The CHIP

Charting the EDA Roadmap

Ahmed Hemani

Welcome to "The Chip!" We conclude 2004 with an article on EDA tools as they pertain to chip design productivity in a way that draws an interesting parallel to Moore's law. We hope you find it informative and interesting.

Send us your contributions to "The Chip," and share your design tips and views with our readers. E-mail your contribution to me at ismail@ece.osu.edu.

Finally, we would like to wish you all a Happy Holiday Season and a Wonderful New Year, 2005!

Mohammed Ismail

Moore's law has been the guiding principle for defining the semiconductor roadmap ever since it was proposed in 1965. Here, we examine and propose a law that plays the role of Moore's law in the very large-scale integration (VLSI) computer-aided design (CAD) field. We also examine the difference in nature of semiconductor productivity and design productivity.

BACKGROUND

In 1965, when Gordon Moore was director of Fairchild Semiconductor's Research and Development, he observed that, for the foreseeable future, integration level in terms of number of transistors per square inch will continue to double every year. Subsequently, this pace slowed down, but the data density has continued to double every 18 months and is the present interpretation of Moore's law. This increase in the integration level is termed "semiconductor productivity"; the same area of semiconductor hosts greater functionality than before and, thereby, is getting more work done.

Design productivity is the ability to design a number of gates per designer per day. Whereas semiconductor productivity keeps increasing at a steady pace following Moore's law, according to the 2003 International Technology Roadmap for Semiconductors (ITRS 2003) [9], improvements in the design productivity is not keeping pace with the improvements in semiconductor productivity, giving rise to what is termed as "productivity gap." According to ITRS 2003, design productivity remains the number one threat to continuation of the semiconductor roadmap.

COMPLEXITY

Three factors contribute to complexity: number of elements, interconnection, and lack of pattern. Semiconductor productivity creates potential for complexity. Application demands for function and performance exploits this potential to create design complexity.

There are two known solutions to reducing design complexity: partitioning and abstraction. Partitioning strives to reduce the complexity by dividing the problems into smaller, more manageable problems. Problems become more manageable because partitioning encapsulates elements that are heavily interconnected and have functional coherence into new higher-level nodes. These nodes introduce a hierarchy that is often multilevel and allows designers to deal with fewer elements that are less heavily interconnected.

Abstraction is the second mechanism to reduce complexity. Whereas hierarchy hides detail, abstraction lacks detail. The lack of details reduces complexity, and the missing details are synthesized, often automatically, to create a more refined design description.

Whereas partitioning and hierarchy as a mechanism to reduce complexity is largely a manual exercise, abstraction lends to automation: the missing details in the abstract design description are automatically synthesized.

Increasing abstraction has been central to evolution of the electronic design automation (EDA) tools and largely responsible for improvements in design productivity.

EDA TOOLS AND DESIGN PRODUCTIVITY

EDA tools have evolved by successively increasing the levels of abstraction used to specify the VLSI systems. Naturally, the evolution started with lower levels of abstraction.

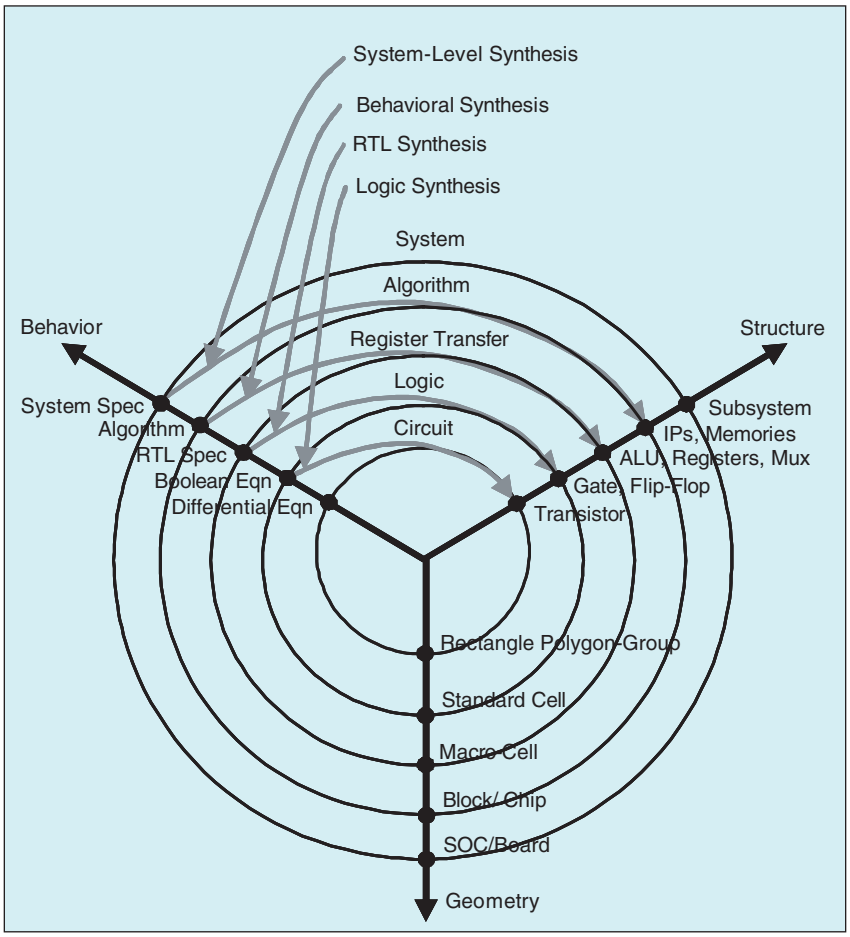
The Gajski Kuhn Y chart, shown in Figure 1, is often used to depict the taxonomy of VLSI systems and discuss EDA tools. It shows three domains as axes and concentric circles as levels of abstraction. The intersections of the domain axes and the abstraction circles are annotated with design elements used to specify the VLSI systems in the intersecting domain and abstraction. Synthesis on the Gajski Kuhn Y chart is defined as sum of three transformations:

- domain transformation, often behavior to structure but also from structure to geometry

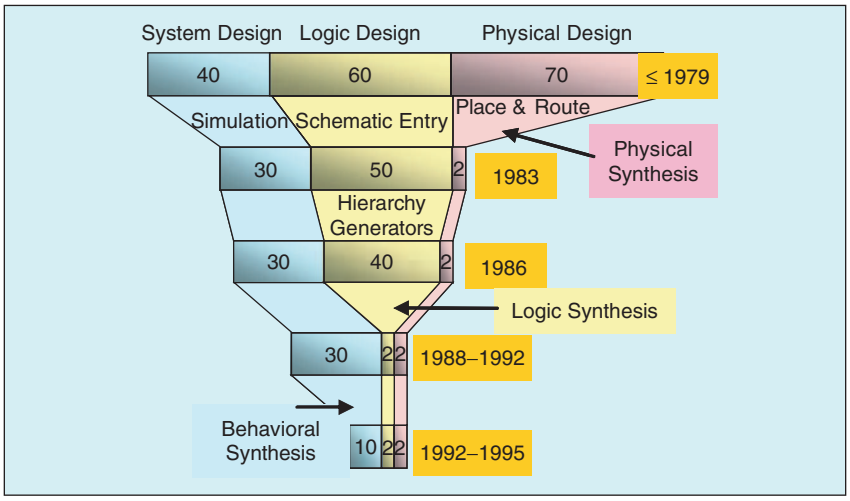
 8755-3996/04/$20.00

- refinement is transformation in abstraction from higher to lower level
- optimization is transformation in the same domain and at the same abstraction level but more optimal with respect to some objective function of complexity, power, and performance.

The Gajski Kuhn Y Chart in Figure 1 shows synthesis associated with successively higher levels of abstraction. The shaded arrows capture the domain transformation and refinement. Optimization is implied but not shown. Synthesis refines the abstract specification in terms of design elements in target domain that are predesigned components stored in a library. As granularity increases with abstraction, synthesis effectively increases the granularity of design elements that are reused. This increase in granularity with abstraction of the design elements used to specify and reused to realize VLSI systems improves design productivity, and there is some good evidence that this happens.



1. Gajski Kuhn Y Chart.



2. Reduction in design effort in person months for a 20-k gate design by raising level of abstraction.

According to [6], design effort for a 20 K gate design, divided into three activities, physical, logic, and system design, showed enormous improvements in design productivity as we successively moved from lower abstraction to higher abstraction (see Figure 2). Before 1979, physical design was the most time-consuming activity, and, with the introduction of physical design tools in 1983 and use of standard cells to abstract away the geometry, the design productivity for physical design improved dramatically.

Ten years later, in 1993, when logic/resistor-transistor logic (RTL) synthesis tools saw widespread adoption and raised the level of abstraction to RTL objects, the logic design activity saw productivity improvement comparable to that of physical design in 1983.

Published in March 1992, Michel et al. [6] forecasted that behavioral synthesis will provide a similar productivity boost for system design, as logic and physical synthesis have done in the past. Instead of behavioral synthesis, intellectual properties (IPs) have found widespread use and improved productivity. This is confirmed by ITRS 2003 [9], which states that in 2003 improvements in design technology allowed the design cost of system-on-chip low-power to be US$20 million instead of the projected US$630 million had there been no innovation in design technology. From mid 1990s to early 2000s, many improvements in design technology have contributed to improvement in design productivity, but the single most important factor has been widespread adoption of IPs as design elements.

The discussion so far has created an

impression that automatic synthesis is the only element of design technology that makes progress and improves design productivity. This is clearly not the case. The primary reason to keep the discussion synthesis centric was that abstraction and synthesis, in many respects, epitomizes a design technology node, and the secondary reason was to not clutter the discussion. As abstraction level is raised, verification tools, analysis tools, estimation and planning tools, design entry, and representation all progress, and only when all of them combine to create a mature methodology, a new design technology node is created and finds widespread adoption among the practitioners. As one would expect, before their widespread adoption, components of a new design technology node get adopted, starting with a few isolated pioneering design groups and then gaining momentum as their success stories spread together with progress in development of the missing methodology components, maturing of the existing ones, and successful integration of them in a methodology.

A comprehensive history of EDA tools is beyond the scope of this article and is covered very well by [2]–[5]. IBM has been a pioneer in the field of EDA, and its contributions has had and continues to have impact on the entire EDA industry, as presented in [2]. The evolution of EDA from an industrial perspective is presented in [3]. History, cycles in EDA industry, and future challenges was presented in an inspiring presentation as a keynote speech at 40th DAC in 2003 and subsequently published as [5].

MOORE'S EDA LAW

A pattern has been observed in the evolution of EDA tools based on raising the abstraction level. This is proposed as a law:

> "Granularity of widely reused design elements for VLSI systems increases by two orders of magnitude every decade."
>
> — *Moore's EDA Law*

Table 1. Design technology nodes according to Moore's EDA law.

Decade	Design Technology Node	Design Element	Granularity	Chip Architecture
2010s	Communication centric platform based design	Subsystems, commn. centric platforms	10^6 X	NOC
2000s	IP/Processor centric platform based design	IPs, e.g., RISC/DSP cores, USB, DSP functions	10^4 X	SOC
1990s	RTL/Logic synthesis	RTL objects, e.g., arithmetic units, registers, multiplexors	10^2 X	Algorithm on a chip
1980s	Physical synthesis	Std. cells, e.g., logic gates, flops, latches	X	
1970s	Manual	Polygons		

This was first presented in August 1998 [7] and later repeated in March 2002 [8]. At the 2003 Design Automation Conference (DAC 2003), Prof. Alberto Sangiovanni Vincentelli presented many instances of ten-year cycles in the EDA industry and also confirmed that raising abstraction is central to improving productivity and, approximately, the move to higher levels of abstraction happens at decade intervals.

To substantiate the proposed law, it is expanded backward over time to verify its correctness in light of the VLSI design automation history and used to predict the future. This is shown in Table 1, which shows a timeline of major design technology nodes at intervals of a decade and associates it with the design elements, relative granularity of the design elements, and the chip architecture style.

The 1970s

VLSI design was largely manual and the design element was polygon.

1980s

This was the first decade when a major design technology node can be said to have taken a foothold. Simple logic functions and sequential devices were the design elements realized as standard cells or in gate arrays. Standard cells and gate arrays helped abstract away the device level geometry and structured and regular placement enabled automatic synthesis of place and route details. In Table 1, these design elements are considered as base elements, and the increase in granularity of other design technology is expressed in multiples of these elements.

The 1990s

Logic synthesis in development since the 1970s saw commercialization in 1987, but the widespread adoption happened in the early 1990s when the use of hardware description languages (HDLs) for specifying VLSI systems gained popularity. Tools unifying RTL and logic synthesis emerged, and the design elements were microarchitectural RTL components, like arithmetic units, registers, multiplexors for datapath, and finite-state machines (FSMs) for control logic. Though microarchitectural components vary considerably in size, the typical gate count for an RTL design element is hundreds of gates, i.e., two orders of magnitude higher granularity compared to design elements in the previous decade.

The complexity that could be integrated in a VLSI system in much of the 1990s was algorithm(s) and characterized by hardwired computation and interconnect. Apart from some marginal reconfigurability, these systems were truly application specific integrated circuits (ASICs), alternatively they could be termed algorithm(s) on a chip.

The 2000s

Behavioral synthesis was commercialized in mid 1990s but failed to get widespread acceptance. IPs emerged in late 1990s as a solution to improve productivity. Typical IPs are reduced-instruction set computer (RISC) or digital-signal processor (DSP) cores,

DSP and communication functionalities like Viterbi, fast Fourier transform (FFT), discrete cosine transform (DCT), audio and video coders and decoders (CODECS), Ethernet Media Access Control (MAC), universal serial bus (USB), etc. Though their usage started in late 1990s, they found widespread acceptance starting from the early 2000s and have since become standard design elements. IPs vary a lot in their gate counts, but the typical figure would be of the order of tens of thousands of gates, thus upholding the proposed law.

IPs as design elements differ from the design elements used by previous technology nodes. Design elements of previous generations got instantiated in an implementation as part of automatic refinement. IPs, on the other hand, are manually instantiated.

As if to compensate for the lack of automatic refinement, IPs that are processor cores have introduced a new dimension of reuse, the programmability, making them processor-centric platforms that can be reprogrammed to implement different applications. VLSI systems with processor cores are often termed system-on-a-chip (SOC). SOCs are characterized by programmable computation, while interconnect is usually hardwired, which limits the extent to which they can be reused.

The 2010s

IPs make it possible to reuse large computational and storage functionalities. This has shifted the design bottleneck to designing the chip level harness to integrate the IPs and hardware macros that are still designed using RTL methodology. Processor-centric platforms are the first step in a more generic platform based design methodology whose essence is to separate architecture from function. The architecture consists of a hardware platform, which standardizes the rules for integrating the hardware IPs, and a software platform, which serves as the application program interface (API) for the different applications that the platform hosts. One such novel communication centric platform, called network on a chip (NOC) [13]–[16], is in early stages of development and likely to mature in a few years time and see widespread adoption sometime after that. While SOCs made computation programmable, NOCs make interconnect programmable and, thus, reusable. If the proposed law holds, early 2010 will see widespread use of multimillion gate subsystems and communication-centric platforms like NOC as reusable design elements. Early evidence of this trend can already be seen as some groups like the one at Philips Semiconductors, where the author works are already pioneering the use of large multimillion gate processor subsystems to succeed processor IPs as reusable elements. PrimeXsys from ARM, SOCit from MIPS Technologies, and CoreWare from LSI Logic are other examples that substantiate the trend towards subsystems as reusable elements. The trend is not limited to infrastructural subsystems typified by processor subsystems but also extends to functional subsystems like modems and protocol stacks for various telecom standards. The WILD product family from NewLogic for IEEE 802.11 is a good example of a functional subsystem IP. Philips Research's Aethereal [16] initiative is a serious research effort to bring the NOC concept to industrial usage. These pioneering efforts are precursors to widespread adoption of these design technology elements.

The Interconnect

As the design elements that are (re)used as building blocks increase in granularity, the input/output (I/O) of these design elements also increase in number and complexity. The I/O of an ARM processor is considerably more complex compared to that of a NAND gate realized as a standard cell. To contain the increase in complexity due to increase in complexity of I/O, the design technology and chip architectures have progressed as follows:

- When gates and flip-flops were replaced by RTL microarchitectural design elements, wires were replaced by signal vectors or buses. This was supported first by schematic capture and later by HDLs.
- When HDLs became popular, automatic synthesis of interconnect elements between design elements saw further progress as data and control flow analysis allowed synthesis tools to infer and instantiate both multiplexors and tristate buses, depending on the language constructs used.
- Interface synthesis is a neat idea to automate synthesis of interconnects among larger IP-size design elements but has unfortunately not been successfully commercialized so far [10]–[12].
- Widespread acceptance of RISC core, like ARM, has created a de facto standard for interconnecting IPs and subsystems in SOCs. These de facto standards, like the advanced microprocessor bus architecture (AMBA), enables third-party IP providers to develop IPs that can be easily plugged into such systems, thus absorbing away the interconnect-related complexity.
- Platform-based design, especially the communication-centric platform like NOC, holds potential to further increase the granularity of the interconnect elements that will be reused by making the chip level interconnect programmable.

Moore's Law Versus Moore's EDA Law

Moore's law roadmaps semiconductor productivity, and a law has been proposed to chart the design productivity. Are they related?

Moore's law, over a decade, makes available $2^{10/1.5} \approx 100$ times more devices, and, according to the proposed law, progress in EDA enables reuse of two orders of magnitude bigger design elements over a decade. In other words, the number-of-elements factor in complexity remains constant over a decade and cannot cause increase in design complexity.

The previous text substantiates that the reused interconnect elements have also similarly increased in granularity and is an unlikely explanation for the productivity gap.

EXPLAINING THE PRODUCTIVITY GAP

To understand the productivity gap, we will take a closer look at verification and differences in the nature of semiconductor and design productivity.

Verification

The last five years has seen a steady rise in verification cost of VLSI systems to the extent that, today, it is widely perceived as the bottleneck and a major source of delays in ASIC/SOC projects.

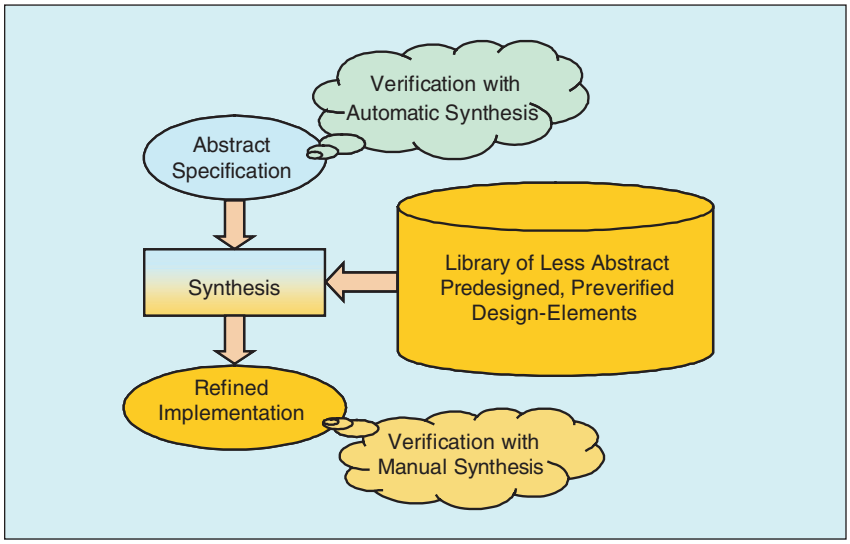
The increasing granularity of reusable design elements, both computational and interconnect, in principle, should benefit not just implementation, but also verification. The task of designing becomes easier and more productive, because the designer has to deal with fewer elements and details. Similarly, the verification task also becomes easier because the verifier only needs to verify the more abstract, less detailed specification; the reused design elements are not only predesigned, they are also preverified. Since the test bench is also written at the same level of abstraction as the abstract specification, it, too, benefits from ignoring the same details that are ignored by the specification (see Figure 3). This works because the synthesis tool is assumed to embody the correct-by-construction principle. This nice property of automatic synthesis has been broken with the use of IP and processor-centric, platform-based design technology node.

With IP-based design, the refined implementation is synthesized manually and requires verification at the more detailed level and does not benefit from the efficiency of verifying at the higher abstraction level (see Figure 3). There may be a more abstract executable specification that roughly corresponds to the more refined specification, but since it is not automatically synthesized, there is no assurance that the two are equivalent. This is a major reason for the verification task becoming a bottle neck.

What is required is

- a language abstract enough to have its atomic operations refined to building blocks of IP-level granularity
- a synthesis process that can refine the abstract atomic operations in terms of IPs or map the atomic operations/functions to processor IPs by synthesizing the software for processor IPs
- synthesizing the interface among IPs or program/configure the interconnect IPs in an NOC-like platform
- synthesizing a memory hierarchy to hold global and local data
- a set of system-level constraints that can be used to guide the refinement process.

Today, rudimentary versions of such systems exist but need considerable progress to fully meet these requirements and are far from widespread adoption.



3. Manual Refinement requires verifying a more detailed specification.

Semiconductor Versus Design Productivity

Semiconductor productivity is machine centric; when a foundry moves to a new technology node, the semiconductor productivity, devices/mm^2, is machined into the fabrication line and does not change with time, place, or the operators. Design productivity is human centric; when a new design technology is adopted, it usually involves a steep learning curve. As the new technology is mastered over time, design teams become more productive. Different design teams use the same technology with varying degrees of success, depending on their experience and background. Therefore, design productivity does change with time and place.

Semiconductor productivity improves by replacing the previous generation process. When a new process is operational, it does not require the previous generation process. Design productivity improves by adopting a new design technology with new level of abstraction, but it does not replace the previous generation technology. On the contrary, the new technology often builds on the previous generation technology. For instance, when the design community adopted logic synthesis, physical synthesis did not become redundant. Instead, it became the back-end to logic synthesis, and remains an essential part of the entire design flow.

The previous differences reflect in the differing life cycles of the semiconductor and design technologies. Semiconductor technology progresses steadily at a rate dictated by Moore's law, creating new nodes every 18 months. Design technology, on the other hand, while matching the overall rate of progress in semiconductor technology as per the proposed law, progresses at an uneven pace, spread over three phases in a ten-year cycle.

The first phase is like childhood and has some teething problems. A new design technology is adopted for its potential to improve design productivity, but often the first time it is applied,

it proves to be less productive than the previous generation design technology. This is why most companies have pilot projects to iron out the wrinkles with a new design technology before it is applied to a time-critical real project.

The second phase is like youth. The technology is mastered and yields productivity gains, which improves with successive applications as the intricacies of the new flow are better understood and tools and methods mature and get better integrated.

Youth does not last forever, and old age descends when the design technology starts getting overwhelmed by complexity and issues not present with the previous generations of semiconductor technologies. Productivity falls, but old habits die hard, and designers resist adopting a new design technology.

Productivity gap is a symptom observed starting from the old age of the previous generation EDA node and continues up to the youth of next generation EDA node and is often reported for the most challenging and complex designs done in the latest semiconductor technology.

This analogy explains the differing design productivity in different phases of the EDA cycle, but old age in humans should not be strictly compared to the old age of EDA cycles. Old age in humans is followed by death, but in the case of EDA, old age becomes the foundation or the back-end for the next generation technology. As semiconductor technology progresses, new issues crop up that require considerable progress in the EDA technology associated with lower levels of abstraction as it is happening now: physical design tools in recent times have seen considerable upgrade in their functionality to effectively serve as a back-end.

CONCLUSION

Complexity in VLSI systems can be traced to progress in semiconductor technology as per Moore's law. The EDA industry has responded to the increasing complexity by successively moving to higher levels of abstraction, which increases the granularity of predesigned and preverified design elements that are reused by the synthesis for refining the abstract specification. Together with the granularity of design elements, the granularity of I/O has also increased. This improves productivity.

A pattern has been observed in the rate at which design technology progresses, and this is expressed as a proposed law. This proposed law is substantiated by evaluating it in light of design technology history, and in the process, design technology nodes in the past, present, and future are defined.

According to the proposed law, progress in design technology equalizes the increase in complexity due to Moore's law over periods of ten years. As a result, we should not see increase in complexity or a productivity gap.

Productivity gap is partly a transient phenomenon that occurs due to differences in the nature of semiconductor and design technologies, which is reflected in their life cycles. Whereas semiconductor technology progresses at a steady pace and has a short 18 month cycle, design technology progresses at an uneven pace and has a decade long life cycle.

Another factor that has contributed to the present productivity gap is the verification problem. It is analyzed as a result of design technology departing from automatic synthesis. To address this problem, design technology innovators will have to raise the abstraction of the specification language and develop synthesis methods to deal with the new abstraction. Unless this happens, the design technology might develop more than a transient productivity gap.

REFERENCES

[1] G. Moore, "Cramming more components into integrated circuits," *Electronics*, vol. 38, no. 8, 1965. Available: ftp://download.intel.com/research/silicon/moorespaper.pdf

[2] J. Darringer, E. Davidson, D.J. Hathaway, M. Lavin, J.K. Morrell, K. Rahmat, W. Rosener, E. Schanzenbach, G. Tellez, and L. Trevillyan, "EDA in IBM: past, present, and future," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1476–1497, Dec. 2000.

[3] D. MacMillen, M. Butts, R. Camposano, D. Hill, and T.W. Wiliams, "An industrial view of electronic design automation," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1428–1448, Dec. 2000.

[4] M.D. Spiller and A.R. Newton, "EDA and the network," in *IEEE/ACM Int. Conf. on Computer Aided Design Tech. Dig.*, 1997, pp. 470–476.

[5] A. Sangiovanni-Vincentelli, "The tides of EDA," *IEEE Design Test Comput.*, vol. 20, pp. 59–75, Nov./Dec. 2003.

[6] P. Michel, U. Lauther, and P. Duzy, *The Synthesis Approach to Digital System Design*. Norwell, MA: Kluwer, 1992.

[7] A. Hemani, "Behavioral synthesis: Problems with the present and promise for the future," Docent Lecture, Aug. 17, 1998. ESDlab, Dept. of Electrons., KTH, Kista, Sweden. [Online]. Available: http://www.ele.kth.se/~axel/presentations/1998/ahmed-hemani-docent-lecture-1998.pdf

[8] A. Hemani, "Industrial requirements for the SOC research and education in the next 5–10 years," Mar. 19 Keynote Speech, Swedish System-on-a-Chip workshop, Mar. 18–19, 2002, Falkenberg, Sweden [Online]. Available: http://www.ele.kth.se/~axel/presentations/2002/ahmed-hemani-SoC-Challenges.pdf

[9] The International Technology Roadmap for Semiconductors, ITRS Roadmap 2003 [Online]. Available: http://public.itrs.net/Files/2003ITRS/Home2003.htm

[10] J. Öberg, A. Kumar, and A. Hemani, "Grammar-based hardware synthesis from port-size independent specifications," *IEEE Trans.. VLSI Syst.*, vol. 8, pp. 184–194, Apr. 2000.

[11] A. Seawright and F. Brewer, "Clairvoyant: A synthesis system for production-based specification," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 172–185, June 1994.

[12] J.A. Rowson and A. Sangiovanni-Vincentelli, "Interface-based design," in *Proc. DAC'97*, pp. 178–183.

[13] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *Proc. IEEE Norchip Conf.*, 2000, pp. 166–173.

[14] W.J. Dally and B. Towles, "Route packets, not wires: Onchip interconnection networks," in *Proc. 38th DAC*, 2001, pp. 684–689.

[15] S. Kumar, A. Jantsch, J.P. Soininen, M. Forsell, M. Millberg, J. Öberg, K. Tiensyrjä, and A. Hemani, "A network on chip architecture and design methodology," in *Proc. IEEE Computer Soc. Annu. Symp. VLSI*, 2002, pp. 105–112.

[16] S.G. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O.P. Gangwal, "Cost-performance trade-offs in networks on chip: A simulation-based approach," in *Proc. Design, Automation, and Test in Europe Conf.*, 2004, vol. 2, pp. 764–769.

Ahmed Hemani is consulting as a systems architect for Phillips Semiconductors in "Eindhoven, Netherlands. E-mail: ahmed_ hemani@yahoo.com.

CD ■