A Position Statement in the Forum on

Future Directions for Design Automation Research

Jason Cong
UCLA Computer Science Department
http://cadlab.cs.ucla.edu/~cong

There are many technical challenges related to further scaling of integrated circuits, from lithography, to fabrication technology, and to novel circuit designs (IC). The advances in these areas are important, and provide the basis for design automation. The design automation, however, focuses on productivity improvement – assuming feasible techniques exist to manufacturing ICs with smaller dimensions and higher density, the goal of design automation is to produce an IC with the optimal or near optimal design, in terms of performance, power, reliability, and cost, with the best utilization of what the underlying fabrication technology can offer and in the shortest possible overall design time (including verification and test). Therefore, I believe that the fundamental issue with design automation is complexity management.

The rapid increase in design complexity has become a serious limiting factor in nanometer system-on-a-chip (SOC) designs. The sharp increase in design complexity is driven by two factors – one is the exponential increase in the number of devices integrated in a single chip, and another is due to need to deal with many new issues, such as interconnect, noise, power, and process variability that are associated with technology scaling. Such "double exponential increase" in design complexity is driving a widening gap between the silicon capacity and the design productivity.

I think that the following three areas are the key for managing the exponential increase of the design complexity: (i) development of more scalable optimization engines, (ii) support of higher level of abstraction, and (iii) enabling efficient design reuse, especially architecture reuse and silicon reuse.

1 Scalable Optimization Engines

Most IC design problems can be formulated as a multi-objective constrained optimization problem. The optimality and scalability of the optimization engine significantly impact the design methodology and design quality. For example, one has to partition a large design into smaller pieces for logic synthesis, as most logic synthesis engines have capacity and scalability limitations. Highly scalable optimization is most needed at the physical design level, where it has been shown that physical hierarchy may not correspond well to the logic hierarchy, and imposing a hierarchical design based on logic hierarchy often lead to suboptimal performance. Our studies in the past a few years showed that existing circuit placement tools and logic optimization tools were surprisingly far from optimal (from 40% to 70X in terms of basic wirelength optimization or area minimization) in some cases when the optimal solutions are known. Note that 30% wirelength reduction is equivalent to the benefit of one process technology scaling or the introduction of copper interconnects, while both required multi-billion dollar investments.

There are two significant challenges to large scale optimization in nanometer IC designs. One is the significant increase in design variables due to simultaneous optimization of performance, power, and reliability. For example, circuit placement needs to be considered jointly with multi-Vdd and multi-Vth island generation, cell sizing, effective channel-length and gate-oxide thickness assignments, clock gating and power gating, buffering, and timing-yield optimization, as they all compete with the same set of timing slacks. Another challenge is the need to support statistical optimization due to the significant device and interconnect variability in nanometer designs.

In order to deal with these challenges, I think it is important for the design automation community to work closely with the theoretical computer science and applied mathematics community. In fact, the two communities enjoyed a period of close interaction with in early 1980s, which led to a number of significant progresses, such as those on VLSI routing, logic optimization, and circuit retiming. Many of them are still used widely in today's design tools. For various reasons, such close collaboration did not continue into 1990's and 2000's decade (although there are sparse, isolated collaborations). I think that it is important to revive such inter-discipline collaborations again, so that the design technology can benefit from recent advances and breakthroughs in many other related areas, such as operation research, mathematical programming, combinatorial optimization, and Monte Carlo optimization. It is also equally important to share the challenges and opportunities in the giga-scale IC designs with the researchers in these fields so that we can attract their attention and interest and build a larger and stronger community in coming up efficient solutions to deal with the rapid increase in design complexity. At UCLA, we enjoyed the close collaboration with the applied mathematics group in developing the multilevel method as a general framework for developing scalable algorithms for IC design.

2 Support Higher Degree of Design Abstraction

Effective complexity management of design automation can also be achieved by raising the design abstraction level. Electronic system-level (ESL) design automation has been identified by Dataquest **Error! Reference source not found.** as the next productivity boost for the semiconductor industry. However, despite some recent success in ESL simulation, the transition to ESL design will not be as well accepted as the one to RTL in 1990s without robust and efficient behavioral synthesis (also known as high-level synthesis) technology that automatically compiles functional and/or algorithmic descriptions into optimized hardware architectures. Although behavioral synthesis has been a topic of research for almost two decades, it has never really caught on among chip designers. We believe that the previous failures are due to the following reasons: (i) Lacking of a compelling reason: the design complexity was still manageable at the RT level a decade ago; (ii) Lack of a solid RTL foundation: the industry was struggling with timing closure between logic and physical designs, and there was no dependable RTL to GDSII flow to support behavior synthesis; (iii) Lack of consideration of physical reality: prior approaches did not address the interaction of high-level abstraction and physical design (e.g., interconnect planning), which often led to unpredictable synthesis and significantly degraded the quality of results. With the rapid increase in design complexity and the availability of robust RTL-to-GDSII flows (finally by the EDA industry), we think it is time to re-visit behavior synthesis again, this time with full consideration of physical reality. I see three significant challenges in supporting higher level of design abstraction:

- Discovery and enabling concurrency: The popular high-level design specification languages at this point is C, C++, or SystemC, which are not very good as describing concurrency. In general, many high-level algorithm designers are not very comfortable with describing concurrency. It is important to develop ESL synthesis tools that can discover and extract the maximum amount of concurrency from the existing high-level specifications. It is also important to develop new specification languages to assist the designs in expressing the concurrency.
- Understanding the physical reality: The ESL synthesis tools need have effective interaction with the implementation tools to assure the correlation of performance, power, and complexity estimation.
- Support of reliability: It will be very helpful if the ESL synthesis tools can consider the reliability issue during behavior-level synthesis by proper utilization of redundancy (including function redundancy and/or time redundancy).

3 Enabling Design Reuse

Another technique to address the design complexity challenge is design reuse. In his keynote speech at DAC'2000, Dr. Classen classified design reuse into four levels: cell reuse, IP reuse, architecture reuse, and silicon reuse. Cell reuse and IP reuse are now well-accepted practices in the industry. A good example of architecture reuse is the evolution of the x86 architectures through multiple generations of Intel processors, and so is the Xtensa architecture platform from Tensilica. The recently advocated platform-based design methodology is one step further along the line of architecture reuse, and we expect it to bring significant productivity gain.

The ultimate form of reuse is silicon reuse, where the same silicon chip can be used for multiple applications. This is achieved through extensive use of on-chip programmable logic and microprocessors. Such programmable chips can be classified as general-purpose programmable chips, or application-specific or domain-specific programmable chips. General-purpose programmable platforms are offered through FPGA vendors, such as Actel, Altera, and Xilinx, which provide FPGAs with embedded processors (hard or soft), embedded memories, and a large amount of programmable logic. On the other hand, domain-specific platforms may include a number of domain-specific customized blocks in addition to general-purpose processors and programmable logic. Such platforms are attractive to many applications, and present a rich potential for flexibility, cost, performance, and design time trade-off. For example, one possible application is to use such a platform to quickly design an application-specific instruction set processor (ASIP), where the embedded processors can support an extensible instruction set, and the programmable logic can implement application-specific instructions.

In addition to the complete silicon reuse, partial silicon reuse is also possible, where several layers of chips (including the device layers and low level metal layers) are pre-fabricated and a few metal layers are available for customization. The well-known gate-array is such an example, so is the *structured ASIC*, which has also been introduced recently (for example by Altera, ChipX, eASIC, Faraday, Flextronics, LSI Logic, and NEC). A typical structured ASIC consists of hard-coded functions, such as memory and microprocessors, and customizable logic gates. It uses a subset of metal/via layers for customization. The design complexity for structured AISCs is reduced as characterization, layout, and optimization for pre-fabricated layers need to be done only once for performance, density, and yield optimization. Structured ASICs also provide a good way to lower the NRE cost and reduce the turnaround time.

The design automation challenges to support architecture and silicon reuse to develop automated performance evaluation tools and system-level compilation and synthesis tools for the underlying heterogeneous, multi-core, programmable systems. This effort is highly related to the needs stated in the previous section on support of higher level of design abstraction. In order to make efficient use of such programmable systems, I believe that the users will go more toward using processor-friendly programming languages, such as C, C++, or SystemC, as the initial system-level specification. Furthermore, a system-level tool for simultaneous processor customization, software optimization, and hardware synthesis will be very important.