Chapter 38, Design Automation for Microelectronics, Springer Handbook of Automation

Deming Chen

Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, dchen@illinois.edu

Abstract

Design automation or computer-aided design (CAD) for microelectronic circuits has emerged since the creation of the integrated circuits (IC). It has played a crucial role to enable the rapid development of hardware and software systems in the past several decades. CAD techniques are the key driving forces behind the reduction of circuit design time and the optimization of the circuit quality. Meanwhile, the exponential growth of circuit capacity driven by the Moore's law[1] prompts new and critical challenges for CAD techniques. In this chapter we will introduce the fundamentals of design automation as an engineering field. We begin with several important processor technologies and several existing IC technologies. We then present a typical CAD flow covering all the major steps in the design cycle. We also cover some important topics such as verification and TCAD. Finally, we introduce some new trends in the design automation field.

1. Introduction

1.1 Background on Microelectronic Circuits

Microelectronic circuits are ubiquitous nowadays. We can find them not only in desktop computers, laptops and workstations, but also in consumer electronics, home and office appliances, automobiles, military applications, and telecommunication applications, etc. Due to different requirements of these applications, circuits are designed differently to pursue unique features suitable for the specific application. In general, these devices are built with two fundamental and orthogonal technologies – *processor technology* and *IC technology*.

Processor technology refers to the architecture of the computation engine used to implement the desired functionality of an electronic circuit. It is categorized into three main branches – *general purpose processors*, *application-specific instruction set processors* and *single-purpose processors* [1]. General-purpose processor, or microprocessor, is a device that executes software through instruction codes. Therefore, they are software-programmable. This processor has a program memory that holds the instructions and a general datapath that executes the instructions. The general datapath consists of one or several general-purpose arithmetic logic units (ALUs). Application-specific instruction set processor (ASIP) is a software-programmable processor optimized for a particular class or domain of applications, such as signal processing, telecommunication, and gaming applications. To fit in the application, the datapath and instructions of such a processor are customized. For example, one type of ASIP, digital signal processor (DSP), may have special-purpose datapath components such as a multiply-accumulate unit, which can perform multiply-and-add using only one instruction. Finally, a single-purpose processor is a digital circuit designed to serve a single purpose – executing exactly one program. It represents an exact fit of the desired functionality and is not software programmable. Its datapath contains only the essential components for this program and there is no program memory. General-purpose processor offers the maximum flexibility in terms of the various applications it can support but with the least efficiency in terms of performance and power consumption. On the contrary, single-purpose processor offers the maximum performance and power efficiency but with the least flexibility in terms of the applications it can support. ASIP offers a compromise between these two extremes.

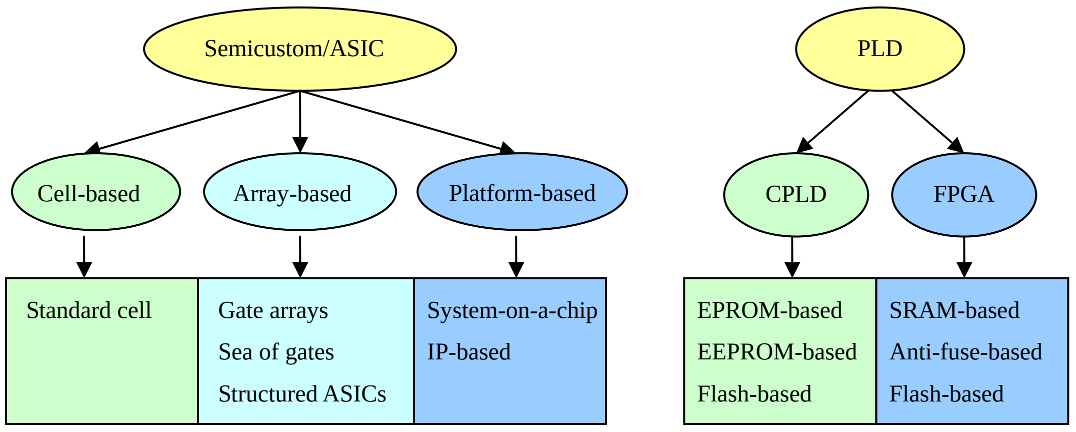
IC technology refers to the specific implementation method or the design style of the processing engine on an IC. It is categorized into three main branches – *full custom*, *semicustom*, and *programmable logic device* (PLD) [2]. Full custom refers to the design style where the functional and physical designs are handcrafted. This would provide the best design quality but also require an extensive effort of a design team to optimize each detailed feature of the circuit. Since the design effort and cost are high, this design style is usually used in high-volume (thus cost can be amortized) or high-performance applications. Semicustom design is also called ASIC (application-specific integrated circuit). It is trying to reduce the design complexity by restricting the circuit primitives to a limited number. Such a restriction allows the designer to use well-designed circuit primitives (gates or functional blocks) and focus on their efficient interconnection. Such a restriction also makes it easier to develop computer-aided design tools for circuit design and optimization and reduce the design time and cost. Today the number of semicustom designs outnumbers custom designs significantly, and some high-performance microprocessors have been designed partially using semicustom style, especially for the control logic (e.g., IBM's POWER-series processors and SUN Microsystems' UltraSPARC T1 processors).

Semicustom designs can be further partitioned into several major classes. Figure 1(a) shows such a partition. *Cell-based design* generally refers to *standard cell design*, where the fundamental cells are stored in a library. Cells often are simple gates and latches, but can be complex gates, memories and logic blocks as well. These cells are pre-tested and

[1] Moore's Law describes an important trend in the history of the semiconductor industry: the number of transistors per unit chip area would be doubled approximately every two years. This observation was first made by the Intel co-founder Gordon E. Moore in a 1965 paper. Moore's Law has been true for the past four decades, and many people believe that it will continue for at least another decade before reaching the fundamental physical limits of device fabrication.

pre-characterized. The maintenance of the library is not a trivial task since each cell needs to be characterized in terms of area, delay, and power over ranges of temperatures and supply voltages [2]. Companies offer standard cell libraries for use with their fabrication and design technologies, and amortize the effort of designing the cell library over all the designs that use it. *Array-based design* in general refers to the design style of constructing a common base array of transistors and personalizing the chip by altering the metallization (the wiring between the transistors) that is placed on top of the transistors. It mainly consists of *gate arrays*, *sea of gates* and *structured ASICs* (refer to Chapter 8 in [3] for details). *Platform-based design* [4][5][6] is referring to the design style that heavily reuses hardware and software intellectual properties (IP), which provide pre-programmed and verified design elements. Rather than looking at IP reuse in a block by block manner, platform-based design aggregates groups of components into a reusable platform architecture. There is a slight difference between *system-on-a-chip* and *IP-based* design. A system-on-a-chip usually incorporates at least one software-programmable processor, on-chip memory, and accelerating function units. IP-based design is more general and may not contain any software-programmable processors. Nonetheless, both styles heavily reuse IPs.

The third major IC technology is the programmable logic device (PLD) technology (Figure 1(b)). In PLD, both transistor and metallization are already fabricated but they are hardware-programmable. Such a programming is achieved through creating and destroying wires that connect logic blocks either by making an anti-fuse[2] or setting a bit in a programmable switch that is controlled by a memory cell. There are two major PLDs – CPLD (complex programmable logic device) and FPGA (field-programmable gate array). The main difference between these two types of PLDs is that the basic programmable logic element in CPLD is PLA (two-level AND/OR array), and the basic element in FPGA is LUT (look-up table). The PLAs are programmed through a mapping of logic functions in a two-level representation unto the AND/OR logic array, and the LUTs are programmed by setting bits in the LUT memory cells that store the truth table of logic functions. In general, CPLD's routing structures are simpler than those of FPGAs. Therefore, the interconnect delay of CPLD is more predictable compared to that of FPGAs. FPGAs usually offer much larger logic capacity than CPLDs mainly because LUTs offer a finer logic granularity than PLAs so they are suitable to be replicated massively to help achieve complex logic designs. Nowadays, a high-end commercial FPGA, such as Altera Stratix III and Xilinx Virtex-5, can contain more than 300K LUTs. Hardware programmability has significant advantages of short design time, low design cost, and fast time-to-market, which become more important when the design is complex. However, PLDs offer less logic density compared to semicustom designs mainly because they occupy a significant amount of circuit area to add in the programming bits [7]. Nonetheless, the number of new design starts using PLDs significantly outnumbers the new semicustom design starts. According to research firm Gartner/Dataquest, in the year 2007, there were nearly 89,000 FPGA design starts, and this number will swell to 112,000 in 2010 — some 25 times that of semicustom/ASIC designs [8]. Figure 1(b) provides further characterization for the implementation styles of CPLDs and FPGAs.



(a) (b)

Figure 1: Classification of (a) semicustom design and (b) PLD design

An important fact is that different IC technologies have different advantages and disadvantages in terms of circuit characteristics. Table 1 lists the comparison of these technologies using some key metrics – NRE cost[3], unit cost, design time, logic density, circuit performance, power consumption, and flexibility (referring to the easiness level of changing the hardware implementation corresponding to design changes). Another important fact is that processor technologies and IC technologies are orthogonal with each other, which means that each of the three processor technologies can be implemented in any of the three IC technologies. Table 2 lists some representative combinations between these two technologies. For instance, general-purpose processors can be implemented using full-custom (Intel Core 2), semicustom (ARM9), or PLD (NIOS II). Each of the nine combinations has the combined features of the two corresponding technologies. For instance, Intel Core 2 represents the top but costly implementation of a general-purpose processor(s), and Altera Viterbi Decoder represents the fast time-to-market version of a single-purpose processor.

[2] An anti-fuse is an open circuit device that becomes a short device when traversed by an appropriate current pulse.

[3] Non-recurring engineering cost: a one time charge for design and implementation of a specific product.

Table 1: Comparison of IC technologies

Metrics	Full custom	Semicustom	PLD
NRE (non-recurring engineering) cost	Very high	Medium-High	Low
Unit cost (low volume)	Very high	Medium-High	Low
Unit cost (high volume)	Low	Low	High
Design time	Very long	Medium-Long	Short
Logic Density	Very high	High	Low-Medium
Circuit performance	Very high	High	Low-Medium
Circuit power consumption	Low	Medium	High
Flexibility	Low	Medium	High

Table 2: The combination between processor technologies and IC technologies

	Processor Types		
IC Tech. Types ⇓	**General Purpose**	**Single Purpose**	**ASIP**
Full custom	Intel Core 2 Quad AMD Opteron	Intel 3965ABG (802.11a/b/g wireless chip) Analog ADV202 (JPEG 2000 Video CODEC)	TI TMS320C6000 (DSP)
Semicustom	ARM9 PowerPC	ATMEL AT83SND1C (MP3 Decoder)	Infineon C166 (Microcontroller)
PLD	Altera NIOS II Xilinx MicroBlaze	Altera Viterbi Decoder (Error Detection)	AllianceCORE C32025 (DSP)

1.2 History of Electronic Design Automation

Electronic design automation (EDA) creates software tools for computer-aided design (CAD) of electronic systems ranging from printed circuit boards (PCBs) to integrated circuits. We describe a brief history of EDA next. Integrated circuits were designed by hand and manually laid out before EDA. This design method obviously could not handle large and complex chips. By the mid-70s, designers started to automate the design process using placement and routing tools. In 1986 and 1987 respectively, Verilog and VHDL were introduced as hardware description languages. Circuit simulators quickly followed these inventions, allowing direct simulation of IC designs. Later on, logic synthesis was developed which would produce circuit netlists for downstream placement and routing tools. The earliest EDA tools were produced academically, and were in the public domain. One of the most famous was the "Berkeley VLSI Tools Tarball", a set of UNIX utilities used to design early VLSI systems. Meanwhile, the larger electronic companies had pursued EDA internally, where the seminal work was done at IBM and Bell Labs. In early-80s, managers and developers spun out of these big companies to start up EDA as a separate industry. Within a few years there were many companies specializing in EDA, each with a slightly different emphasis. Many of these EDA companies were merged with one another along the years. Currently, the major EDA companies include Cadence, Magma, Mentor Graphics and Synopsys. The total annual revenue of EDA is close to six billion US dollars.

According to the International Technology Roadmap for Semiconductors, the IC technology scaling driven by the Moore's Law will continue to evolve and dominate the semiconductor industry for at least another 10 years. This will lead to over 14 billion transistors integrated on a single chip in the 18nm technology by the year 2018 [9]. Such a scaling, however, has already created a large design productivity gap due to inherent design complexities and deep submicron issues. The study by a research consortium SEMATECH shows that although the level of on-chip integration, expressed in terms of the number of transistors per chip, increases at an approximate 58% annual compound growth rate, the design productivity, measured in terms of the number of transistors per staff-month, grows only at a 21% annual compound rate. Such a widening gap between the IC capacity and the design productivity presents critical challenges and also opportunities for the CAD community. Better and new design methodologies are needed to bridge such a gap.

2. Techniques of Electronic Design Automation

EDA can work on digital circuits and analog circuits. In this article, we will focus on EDA tools for digital integrated circuits because they are more prominent in the current EDA industry and occupy the major portion of the EDA market. For analog and mixed-signal circuit design automation, readers are referred to [10] and [11] for more details. Note that we can only briefly introduce the key techniques in EDA. Interested readers can refer to [12][13][14] for more details.

2.1 System-Level Design

Modern system-on-a-chip or FPGA designs contain embedded processors (hard or soft[4]), busses, memory, and hardware accelerators on a single device. On the one hand, these types of circuits provide opportunities and flexibilities for system

[4] These embedded processors are software-programmable IP cores. Hard processors are built with full-custom or semicustom technologies, and soft processors are built with PLD implementations (refer to Table 2).

designers to develop high-performance systems targeting various applications. On the other hand, they also immediately increase the design complexity considerably as mentioned in the Introduction section. To realize the promise of large system integration, a complete tool chain from concept to implementation is required. System-level and behavior-level synthesis techniques are the building blocks for this automated system design flow. System-level synthesis compiles a complex application in a system-level description (such as in C or SystemC) into a set of tasks to be executed on various software-programmable processors (referred as *software*), or a set of functions to be implemented in single-purpose processors (referred as *customized hardware* or simply *hardware*), together with the communication protocols and the interface logic connecting different components. Such capabilities are part of the *electronic system-level* (ESL) design automation emerged recently to deal with the design complexity and improve design productivity. The design challenges in ESL are mainly on effective hardware/software partitioning and co-design, system integration, and related issues such as the standardization of IP integration, system modeling, performance/power estimation, and system verification, etc.

Figure 2 illustrates a global view of the ESL design flow. The essential task is the hardware/software co-design, which requires hardware/software partitioning and incorporates three key synthesis tasks – *processor synthesis*, *interface synthesis*, and *behavioral synthesis*. Hardware/software partitioning defines the parts of the application that would be executed in software or hardware. Processor synthesis for software-programmable processors usually involves instantiation of processor IP cores or generation of processors with customized features (customized cache size, datapath, bitwidth, or pipeline stages, etc). Behavioral synthesis is also called high-level synthesis. It is a process that takes a given behavioral description of a hardware circuit and produces an RTL design automatically. We will introduce more details about behavioral synthesis in the next section. Every time the designer explores a different system architecture with hardware/software partitioning, the system interfaces must be redesigned. Interface synthesis is the process of automatic derivation of both the hardware and software interfaces to bind hardware/software elements together and permit them to communicate correctly and efficiently. Interface synthesis results need to meet bandwidth and performance requirements. The end product of ESL is an integrated system-level IC (e.g., system-on-a-chip, system in an FPGA, etc.) that aggregates software-programmable processors, customized hardware and interface logic to satisfy the overall area, delay, and power constraints of the design. Interested readers can refer to [1][12][13] and [15]-[23] for further study.

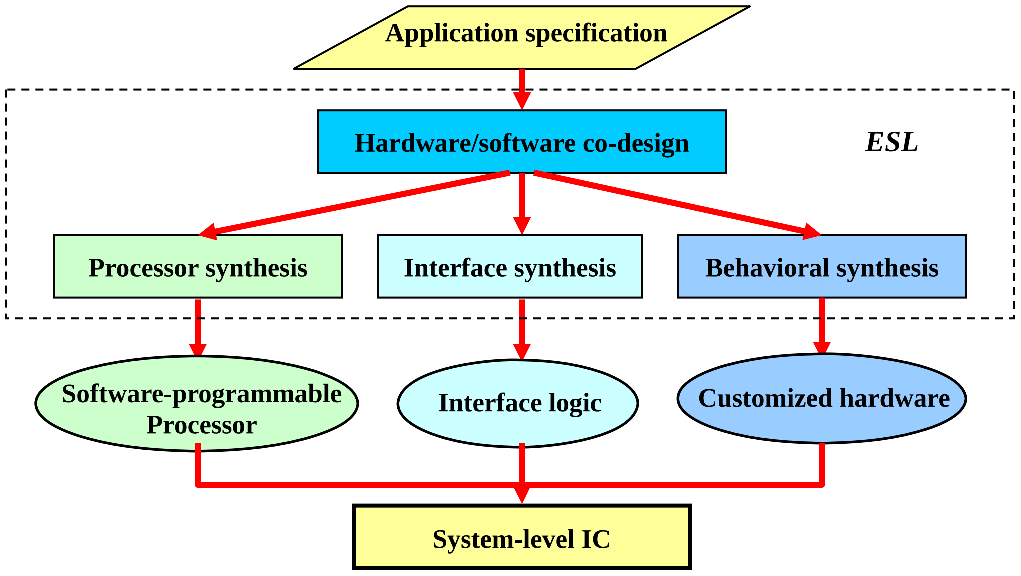


Figure 2: ESL (electronic system-level) design flow

2.2 Typical Design Flow

The majority of the development effort for CAD techniques is devoted to the design of single-purpose processors using semicustom or PLD IC technologies. We will introduce a typical design flow step by step as shown in Figure 3.

Behavior Synthesis. The basic problem of behavior synthesis or high-level synthesis is the mapping of a behavioral description of a circuit into a cycle-accurate RTL design consisting of a *datapath* and a *control unit*.[5] A datapath is composed of three types of components: *functional units* (e.g., ALUs, multipliers, and shifters), *storage units* (e.g., registers and memory), and *interconnection units* (e.g., buses and multiplexers). The control unit is specified as a finite state machine which controls the set of operations for the datapath to perform during every control step (clock cycle). The behavioral synthesis process mainly consists of three tasks: *scheduling*, *allocation*, and *binding*. Scheduling determines when a computational operation will be executed; allocation determines how many instances of resources (functional units, registers, or interconnection units) are needed; binding binds operations, variables, or data transfers to

[5] Designers can skip behavioral synthesis and directly write RTL codes for circuit design. This design style is facing increasing challenges due to the growing complexity of circuit design.

these resources. In general, it has been shown that the code density and simulation time can be improved by 10X and 100X, respectively, when moved to the behavior-level synthesis from RTL synthesis [23]. Such an improvement in efficiency is much needed for design in the deep submicron era. Figure 4 shows the scheduling and the binding solution for a computation *y= (a+b+c)*(d+e)*. Figure 4(a) shows the scheduling result, where CS means control step or clock cycle number. Figure 4(b) shows the binding solution for operations, which is a mapping between operations and functional units (t1, t2, t3 are temporary values). Figure 4(c) shows the final datapath. Note that the marks S1, S2, and S3 in the multiplexers indicate how the operands are selected for control steps 1, 2 and 3 respectively. A controller will be generated accordingly (not shown in the figure) to control the data movement in the datapath.

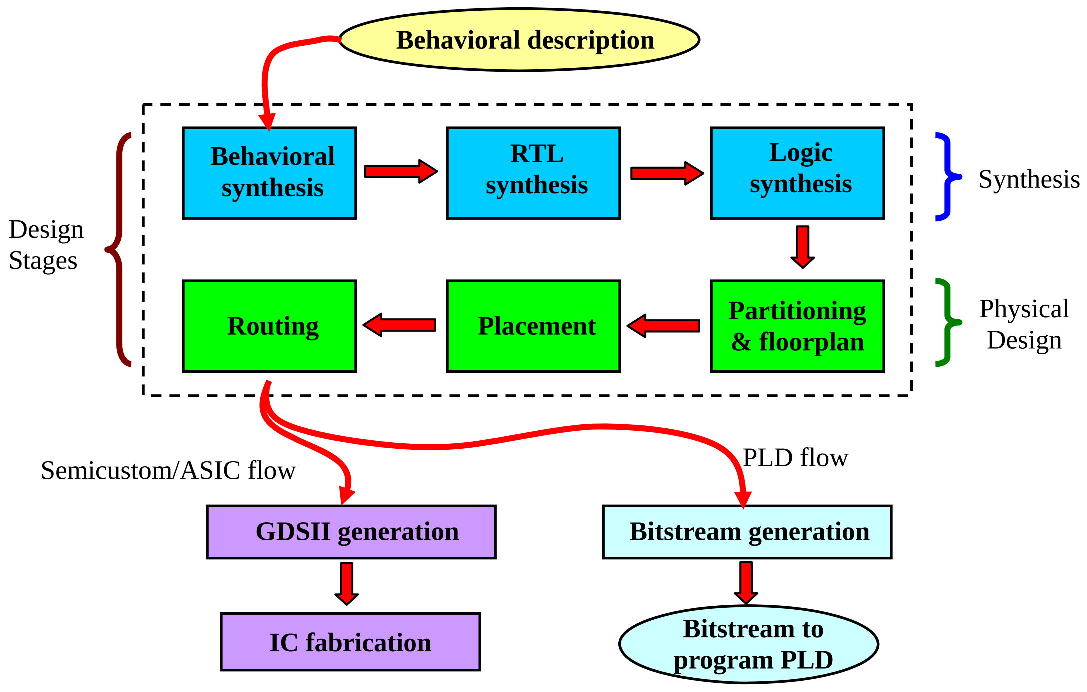
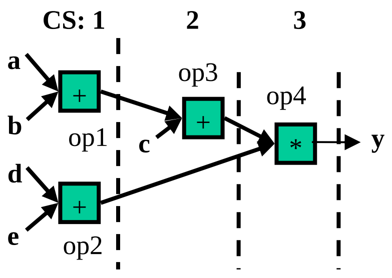


Figure 3: A typical design flow

Behavior synthesis is a well-studied problem [2][24]-[27]. Most of the behavioral synthesis problems are NP-hard problems due to various constraints, including latency and resource constraints. The subtasks of behavioral synthesis are highly interrelated with one another. For example, the scheduling of operations is directly constrained by resource allocation. Behavioral synthesis also faces challenges on how to connect better to the physical reality. Without physical layout information, the interconnect delay cannot be accurately estimated. In addition, there is a need of powerful data-dependence analysis tools to analyze the operational parallelism available in the design before one can allocate proper amount of resources to carry out the computation in parallel. In addition, how to carry out memory partitioning, bitwidth optimization, and memory access pattern optimization, together with behavioral synthesis for different application domains are important problems. Given all these challenges, much research is still needed in this area. Some recent representative works are presented in [28]-[35].

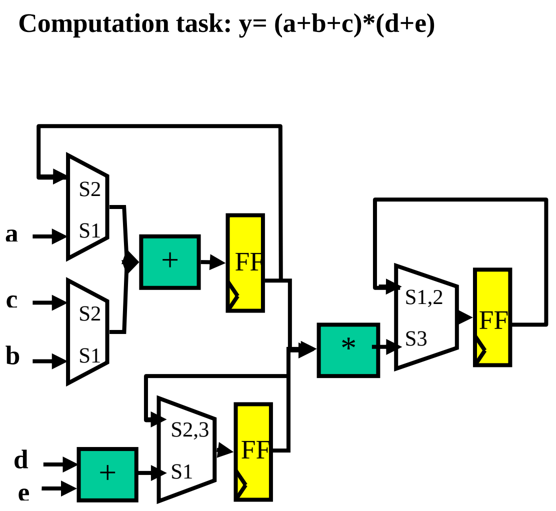
RTL Synthesis. The next step after behavioral synthesis is RTL synthesis. RTL synthesis performs optimizations on the register-transfer-level design. Input to an RTL synthesis tool is a Verilog or VHDL design that includes the number of datapath components, the binding of operations/variables/transfers to datapath components, and a controller that contains the detailed schedule of computational, I/O, and memory operations. In general, an RTL synthesis tool would use a front-end parser to parse the design and generate an intermediate representation of the design. Then, the tool can traverse the intermediate representation and create a netlist that consists of typical circuit sub-structures including memory blocks, *if* and *case* blocks, arithmetic operations, and registers, etc. Next, synthesis and optimization can be performed on this netlist, which can include examining adders and multipliers for constants, operation sharing, expression optimization, collapsing multiplexers, and re-encoding finite state machines for controllers, etc. Finally, an inferencing stage can be invoked to search for structures in the design that could be mapped to specific arithmetic units, memory blocks, registers and other types of logic blocks from an RTL library. The output of the RTL synthesis provides such a mapped netlist. For the controller and glue logic, generic Boolean networks can be generated. RTL synthesis may need to consider the target IC technologies. For example, if the target IC technology is PLD, the regularity of PLD logic fabric offers opportunities for directly mapping datapath components to PLD logic blocks, producing regular layout, and reducing chip delay and synthesis runtime [36]. There are interesting research topics for further study in RTL synthesis, such as retiming for glitch power reduction, resource sharing for multiplexer optimization, and layout-driven RTL synthesis, just to name a few.



(a) Scheduling solution

Op1 (t1=a+b) → ADD1
Op2 (t2=d+e) → ADD2
Op3 (t3=t1+c) → ADD1
Op4 (y=t2*t3) → MUL1

(b) Binding solution



(c) Final datapath

Figure 4: A behavioral synthesis example derived from [26]

Logic Synthesis. Logic synthesis is the task of generating a structural view of the logic-level implementation of the design. It can take the generic Boolean network generated from the RTL synthesis and perform logic optimization on top of it. Such optimizations include both sequential logic optimization and combinational logic optimization. Typical sequential optimization includes finite-state machine encoding/minimization and retiming for the controller, and typical combinational optimization includes constant propagation, redundancy removal, logic network restructuring and optimization, and don't-care based optimizations. Such optimizations can also be carried out either in a general sense or targeting a specific IC technology. General optimization is also called *technology-independent optimization* with objectives like minimizing the total amount of gates or reducing the logic depth of the Boolean network. Famous examples include the two-level logic minimizor ESPRESSO [37], sequential circuit optimization system SIS [38], BDD (binary decision diagram)-based optimizations [39], and SAT (satisfiability)-based optimizations [40]. Logic optimization targeting a specific IC technology is also called *technology-dependent optimization*. The main task in this type of optimization is *technology mapping*, which transforms a Boolean network into an interconnection of logic cells provided from a cell library. Figure 5 demonstrates an example of mapping a Boolean network into an FPGA. In Figure 5, each subcircuit in the dotted box is mapped into a 3-input LUT. Representative works in technology mapping include DAGON [41], FlowMap [42], ABC [43], and others (e.g. [44][45][46]). Logic synthesis is a critical step in the design flow. Although this area in general is pretty mature, new challenges need to be addressed such as fault-aware logic synthesis and logic synthesis considering circuit parameter variations. Synthesis for specific design constraints is also challenging. One example is synthesis with multiple clock domains and false paths [47].[6]

Partitioning and floorplan. We now get into the domain of physical design (Figure 3). The input of physical design is a circuit netlist, and the output is the layout of the circuit. Physical design includes several stages such as *partitioning*, *floorplan*, *placement*, and *routing*. Partitioning is usually required for multi-million-gate designs. For such a large design, it is not feasible to layout the entire chip in one step due to the limitation of memory and computation resources. Instead, the circuit will be first partitioned into sub-circuits (blocks), and then these blocks can go through a process called floorplan to set up the foundation of a good layout. A disadvantage of the partitioning process, however, is that it may degrade the performance of the final design if the components on a critical path are distributed into different blocks in the design [48]. Therefore, setting timing constraints are important for partitioning. Meanwhile, partitioning should also work to minimize the total number of connections between the blocks to reduce global wire usage and interconnect delay. Representative partitioning works include Fiduccia-Matheysses (FM) partitioning [49] and hMETIS [50]. Other works (e.g. [51][52]) are well known as well.

Floorplan will select a good layout alternative for each block and for the entire chip as well. Floorplan will consider the area and the aspect ratio of the blocks, which can be estimated after partitioning. The number of terminals (pins) required by each block and the nets[7] used to connect the blocks are also known after partitioning. In order to complete the layout, we need to determine the shape and orientation of each block and place them on the surface of the layout. These blocks should be placed in such a way to reduce the total area of the circuit. Meanwhile, the pin-to-pin wire delay needs to be optimized. Floorplan also needs to consider whether there is sufficient routing area between the blocks so that the routing

[6] False paths will not be activated during normal circuit operation, and therefore can be ignored. Multi-cycle paths refer to signal paths that carry a valid signal every few clock cycles, and therefore have a relaxed timing requirement.

[7] The net is an important concept in physical design. It represents one wire (or a group of connected wires) that connect a set of terminals (pins) so these terminals will be made electrically equivalent.

algorithms can complete the routing task without experiencing routing congestions. Partitioning and floorplan are optional design stages. They are required only when the circuit is highly complex. Usually, physical design for PLDs can skip these two stages.[8] Some floorplanning works are [53]-[56].

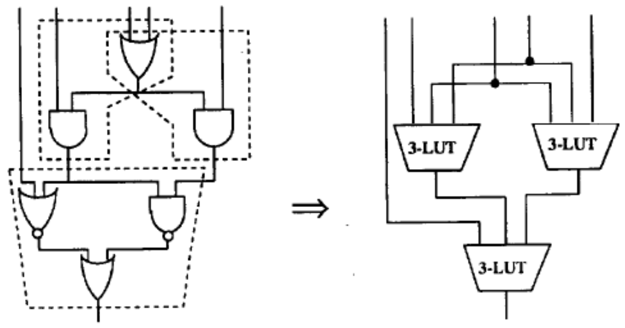


Figure 5: An example of technology mapping for FPGAs [42]

Placement. Placement is a key step in the physical design flow. It deals with the similar problem as floorplan — determining the positions of physical objects (logic blocks and/or logic cells) on the layout surface. The difference is that in placement, we can deal with a large number of objects (up to millions of objects) and the shape of each object is predetermined and fixed. Therefore, placement is a scaled and restricted version of the floorplan problem and is usually applied within regions created during floorplanning. Placement has a significant impact on the performance and routability of a circuit in nanometer design because a placement solution, to a large extent, defines the amount of interconnects, which have become the bottleneck of circuit performance. Figure 6 shows a simple example of a placement problem [48]. It shows two different placements for the same problem. The wire congestion in Figure 6(a) is much less than that in Figure 6(b). Thus, the solution in 6(a) can be considered more easily routable than that in 6(b). In placement, several optimization objectives may contradict each other. For example, minimizing layout area may lead to increased critical path delay and vice versa. Placement, like most of other physical design tasks, is an NP-hard problem and hence, the algorithms used are generally heuristic in nature. Because of the importance of placement, extensive amount of research has been carried out in the CAD community. Placement algorithms can be mainly categorized into simulated annealing-based (e.g., [57][58]), partitioning-based (e.g., CAPO [59]), analytical placement (e.g., BonnPlace [60]), and multilevel placement (e.g., mPL [61]). Some other well known placers include FastPlace [62], grid warping [63], Dragon [64], NTUplace [65], and APlace [66]. In general, for small placement instances (<50K movable objects), simulated annealing is successful. For routability-driven placement of mid-size instances (up to100-200K movable objects), partitioning-based placement does well. However, for large instances (above 200K and into the millions), especially with many fixed pins/IP blocks, analytical and multilevel placement methods are the most successful.

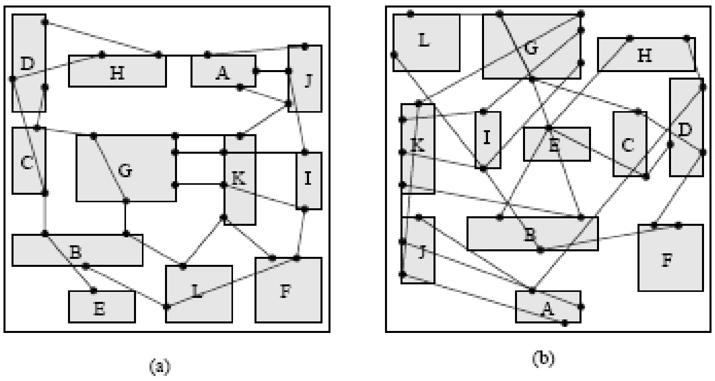


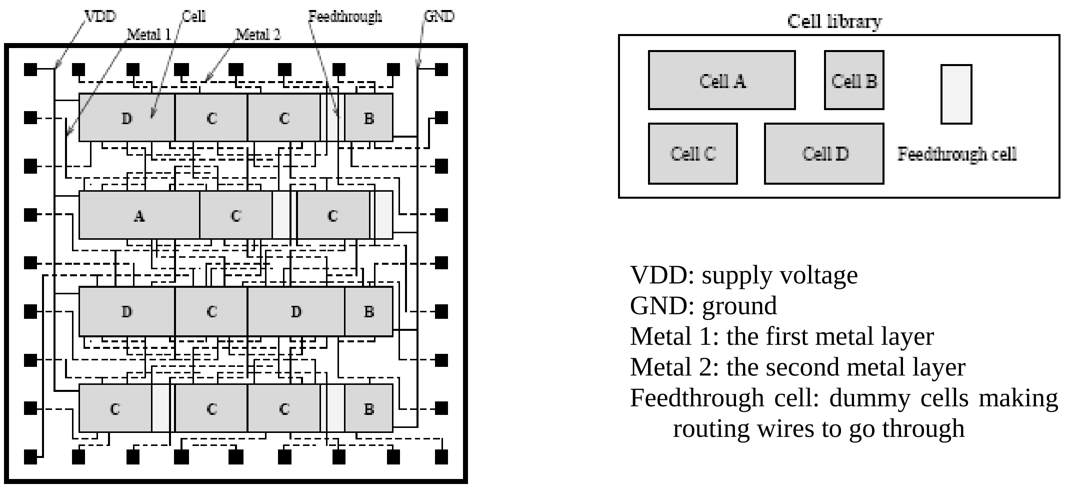
Figure 6: Two different placements of the same problem [48]

It is worth mentioning that there is research on placement/floorplan-driven synthesis (also called *physical synthesis*). Once placement is available, interconnects are defined and may become a performance bottleneck. These interconnect delay values however can be fed back to the synthesis stages so that further optimization can be carried out in the presence of interconnect delays, including operations rebinding, logic restructuring, and remapping, etc. After such operations, an incremental placement step is needed to finalize the placement again given the new synthesis results. Placement-driven optimization is optional, but may improve design performance considerably.

[8] The PLD design flow, especially for hierarchical-structured FPGAs, would require a clustering design stage after the technology mapping stage. The clustering stage would gather groups of LUTs into logic blocks (e.g, each logic block contains 10 LUTs). The netlist of logic blocks is then fed to a placement engine to determine the locations of the logic blocks on the chip.

Routing. After placement, the routing stage determines the geometric layouts of the nets to connect all the logic blocks and/or logic cells together. Routing is the last step in the design flow before either creating the GDSII[9] file for fabrication in the semicustom/ASIC design style or generating the bit-stream to program the PLD (Figure 3). The objective of routing can be reducing the total wire length, minimizing the critical path delay, minimizing power consumption, or improving manufacturability, etc. A deep-submicron VLSI chip may contain tens of millions of gates. As a result, millions of nets have to be routed while each net may have hundreds of possible routes. This makes the routing problem computationally expensive and hard. To deal with such complexity, current solution is to divide routing into two phases: *global routing* and *detailed routing*. Global routing will generate a coarse route for each net, which basically assigns routing regions to each net without specifying the actual geometric layout of the net. Detailed routing then finds the actual geometric layout of each net within the assigned regions.

The global routing problem is typically studied as a graph problem with different graph models, including grid model, checkerboard model, and channel intersection model. Also, there are two kinds of approaches to solve global routing problem: *sequential* and *concurrent*. The sequential approach routes the nets one by one following an order determined by some criteria such as the nets' criticality or number of terminals. Some important algorithms include maze routing [67], line-probe [68], shortest path-based, negotiation-based [69], and Steiner tree-based [70] algorithms. The concurrent approach avoids the ordering problem by considering routing of all the nets simultaneously. It usually follows a hierarchical partitioning of the problem instance into smaller sub-instances, which can be solved by integer programming. Detailed routing is usually solved incrementally by routing a net one region at a time in a predefined order considering number of terminals, net width and type, pin locations, via restrictions, and number of metal layers, etc. There have been extensive amount of research published for routing algorithms [12][48][71]. Figure 7 illustrates the final layout of a simple standard cell design.



VDD: supply voltage
GND: ground
Metal 1: the first metal layer
Metal 2: the second metal layer
Feedthrough cell: dummy cells making space for routing wires to go through

Figure 7: The final layout of a simple standard cell-based design [48]

2.3 Verification and Testing

The major design steps in Section 2.2 focus on design implementation. Implementation is a transformation process, which converts a design from a more abstract level into a lower, more detailed level. Verification, on the other hand, is the task to verify that such a transformation is done correctly. Also, due to manufacturing imperfections, each fabricated chip needs to go through a testing procedure to make sure it is functioning as desired. Verification and testing are essential for the timely delivery of correct ICs. These steps can occupy more than half of the total design time. We briefly introduce these topics here. Interested readers can refer to [3][12][13] for further details.

2.3.1 Verification

Ideally, verification should be carried out after each implementation step to catch any design errors. Otherwise, the errors can propagate to the lower design levels and may eventually lead to faulty manufacturing masks, which then requires a design respin and generates a large cost overhead. Verification can be carried out in several different ways, namely *simulation*, *formal verification*, *emulation*, and *post-silicon validation*.

Simulation. Simulation uses mathematical models to simulate the behavior of an actual electronic device or circuit. In general, people obtain typical input vectors (stimuli), track the propagation of these input values through the circuit, and check whether the simulated outputs are identical to the intended outputs of the circuit. Once mismatches are identified, the errors need to be localized and fixed. Simulations can be carried out at different levels (e.g., system level, RT level, or gate level). Usually, one cannot afford exhaustive simulation using all the input vectors because it would be very slow (the number of input vectors is an exponential function of the number of total inputs). Thus in practice, only a subset of

[9] GDSII is a database file format used as the industry standard for IC layout data exchange.

all the input vectors is used. As a result, how to select such a subset becomes extremely important — only the most relevant vectors should be selected for the maximum simulation coverage[10], which then leads to high fault coverage. However, the downside of this compromise is that some corner case design errors may remain undetected if testing vectors are not properly selected.

Simulation is also widely used to analyze the timing of the circuit, especially for analog and mixed-signal circuits. Simulation-based timing analysis is able to consider the correlation among the circuits' inputs and avoid timing analysis hurdles due to false paths. However, the main concern of this approach is its runtime complexity, especially for large and complex digital circuits. A popular replacement is the static timing analysis (STA), which is carried out in an input-independent manner and tries to find the worst-case delay of the circuit over all possible input combinations. The computational complexity is liner in the number of edges in the circuit netlist. However, due to the *static* feature of STA, it is vulnerable to false paths, which the STA may treat as critical, but in reality they may never be sensitized. A series of works have been dedicated to deal with this problem [72][73][74].

Formal Verification. Instead of simulation, formal verification strives to prove the correctness of a circuit implementation using formal methods of mathematics. Used correctly, this method can decrease the verification time as well as guarantee the correctness. However, due to the intrinsic difficulty of the problem, this proof-based method cannot scale to very large designs. There are two main techniques for formal verification, namely *Model Checking* and *Equivalence Checking*. Model checking verifies that a design satisfies certain properties. It generally requires that the designer knows what desired properties the design should have. Equivalence checking compares two implementations (one of them is known to be correct) and proves that they are functionally equivalent. For example, two Boolean functions can be compared using their ROBDD (reduced ordered binary decision diagram) representations, and two finite-state machines can be compared using their state diagrams.

Emulation and Post-silicon Validation. Emulation is to implement a prototype of the design using a PLD (Section 1.1). Once the prototype is completed, the verification process applies input vectors to the PLD and compares its outputs with the intended outputs. This is similar to simulation in principle, but is much faster due to the significant hardware acceleration effect provided by the hardware resources in the PLD. Thus, designers can afford to operate on more input vectors for better verification results. A drawback of this approach is that for each design, designers need to spend time and effort to come up with its prototype upfront before emulation starts (system-level and behavior-level design automation can speed up this process). Post-silicon validation uses actual fabricated chips and tests them at full speed. This method is obviously the fastest but is also the most expensive, among all the verification methods.

2.3.2 Testing

Verification works on the functional or timing aspect of the design before manufacturing. Even if the chip is without design errors, hardware defects may occur during the manufacturing process. A number of factors such as optical proximity effects, airborne impurities, and processing material flaws during fabrication can result in defective transistors and interconnect. There are also hard errors caused by sophisticated mechanisms, such as those caused by antenna, thermal and inductive effects. Thus, testing after manufacturing is essential. Since each individual chip must be tested, testing can be very time consuming and expensive. Testing can be also challenging because the behavior of a million-gate VLSI chip has to be tested using only a small number of pads (e.g., < 100).

People have identified fault models for testing purposes. The best-known is the model of *stuck at* faults, where a fault causes the signal of a wire to be fixed at the value 0 or at the value 1. With this model, one tries to observe the faults' effect on the circuit behavior instead of directly detecting a physical defect. The main issues and techniques involved with testing include the following. 1) ATPG (automatic test pattern generation): This technique automatically selects high-quality test vectors to minimize the testing time of each chip on the tester. 2) Controllability and observability of signals[11]: Testing ideally would check every gate in the circuit to prove it is not stuck. Therefore, it is desirable to design the chip to increase gate observability and controllability. 3) Scan chain: Convert each flip-flop to a scan register, which has normal mode and scan mode. In the scan mode, specific values can be scanned in for testing purposes. 4) BIST (build-in self-test): Circuits test themselves. The circuit contains extra testing circuitry, which in the testing mode generates input vectors to test the circuit by itself.

2.4 Technology CAD

Technology computer-aided design (TCAD) is an important branch in CAD, which carries out numeric simulations of semiconductor processes and devices. Process TCAD takes a process flow, including essential steps such as ion implantation, diffusion, etching, deposition, lithography, oxidation, and silicidation, and simulates the active dopant distribution, the stress distribution, and the device geometry. Mask layout is also an input for process simulation. The layout can be selected as a linear cut in a full layout for a two-dimensional simulation or a rectangular cut from the layout for a three-dimensional simulation. Process TCAD produces a final cross-sectional structure. Such a structure is then

[10] A percentage reporting the ratio of output ports actually toggling between 1 and 0 during simulation, compared to the total number of output ports present in the circuit.

[11] *Controllability*: ease of forcing an internal logic gate to 0 or 1 by driving input pins of the chip. *Observability*: ease of observing an internal logic gate by watching external output pins of the chip.

provided to device TCAD for modeling the device electrical characteristics. The device characteristics can be used to either generate the coefficients of compact device models or develop the compact models themselves. These models are then used in circuit simulators, such as the SPICE simulator, to model the circuit behavior. Because of the detailed physical modeling involved, TCAD is mostly used to aid the design of single devices.

TCAD has become a critical tool in the development of next-generation IC processes and devices. The reference [75] summarizes applications of TCAD in four areas. 1) Technology Selection: TCAD tools can be used to eliminate or narrow technology development options prior to starting experiments. 2) Process Optimization: Tune process variables and design rules to optimize performance, reliability, cost, and manufacturability. 3) Process Control: Aid the transfer of a process from one facility to another (including from development to manufacturing) and serve as reference models for diagnosing yield issues and aiding process control in manufacturing. 4) Design Optimization: Optimize the circuits for cost, power, performance, and reliability. The challenge for TCAD is that the physics and chemistry of fabrication processes are still not well understood. Therefore, TCAD cannot replace experiments except in very limited applications so far. It is worth mentioning that electromagnetic field solvers are considered as part of TCAD as well. These solvers solve Maxwell's equations, which govern the electromagnetic behavior, for the benefits of IC and PCB design. For example, one objective is to help account accurately for parasitic effects of complicated interconnect structures.

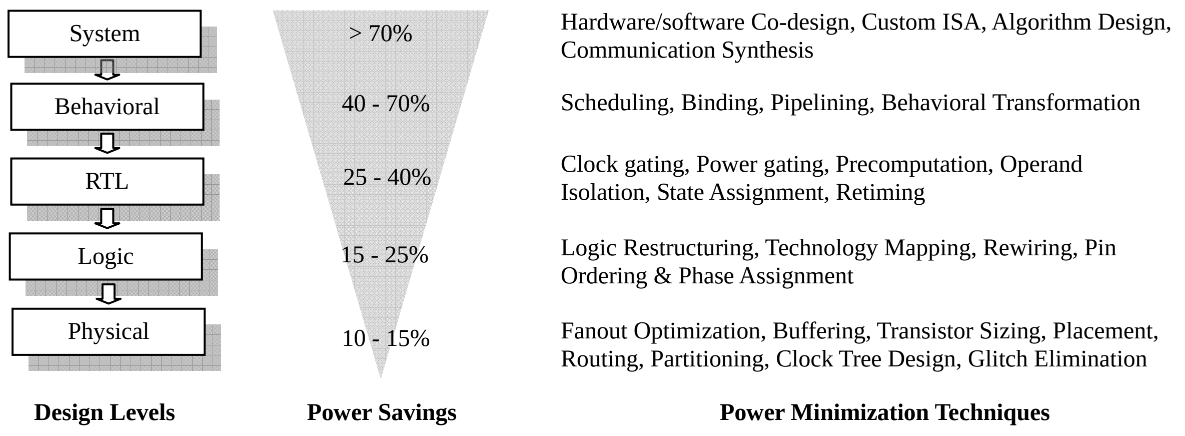

System
Behavioral
RTL
Logic
Physical
> 70%
40 - 70%
25 - 40%
15 - 25%
10 - 15%
Hardware/software Co-design, Custom ISA, Algorithm Design, Communication Synthesis
Scheduling, Binding, Pipelining, Behavioral Transformation
Clock gating, Power gating, Precomputation, Operand Isolation, State Assignment, Retiming
Logic Restructuring, Technology Mapping, Rewiring, Pin Ordering & Phase Assignment
Fanout Optimization, Buffering, Transistor Sizing, Placement, Routing, Partitioning, Clock Tree Design, Glitch Elimination
Design Levels
Power Savings
Power Minimization Techniques

Figure 8: Power Saving Opportunities on Different Design Levels [80]

2.5 Design for Low Power

With the exponential growth of the performance and capacity of integrated circuits, power consumption has become one of the most constraining factors in the IC design flow. There are three power sources in a circuit: 1) *switching power*, 2) *short-circuit power*, and 3) *static* or *leakage power*. The first two types of power can only occur when a signal transition takes place at the gate output; together they are called *dynamic power*. There are two types of signal transitions: one is the signal transition necessary to perform the required logic functions; the other is the unnecessary signal transition due to the unbalanced path delays to the inputs of a gate (called *spurious transition* or *glitch*). Static power is the power consumption when there is no signal transition for a gate. As technology advances to feature sizes of 90*nm* and below, static power starts to become a dominating factor in the total chip power dissipation. Design for low power is a vast research topic involving low-power device/circuit/system architecture design, device/circuit/system power estimation, and various CAD techniques for power minimization [76][77][78][79]. Power minimization can be performed in any of the design stages. Figure 8 shows the power saving techniques and power saving potentials during each design level [80]. Note that some techniques are not unique to only one design level. For example, glitch elimination and retiming can be applied to logic-level design as well.

3. New Trends

Due to technology scaling, nanoscale process technologies are fraught with non-idealities such as process variations, noise, soft errors, leakage and others. Designers are also facing unprecedented design complexity due to these issues. CAD techniques need new innovations to continue to deliver high-quality IC designs in a short period of time. Under this vision, we introduce some new trends in CAD below.

Design for manufacturing (DFM). Nanometer IC designs are deeply challenged by manufacturing variations. The industry is currently using the 193*nm* photo lithography for the fabrication of ICs in 130*nm* and down (to 32*nm* or even 22*nm*). Therefore, it is challenging for the photo lithography process to precisely control the manufacturing quality of the circuit features. There are other manufacturing/process challenges, such as topography variations, random defects due to missing/extra material, and via void/failure, etc. DFM will take the manufacturing issues into the design process to improve the circuit manufacturability and yield. The essential task in DFM is the development of resolution enhancement techniques (RETs), such as tools for optical proximity correction (OPC) and phase-shift mask (PSM) [81]-[85]. As an example, Figure 9 shows the OPC optimization for a layout, which manipulates mask geometry to compensate for image distortions. Another area is to develop efficient ECO (engineering change order) tools, so that when some changes need to be made, as few layers as possible need to be modified [86][87]. Meanwhile, post-silicon debug and repair techniques are gaining importance as well [88][89].

Statistical static timing analysis (SSTA). Large variation in process parameters makes worst-case design too expensive in terms of power and delay. Meanwhile, nominal case design will result in a loss in yield as performance specifications may not be met for a large percentage of chips. SSTA is an effort to specifically improve performance yield to combat manufacturing variations. SSTA treats the delay of each gate as a random variable and propagates gates' probability density functions (*pdfs)* through the circuit to create a *pdf* of the output delay random variable. Spatial correlations among the circuit components need to be considered. A vast amount of research has been reported (e.g., [90]-[98]) in the past five years. SSTA is critical to guide statistical design methodologies. An important application is SSTA-driven placement and routing to improve the performance yield.

Design for nanotechnology. Sustained exponential growth of complex electronic systems will require new breakthroughs in fabrication and assembly with controlled engineering of nanoscale components. *Bottom-up* approaches, in which integrated functional device structures are assembled from chemically synthesized nanoscale building blocks (so-called nanomaterials), such as *carbon nanotubes*, *nanowires*, and other molecular electronic devices, have the potential to revolutionize the fabrication of electronic systems. Nanoelectronic circuits always have a certain percentage of defects as well as nanomaterial-specific variations over and above process variations introduced by lithography. Using simplified nanodevice assumptions and traditional scaled design flows will lead to suboptimal and impractical nanocircuits designs and inaccurate system evaluation results. For nanotechnology to fulfill its promise, there is a need to understand and incorporate nano-specific design techniques, such as nanosystems modeling, statistical approaches and fault tolerant design, systematically from devices all the way up to systems. Initial effort has been made in this important area [99]-[107], but much more research has to be done to enable the large integration capability of nanosystems.

Design for 3D ICs. One promising way to improve circuit performance, logic density or power efficiency is to develop three-dimensional integration, which increases the number of active die layers and optimizes the interconnect network vertically [108]-[115]. Potentially, 3D IC provides improved bit bandwidth with reduced wire length, delay and power. There are different bonding technologies for 3D ICs, including die-to-die, die-to-wafer, and wafer-to-wafer, and the two parties can be bonded face-to-face or face-to-back. One disadvantage of the 3D IC is its thermal penalty. The 3D stacks will increase heat density, leading to degraded chip performance and reliability if not handled properly.

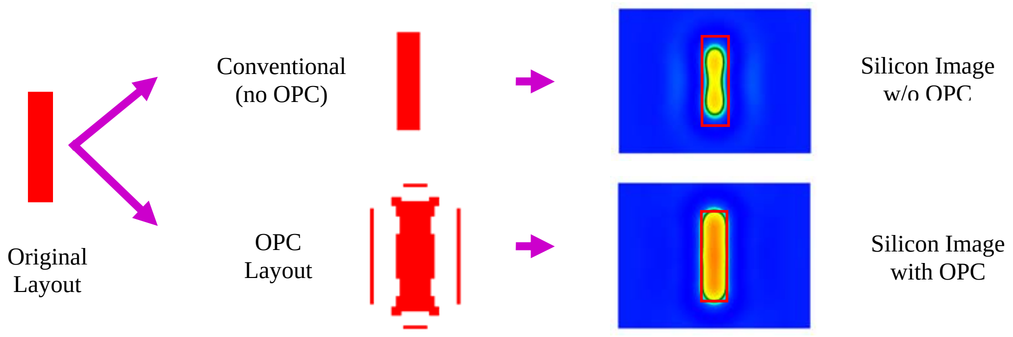


Figure 9: An illustration for optical proximity correction [85]

Design for reliability. Besides the fabrication defects, soft errors and aging errors have emerged as the new sources of circuit unreliability for nanometer circuit designs. A soft error occurs when a cosmic particle, such as a neutron, strikes a portion of the circuit upsetting the state of a bit. Aging errors are due to the wearing effect of an operating circuit. As device dimensions scale down faster than the supply voltage [9], the resulting high electric fields combined with temperature stresses lead to device aging and hence failure. Especially, transistor aging due to NBTI (negative bias temperature instability) has become the determining factor in circuit lifetime. Reliability analysis and error mitigation techniques under soft errors, aging effects, and process variations have been proposed [116]-[126]. Ultimately, chip reliability would need to become a critical design metric incorporated into the main-stream CAD methodologies.

Design with parallel computing. An important way to deal with the design complexity is to take advantage of the latest advances of parallel computing with multicore computer systems so computation can be carried out in parallel for acceleration. Although there were some studies on parallel CAD algorithms (e.g., [127][128]), much more work is needed to come up with parallel CAD algorithms to improve design productivity.

Design for NoC (network on chip). The increasing complexity and heterogeneity of future SoCs prompt significant system scalability challenge using conventional on-chip communication schemes, such as the point-to-point (P2P) and the bus-based communication architectures. NoC emerged recently as a promising solution for the future [129]-[133]. In a NoC system, modules such as processor cores, memories, and other IP blocks exchange data using a network on a single chip. NoC communication is constructed from a network of data links interconnected by switches (or routers) such that messages can be relayed from any source module to any destination module. Because all links in the NoC can operate simultaneously on different data packets, a high level of parallelism can be achieved with a great scaling capability. However, many challenging research problems remain to be solved for NoC, from the design of the physical link through the network-level structure, and all the way up to the system architecture and application software.

4. Conclusions

Electronic design automation or computer-aided design as an engineering field has been evolving through the past several decades since its birth shortly after the invention of integrated circuits. On the one hand, it has become a mature engineering area to provide design tools for the electronic semiconductor industry. On the other hand, many challenges and unsolved problems still remain in this exciting field as on-chip device density continues to scale. As long as electronic circuits are impacting our daily lives, design automation will continue to diversify and evolve to further facilitate the growth of semiconductor industry and revolutionize our future.

References

[1] F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, Wiley, 2002.

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

[3] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective (3rd Edition)*, Addison Wesley, 2004.

[4] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System Level Design: Orthogonalization of Concerns and Platform-Based Design,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 19, No. 12, Dec. 2000.

[5] A. Sangiovanni-Vincentelli and G. Martin, “A Vision for Embedded Systems: Platform-Based Design and Software Methodology,” *IEEE Design and Test of Computers*, Vol 18, No 6, pp. 23-33, 2001.

[6] G. Martin and H. Chang, *Winning the SoC Revolution: Experiences in Real Design*, Kluwer, 2003.

[7] I. Kuon and J. Rose, “Measuring the Gap between FPGAs and ASICs” *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 26, NO. 2, 2007.

[8] D. Orecchio, “FPGA Explosion will Test EDA,” *Electronic Design Update*, June 18, 2007.

[9] International Technology Roadmap for Semiconductors, *http://www.itrs.net/*.

[10] G.G. E. Gielen and R.A. Rutenbar, “Computer-aided design of analog and mixed-signal integrated circuits,” *Proceedings of the IEEE*, Vol. 88, Issue 12, Dec. 2000.

[11] P. Wambacq, G. Vandersteen, J. Phillips, J. Roychowdhury, W. Eberle, B. Yang, D. Long, A. Demir, “CAD for RF circuits,” *Design, Automation and Test in Europe*, 2001.

[12] L. Scheffer, L. Lavagno, and G. Martin, (editors), *Electronic Design Automation For Integrated Circuits Handbook* - 2 Volume Set, CRC, 1 edition, 2006.

[13] D. Jansen et al. (editors), *The Electronic Design Automation Handbook*, Kluwer Academic Publishers, 2003.

[14] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar (editors), *The Handbook of Algorithms for VLSI Physical Design Automation*, CRC Press, 2007.

[15] S. Kumar, J. Aylor, B.W. Johnson, and W. A. Wulf, *The Codesign of Embedded Systems: A Unified Hardware/Software Representation*, Kluwer Academic Publishers, 1996.

[16] Y. T. Li and S. Malik, *Performance Analysis of Real-Time Embedded Software*, Kluwer Academic Publishers, 1999.

[17] G. De Micheli, R. Ernst, and W. Wolf, eds., *Readings in Hardware/Software Co-Design*, Morgan Kaufmann, 2001.

[18] F. Balarin, H. Hsieh, L. Lavagno, C. Passerone, A. Pinto, A. Sangiovanni-Vincentelli, Y. Watanabe, and G. Yang, “Metropolis: A Design Environment for Heterogeneous Systems,” Chap. 16, A. Jerraya and W. Wolf, eds., *Multiprocessor Systems-on-Chips*, Morgan Kaufmann, 2004.

[19] R. P. Dick and N. K. Jha, “MOCSYN: Multi-objective core-based single-chip system synthesis,” *IEEE Design Automation and Test in Europe Conf.*, Feb. 1999.

[20] P. Petrov and A. Orailoglu, “Tag compression for low power in dynamically customizable embedded processors,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(7), pp. 1031-1047, 2004.

[21] S. P. Levitan and R. R. Hoare, “Structural Level SoC Design Course,” (lecture notes), *The Technology Collaborative*, Pittsburgh, PA, 1991.

[22] B. Bailey, G. Martin, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*, Morgan Kaufmann/Elsevier, 2007.

[23] K. Wakabayashi and T. Okamoto, “C-based SoC design flow and EDA tools: an ASIC and system vendor perspective,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 19(12), pp.1507-1522, Dec. 2000.

[24] D. Gajski, N. Dutt, and A.Wu, *High-level synthesis: Introduction to chip and system design*. Kluwer Academic Publishers, 1992.

[25] A. Raghunathan, N. K. Jha, and S. Dey, *High-level power analysis and optimization*. Kluwer Academic Publishers, 1998.

[26] J. P. Elliott, *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design*, Kluwer, 1999.

[27] R. Camposano and W. Wolf, *High-level VLSI synthesis*. Springer-Verlag New York, 2001.

[28] J. Chang and M. Pedram, *Power Optimization and Synthesis at Behavioral and System Levels Using Formal Methods*, Kluwer Academic Publishers, Boston, 1999.

[29] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Brodersen, “Hyper-LP: A System for Power Minimization Using Architectural Transformations,” *The Best of ICCAD, 20 Years of Excellence in Computer-Aided Design*, A. Kuehlman Ed., Kluwer Academic Publishers, 2003.

[30] S. Gupta, N. D. Dutt, R. Gupta, and A. Nicolau, *SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits*, Kluwer Academic Publishers, Norwell, MA, 2004.

[31] S. Memik, E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh, "A Scheduling Algorithm for Optimization and Planning in High-level Synthesis," *ACM Trans. on Design Automation of Electronic Systems*, Vol. 10, No. 1, January 2005.

[32] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-Level Synthesis under Multi-Cycle Interconnect Delay," *Asia and South Pacific Design Automation Conf.*, Jan. 2001.

[33] P. Brisk, A. Verma, and P. Ienne, "Optimal polynomial-time interprocedural register allocation for high-level synthesis and ASIP design," *Intl. Conf. on Computer Aided Design*, 2007.

[34] D. Chen, J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, "xPilot: A Platform-Based Behavioral Synthesis System," *Proceedings of SRC Techcon Conf.*, 2005.

[35] F. Wang, X. Wu, and Y. Xie, "Variability-Driven Module Selection with Joint Design Time Optimization and Post-Silicon Tuning," *Asia-South Pacific Design Automation Conf.*, Jan. 2008.

[36] T. J. Callahan, P. Chong, A. DeHon, and J. Wawrzynek, "Fast Module Mapping and Placement for Datapaths in FPGAs," *Intl. Symp. on FPGAs*, 1998.

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

[38] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," *Memo. UCB/ERL M92/41*, Univ. of California at Berkeley, 1992.

[39] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, Vol. C-35, No. 8, pp.677-691, Aug. 1986.

[40] J. Marques Silva and K. Sakallah, "Boolean satisfiability in electronic design automation," *Design Automation Conf.*, 2000.

[41] K. Keutzer. "DAGON: Technology mapping and local optimization," *IEEE/ACM Design Automation Conf.*, June 1987.

[42] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1-12, January 1994.

[43] Berkeley ABC, *A System for Sequential Synthesis and Verification*, http://www.eecs.berkeley.edu/~alanmi/abc/.

[44] C-W. Kang, A. Iranli, and M. Pedram, "A synthesis approach for coarse-grained, antifuse-based FPGAs," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 26, No. 9, pp. 1564-1575, 2007.

[45] A. Ling, D. Singh, and S. Brown, "FPGA PLB Architecture Evaluation and Area Optimization Techniques using Boolean Satisfiability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol 26, No 7, July 2007.

[46] A. K. Singh, M. Mani, R. Puri, and M. Orshansky, "Gain-based technology mapping for minimum runtime leakage under input vector uncertainty," *Design Automation Conf.*, 2006.

[47] L. Cheng, D. Chen, D. F. Wong, M. Hutton, and J. Govig, "Timing Constraint-driven Technology Mapping for FPGAs Considering False Paths and Multi-Clock Domains," *Intl. Conf. on Computer-Aided Design*, Nov. 2007.

[48] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1999.

[49] C. M. Fiduccia and R. M. Matheysses, "A linear-time heuristic for improving network partitions," *IEEE/ACM Design Automation Conf.*, pp.175–181, 1982.

[50] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI Domain," *IEEE/ACM Design Automation Conf.*, 1997.

[51] Y. C. Wei and C. K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 10, pp. 911-921, July 1991.

[52] H. Liu and D. F. Wong, "Network-Flow-Based Multiway Partitioning with Area and Pin Constraints," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 17, No. 1, January 1998.

[53] L. Stockmeyer, "Optimal Orientation of Cells in Slicing Floorplan Designs," *Information and Control*, Vol. 57, Issue 2-3, 1984.

[54] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," *Design Automation Conf.* 1986.

[55] P. Sarkar and C. K. Koh, "Routability-Driven Repeater Block Planning for Interconnect-Centric Floorplanning," *IEEE Trans. on CAD of Integrated Circuits and Systems (Special Issue on Physical Design)*, 20(5), pp. 660-671, 2001.

[56] M. Healy, M. Vittes, M. Ekpanyapong, C. Ballapuram, S. K. Lim, H. Lee, and G. Loh, "Multi-Objective Microarchitectural Floorplanning for 2D and 3D ICs," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 26, No. 1, pp. 38-52, 2007.

[57] W. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 14(3), 349-359, March 1995.

[58] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1999.

[59] A. Caldwell, A. B. Kahng, and I. Markov, "Can recursive bisection produce routable placements?" *IEEE/ACM Design Automation Conf.*, pp.477–482, 2000.

[60] U. Brenner and A. Rohe, "An effective congestion-driven placement framework," *Intl. Symp. on Physical Design*, Apr 2002.

[61] T. Chan, J. Cong, T. Kong, and J. Shinnerl, "Multilevel circuit placement," *Multilevel Optimization in VLSICAD*, Kluwer, Boston, Mass., Chap. 4. 2003.

[62] C. Chu and N. Viswanathan, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model," *Intl. Symp. on Physical Design*, pp.26–33. 2004.

[63] Z. Xiu, J. Ma, S. Fowler, and R. Rutenbar, "Large-scale placement by grid warping," *Design Automation Conf.*, 2004.

[64] T. Taghavi, X. Yang, B. Choi, M. Wang, and M. Sarrafzadeh, "Dragon2006: blockage-aware congestion-controlling mixed-size placer," *Intl. Symp. on Physical Design*, 2006.

[65] T. Chen, Z. Jiang, T. Hsu, H. Chen, and Y. Chang, "NTUplace2: a hybrid placer using partitioning and analytical techniques," *Intl. Symp. on Physical Design*, April 2006.

[66] A. B. Kahng, S. Reda, and Q. Wang, "APlace: A General Analytic Placement Framework," *Intl. Symp. on Physical Design*, 2005.

[67] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. on Electronic Computers*, 1961.

[68] D. W. Hightower, "A solution to the line routing problem on a continuous plane," *Proc. of Design Automation Workshop*, 1969.

[69] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-based Performance-driven Router for FPGAs," *Intl. Symp. on FPGAs*, 1995.

[70] C. Chu and Y. C. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(1), pp. 70-83, 2008.

[71] J. Hu and S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integration: VLSI J.*, 31, 1-49, 2001.

[72] P. McGeer and R. Brayton, "Efficient algorithms for computing the longest viable path in a combinational network," *Design Automation Conf.* 1989.

[73] P. Ashar, S. Dey, and S. Malik, "Exploiting multicycle false paths in the performance optimization of sequential logic circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 14(9), 1067-1075, 1995.

[74] S. Zhou, B. Yao, H. Chen, Y. Zhu, M. Hutton, T. Collins, S. Srinivasan, N. Chou, P. Suaris, and C. K. Cheng, "Efficient Timing Analysis with Known False Paths Using Biclique Covering," *IEEE Trans. on CAD of Integrated Circuits and Systems*, May 2007.

[75] J. Mar, "The application of TCAD in industry," *Intl. Conf. on Simulation of Semiconductor Processes and Devices*, 1996.

[76] W. Nebel and J. Mermet (editors), *Low Power Design in Deep Submicron Electronics*, Springer, Dec. 1997.

[77] K. Roy and S. Prasad, *Low-Power CMOS VLSI Circuit Design*, Wiley-Interscience, Feb. 2000.

[78] M. Pedram and J. M. Rabaey, *Power Aware Design Methodologies*, Springer, 2002.

[79] R. Puri, L. Stok, J. Cohn, D. Kung, D. Pan, D. Sylvester, A. Srivastava, and S. Kulkarni, "Pushing ASIC Performance in a Power Envelope," *Design Automation Conf.*, 2003.

[80] M. Pedram, "Low Power Design Methodologies and Techniques: An Overview," *http://atrak.usc.edu/~massoud/*, Mar. 1999.

[81] P. Yu, S. X. Shi, and D. Z. Pan, "True Process Variation Aware Optical Proximity Correction with Variational Lithography Modeling and Model Calibration," *The Journal of Microlithography, Microfabrication, and Microsystems (JM3), Special Edition of Resolution Enhancement Techniques and Design for Manufacturability*, Sept. 2007.

[82] P. Berman, A. B. Kahng, D. Vidhani, H. Wang, and A. Zelikovsky, "Optimal Phase Conflict Removal for Layout of Dark Field Alternating Phase Shifting Masks," *Intl. Symp. on Physical Design*, April 1999.

[83] L. W. Liebmann, G. A. Northrop, J. Culp, and M. A. Lavin, "Layout Optimization at the Pinnacle of Optical Lithography," *SPIE Design and Process Integration for Electronic Manufacturing*, Feb. 2003.

[84] F. M. Schellenberg and L. Capodieci, "Impact of RET on Physical Layouts," *Intl. Symp. on Physical Design*, 2001.

[85] L. Huang and D. F. Wong, "Optical proximity correction (OPC): friendly maze routing," *Design Automation Conf.*, 2004.

[86] Y. M. Kuo, Y. T. Chang, S. C. Chang, and M. Marek-Sadowska, "Engineering change using spare cells with constant insertion," *Intl. Conf. on Computer Aided Design*, 2007.

[87] S. Ghiasi, "Incremental component implementation selection: enabling ECO in compositional system synthesis," *Intl. Conf. on Computer Aided Design*, 2007.

[88] A. Krstic, L. C. Wang, K. T. Cheng, and T. M. Mak, "Diagnosis-Based Post-Silicon Timing Validation Using Statistical Tools and Methodologies," *Intl. Test Conf.*, 2003.

[89] K. H. Chang, I. Markov, and V. Bertacco, "Automating post-silicon debugging and repair," *Intl. Conf. on Computer Aided Design*, 2007.

[90] S. Sapatnekar, *Timing*, Springer, 2004.

[91] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*, Springer Publishers, New York, 2005.

[92] V. Mehrotra, S. Sam, D. Boning, A. Chandrakasan, R. Vallishayee, and S. Nassif, "A methodology for modeling the effects of systematic within-die interconnect and device variation on circuit performance," *Design Automation Conf.*, 2000.

[93] M. Guthaus, N. Venkateswaran, C. Visweswariah, and V. Zolotov, "Gate sizing using incremental parameterized statistical timing analysis," *Intl. Conf. on Computer Aided Design*, 2005.

[94] J. Le, X. Li, and L. T. Pileggi, "STAC: Statistical timing analysis with correlation," *IEEE/ACM Design Automation Conf.*, 2004.

[95] M. Orshansky and A. Bandyopadhyay, "Fast statistical timing analysis handling arbitrary delay correlations," *IEEE/ACM Design Automation Conf.*, 2004.

[96] V. Khandelwal and A. Srivastava, "A General Framework for Accurate Statistical Timing Analysis Considering Correlations," *IEEE/ACM Design Automation Conf.*, 2005.

[97] A. Ramalingam, A. K. Singh, S. R. Nassif, M. Orshansky, and D. Z. Pan, "Accurate Waveform Modeling using Singular Value Decomposition with Applications to Timing Analysis," *Design Automation Conf.*, 2007.

[98] J. Xiong, V. Zolotov, and L. He, "Robust Extraction of Spatial Correlation," *Intl. Symp. on Physical Design*, 2006.

[99] J. Heath, P. Kuekes, G. Snider, and S. Williams, "A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology," *Science*, Vol. 280, pp. 1716-1721, 12 June 1998.

[100] A. DeHon, "Seven strategies for tolerating highly defective fabrication," *IEEE Design & Test of Computers*, Vol. 22, 2005.

[101] J. Deng, N. Patil, K. Ryu, A. Badmaev, C. Zhou, S. Mitra, and H. S. Wong, "Carbon Nanotube Transistor Circuits: Circuit-Level Performance Benchmarking and Design Options for Living with Imperfections," *IEEE Intl. Solid-State Circuits Conf.*, 2007.

[102] S. C. Goldstein and M. Budiu, "NanoFabric: Spatial Computing using Molecular Electronics," *Intl. Symp. on Computer Architecture*, pp. 178-189, 2001.

[103] D. B. Strukov and K. K. Likharev, "A reconfigurable architecture for hybrid CMOS/Nanodevice circuits," *Intl. Symp. on FPGAs*, 2006.

[104] A. Raychowdhury, A. Keshavarzi, J. Kurtin, V. De, and K. Roy, "Analysis of Carbon Nanotube Field Effect Transistors for High Performance Digital Logic - Modeling and DC Simulations," *IEEE Trans. on Electron Devices*, Vol. 53, Issue 11, Nov. 2006.

[105] W. Zhang, N. K. Jha, and L. Shang, "NATURE: A CMOS/nanotube hybrid reconfigurable architecture," *Design Automation Conf.*, 2006.

[106] M. Ben Jamaa, K. Moselund, D. Atienza, D. Bouvet, A. Ionescu, Y. Leblebici, and G. De Micheli, "Fault-tolerant multi-level logic decoder for nanoscale crossbar memory arrays," *Intl. Conf. on Computer Aided Design*, 2007.

[107] A. Nieuwoudt, M. Mondal, and Y. Massoud, "Predicting the Performance and Reliability of Carbon Nanotube Bundles for On-Chip Interconnect," *Asian and South Pacific Design Automation Conf.*, 2007.

[108] C. Ababei, P. Maidee, and K. Bazargan, "Exploring Potential Benefits of 3D FPGA Integration," *Field Programmable Logic and Application*, vol. 3203/2004: Springer Berlin / Heidelberg, pp. 874-880, 2004.

[109] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-D ICs: a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration," *Proceedings of the IEEE*, Vol. 89, no. 5, pp. 602-633, 2001.

[110] M. Lin, A. El Gamal, Y. C. Lu, and S. Wong, "Performance Benefits of Monolithically Stacked 3D-FPGA," *Intl. Symp. on FPGAs*, 2006.

[111] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon, "Demystifying 3-D ICs: The pros and cons of going vertical," *IEEE Design and Test of Computers*, Vol. 22, no. 6, pp. 498–510, Jun. 2005.

[112] C. Dong, D. Chen, S. Haruehanroengra, and W. Wang, "3-D nFPGA: A Reconfigurable Architecture for 3-D CMOS/Nanomaterial Hybrid Digital Circuits," *IEEE Trans. on Circuits and Systems I,* Vol. 54, Issue 11, pp. 2489-2501, 2007.

[113] Y. Xie, G. Loh, B. Black, and K. Bernstein, "Design Space Exploration for 3D Architecture," *ACM Journal of Emerging Technologies for Computer Systems*, Vol. 2. No. 2, pp.65-103, April 2006.

[114] J. Cong, Y. Ma, Y. Liu, E. Kursun, and G. Reinman, "3D Architecture Modeling and Exploration," *Intl. VLSI/ULSI Multilevel Interconnection Conf.*, Sept. 2007.

[115] M. Pathak and S. K. Lim, "Thermal-aware Steiner Routing for 3D Stacked ICs," *Intl. Conf. on Computer Aided Design*, 2007.

[116] M. Zhang and N. R. Shanbhag, "Soft error-rate analysis (SERA) methodology," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 25, No. 10, pp. 2140-2155, Oct. 2006.

[117] N. Miskov-Zivanov and D. Marculescu, "MARS-C: Modeling and Reduction of Soft Errors in Combinational Circuits," *Design Automation Conf.*, 2006.

[118] R. R. Rao, K. Chopra, D. Blaauw, and D. Sylvester, "An Efficient Static Algorithm for Computing the Soft Error Rates of Combinational Circuits," *Design, Automation and Test in Europe*, 2006.

[119] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust System Design with Built-In Soft Error Resilience," *IEEE Computer*, Vol. 38, Number 2, pp. 43-52, Feb. 2005.

[120] B. Paul, K. Kang, H. Kufluoglu, A. Alam, and K. Roy, "Impact of NBTI on the Temporal Performance Degradation of Digital Circuits," *IEEE Electron Device Letters*, Aug. 2005.

[121] D. Marculescu, "Energy Bounds for Fault-Tolerant Nanoscale Designs," *Design, Automation and Test in Europe*, Feb. 2005.

[122] W. Wang, S. Yang, S. Bhardwaj, R. Vattikonda, S. Vrudhula, F. Liu, and Y. Cao, "The Impact of NBTI on the Performance of Combinational and Sequential Circuits," *IEEE/ACM Design Automation Conf.*, 2007.

[123] W. Wu, J. Yang, S. X. D. Tan, and S. L. Lu, "Improving the reliability of on-chip caches under process variations," *Intl. Conf. of Computer Design*, 2007.

[124] A. Mitev, D. Canesan, D. Shammgasundaram, Y. Cao, and J. M. Wang, "A Robust Finite-Point Based Gate Model Considering Process Variations", *Intl. Conf. on Computer Aided Design*, 2007.

[125] S. Sarangi, B. Greskamp, and Josep Torrellas, "A Model for Timing Errors in Processors with Parameter Variation," *Intl. Symp. on Quality Electronic Design*, 2007.

[126] L. Cheng, Y. Lin, L. He, and Y. Cao, "Trace-Based Framework for Concurrent Development of Process and FPGA Architecture Considering Process Variation and Reliability," *Intl. Symp. on FPGAs*, 2008.

[127] P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design*, Prentice-Hall, Inc., 1994.

[128] A. Ludwin, V. Betz, and K. Padalia, "High-Quality, Deterministic Parallel Placement for FPGAs on Commodity Hardware," *Intl. Symp. on FPGAs*, 2008.

[129] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools*, Morgan Kaufmann, 2006.

[130] A. Jantsch and H. Tenhunen (editors), *Networks on Chip*, Kluwer, 2003.

[131] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist, "Network on a chip: An architecture for billion transistor era," *IEEE NorChip Conf.*, 2000.

[132] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. on Design Automation of Electronic Systems*, Vol.12, No.3, Aug. 2007.

[133] H. Wang, L. S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," *Intl. Symp. on Microarchitecture*, Nov. 2003.