# Impact of Interoperability on CAD-IP Reuse: An Academic Viewpoint

Andrew B. Kahng
CSE and ECE Departments
University of California
San Diego
La Jolla CA 92093-0114
abk@cs.ucsd.edu

Igor L. Markov
EECS Department
The University of Michigan
1301 Beal Ave.
Ann Arbor, MI 48109-2122
imarkov@eecs.umich.edu

## ABSTRACT

Mind-boggling complexity of EDA tools necessitates reuse of intellectual property in any large-scale commercial or academic operation. However, due to the nature of software, a *tool component* remains an ill-defined concept, in contrast to a *hardware component* (core) with its formally specified functions and interfaces. Furthermore, EDA tasks often evolve rapidly to fit new manufacturing contexts or new design approaches created by circuit designers; this leads to moving targets for CAD software developers. Yet, it is uneconomical to write off tool reuse as simply an endemic "software problem". Our main message is that CAD tools should be planned and designed in terms of reusable components and glue code. This implies that industrial and academic research should focus on (1) formulating practical tool components in terms of common interfaces, (2) implementing such components, and (3) performing detailed evaluations of such components. While this is reminiscent of hardware reuse, most existing EDA tools are designed as stand-alone programs and interface through files.

## 1. INTRODUCTION

The well-documented *design productivity gap* establishes, through historical data, that performance of electronic design automation (EDA) tools lags behind the capabilities of silicon. Depending on the market segment and development strategy, this mismatch can lead to ever-growing design teams and unsatisfiable human resource demands, soaring software costs and ever-steeper learning curves, slipped tape-out schedules and missed yield/performance targets - all contributing to loss of revenue. The design productivity gap can be attributed to the sheer complexity of EDA tasks with respect to both software integration and computational effort. Thus duplication of development, testing or evaluation effort carries a heavy price tag in terms of human and financial resources. Reuse of intellectual property becomes an absolute requirement for any large-scale operation.

A related concept in software engineering is that of a *software component*, and it is natural to specialize this concept to EDA tools. Unfortunately, the concept of a *tool component* is less well-defined than that of a *hardware component* (core), with its formally specified functions and interfaces. While this partly due to the nature of software, writing off tool reuse as a "software problem" would be a mistake. EDA tools must be frequently upgraded to address new manufacturing contexts (e.g., high process variability in subwavelength optical lithography) or new demands set by circuit designers. Such upgrades may involve changes to the interface and even the expectations for what the component does. They are often driven by expert users: many features in commercial EDA products are prototyped and/or suggested by circuit designers during the course of learning, configuring and monitoring EDA software. Also, the best designers are often those who are well-versed in CAD algorithms, and the best CAD engineers have designed chips.

In this paper, we focus on the *interoperability* of dynamically evolving CAD tools. Our main suggestion is that interoperability should be a planning consideration, rather than an afterthought, in the software design process. We discuss recent experiences of interoperability and design with tool components and conclude that CAD tools should be planned and designed in terms of reusable components and glue code. While it is hardly possible to suggest recipes of strict guidelines for design with components, immediate implications are that industrial and academic research should focus on (1) formulating practical tool components in terms of common interfaces, (2) implementing such components, and (3) performing detailed evaluations of such components. While this is reminiscent of hardware reuse, most existing EDA tools have been designed as standalone programs, and presently interface with other tools through files. We suggest that old tools be *re-factored into new tools* (via extraction of maximally generic, high-performance functional components) rather than *connected with new databases* through API layers or design exchange formats (via parsers). Given the complexity of EDA software and the rapid evo-

Proceedings of the Fourth International Symposium on Quality Electronic Design (ISQED'03)
0-7695-1881-8/03 $17.00 © 2003 IEEE

lution of design problems, there is a fundamental danger of misinterpreting design features, overlooking hidden assumptions, and hitting unexpected incompatibilities. These pitfalls only reinforce the need for well-tested, widely used high-performance components.

The remainder of this paper is organized as follows. In Section 2 we discuss various levels of CAD tool interoperability, mostly in an academic context that is distinguished by vendor-neutrality, limited development resources and very small average experience of circuit designers and CAD tool developers. Section 3 then provides two illustrations from recent experiences, namely, an example of loose interoperability and a sample academic open-source project. In Section 4 we formulate what we see as current challenges to interoperability and CAD IP reuse. Conclusions and on-going work are given in Section 5.

## 2. LEVELS OF INTEROPERABILITY

Inter-tool communication through design exchange files is very common and has obvious advantages over other methods. It is most straightforward, can be learned by example and supports regression testing well. One can easily capture intermediate results for sanity-checking or archival purposes. File-based communication supports software modularity since communicating tools do not need to be connected explicitly; in theory, they can be written in different languages and run on different platforms. The main perceived drawback of file-based communication is its speed.

Alternatives to file-based inter-tool communication rely on in-memory data structures, often object-oriented databases. This avoids the overhead of serializing and then parsing design information, and hence requires less runtime and disk space. When small circuits are designed on workstations with large and fast disks, these advantages may be negligible. At the other extreme, the savings from unified object-oriented databases may be limited due to asymptotic computational complexity. We observe that serialization and context-insensitive parsing of VLSI design data is typically feasible in time linear in database size (bytes). However, most design steps automated by EDA software typically take superlinear time in terms of database size. Therefore, if parsers and converters are implemented well and if EDA tools do not finish prematurely, communication through in-memory data structures will not noticeably accelerate the overall design process.[1] We expect that such asymptotic arguments are going to gain significance and overcome lesser trends with further growth of integrated circuits.

[1]The cost of large and fast disks has decreased to the point where it is negligible compared with the salaries of circuit designers and CAD engineers - or even the support costs of graduate students. Many important operations such as archiving, check-pointing and error recovery require serialization and parsing, thus making in-memory communication irrelevant.

Despite the above observations, our actual belief is that in-memory communication has marked advantages over file-based communication. Significant time-sinks and hidden costs of file-based communication are seen in the integration failures and various overheads that are associated with even milder inconsistencies of design exchange formats. Given the passage of years, thousands of users, and hundreds of flows and companies, the flexibility of file-based communication breeds incompatible format versions, parsers and file writers. For example, Cadence's LEF/DEF parsers tolerate a stunning number of deviations from the official Cadence LEF/DEF standard, e.g., (i) DEF keywords used as names for design objects, or (ii) design object names starting with digits rather than letter characters. Official format descriptions may have ambiguities that different parsers and file writers resolve differently (treatments of escaped names in many formats, or the various subversions of the GDSII Stream format, are classic examples). Indeed, it is not uncommon to see bizarre utilities such as "Verilog to Verilog translation" in any multi-company or multi-tool design flow. Anecdotal evidence indicates that such mismatches become noticeable in large-scale EDA tool development and may, in principle, lead to costly high-profile failures of the miles-versus-kilometers variety. While in-memory communication retains many of these drawbacks, it leaves significantly less freedom of interpretation and typically implies some kind of executable infrastructure that can verify or at least sanity-check the inter-tool communication process.

Importantly, in-memory communication *does not* require access to source code: it requires only that an API be documented. This requirement is comparable to having a good description of a design exchange format, but typically offers more to the user. API-based integration can be performed using software libraries, either static or shared (known as `.a` and `.so` files on Unix/Linux and `.dll` files on Windows). We believe that library-based integration of EDA tools is currently underexplored, despite the fact that linking to libraries has been very well-supported on all major operating systems for many years.[2] Finally, depending on the context, a possible boon to the use of in-memory communication is the availability of a high-quality design database that not only serves as a repository, but also provides basic services with respect to the design data stored in it. Such a database – OpenAccess [16] – has been recently proposed by SI[2] and is currently endorsed by a variety of companies ranging from Cadence to IBM. In fact, its source code is available through `http://www.si2.org/openaccess`. This may be useful if one needs to disambiguate the semantics of published APIs or suggest changes to the developers of Open Access.

[2]Of course, file-based communication does not require explicit software integration (linking to libraries or recompilation) and is therefore easier. Thus, for highly experimental projects where integration with existing design flows is known to be a non-problem, we see no reasons to avoid simple ad-hoc file-based communication.

## 3. RECENT EXPERIENCES

Proposed approaches to interoperability must, to be viable, acknowledge recent related practices. We now review recent experiences with (i) loose integration between tools developed in many different research groups, and (ii) tighter in-memory integration in an academic open-source project.

### 3.1 Loose Interoperability: The GSRC Bookshelf

The MARCO Gigascale Silicon Research Center (GSRC) [13] has for several years sought to enable CAD-IP reuse via the GSRC Bookshelf of Fundamental CAD Algorithms [9] (`http://www.gigascale.org/bookshelf/`). [3] The Bookshelf's original motivation was the "design *technology* productivity gap" – the lack of reusable components that enable rapid development and deployment new EDA capability – which contributes to the well-known "design productivity gap" cited above. Other motivations included recent experiences from such areas as electronic publishing, software repositories, algorithm evaluation methodologies [3, 5, 6], benchmarking [4, 1, 10], software engineering, combinatorial optimization [19], and open-source and free-software movements [18]. Original aims of the project included (i) establishment of a new *electronic medium* oriented toward implementations of fundamental VLSI CAD algorithms and their evaluation, (ii) common open standards for data representations and evaluation methodologies, and even (iii) community-wide culture change that credits leading-edge empirical results on par with theoretical results.

Today, the Bookshelf embodies a "publication medium" for CAD-IP that is focused on algorithm implementations, evaluation and related information. This medium is akin to an electronic publication, but also enjoys features of a software repository. In particular, its structure and processes connote "high-quality, reviewed, archival, citable and relatively hard to change once published", in concert with a "preview" (or "contrib") section where new contributions can be made visible by authors at any time and stay visible regardless of editorial decisions. The Bookshelf stresses the value of advances in implementations, not just the value of underlying theoretical innovations.

As the Bookshelf has become more established, several impacts have emerged. Specifically, it is: (i) a means of collecting and disseminating leading-edge knowledge on optimization algorithms; (ii) a usage-centric "community memory"; (iii) an infrastructure for reuse of CAD-IP that encourages uniform look-and-feel of contributions within each type of CAD-IP as well as interfaces, documentation and common testing/evaluation methods; (iv) a means of lowering barriers to entry (for all users, whether in academia or startups) caused by implementation complexity in mature CAD domains such as placement or logic synthesis; and (v) a means of accelerating the evolution and maturation of particular CAD domains.

In brief, the Bookshelf structure is as follows. *Slots* in the Bookshelf represent individual areas of VLSI CAD, such as graph coloring, Boolean satisfiability, technology mapping, hypergraph partitioning, block shaping and packing, global routing, scheduling, etc. Individual submissions are represented by *entries* embedded in slots. The main types of entries by function are:

- generic problems (standard file formats, standard in-memory representations (classes) to be passed to optimization engines, integrated I/O including parsers of standard formats, standard benchmark instances, and reasonably good or interesting solutions),
- reference solver implementations (usable in successful applications, comparable to best-known solvers, and supportive of modifications and performance analysis),
- independent evaluators, and
- heuristic evaluation and comparison methodologies (descriptions of testing procedures and best known results, precepts for experimental evaluation of metaheuristics, and references to relevant optimizers and benchmarks).

### 3.2 An Academic Open-Source Project

We now give an overview of a sample subject of EDA integration – an open-source academic project focused on circuit partitioning and layout: UCLA Physical Design Tools (PDTools) [17]. Most of the original PDTools development work was done by Andrew Caldwell and Igor Markov at UCLA, under the guidance of Professor Andrew Kahng. The publicly available software distribution, now maintained at the University of Michigan in Professor Markov's group, includes such tools as the MLPart circuit partitioner and the

[3] Caldwell et al. [9] note the following components of CAD-IP. (i) Data models that provide consistent semantics and structuring of data along different steps of design flows (ideally with an accompanying canonical API). (ii) Mathematical problem formulations for optimization and constraint satisfaction that isolate the fundamental difficulties in particular design tasks, and that encourage reuse of solvers. (iii) Use models and context descriptions for problem formulations. (iv) Testcases with corresponding high-quality solutions – both for individual problem formulations and integrated tool flows. (v) Algorithm descriptions and theoretical analyses. (vi) Executable implementations, not only for solvers (algorithms) but also for parsers and converters of interchange formats, legality checkers and cost function evaluators of solutions, etc. (vii) Leading-edge performance results, as well as standard comparison and evaluation methodologies for algorithms. These are needed to ensure that newly proposed methods are indeed improvements over previous methods. (viii) Software design and implementation methodologies. (ix) Source code of public-domain implementations.

large-scale circuit placer Capo [8]. The internal architecture of PDTools is highly modular; it is based on fundamental data structures and a number of interoperable solvers that operate on those data structures. While this development style at times requires additional programming effort, it simplifies debugging and performance optimization in the long run: bottleneck analysis is especially improved since the software architecture allows quick zooming in on buggy components and performance bottlenecks.

UCLA PDTools are distributed under the MIT open-source license, which requires neither fee nor author notification for any uses of the software, but stipulates only that the license be redistributed with all derivatives. The tools have LEF/DEF interface and run on Linux, Solaris and Windows. Historically, a serious effort was made to provide robust installation scripts, support revision control and ensure code clarity, but recent improvements and ongoing maintenance mostly focus on the quality of results. UCLA PD tools are available online from the MARCO GSRC Bookshelf gigascale.org, openeda.org, opencollector.org, etc. Academic users of PD Tools work at CMU, Georgia Tech, UCLA, UCSD, UCSB, UCSD, UMinn, UMich, etc. Known industrial users come from IBM, Intel, Cadence, Philips, Synplicity, AmmoCore and other companies.

To "guess-timate" potential needs for integration, we provide additional details on typical uses known to us. Some academic users work on competing tools and are interested in performance comparisons [15, 21, 20] In particular, homegrown sub-solvers can be plugged in place of existing software components and evaluated in a broader context. A related use model seeks improvement through pre- and post-processing [14]. For more comprehensive and realistic evaluation, individual tools are often embedded into larger design flows [2]. Some academic researchers modified UCLA PD Tools to work in contexts not originally considered by authors, notably 2.5-dimensional integration on chip [11].

Use models in the industry are somewhat different. It is typical to use source code for prototyping commercial tools (e.g., Cadence) and for educating industrial developers in new algorithms (IBM, Intel, Synplicity). Modularity greatly helps in this context because it simplifies performance evaluations and improves the understanding of source code. More research-like uses include experiments with novel Physical Design flows such as RTL-based placement before synthesis to increase the accuracy of wireload estimates. Executables are often used to benchmark and validate internal tools (Intel, IBM) and sometimes as back-up options in commercial design flows (IBM, Synplicity).

While the integration of UCLA PD tools with other software is not a solved problem, we would also like to mention several other lessons we learned from this project. First, we confirmed through our experience that incorporating modularity and interoperability into EDA software by design pays off: it leads to more robust and higher-quality software that is easier to tune, retarget, optimize and integrate with other tools. Second, modular and interoperable software improves user learning curve and, in fact, can be used as an aid in teaching sophisticated EDA algorithms. Third, modular and interoperable software is convenient in terms of distributed development because it encourages new components to be created and tested independently, followed by an integration step. Component-based development gives authors a sense of accomplishment. It also makes it easier for management to plan and evaluate accomplishments. Finally, we note that while these suggestions are fairly obvious, a typical development approach for a new tool exclusively focuses on a working prototype by all costs, and attempts to re-factor the prototype into a functional EDA tool. We suggest that this mentality be changed.

# 4. CURRENT CHALLENGES

With many academic research groups working today on competitive EDA tools, evaluation methodologies and embeddings into design flows become more important, leading to interoperability concerns. We note that file-based communication between EDA tools is acceptable in a typical research environment because the circuits designed are smaller and researchers do not typically race to meet stringent tape-out requirements. Researchers can also factor I/O time out of their experimental evaluation. Moreover, having human-readable snapshots from tool flows is convenient for debugging as well as for sanity-checking of individual tools and tool flows. Nevertheless, there are also strong arguments for in-memory integration of academic tools. They revolve around the benefits provided by high-quality open databases backed by the industry, such as OpenAccess v2.0 whose source code is publicly available as of January 2003 [16]. The promise of such databases, bundled with various parsers, is to provide a standard design repository with standard semantics and basic services. This could reduce start-up costs for tool developers, make tool comparisons more straightforward, and dramatically simplify the adoption of academic research outputs in industry. The main challenge to the adoption of standard design repositories such as OpenAccess 2.0 is the cost-benefit analysis. *Wide adoption can only be successful if users can offset learning curves by benefits such as the availability of realistic design examples and, most importantly, up-to-date industrial design flows.* In other words, new design repositories should come with detailed illustrations of best practices such that researchers can insert their point tools into known-good design flows and evaluate the overall performance on representative benchmarks. Studies based on "best practices" would leverage interoperability provided by new design repositories and associated services.

Regardless of the type of integration, serious design technology bottlenecks may be associated with glue code such as

file converters, API translation layers, control code, PERL scripts, etc. We believe that these bottlenecks can be addressed with additional degrees of automation that, unlike a typical PERL script, would support high-level integration concepts like "run until no improvement". Such automation would require an abstract and system-independent representation of design flows to ensure portability. An infrastructure for integration could automate software installation from source code, scheduling and launching of tool runs, followed by reporting of vital statistics in automatically formatted tables. As an end result, design flow prototyping and evaluation would be considerably simplified by leveraging interoperability of plug-and-play tool components.

Another challenge must be addressed simultaneously with the ones described above is the development of component-based algorithms and design frameworks, both in general terms and implemented in terms of common infrastructure. Such contributions should accelerate fundamental research in EDA algorithms and help its adoption in the industry. To this end, [7] proposes a component-based interpretation of the 20-year old Fiduccia-Mattheyses algorithm and shows that detailed experiments enabled by it expose serious implementation pitfalls. Importantly, the implementation of the Fiduccia-Mattheyses heuristic reported in [7] and available in the UCLA PDtools [17] is among the strongest reported to date. In terms of frameworks, a recent paper from IBM [12] proposes to view, implement and evaluate algorithms for placement and synthesis as transformations that can be chained in various ways. We point out that such a framework, besides looking "clean" and "right" from an academic viewpoint, would be very convenient to implement and evaluate as it would leverage common databases and other infrastructure. It should also be easy to parallelize the implementation of such a framework among multiple developers.

## 5. CONCLUSIONS

In this work we discussed interoperability requirements for EDA software from an academic point of view. In particular, we identified important drivers behind the push for interoperability as well as key obstacles. We argued that while specific, highly experimental research projects may, in principle, not need interoperability with existing tools, more near-term academic research would greatly benefit from such interoperability. A survey of recent experiences with integration and distributed development shows that so far interoperability has been achieved through file-based interfaces. However, such interfaces are both limited and error-prone. Among the main challenges for future interoperability is to develop and ensure wide adoption of common infrastructures enabling tighter integration. Importantly, such software engineering efforts must be supported by the development of new component-based algorithms as well as the reinterpretation of old algorithms in modular terms. Finally, we believe that the EDA development process must undergo a culture change and adopt component-based approaches to design, implementation, evaluation, maintenance, integration and evolution of EDA tools. In particular, it is important to develop and catalogize high-quality open-source tool components that can be used in education and research.

## 6. REFERENCES


[1] Collaborative Benchmarking Laboratory at NCSU, `http://www.cbl.ncsu.edu/www/`.

[2] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-blocks Using Floorplanning and Standard-Cell Placement", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, pp. 12-17, April 2002.

[3] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende and W. R. Stewart, "Designing and Reporting on Computational Experiments with Heuristic Methods", *technical report* (extended version of *J. Heuristics* paper), June 27, 1995.

[4] F. Brglez, "ACM/SIGDA Design Automation Benchmarks: Catalyst or Anathema?", *IEEE Design and Test*, 10(3) (1993), pp. 87-91.

[5] F. Brglez, "Design of Experiments to Evaluate CAD Algorithms: Which Improvements Are Due to Improved Heuristic and Which are Merely Due to Chance?", *Technical report* CBL-04-Brglez, NCSU Collaborative Benchmarking Laboratory, April 1998.

[6] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Hypergraph Partitioning for VLSI CAD: Methodology for Heuristic Development, Experimentation and Reporting", in *Proc. ACM/IEEE Design Automation Conf.*, June 1999.

[7] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Design and Implementation of Move-Based Heuristics for VLSI Hypergraph Partitioning", *ACM Journal on Experimental Algorithms*, vol. 5, 2000. `http://www.jea.acm.org/2000/CaldwellDesign/`

[8] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Produce Routable Placements?", *Proc. IEEE/ACM Automation Conf.*, Los Angeles, June 2000, pp. 477-482.

[9] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Toward CAD-IP Reuse: The MARCO GSRC Bookshelf of Fundamental CAD Algorithms", *IEEE Design and Test of Computers* 19(3) (2002), pp. 70-79.

[10] CATS: Combinatorial Algorithm Test Sets, 1999, `http://www.jea.acm.org/CATS/`

[11] Y. Deng and W. Maly, "Interconnect Characteristics of 2.5-D System Integration Scheme", *Proc. Intl. Symp. on Physical Design 2002*, p. 171-176.

[12] W. Donath et al., “Transformational Placement and Synthesis”, *Proc. Design Automation and Test in Europe (DATE) 2000*, pp. 194-201.

[13] The Gigascale Silicon Research Center, `http://gigascale.org`.

[14] B. Hu and M. Marek-Sadowska, “Congestion Minimization During Placement Without Estimation”, *Intl. Conf. on Computer-Aided Design (ICCAD) 2002*.

[15] P. H. Madden, “Reporting of Standard Cell Placement Results”, *Proc. Int’l Symp. on Physical Design*, 2001, pp. 30–35.

[16] Open Access Online, `http://www.si2.org/openaccess/`

[17] UCLA Physical Design tools, http://vlsicad.cs.ucla.edu/software/PDtools

[18] D. Rosenberg, “The Open Source Software Licensing Page”, The Worl-Wide Web, 1998, http://www.stromian.com/Open_Source_Licensing.htm

[19] S. S. Skiena, “The Stony Brook Algorithm Repository”, 1997. `http://www.cs.sunysb.edu/~algorith/`

[20] X. Yang, B.-K. Choi and M. Sarrafzadeh, “Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement”, *Proc. Intl. Symp. on Physical Design 2002*, p. 42-47.

[21] M. C. Yildiz and P. H. Madden, “Improved Cut Sequences for Partitioning Based Placement”, *Proc. Design Automation Conference*, 2001, pp. 776–779.