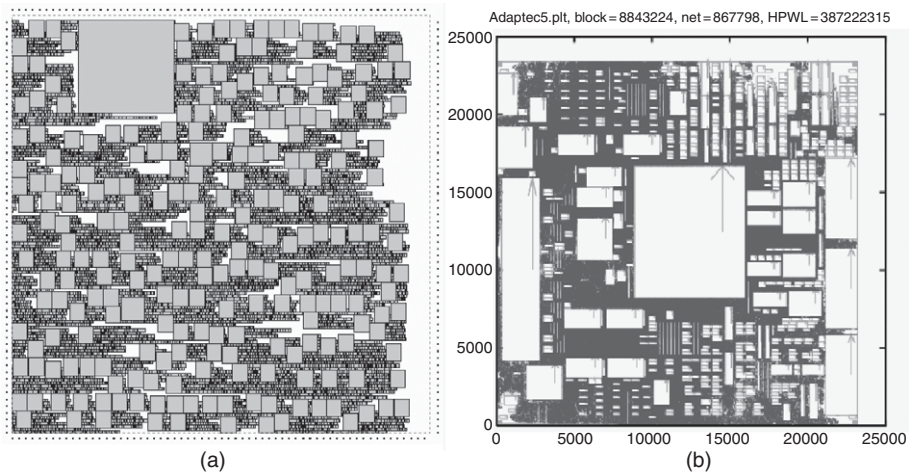

FIGURE 1.16

Two IBM placement examples: (a) The ibm01 circuit with 12,752 cells and 247 macros. (b) The adapetc5 circuit with 842 K cells, 646 macros, and 868 K nets.

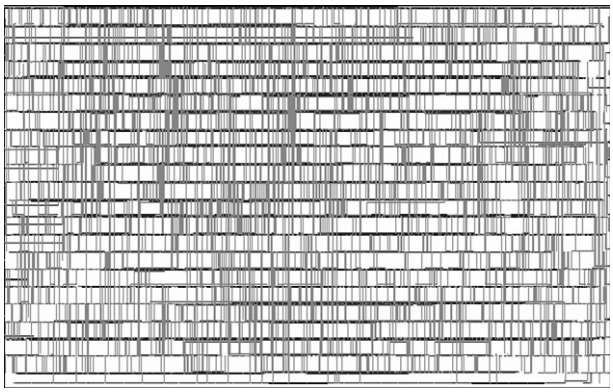

FIGURE 1.17

A two-layer routing example with 8109 nets. All horizontal wires are routed on one layer, and so are vertical ones.

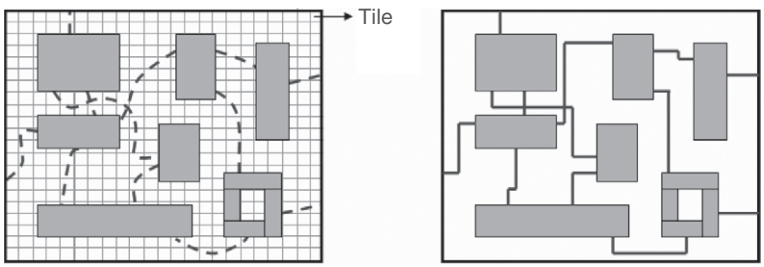

FIGURE 1.18

Global routing and detailed routing.

Typically, routing is a very complex problem. To make it manageable, a traditional routing system usually uses the two-stage technique of **global routing** followed by **detailed routing**. Global routing first partitions the entire routing region into tiles (or channels) and decides tile-to-tile paths for all nets while attempting to optimize some specified objective functions (*e.g.*, the total wire length and the critical timing constraints). Then, guided by the results of global routing, detailed routing determines actual tracks and routes for all nets according to the design rules. See Figure 1.18 for an illustration of the global and detailed routing [Ho 2007].

1.4.4 **Synthesis of clock and power/ground networks**

The specifications for clock and power/ground nets are significantly different from those for general signal nets. Generic routers cannot handle the requirements associated with clock and power/ground nets well. For example, we

often need to synchronize the arrivals of the clock signals at all functional units for clock nets and minimize the IR (voltage) drops while satisfying the current density (electromigration) constraint for power/ground nets. As a result, it is desirable to develop specialized algorithms for routing such nets.

Two strategies are used to implement a digital system: synchronous and asynchronous systems. In a typical synchronous system, data transfer among circuit components is controlled by a highly precise clock signal. In contrast, an asynchronous system usually applies a data signal to achieve the communication for data transfer. The synchronous system dominates the on-chip circuit designs mainly because of its simplicity in chip implementation and easy debugging. Nevertheless, the realization and performance of the synchronous system highly rely on a network to transmit the clock signals to all circuit components that need to be synchronized for operations (*e.g.*, triggered with a rising edge of the clock signal). Ideally, the clock signals should arrive at all circuit components simultaneously so that the circuit components can operate and data can be transferred at the same time. In reality, however, the clock signals might not reach all circuit components at the same time. The maximum difference in the arrival times of the clock signals at the circuit components, referred to as **clock skew**, should be minimized to avoid the idleness of the component with an earlier clock signal arrival time. The smaller the clock skew, the faster the clock. Consequently, a **clock-net synthesis** problem arises from such a synchronous system: routing clock nets to minimize the clock skew (preferably zero) and delay [Tsay 1993]. More sophisticated synchronous systems might intentionally schedule nonzero clock skew to further reduce the clock period, called **useful clock skew**. More information can be found in Chapter 13. There are also some other important design issues for clock-net synthesis, for example, total wire length and power consumption optimization.

Example 1.1 Figure 1.19 shows two clock networks. The clock network in Figure 1.19a incurs a skew of 16 units and the maximum delay of 30 units, whereas the clock network in Figure 1.19b has zero clock skew and the same delay as that in Figure 1.19a.

For modern circuit design, the power and ground networks are usually laid out on metal layers to reduce the resistance of the networks. See Figure 1.20 for a popular two-layer meshlike power/ground network, in which parallel vertical power (V_{DD}) and ground (GND) lines run on the metal-4 layer, connected by horizontal power and ground lines on the metal-5 layer. All the blocks that need power supply or need to be connected to ground can thus connect to the appropriate power and ground lines.

The power and ground lines are typically much wider than signal nets because they need to carry much larger amounts of current. Therefore, we need to consider the wire widths of power/ground networks for the area requirement. As technology advances, the metal width decreases while the global wire length increases. This trend makes the resistance of the power line

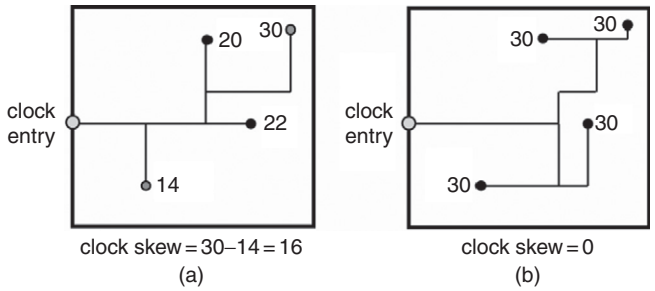

FIGURE 1.19

Two clock networks: (a) Clock network with a skew of 16 units and the maximum delay of 30 units. (b) Clock network with zero skew and 30-unit delay.

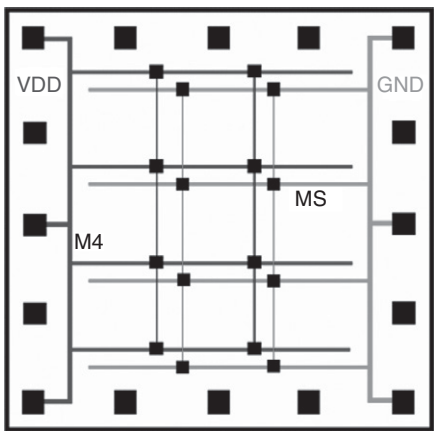

FIGURE 1.20

A typical power/ground network.

increase substantially. Furthermore, the threshold voltage scales nonlinearly, raising the ratio of the threshold voltage to the supply voltage and making the voltage drop in the power/ground network a serious challenge in modern circuit design. Because of the voltage drop, supply voltage in logic may not be an ideal reference. This effect may weaken the driving capability of logic gates, reduce circuit performance, slow down slew rate (and thus increase power consumption), and lower noise margin. As a result, power/ground network synthesis attempts to use the minimum amount of wiring area for a power/ground network under the power-integrity constraints such as voltage drops and electromigration. There are two major tasks for the synthesis: (1) power/ground network topology determination to plan the wiring topology of a power/ground network and (2) power/ground wire sizing to meet the current density and reliability constraints [Sait 1999; Sherwani 1999; Tan 2003].

Example 1.2 Figure 1.21a shows a chip floorplan of four modules and the power/ground network. As shown in the figure, we refer to a pad feeding supply voltage into the chip as a power pad, the power line enclosing the floorplan as a core ring, a power line branching from a core ring into modules inside as a power trunk, and a pin in a module that absorbs current (connects to a core ring or a power trunk) as a P/G pin. To ensure correct and reliable logic operation, we will minimize the voltage drops from the power pad to the P/G pins in a power/ground network. Figure 1.21a shows an instance of voltage drop in the power supply line, in which the voltage drops by almost 26% at the rightmost P/G pin. Figure 1.21b shows that by having a different chip floorplan, the worst-case voltage drop is reduced to approximately 5% [Liu 2007]. Recent research showed that a 5% voltage drop in supply voltage might slow down circuit performance by as much as 15% or more [Yim 1999]. Furthermore, it is typical to limit the voltage drop within 10% of the supply voltage to guarantee proper circuit operation. Therefore, voltage drop is a first-order effect and can no longer be ignored during the design process.

1.5 CONCLUDING REMARKS

The sophistication and complexity of current electronic systems, including ***printed circuit boards*** (PCBs) and ***integrated circuits*** (ICs), are a direct result of ***electronic design automation*** (EDA). Conversely, EDA is highly dependent on the power and performance of ICs, such as microprocessors and RAMs used to construct the computers on which the EDA software is executed. As a result, EDA is used to develop the next generation of ICs, which, in turn, are used to develop and execute the next generation of EDA, and so on in an ever-advancing progression of features and capabilities.

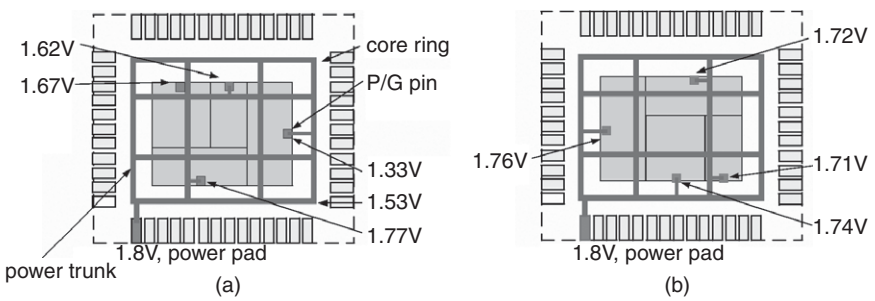

FIGURE 1.21

Two floorplans with associated power/ground network structures: (a) Worst-case voltage drop at the P/G pins approximately 26% of the supply voltage. (b) Worst-case voltage drop approximately only 5% [Liu 2007].

The current drivers for EDA include such factors as manufacturing volume, die size, integration heterogeneity, and increasing complexity [SIA 2005]. The primary influencing factors include ***system-on-chips*** (SOCs), microprocessors, ***analog/mixed-signal*** (AMS) circuits, and embedded memories, as well as continuing increases in both silicon and system complexity. Silicon complexity results from process scaling and introduction of new materials and device/interconnect structures. System complexity results from increasing transistor counts produced by the smaller feature sizes and demands for increased functionality, lower cost, and shorter time-to-market. Collectively, these factors and influences create major EDA challenges in the areas of design and verification productivity, power management and delivery, manufacturability, and manufacturing test, as well as product reliability [SIA 2005]. Other related challenges include higher levels of abstraction for ESL design, AMS codesign and automation, parametric yield at volume production, reuse and test of ***intellectual property*** (IP) cores in heterogeneous SOCs, cost-driven design optimization, embedded software design, and design process management. The purpose of this book is to describe more thoroughly the traditional and evolving techniques currently used to address these EDA challenges [SIA 2006, 2007].

The remaining chapters provide more detailed discussions of these topics. For example, general CMOS design techniques and issues are presented in Chapter 2 and fundamental design for testability techniques for producing quality CMOS designs are provided in Chapter 3. Most aspects of EDA in synthesis (including high-level synthesis, logic synthesis, test synthesis, and physical design), verification, and test, rely heavily on various algorithms related to the specific task at hand. These algorithms are described in Chapter 4. Modeling of a design at the *electronic system level* (ESL) and synthesis of the ESL design to the high level are first presented in Chapter 5. The design then goes through logic synthesis (Chapter 6) and test synthesis (Chapter 7) to generate a testable design at the gate level for further verification before physical design is performed. Design verification that deals with logic and circuit simulation is presented in Chapter 8, and functional verification is discussed in Chapter 9. The various aspects of physical design are addressed in Chapter 10 (floorplanning), Chapter 11 (placement), Chapter 12 (routing), and Chapter 13 (synthesis of clock and power/ground networks). Finally, logic testing that includes the most important fault simulation and test generation techniques to guarantee high product quality is discussed in Chapter 14 in detail.

1.6 EXERCISES

1.1. (Design Language) What are the two most popular *hardware description languages* (HDLs) practiced in the industry?

1.2. (Synthesis) Synthesis often implies high-level synthesis, logic synthesis, and physical synthesis. State their differences.

1.3. (Verification) Give three verification approaches that can be used to verify the correctness of a design. State the differences between model checking and equivalence checking.

1.4. (Fault Model) Assume a circuit has a total of n input and output nodes. How many single stuck-at faults, dominant bridging faults, 4-way bridging faults, and multiple stuck-at faults are present in the circuit?

1.5. (Design for Testability) Assume a sequential circuit contains n flip-flops and each state is accessible from an initial state in m clock cycles. If a sequential ATPG is used and p test patterns are required to detect all single stuck-at faults in the design, how many clock cycles would be required to load the sequential circuit with predetermined states? If all flip-flops have been converted to scan flip-flops and stitched together to form one scan chain, how many clock cycles would be required to load the combinational circuit with predetermined states?

1.6. (Testing) State the differences between fault simulation and test generation. Give three main reasons each why sequential test generation is difficult and why the industry widely adopts scan designs.

1.7. (Design Flow) As technology advances, interconnects dominate the circuit performance. When are the interconnect issues handled during the traditional VLSI design flow? How can we modify the design flow to better tackle the interconnect issues?

1.8. (Clock-net Synthesis) Give the clock entry point p_0 located at the coordinate (3, 0) and four clock pins p_1, p_2, p_3, and p_4 located at (1, 1), (5, 1), (1, 5), and (5, 5), respectively. Assume that the delay is proportional to the path length and the wire can run only on the grid lines. Show how to interconnect the clock entry point p_0 to the other four clock pins p_i, $1 \leq i \leq 4$, such that the clock skew is zero and the clock delay is minimized. What is the resulting clock delay?

1.9. (Programmable Logic Array) A shorthand notation commonly used for ***programmable logic arrays*** (PLAs) and combinational logic in ***programmable logic devices*** (PLDs) is illustrated in Figure 1.22, which

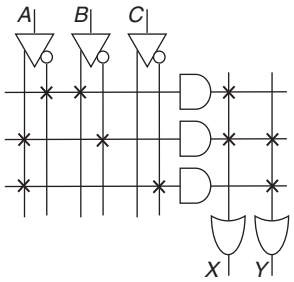

FIGURE 1.22

Shorthand notation for the connection array in Figure 1.6 and the PLA implementation in Figure 1.7.

corresponds to the connection array in Figure 1.6 and the PLA implementation in Figure 1.7. Give the connection array and draw the PLA shorthand diagram and PLA transistor-level implementation for the following set of Boolean equations, sharing product terms where possible:

$$O_2 = I_3 \bullet I_2$$
$$O_1 = I_3 \bullet I_2 \bullet I_1' \bullet I_0' + I_3' \bullet I_2' \bullet I_1 \bullet I_0$$
$$O_0 = I_3 \bullet I_2 + I_2 \bullet I_1 + I_1 \bullet I_0 + I_3 \bullet I_0$$

ACKNOWLEDGMENTS

We wish to thank Professor Ren-Song Tsay of National Tsing Hua University, Professor Jie-Hong (Roland) Jiang of National Taiwan University, and Professor Jianwen Zhu of University of Toronto for reviewing the Logic Design Automation section; Professor Wen-Ben Jone of University of Cincinnati for reviewing the Test Automation section; and Professor James C.-M. Li, Wen-Chi Chao, Po-Sen Huang, and Tzro-Fan Chien of National Taiwan University for reviewing the manuscript and providing very helpful comments.

REFERENCES

R1.0 Books

[Abramovici 1994] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design,* IEEE Press, Revised Printing, Piscataway, NJ, 1994.

[Brayton 1984] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis,* Kluwer Academic, Boston, 1984.

[Bushnell 2000] M. L. Bushnell and V. D. Agrawal, Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*,* Springer, Boston, 2000.

[De Micheli 1994] G. De Micheli, *Synthesis and Optimization of Digital Circuits,* McGraw-Hill, New York, 1994.

[Devadas 1994] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis,* McGraw-Hill, New York, 1994.

[Dutton 1993] R. Dutton and Z. Yu, *Technology CAD: Computer Simulation of IC Processes and Devices,* Kluwer Academic, Boston, 1993.

[Gerez 1998] S. Gerez, *Algorithms for VLSI Design Automation,* John Wiley & Sons, Chichester, England, 1998.

[Ho 2007] T.-Y. Ho, Y.-W. Chang, and S.-J. Chen, *Full-Chip Nanometer Routing Techniques,* Springer, New York, 2007.

[IEEE 1076-2002] *IEEE Standard VHDL Language Reference Manual,* IEEE, Std. 1076-2002, IEEE, New York, 2002.

[IEEE 1463-2001] *IEEE Standard Description Language Based on the Verilog Hardware Description Language,* IEEE, Std. 1463-2001, IEEE, New York, 2001.

[Jha 2003] N. Jha and S. Gupta, *Testing of Digital Systems,* Cambridge University Press, London, 2003.

[Keating 1999] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs,* Springer, Boston, 1999.

[Ledgard 1983] H. Ledgard, *Reference Manual for the ADA Programming Language,* Springer, Boston, 1983.

[McCluskey 1986] E. J. McCluskey, *Logic Design Principles: With Emphasis on Testable Semiconductor Circuits,* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Mead 1980] C. Mead and L. Conway, *Physical Design Automation of VLSI Systems,* Addison Wesley, Reading, MA, 1980.

[Nam 2007] G.-J. Nam and J. Cong, editors, *Modern Circuit Placement: Best Practices and Results,* Springer, Boston, 2007.

[Plummer 2000] J. D. Plummer, M. Deal, and P. Griffin, *Silicon VLSI Technology–Fundamentals, Practice and Modeling,* Prentice-Hall, Englewood Cliffs, NJ, 2000.

[Preas 1988] B. Preas and M. Lorenzetti, *Physical Design Automation of VLSI Systems,* Benjamin/Cummings, Menlo Park, CA, 1997.

[Sait 1999] S. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice,* World Scientific Publishing Company, 1999.

[Scheffer 2006a] L. Scheffer, L. Lavagno, and G. Martin, editors, *EDA for IC System Design, Verification, and Testing,* CRC Press, Boca Raton, FL, 2006.

[Scheffer 2006b] L. Scheffer, L. Lavagno, and G. Martin, editors, *EDA for IC Implementation, Circuit Design, and Process Technology,* CRC Press, Boca Raton, FL, 2006.

[Sherwani 1999] N. Sherwani, *Algorithms for VLSI Physical Design Automation,* 3rd Ed., Kluwer Academic, Boston, 1999.

[Stroud 2002] C. Stroud, *A Designer's Guide to Built-In Self-Test,* Springer, Boston, 2002.

[Wang 2006] L.-T. Wang, C.-W. Wu, and X. Wen, editors, *VLSI Test Principles and Architectures: Design for Testability,* Morgan Kaufmann, San Francisco, 2006.

[Wang 2007] L.-T. Wang, C. Stroud, and N. Touba, editors, *System-on-Chip Test Architectures: Nanometer Design for Testability,* Morgan Kaufmann, San Francisco, 2007.

[Wile 2005] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification,* Morgan Kaufmann, San Francisco, 2005.

R1.1 Overview of Electronic Design Automation

[Cadence 2008] Cadence Design Systems, http://www.cadence.com, 2008.

[DAC 2008] Design Automation Conference, co-sponsored by Association for Computing Machinery (ACM) and Institute of Electronics and Electrical Engineers (IEEE), http://www.dac.com, 2008.

[Kilby 1958] J. Kilby, Integrated circuits invented by Jack Kilby, Texas Instruments, Dallas, TX, http://www.ti.com/corp/docs/company/history/timeline/semicon/1950/docs/58ic_kilby.htm, September 12, 1958.

[Mentor 2008] Mentor Graphics, http://www.mentor.com, 2008.

[MOSIS 2008] The MOSIS Service, http://www.mosis.com, 2008.

[Naffziger 2006] S. Naffziger, B. Stackhouse, T. Grutkowski, D. Josephson, J. Desai, E. Alon, and M. Horowitz, The implementation of a 2-core multi-threaded Itanium family processor, *IEEE J. of Solid-State Circuits Conf.*, 41(1), pp. 197–209, January 2006.

[Ochetta 1994] E. Ochetta, R. Rutenbar, and L. Carley, ASTRX/OBLX: Tools for rapid synthesis of high-performance analog circuits, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 24–30, June 1994.

[SIA 2005] SIA, *The International Technology Roadmap for Semiconductors*: *2005 Edition*, Semiconductor Industry Association, San Jose, CA, http://public.itrs.net, 2005.

[SIA 2006] *SIA*, *The International Technology Roadmap for Semiconductors: 2006 Update*, Semiconductor Industry Association, San Jose, CA, http://public.itrs.net, 2006.

[Stackhouse 2008] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles, A 65nm 2-billion-transistor quad-core Itanium processor, in *Digest of Papers*, *IEEE Int. Solid-State Circuits Conf.,* pp. 92, February 2008.

[Stroud 1986] C. Stroud, R. Munoz, and D. Pierce, CONES: A system for automated synthesis of VLSI and programmable logic from behavioral models, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 428–431, November 1986.

[Synopsys 2008] Synopsys, http://www.synopsys.com, 2008.

[SystemC 2008] SystemC, http://www.systemc.org, 2008.

[SystemVerilog 2008] SystemVerilog, http://systemverilog.org, 2008.

R1.2 Logic Design Automation

[Velev 2001] M. N. Velev and R. Bryant, Effective use of Boolean satisfiability procedures in the formal verification of scalar and VLIW microprocessors, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 226–231, June 2001.

R1.3 Test Automation

[Eichelberger 1978] E. Eichelberger and T. Williams, A logic design structure for LSI testability, *J. of Design Automation and Fault-Tolerant Computing*, 2(2), pp. 165–178, February 1978.

[Sato 2005] Y. Sato, S. Hamada, T. Maeda, A. Takatori, Y. Nozuyama, and S. Kajihara, Invisible delay quality–SDQM model lights up what could not be seen, in *Proc. IEEE Int. Test Conf.*, Paper 47.1, November 2005.

[Touba 2006] N. A. Touba, Survey of test vector compression techniques, *IEEE Design & Test of Computers*, 23(4), pp. 294–303, July-August 2006.

[Williams 1983] T. Williams and K. Parker, Design for testability—A survey, *Proceedings of the IEEE*, 71(1), pp. 98–112, January 1983.

R1.4 Physical Design Automation

[Chang 2004] Y.-W. Chang and S.-P. Lin, MR: A new framework for multilevel full-chip routing, *IEEE Trans. on Computer-Aided Design*, 23(5), pp. 793–800, May 2004.

[Chang 2007] Y.-W. Chang, T.-C. Chen, and H.-Y. Chen, Physical design for system-on-a-chip, in *Essential Issues in SOC Design*, Y.-L. Lin, editor, Springer, Boston, 2007.

[Chen 2006] T.-C. Chen and Y.-W. Chang, Modern floorplanning based on B*-trees and fast simulated annealing, *IEEE Trans. on Computer-Aided Design*, 25(4), pp. 637–650, April 2006.

[Chen 2008] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints, *IEEE Trans. on Computer-Aided Design*, 27(7), pp. 1228–1240, July 2008.

[Liu 2007] C.-W. Liu and Y.-W. Chang, Power/ground network and floorplan co-synthesis for fast design convergence, *IEEE Trans. on Computer-Aided Design*, 26(4), pp. 693–704, April 2007.

[Tan 2003] S. X-D. Tan and C.-J. R. Shi, Efficient very large scale integration power/ground network sizing based on equivalent circuit modeling, *IEEE Trans. on Computer-Aided Design*, 22(3), pp. 277–284, March 2003.

[Tsay 1993] R.-S. Tsay, An exact zero-skew clock routing algorithm, *IEEE Trans. on Computer-Aided Design*, 12(2), pp. 242–249, February 1993.

[Yim 1999] J. S. Yim, S. O. Bae, and C. M. Kyung, A Floorplan-based planning methodology for power and clock distribution in ASICs, in *Proc. ACM/IEEE Design Automation Conf.*, 766–771, June 1999.

R1.5 Concluding Remarks

[SIA 2005] SIA, *The International Technology Roadmap for Semiconductors: 2005 Edition,* Semiconductor Industry Association, San Jose, CA, http://public.itrs.net, 2005.

[SIA 2006] SIA, *The International Technology Roadmap for Semiconductors: 2006 Update,* Semiconductor Industry Association, San Jose, CA, http://public.itrs.net, 2006.

[SIA 2007] SIA, *The International Technology Roadmap for Semiconductors: 2007 Edition,* Semiconductor Industry Association, San Jose, CA, http://public.itrs.net, 2007.