Perspectives

NSF Workshop on EDA: Past, Present, and Future (Part 2)

Robert Brayton
University of California, Berkeley

Jason Cong
University of California, Los Angeles

THIS TWO-PART ARTICLE reviews the July 2009 National Science Foundation (NSF) Workshop on EDA. Part 1, which appeared in the March/April issue of *Design & Test*, discussed the workshop objectives, electronic design automation definition and history, and EDA funding sources. Part 2 discusses the foundational areas of EDA, key EDA challenges, emerging areas that may benefit from the EDA technologies, educational perspective, theory and EDA, and, finally, recommendations to the NSF.

Foundational areas for future EDA support

The National Science Board (NSB) and the US Congress recently mandated the National Science Foundation to explore possibilities of funding transformative research. However, while accepting the need for and encouraging transformative technologies, NSF program directors and the research community generally recognize that, historically, progress has been achieved by consistently pushing important areas over time. For example, in EDA, model checking involved 28 years of research with long periods of solid (but sometimes incremental) progress. A similar trend occurred with model reduction. In funding research, it's important to keep in mind that many fundamental breakthroughs occur typically after years of steady progress; moreover, the exact nature and impact of a genuine breakthrough are often difficult to predict.

Verification and model checking

Verification is an essential and increasingly important part of EDA, and formal model checking has become a major contributor to verification. Model checking arose from basic research in the programming languages and logics of the programs community. Early papers were published in POPL (Symposium on Principles in Programming Languages) and LICS (Logic in Computer Science). Only later was it realized that model checking could be used for verifying the correctness of sequential circuit designs. Indeed, model checking and other state-exploration techniques are probably easier to implement in computer hardware than in software. Hardware companies have used model checking since the mid-1990s, while software companies like Microsoft have only recently begun to take a serious interest in formal verification.

Model checking is also a good example of a case in which initial support by NSF paid off. The first papers were written in 1981 and 1982. Research was supported entirely by NSF until the early 1990s. When the power of symbolic model checking with OBDDs (ordered binary decision diagrams) was recognized, the Semiconductor Research Corporation (SRC) also began to fund research in this area. Computer companies did not directly support research on model checking at universities, and EDA companies such as Cadence and Synopsys did not develop commercial model checking tools until 2000, when bounded model checking with fast propositional SAT algorithms was proposed.

The current rash of fast SAT algorithms, now used for many purposes in EDA as well as other fields, were also developed primarily by EDA researchers in universities, for example, GRASP SAT-solver developed at the University of Michigan and Chaff SAT-solver developed at Princeton University, where this recent research was motivated by EDA applications.

0740-7475/10/$26.00 © 2010 IEEE Copublished by the IEEE CS and the IEEE CASS

Formal verification is inherently multidisciplinary; researchers must have a strong background in mathematics and theory. They need to know about mathematical logic, automata theory, OBDDs, SAT algorithms, decision procedures (like linear real arithmetic), programming language theory, models for concurrency, static program analysis, and symbolic evaluation. To deal with analog and mixed-signal circuits operating in noisy environments, they will need to know differential equations, probability theory, and stochastic processes. A strong background in computer science (CS) theory is necessary, but not sufficient. Deep knowledge of digital design and computer hardware is needed, often involving joint projects between CS and electrical and computer engineering groups or involving close work with someone at a company interested in using the techniques. Since companies are often reluctant to release their designs to researchers in academia, summer internships for graduate students at companies are often key.

Much important research still needs to be done in verification, particularly in these areas:

- *Scalability.* This will always be a problem because of the state explosion in highly concurrent systems.
- *Embedded systems*. Those that involve discrete and continuous behavior require a breakthrough.
- *Correctness of analog and mixed-signal circuits*. Little research has been done on establishing this.
- *Assertions about non-functional properties, such as power.* The assertions might be checked, but little research has been done thus far.
- *Developing tools that typical engineers can use*. This is a major challenge. It's difficult to write specifications in a notation-like temporal logic and to specify the environment in which a device to be verified will operate. Counter-examples can be quite long, involving many state variables and inputs, and locating its cause can be challenging.

Synthesis research

The development of logic synthesis before the early 1980s led to the use of higher levels of design description, such as the RTL, which is almost universally used today. High-level synthesis development has had an almost equally long history, while physical synthesis capabilities were developed later in the 1990s to cope with the timing closure problems in deep-submicron circuit designs. Over the years, synthesis capabilities have been extended significantly, in scale, speed, and quality of results. Also, as newer methods and devices for implementing logic, such as FPGAs, were developed, synthesis methods have been adapted and devised. Methods for front-end estimation have played a critical role in allowing earlier design convergence and have yielded better results.

The industry's acceptance of synthesis methods depends on their synthesis results to be as good as or better than the results by manual methods. Also, synthesis techniques have been crucial to the acceptance of new design styles or circuit fabrics. For example, several start-up ventures have invented circuit fabrics that, on paper and with some circuits implemented by hand, looked promising. However, the effort to automate logic synthesis into these fabrics was vastly underestimated and resulted in the ventures' failure.

Although synthesis is frequently regarded as being mature and not needing further support, it is fundamental to EDA and still needs to be exploited and extended to larger ranges of scalability. Synthesis is also intrinsically intertwined with verification; for example, some synthesis methods are not used because they can't be verified formally, while the formal verification task can be simplified if it is given the list of transformations performed by the synthesis tool, and if the task harnesses equally strong methods for circuit reduction, such as induction. Although synthesis can be made scalable by partitioning the problem into smaller parts, this can limit the quality of results. Thus, extending the scalability of the underlying algorithms can improve overall quality and make resulting designs more competitive. Such development will aid both synthesis and verification because both areas share many of the fundamental algorithms, such as SAT, BDDs, representation and manipulation of logic, abstraction, speculation, interpolation, induction, and so on.

Another development is the synthesis for different objectives. Early synthesis was aimed at decreasing area and delay. More recently, other objectives have come into play, such as power, noise, thermal control, verifiability, manufacturability, variability, and reliability. Consequently, additional criteria will emerge as new technologies develop, and new models and optimization techniques will be needed to address such requirements.

Finally, synthesis is being challenged by a rising level of abstraction. For example, current SoCs can accommodate thousands of simple processor cores (such as Intel 386–equivalent processors). Instead of synthesizing a design into a network of standard cells (the current practice), we may consider synthesizing it into a network of processors or into a mixture of processors and standard cells. In this case, a behavior specification based on C/C++ or SystemC is a more natural starting point, as such specifications are more software friendly. New modeling, synthesis, and verification techniques are needed to support such new design methodology and a new level of design abstraction.

Programming language research

A good programming language can be crucial to achieving scalable design methodologies. The consensus is that the language design aspect of Verilog and VHDL leaves much to be desired. Unfortunately, the current EDA response seems to be "kitchen sink" languages (e.g., SystemVerilog) that combine every aspect of every language (Verilog, e, Vera, and so on) into the language definition. This complexity slows down the standardization process, and the definition release of languages like SystemVerilog v. 3.1a, which needed three major revisions just to get the "core" language right.

In contrast, the programming language and software world has much to offer: clean core semantics, types and modularity, core languages together with well-designed libraries, static analysis, and so on. A secondary benefit of a clean language design is that it might be easier to work with by using tools within academia. At the moment, the investment to build the front end for SystemVerilog is enormous and hard to justify in an academic context.

Languages can be a great opportunity for industry/academic alliances. Language innovations are more easily begun within academia, which has the flexibility to start new projects relatively quickly and unimpeded by corporate requirements, while a company has so much legacy code that it won't change, or it won't readily consider new languages. However, industry can provide challenging designs to judge the effectiveness of the language design or to motivate language constructs; ideally, academics can adopt the best features. A good demonstration of research ideas from programming language and verification communities that is now influencing industrial design is Bluespec, a new atomic-transaction-based language for circuit specification.

Analog and mixed-signal design

Automation techniques for systems increasingly made up of analog circuits lag behind systems in the digital realm. This is particularly critical given two quantitative facts:

- 66% of today's ICs are mixed-signal designs—that is, they incorporate analog components; and
- numerous studies show that the analog portions of these designs are most frequently at fault when chips fail at first silicon.

Moreover, scaling to the nanoscale level has led to the breakdown of clean digital abstractions to the point that, at the level of physical implementation, the tools and techniques used for digital design are closely related to those for mixed-signal and RF design. We expect this trend to accelerate.

Against this backdrop, it is essential to continue to invest in fundamental solutions for the growing non-digital side of EDA. Improvements will rely on foundational mathematical research to bring increased automation. Where human experts rely on intuition, design automation tools rely on a diverse range of aggressive optimization formulations: numerical, combinatorial, geometric. A unique attribute of these analog problems is that concurrent consideration of functional, electrical, and geometric aspects is necessary to compete with the manual designs. Experts often can transform large mixed-signal problems into small, workable abstractions that preserve the essential design features.

Nonlinear model reduction

Progress over the past 20 years has rendered the analogous model order reduction problems solvable for linear systems (e.g., via robust Padé approximations) and for finite-state digital systems (e.g., initially via symbolic model checking with BDDs, later with Boolean SAT-based bounded model checking). Unfortunately, no such robust theory exists for the more common nonlinear cases that dominate in the mixed-signal and low-level digital realm. Nor are there complete solutions for designing or assuring correct behavior of these analog systems in the presence of unavoidable statistical manufacturing fluctuations. Digital systems are designed to hide the physics

of the fabrication process; analog systems are designed to exploit these behaviors.

As fundamental components move further into the nanoscale regime, analog systems are increasingly vulnerable to these small upsets. Thus, great opportunities exist in building the next generation of fundamental numerical, combinatorial, and geometric algorithms to handle these essential designs. Moreover, improvements in fundamental verification and in algorithmic simulation are mutually reinforcing. Larger, more complex artifacts can be designed and used only when their component interactions can be guaranteed with assurance; synthesis tools that rely on iteratively exploring a huge number of different solution configurations can be targeted to larger and more challenging designs only when each solution candidate can be evaluated in one second, instead of one hour or even one day. Also, nonlinear model reduction will be a key enabler in verification and model checking of hybrid systems.

Key EDA challenges

We see a number of EDA challenges within five areas: scalable design methodologies, scalable design synthesis and validation/verification, dealing with new technology, designing with uncertainty and fragility, and new classes of algorithms.

Scalable design methodologies

At its core, design automation is about developing design methodologies that best address the emerging technology challenges. ASIC design methodology based on standard cells and synchronous timing was crucial in providing the foundational basis for the design tools that eventually enabled it. Future nanoscale designs face significant complexity challenges, which demand design methodologies that will continue to scale in the face of increasing complexity. The following five attributes are critical for success.

Disciplined, predictable design. The design process must be highly disciplined; only then does it lend itself to automation. Further, predictability is important, both in the design process and in design quality. A predictable design process leads to predictable design schedules, and a predictable design quality leads to dependable results. Current design flows might not converge, or they might have unpredictable tool runtimes, or provide unexpected results.

Intuitive design environments, simplified user interfaces. Current design environments are too complex, with too many parameters that designers cannot understand. This is limiting and must be replaced with intuitive, simplified interfaces that can lead to fast design cycles, enabling designers to better explore the design space.

Appropriate abstractions. An important component of ASIC design methodology has been the separation of electrical concerns from the logical aspects of hardware design. With nanoscale scaling, this separation is threatened. Hardware designers must increasingly understand the underlying electrical models to better predict circuits' power consumption and reliability. This diminishing separation of concerns makes it harder to have multiple entry points into the design flow based on a designer's expertise; this limits the pool of designers and, eventually, the number of designs. Automatic abstraction techniques can play an increasingly larger role in helping with this. Our current block-based signoff techniques may not be sufficient to allow efficient design (in manpower and area) in the face of an increasing region of influence from physical effects.

Scalable design methodologies along the axis of different circuit functions. One example of such needs is that the lack of automation for embedded arrays poses an increasing problem. Arrays (by virtue of size and number) are fertile ground for automation; the need for efficient design is great, but the highly structured nature of the circuits lends itself to automation. Another example of such needs is analog circuit design, which also requires specialized design flows.

Standardized interfaces. It is essential that we have open interfaces for different parts of the design flow. Proprietary formats from vendors impede progress in our field.

Scalable design synthesis and validation/verification

Existing design synthesis and verification techniques are no longer scalable. The techniques' costs and limitations are restricting the design of future computing platforms and even the inclusion of additional features in existing platforms. If this is not suitably addressed, the impact will be felt in all fields of

computer science. This directly strikes at the economic basis of Moore's law—increased computation per unit cost with each generation of Moore's law scaling, which in turn drives the development of novel applications that exploit this cheaper computation. The profit from such new applications indirectly funds the R&D for further scaling. If this cycle can no longer be counted on, it will severely limit future advancement of computer science. Multiple possible solution directions remain to be explored:

- Higher level of design abstraction
- Better design principles and languages
- Formal and semiformal verification techniques
- Runtime validation
- Extending the scalability of the underlying fundamental algorithms

Dealing with new technology

Design in the late- and post-silicon era must deal with new devices as well as with manufacturing technology. Important changes in the future include 3D designs, graphene-based and other emerging devices, and new lithography techniques. These changes will require new techniques and tools for modeling, synthesis, and simulation.

Designing with uncertainty and fragility

The combination of uncertainty and fragility adds a critical dimension to existing design methodologies because reliable circuits and systems will have to be built using unreliable fabrics. In addition, uncertainty and fragility arise from physics; for example, there are increased soft errors in finer geometries due to hits from energized particles, and there are uncertainties in manufacturing in fine geometries due to system and random variations. These contribute a diverse set of failure modes that will need to be accounted for at appropriate levels of the design.

New classes of algorithms

Several new classes of algorithms need to be explored for development of scalable design methodologies.

- *Linear/sublinear algorithms*. Increasing complexity and problem scale cause quadratic, or in some cases even log-linear algorithms, to have unacceptable runtimes. A significant push is needed in exploring linear and sublinear algorithms across design tools.
- *Incremental algorithms*. Algorithms that can recognize and exploit incremental design changes can significantly help reduce design time.
- *Parallel algorithms*. As we move to the multi/many-core era, it is critical that EDA algorithms be able to exploit future platforms.
- *Deterministic algorithms*. With long design cycles, it is important for the algorithms to be deterministic. This is critical in reproducing results and providing predictability in the design process, and is especially critical for parallel programs in which races are an important contributing factor to nondeterminism.
- *Design for security*. To ensure data privacy, it is increasingly important that designs be resilient in the face of security attacks. Examples of this are various forms of side channel attacks. These can be tackled as a potential part of the design process.

Emerging areas and EDA technology

Given the new definition of EDA that the NSF EDA workshop agreed on,[1] we feel that areas involving highly complex systems requiring modeling, analysis, and transformation among different levels of abstraction are good candidates that can benefit from EDA technologies. The following six areas, often in emerging fields, are examples.

1. *Biology systems*. Biology will transform into a precise, quantitative, bottom-up, predictive discipline, much as physics and engineering did over the last century. Such systems feature many individual entities that interact extensively and are organized hierarchically, where logical functionality arises from the hybrid interplay of discrete and continuous-time dynamics. Understanding how most biological systems work will require numerical and Boolean tools for analysis and design, similar to how CAD tools became indispensable to VLSI design. In biology, effective leveraging of, for example, computational, abstraction, and verification techniques from EDA will be critical for catalyzing progress in core science areas such as systems and synthetic biology.
   - *System biology* captures interactions of discrete molecular elements (genes, proteins, etc.)

that lead to collective properties at multiple levels (metabolic function, organ function, etc.). The outcome might be a predictive network hypothesis (the biological circuit) that correlates system behavior to lower-level structures.

- *Synthetic biology* focuses on modification of biological systems at the molecular level to achieve new functions, such as bacteria that can attack cancer cells, or to lead to new biofuels. The EDA community has already made key contributions (e.g., automated abstraction techniques to improve simulation efficiency, BDD technology applied to accelerate drug design, oscillator phase macromodels for quantitative understanding of circadian systems, etc.). This trend will accelerate using compelling aspects of EDA—deep foundational underpinnings and tools for large-scale systems—to enable fundamental progress and new discoveries in the core science area of biological systems.

2. *Emerging computing, communications, and storage fabrics and manufacturing substrates*. Of particular note are nanoelectronics, nanophotonics, nanoelectromechanical systems,[2] and flexible electronics,[3,4] in which diversity of functionality, manufacturability, variability, and reliability provides challenges and opportunities for modeling, analysis, synthesis, and integration.
3. *Analysis, characterization, and potential design of hybrid electronic and biological systems*. Examples include biological neural networks with integrated digital and analog electronics-based stimuli and readouts.
4. *Cyber-physical systems*. Such systems consist of the interaction of vast numbers of embedded systems, often requiring real-time control, such as intelligent transportation systems.
5. *Datacenter design and optimization*. It is increasingly important yet difficult to address performance, energy, reliability, interconnection, and cooling issues involving thousands to millions of entities (servers, storage, and I/O devices) in a modern large-scale datacenter under a dynamic workload.
6. *Software systems*.[5] These are especially relevant in the following two areas: scalable and more precise large-scale software analysis, and tools and methodologies to extract and manage concurrency. Current EDA technology has strongly supported the high degree of concurrency in IC and electronic system designs.

This list can serve as suggestions for applying EDA techniques to adjacent/emerging fields, but it is by no means exhaustive. For example, NSF workshop attendees discussed the possibility of applying EDA technologies to the design and analysis of social networks and to the design and analysis of quantum information processing systems, especially quantum communication and quantum cryptography. However, the applicability and benefits of EDA technology to these fields are less certain in comparison to what other techniques, which address problems in these fields, can offer. Therefore, these areas are not included in the NSF workshop's initial list of recommendations. Further joint investigations would be needed with other domain experts from physics, information science, and so on to develop a sharper focus and a more convincing justification, as well as to establish a consensus on feasibility and identify verifiable order-of-magnitude improvements.

The most readily transferable EDA assets to adjacent/related disciplines include the following:

- Modeling and analysis at multiple levels of abstraction
- Synthesis and transformation with optimization
- Validation of functional and structural properties as well as performance requirements
- Use of formal methods

The vast knowledge accumulated in these areas for solving IC and electronic system design problems can be applied rather directly to solve any of the new applications listed.

Educational perspective on EDA

Graduate students from US institutions apparently receive a different perspective on EDA than do those graduate students overseas. In the US, there is little teaching of core EDA topics—such as logic synthesis, physical design, simulation (discrete and analog), testing, and formal verification—at the undergraduate level compared to what is taught at leading universities overseas. Consequently, fewer domestic students apply to graduate schools in EDA, and some who might have been attracted by the

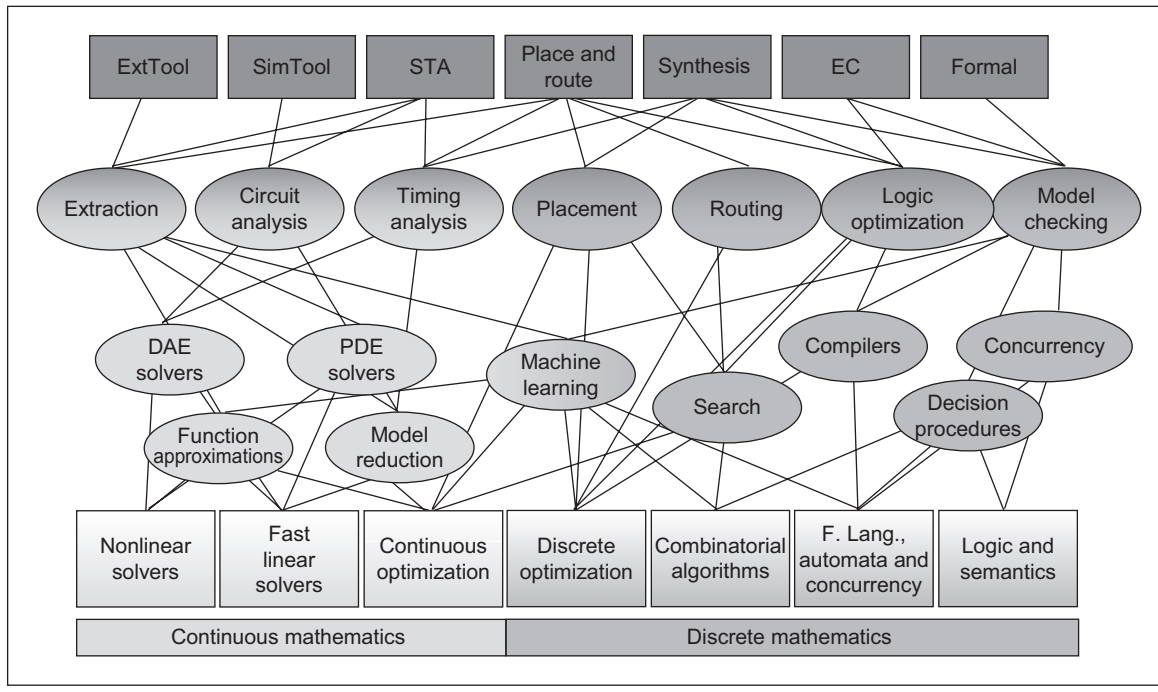

Figure 1. Fundamental areas and domain knowledge in EDA. (Courtesy Andreas Kuehlmann, Cadence Design Systems, Inc.)

subject matter are lost to other disciplines. The workshop attendees agreed that a good design background is important in EDA, and for the most part, such courses are being offered; however, these mainly teach the use of canned CAD tools and cannot cover EDA algorithm topics in any depth. Workshop attendees felt that a good senior-level introductory CAD class could be developed and offered more broadly in the US.

Exactly what subset to teach is a challenge because EDA is a broad, interdisciplinary field that continues to expand; for example, embedded systems is a relatively new EDA topic. An ideal undergraduate course should develop this breadth but avoid being just an enumeration of disparate topics; it should emphasize a set of problem areas containing common underlying algorithmic themes. This would allow in-depth exposure to some algorithms and also introduce the algorithmic and theoretic foundations of EDA.

At the graduate level, few universities have the manpower to address all possible EDA topics. Figure 1 illustrates the skill sets an employer in the EDA field needs and that delineate the kinds of skills and knowledge that should be taught. The top layer lists the set of products that are part of an EDA company's current offerings. These include extraction, simulation, static timing analysis, place and route, synthesis, engineering change, and formal verification. The next layers of the graph (oval nodes) show software that is used in these tools. For instance, synthesis needs timing analysis, placement, logic synthesis, and model checking. Extraction needs function-approximation methods, PDE solvers, model-order reduction, and machine learning. The next layer lists the academic disciplines required by the people who implement state-of-the-art tools in the listed areas. For example, discrete optimization is used in machine learning, placement, routing, search, and logic optimization. The bottom layer categorizes the underlying mathematics as either continuous or discrete.

It was informative to look at a similar graph (not included in this article) for some of the adjacent or emerging technologies that might be part of the future. That graph differed only in the first layer and the interdependencies. Some of the future technologies listed were multidomain microsystems (such as micromechanics), new device and process modeling, software verification, systems biology, parallel computation, and so on. In addition, the types of complexities to be met, and the problem scale to be addressed, will be similar to those already encountered in EDA and so have already been solved to some extent.

The workshop's conclusion is that training in EDA fundamentals and experience with EDA-scale problems are required for continuing technological innovations. Of course, not all of the domain knowledge need be taught by EDA personnel, and certainly existing courses in electrical engineering and computer science should be incorporated in a graduate program. Even for the main core EDA courses, there are sufficient numbers of neither professors nor students to warrant courses being taught each year. A suggestion was that a graduate EDA curriculum should offer relevant in-depth courses in alternate years. In addition, it was suggested that there be a definite movement toward "nano-widget" courses (covering nanowires, nanoswitches, nanomemories, etc.). Similarly, it was suggested students be encouraged to take these courses and professors be encouraged to participate, prepare, and adapt EDA material to supplement such courses.

The EDA workshop group noted that, in Europe, fewer actual graduate courses are taught because the education depends more on self-study, seminars, and learning while students conduct research. However, it was not recommended that the US follow this approach because offering a large variety of graduate-level courses to students is believed to be an educational advantage.

A serious impediment for advancing EDA education in universities and, to an extent, in research (more so at smaller universities), is the demise of funding for MOSIS (Metal Oxide Semiconductor Implementation Service). In the past, NSF and DARPA jointly funded this program, which provided access to fabrication of prototype and low-volume production quantities of ICs. MOSIS lowers the fabrication cost by combining designs from many customers onto multiproject wafers. Although the expenses for funding such a program have risen dramatically, to maintain US competitiveness it would be important to resurrect this support. While funding this could be expensive, NSF mechanisms that support infrastructure could be used. These include the CISE Computing Research Infrastructure (CRI) program, the NSF-wide Major Research Instrumentation (MRI) program, and the office of Cyber-Infrastructure (OCI). NSF MOSIS funds for the use of the DARPA-secure foundry, or even foreign foundries, could be investigated.

Another topic discussed at the workshop was the need to combat negative perceptions and negative—but realistic—observations about the EDA field: Many EDA companies are hurting financially, and job opportunities are down.

- EDA summer internships are limited.
- Venture capital for start-ups in EDA has decreased significantly. These have been a vital component for EDA innovation; moreover, start-ups have served as major centers for R&D and employment of PhDs.
- Faculty positions in EDA are tight, aggravated by the difficulty of obtaining funding to support research and students.
- Student interest in EDA as a career has decreased in recent years.
- Industrial EDA research efforts have dropped off with the dissolution of high-quality research groups at Cadence and Synopsys, while the large system design companies have throttled back on the research components of their activities.
- Transition of academic research to industry is much harder than it used to be. Technologies are more complex, and it's harder to integrate new ideas in the sophisticated and mature software offered by EDA vendors.

On the positive side, six factors help offset the negative observations:

- The EDA field will not disappear and it cannot stagnate. It is vital for supporting the design of complex systems such as microchips. EDA expertise is needed by EDA software companies and by large system design houses. Start-ups will continue to be valuable in finding niche applications and researching solutions, and nurturing core EDA technologies as well as emerging ones. These should see a resurgence as the economy improves.
- Cooperation between industry and academia remains high, probably among the highest of any discipline within computer science and engineering.
- As technology shrinks, the problems become more difficult, so not less but more EDA activity is required. This increased complexity puts more emphasis on modeling, model reduction, abstraction, and so on—techniques in which expert EDA personnel are well versed.
- EDA engineers are well paid, apparently better than most other types of engineers. Moreover, if

the trend of fewer students entering the field continues, then supply and demand will assert itself; demand and salaries will increase.
- EDA training in its various disciplines, including complex problem solving, will be valuable as new growth areas come into play, such as nanotechnologies, biology and other life science applications, new energy technologies, new energy conservation methods such as smart buildings, and perhaps new applications in the financial sector. Indeed, we see that students with EDA backgrounds were hired into these sectors.
- Aside from the emerging areas, EDA continues with its own hot areas. These include, for instance, system-level design, embedded software, design for manufacturing including lithographic and scaling problems, issues of robustness and unreliable components, parallelism, design and application of many-core (1,000+) processors, application of probabilistic methods to enhance scaling of algorithms and problem solutions, and new methods for derivative and incremental design.

The bottom line is that EDA continues to evolve, expanding into exciting areas with lots of new technology on the horizon (nano, bio, optical, etc.). Tremendous but exciting challenges are looming. EDA provides a flexible foundation for solving future problems in the design of complex and large systems, with many ideas transferable to other fields, and EDA researchers are still well paid. Students should keep these aspects in mind for the time, four or five years in the future, when they might expect to graduate. EDA professors need to convey the idea that EDA is more than just supporting semiconductor implementation flows; it is broadly about the optimization, implementation, and verification of complex systems using deep knowledge of an underlying technology.

Theory and EDA

The theory and algorithm community within engineering has had a long history of involvement with EDA efforts, dating back to VLSI work in the 1980s. Floorplanning and layout/routing problems, among others, were studied extensively. Over time, many key algorithmic questions in these areas were either resolved or reduced to hard (and open) problems, and the strong initial connections between the theory and algorithm communities became weaker.

Evolution of EDA and theoretical methods suggests several key directions for fruitful collaborations between the algorithm and EDA communities, beginning with SAT solver effectiveness.

Effectiveness of SAT solvers

In the EDA community, satisfiability (SAT) solvers have been engineered to the point where SAT, rather than being viewed as a hard NP-complete problem, is regarded as an "easy" subroutine used to lessen other hard problems. However, we do not yet have a clear understanding of why the SAT instances encountered in EDA problems are easier to solve. Such a study has both theoretical and practical merit. From a theoretical perspective, a better understanding of "easy" instances of SAT will naturally further develop our understanding of algorithms in the exponential time regime. This is an area that has been sorely underdeveloped in the algorithm community. EDA applications provide both a testbed of instances as well as concrete applications for this work.

In related work, research to determine the threshold for random SAT has been extensive. Specifically, researchers have looked into determining the precise ratio of variables to clauses to determine the exact point above which a random SAT formula is almost always satisfiable, and below which it is almost always not satisfiable. Although random instances are not likely to be the same as those encountered in practice, the insights gleaned from this research are likely to be very useful.

New algorithm design paradigms for EDA applications

Randomization and approximations have been among the most exciting developments in the realm of algorithm design over the past 20 years. Randomization usually allows the design of very simple and practical algorithms with strong (probabilistic) guarantees on performance. Many problems in the EDA pipeline include elements like state space exploration, or searches in high-dimensional parameter spaces. These problems can benefit greatly from randomized methods. Repeatability is an important concern for such applications, and designers should consider this issue when designing randomized algorithms (proper seed or trace management can address this issue quite easily).

Approximation techniques are also a powerful way of efficiently obtaining guaranteed quality bounds for

an optimization. Approximation techniques are mature and could be brought to bear on a number of the problems discussed.

Another issue is that of incremental (or progressive) methods. The "waterfall" model of EDA treats the design process as a pipeline with outputs from a phase progressively feeding into the next phase. It is important, when operating such a pipeline, that small changes made in one stage do not cause drastic changes in the final design. Incremental (and streaming) methods in algorithms are designed to adapt solutions to slight changes in data. Progressive methods are designed to provide better-quality solutions as more time is invested in the computation. These two paradigms could benefit EDA pipeline design immensely.

New computational models

EDA systems are compute-intensive, especially in the verification stages. Desktop computers now typically have several levels of caches, and are rapidly migrating to a multicore architecture that combines parallelism and a caching hierarchy. Achieving algorithm efficiency under these new architectures requires attention to cache efficiency and parallelism in addition to the traditional measure of the number of operations executed by the algorithm (i.e., the sequential running time). It is also desirable for the code to be portable across different architectures while maintaining efficiency. The cache-oblivious setting offers a simple, attractive model for sequential, portable, cache-efficient algorithm design, and many provably efficient algorithms have been designed in this model and experimentally run efficiently on modern uniprocessors with a cache hierarchy.

In the past couple years, attention has turned to modeling multicores effectively, and some promising results have been developed for basic algorithms in the theory community. EDA would benefit from incorporating efficient cache-oblivious[6-8] and multicore[9,10] techniques into EDA algorithms. These techniques promise much faster parallel multicore implementations for EDA problems than currently exist, and could significantly extend the scale of problems that can feasibly be solved.

New classes of algorithms

New classes of algorithms are needed throughout the EDA field. These include sublinear algorithms, incremental algorithms, and the cache-oblivious and multicore algorithms we've mentioned. Sublinear algorithms would be beneficial in EDA in cases where a large number of candidate solutions must be examined to determine if they satisfy a given property. Often in such cases, very fast (sublinear time) algorithms can reject any candidate solution that is significantly far from the desired property. The use of such algorithms could help to significantly speed up EDA computations. Incremental and fully dynamic algorithms[11,12] are other classes that have been well studied in theoretical algorithm design, and they hold much promise for EDA. Here, when components are added or modified in a large design, designers seek algorithms that incorporate the change with runtime that is much smaller than the size of the design. These algorithms typically run in time that is a function of the size of the change being made, and that is a very slow-growing function of the design's size. In fully dynamic algorithms, most incremental and decremental changes are allowed. Many fast, fully dynamic algorithms have been developed for basic problems, such as graph connectivity and minimum spanning tree. This is an area that potentially could speed up EDA algorithms significantly.

Additional considerations

Three other elements to promote EDA/algorithm researcher collaboration are as follows.

- *Engagement of theorists through EDA benchmarks*. EDA research generates large-scale benchmarks (e.g., SAT instances arising from verification) that are of value to the algorithm community. This engagement can help provide a better formal understanding of effective EDA algorithms.
- *Robustness in design*. The theory community can provide insights into design techniques with formally provable robustness properties, even in the face of security attacks.
- *Statistical design*. With statistical variations playing an increasingly important role in design, there are opportunities in new techniques for modeling, analysis, and design (optimization) with statistical variations.

Recommendations to NSF

The EDA workshop has made recommendations in three main areas—research, education, and industry collaboration—for improving future design automation programs.

Research programs

The EDA workshop has identified five types of new funding programs to support the following research initiatives:

1. *Mid-scale or large-scale research efforts that couple design with EDA.* Many workshop attendees pointed out that current design automation (DA) researchers no longer have direct interactions with circuit/system designers. Thus the researchers have neither first-hand experience with the new design problems nor direct feedback on the fruitfulness of their research. However, modern SoC designs in nanoscale technologies require design teams of substantial size. Therefore, it is important to create large funding opportunities to support innovative design projects and couple them with leading-edge DA researchers. Such funding may be established jointly by the Experimental System Program and the DA Program at the NSF Computer & Information Science & Engineering (CISE) Directorate.
2. *Joint research programs between research groups from universities, commercial EDA companies, and large systems houses.* These programs would alleviate some of the recently diminished research in EDA.
3. *Shared infrastructure for design and DA.* This would provide additional collaborations between industry and academia. However, significant resources are needed at universities that would produce noncommensurate research benefits. (It was pointed out at the workshop that infrastructure development can be achieved by university groups, even with limited industrial support.) The NSF CRI/MRI programs exist to help with this, but they need to be better utilized.
4. *Exploration of DA for emerging areas.* NSF should substantially increase its funding to the DA program so that it can form partnerships with other research programs in NSF to explore the new frontier for DA, in particular, with the following initiatives:
   - Joint program with the CPS program to explore design automation for cyber-physical systems.
   - Joint program with architecture and networking programs to explore data center design and optimization.
   - Joint program with the software division to investigate DA techniques for scalable and more precise large-scale software analysis, and tools and methodologies to extract and manage concurrency.
   - Joint programs with biological sciences to initiate programs in DA for system biology and synthetic biology.
   - Joint programs with the NSF's Engineering Directorate to explore DA for emerging computing, communications, and storage fabrics, and manufacturing substrates.
5. *Interaction between DA and theory communities, as well as interaction between DA and mathematical sciences.* Such collaborations do exist, but are done in an ad hoc fashion. A well-formalized program within the DA program will greatly facilitate such collaborations.

Education programs

The EDA workshop has recommended three programs in the following areas:

1. *Support for development of a senior-level EDA course.* This would emphasize the underlying algorithmic and theoretic foundations of EDA while motivating EDA's breadth and flexibility with specific interesting applications. Materials might be broadly submitted by many faculties to a central principal investigator who would meld the contributions into a viable semester course and make the materials available online.
2. *Support from NSF to develop shared courseware infrastructure in EDA.* Some faculty members have had exposure to Connexions (see http://cnx.org), an open platform for course sharing, which might be used.
3. *An increased post-doc program to alleviate the lack of research positions for new graduates.* Such a program was perhaps part of the stimulus effort but was quite limited and not specific to EDA.

Collaboration with industry

The EDA workshop has identified four programs to enhance industry and academic collaboration.

1. *An enhanced program to support longer-term faculty/industry interactions.* A tight connection with industrial reality and practice has always been crucial in EDA. IP houses and the IDMs (integrated device manufacturers) jealously guard their data (e.g., design data, test data) while the fabrication costs in the latest technologies are

tremendous. Access to technologies, which is important for academic research, can be seeded by enhanced faculty stays in industry or, conversely, by technical leaders from industry visiting academia. There must be a commitment by industry to implement designs for the purpose of outside research. This could be enabled by matching NSF and industry contributions. In the Engineering Directorate, there is a GOALI (Grant Opportunities for Academic Liaison with Industry) program to enable this; perhaps a similar program is needed for CISE.

2. *An enhanced program to support summer students working at EDA companies*. Students would be located physically at a company. Proposals for funding would be a joint effort between a faculty member and a research staff person at the company. This program could include small start-ups as well as EDA vendors and large system design houses.
3. *A program to help faculty members and graduate researchers spin off start-ups to commercialize successful research projects*. This would be similar to an SBIR program but more focused on EDA. The goal would be to help make the transition from a research paper or prototype to first customer adoption, so that venture capitalists or large EDA companies could take over from there.
4. *A program to help marry faculty to existing start-ups*. (This recommendation relates to the preceding three programs.) This would encourage new ventures in EDA-type activities.

We estimate that a 2.5× increase in the current funding level is needed to support these new initiatives. Part of it can, we hope, be shared with other programs in the NSF's CISE or other directorates. We also hope to see continued partnership and cost-sharing with industry, such as the multicore program with SRC. ■

Acknowledgments

This workshop was sponsored by the National Science Foundation CISE/CCF Division under grant CCF-0930477. We thank the program director, Sankar Basu, for his support of this workshop and for providing valuable feedback to the drafts of this article. We thank all workshop speakers and participants for their interesting and stimulating presentations and discussions; the lists of speakers and participants are available at http://cadlab.cs.ucla.edu/nsf09/. In particular, we thank Sharad Malik (Princeton University), Pinaki Mazumder (NSF), and Suresh Venkatasubramanian (University of Utah) for serving as group discussion leaders. Their summary reports of the group discussions form the basis of many parts of this article. Finally, we are grateful to UCLA staff members Alexendra Luong and Cassandra Franklin for their support in organizing the workshop, and also to Janice Wheeler for editing this article.

References

1. R. Brayton and J. Cong, "NSF Workshop on EDA: Past, Present, and Future (Part 1)," *IEEE Design & Test,* vol. 27, no. 2, 2010, pp. 68-74.
2. A. Korkin and F. Rosei, eds., *Nanoelectronics and Photonics: From Atoms to Materials, Devices, and Architectures,* Springer, 2008.
3. FlexTech Alliance; http://www.flextech.org/.
4. T.-C. Huang and K.-T. Cheng, "Design for Low Power and Reliable Flexible Electronics: Self-Tunable Cell-Library Design," *IEEE/OSA J. Display Technology,* vol. 5, no. 6, 2009, pp. 206-215.
5. J.R. Larus et al., "Righting Software," *IEEE Software,* vol. 21, no. 3, 2004, pp. 92-100.
6. M. Frigo et al., "Cache-Oblivious Algorithms," *Proc. 40th IEEE Symp. Foundations of Computer Science* (FOCS 99), IEEE CS Press, 1999, pp. 285-297.
7. L. Arge et al., "An Optimal Cache-Oblivious Priority Queue and Its Application to Graph Algorithms," *SIAM J. Computing,* vol. 36, no. 6, 2007, pp. 1672-1695.
8. R. Chowdhury and V. Ramachandran, "The Cache-Oblivious Gaussian Elimination Paradigm: Theoretical Framework, Parallelization and Experimental Evaluation," *Proc. ACM Symp. Parallelism in Algorithms and Architectures* (SPAA 07), ACM Press, 2007, pp. 71-80.
9. R. Chowdhury and V. Ramachandran, "Cache-Efficient Dynamic Programming Algorithms for Multicores," *Proc. ACM Symp. Parallelism in Algorithms and Architectures* (SPAA 08), ACM Press, 2008, pp. 207-216.
10. L.G. Valiant, "A Bridging Model for Multi-Core Computing," *Proc. 16th European Symp. Algorithms* (ESA 08), LNCS 5193, Springer-Verlag, 2008, pp. 13-28.
11. J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity," *J.ACM,* vol. 48, 2001, pp. 723-760.
12. C. Demetrescu and G.F. Italiano, "A New Approach to Dynamic All Pairs Shortest Paths," *JACM,* vol. 51, 2004, pp. 968-992.

Robert Brayton is Cadence Distinguished Professor of Engineering at the University of California, Berkeley. His research interests include combinational and sequential logic synthesis for area, performance, and testability; formal design verification, and logical/physical synthesis for DSM designs. He has a PhD in mathematics from MIT. He is a member of the National Academy of Engineering, and a Fellow of IEEE and the AAAS.

Jason Cong is Chancellor's Professor and director of the Center for Domain-Specific Computing at the University of California, Los Angeles. His research interests include synthesis of VLSI circuits and systems, programmable systems, novel computer architectures, and nano-systems. He has a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a Fellow of the ACM and IEEE.

■ Direct questions and comments about this article to Jason Cong, UCLA Computer Science Department, 4731J Boelter Hall, Los Angeles, CA 90095; cong@cs.ucla.edu.