Early Research Experience With OpenAccess Gear: An Open Source Development Environment For Physical Design

Zhong Xiu* David A. Papa† Philip Chong‡ Christoph Albrecht‡
Andreas Kuehlmann‡ Rob A. Rutenbar* Igor L. Markov†

*Carnegie Mellon University, E&CE Dept., Pittsburgh, PA, USA
†University of Michigan, Dept. of EECS, Ann Arbor, MI, USA
‡Cadence Berkeley Labs, Berkeley, CA, USA

ABSTRACT

Physical design EDA research in academia has historically been based on infrastructure developed independently by individual contributors. This has led to fragmentation in the community, where interaction, data interchange and comparison of results between tools are difficult. We discuss our early experience with the *OpenAccess Gear* system, an open source software initiative intended to provide pieces of the critical integration and analysis infrastructure that are taken for granted in proprietary tools, but often wholly absent in research tools. Built on top of the widely available OpenAccess database, OA Gear provides components such as industrial-strength static timing analysis and extensible layout and netlist visualization. We discuss preliminary results from two on-going research efforts that have adopted OA Gear as their infrastructure: retrofitting the University of Michigan Capo placer into this environment, and the addition of a timing-driven capability to the Carnegie Mellon Warp placer.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Design, Experimentation, Standardization

Keywords

EDA, database, open source, physical design, placement, timing

1. INTRODUCTION

The physical design research community is highly fragmented. Individual academic tool developers tend to implement their own infrastructure, distinct from other people's works. For example, four leading-edge academic placement tools (Capo [10, 8], Warp1 [22], Dragon [21], and Feng Shui [25]) are each implemented on their own design database. Others have also commented on the fragmentation within the physical design community, for instance [18].

This deep fragmentation is the root of several problems in physical design research today. First, building the infrastructure requires a significant effort from already busy researchers. At the least, each individual design database requires the development of parsers and translators for processing standard file formats to that database. Second, without common infrastructure, it becomes difficult to integrate tools into larger flows, or to extend tools with additional functionality. As an example, it is difficult to tightly couple a timing engine with a placement tool to create a timing-driven placement experiment, if the timer and placer do not share a common design database. Finally, the lack of common infrastructure makes comparison of results between different tools problematic. Both [7] and [18] discuss this particular issue in detail.

The industry-standard OpenAccess (OA) database was developed to provide a common EDA infrastructure for physical design tools [1, 9]. While originally intended for adoption within industry, the release of OA as free open source makes it an ideal candidate for academic use. Moreover, adoption of a common database in the physical design community yields benefits for everyone, minimizing problems of fragmentation, easy result comparison, and the perennial problem of incompatible benchmark formats. However, despite the availability of OA for several years now, new academic tools are still built on top of *ad hoc* infrastructure.

The main shortcoming of OA in academic research has been the lack of a supporting environment of software components with higher levels of functionality: industrial-strength analysis, easy integration and visualization. While OA provides extensive support for low-level design database operations, this is of no help for, say, a graduate student researcher looking for a timing engine to plug into a placer. Industrial users could be expected to be able to build their own high-level components on top of OA by themselves, or buy them from other parties, but at the academic level such resources are lacking.

Because of this "infrastructure gap" in OA, we have initiated the *OpenAccess Gear* (OA Gear) project. OA Gear aims to provide an open source development environment with a library of tools and software components which algorithm EDA designers, both in academia as well as in industry, can use to extend or improve their own work.

The paper is organized as follows. Section 2 first surveys the OA Gear system itself and its core components. To illustrate how OA Gear can be exploited by academic research, Section 3 discusses three experiences using the tools: (1) prototyping a simplistic buffer insertion flow; (2) quickly integrating Capo, a mature and well-known placer, without committing to a complete rewrite of the tool; and (3) extending the recently introduced Warp placer to include a fairly complete timing-driven flow, done as a native integration. Section 4 offers concluding remarks.

2. OA GEAR COMPONENTS

Currently, OA Gear consists of four components:

- **Static timing analyzer** (OA Gear Timer)
- **User interface** (OA Gear Bazaar)
- **Benchmarks** (free and restricted cases)
- **Standard cell placer** (Capo)

These components were chosen for their perceived utility to the physical design community. We describe the first three of these in detail in this section. We describe the Capo integration in the following section.

2.1 OA Gear Basics

OA Gear is written in C++, and runs on any platform on which OA itself is available.

An early decision was made to have OA Gear be fully open source, free for any use. This has many advantages in an academic research setting. First, the transparency of open source makes it ideal for scientific exploration, where reproducibility of results is critical. Second, published papers often hide implementation details due to space limitations, leaving the actual source code as the only document describing complex algorithms completely. Third, open source encourages others to build on existing work and return improvements back to the community, benefiting all. The advantages of open source are well-known in the physical design community; see for instance [7, 18].

OA Gear is an ongoing project. The current components are an initial seed, to help promote the adoption of OA in the research community. In addition to continued development of OA Gear on our part, contributions of software from the community are welcome. We envision users of OA Gear contributing components representing their own research work, building up the OA Gear toolkit and making it even more valuable for others, while at the same time promoting their own research. Section 4 lists some of the components we hope will be added to OA Gear in future.

OA Gear was initially released on November 19, 2004. The project home page can be found at [2].

2.2 OA Gear Timer

Timing is a much-neglected area in academic physical design research, with a lack of generally accepted infrastructure [7]. The essential difficulties are:

- **Accuracy versus infrastructure effort**: At the beginning of a research project, one would like to validate *quickly* the utility of a new timing idea, without the effort of a complete industrial flow integration. Bluntly put, we would prefer *not* to spend a year integrating the necessary tool infrastructure, only to find out after the fact that our idea does *not* work. We seek to reduce the barriers to experimentation with more realistic technologies, timing models, and flows.
- **Comparability**: We also seek to make it easier to compare "apples to apples" among different research layout tools, as these tools mature and add capabilities. Though much abused and over-interpreted, core area and half-perimeter wirelength are at least reasonably useful as means of comparing layout quality. This is much less true for timing results, which make more serious demands on not only placement, but cell and technology models, routing models, timing verification tools, etc.

Some more mature tools, for instance APlace [16], Capo [15], and Dragon [24], have already made these serious integration efforts, and use a mix of academic (e.g. place, timing optimize, etc.) and commercial flow components (e.g. legalize, global and detailed route, timing analysis). Once in place, such academic/commercial flow "hybrids" can be enormously useful. However, tightly integrating industrial tools into an academic project can be a large task, given the potential differences between the underlying design databases. Interaction with core components may be limited to slow, inefficient file transfers. Finally, use of an industrial analysis tool such as a static timing engine can be problematic when comparing results, as such tools sometimes come with licensing restrictions preventing such comparisons. Our goal is to provide infrastructure to make future efforts easier, less costly in resources to complete, and moreover, easier to compare across different research groups.

Another approach which is sometimes taken in academic projects is to incorporate a static timing engine written specifically for that project; for example [13] and [14] take this approach. Not only is this a work-intensive undertaking, but often such code has very little potential for reuse outside of the original project. Ensuring correctness or fidelity to actual timing results can be difficult, as physical design researchers generally are not interested in learning the finer details of timing analysis. Finally, compatibility with industry standard data formats can also be a significant problem with such an approach. Therefore, OA Gear tries to provide a flexible, shared timing tool to avoid complete reimplementation on a per-project basis.

To address the lack of timing infrastructure in the EDA research community, OA Gear provides a static timing analysis tool called *OA Gear Timer*, which is comparable with industrial offerings. A few of the key features of OA Gear Timer are support for industry-standard timing library and constraint file formats, extensible wire delay modeling, and incremental timing analysis capabilities. We can summarize briefly the main features in the following bullets.

- **Approach**: OA Gear Timer follows generally accepted standard techniques for static timing analysis. Arrival and required arrival times and signal slew rates are maintained for all nodes in the circuit. Separate timing figures are kept for rising and falling signals. Internal gate delays utilize the standard interpolated two-dimensional lookup table based on output load and input signal slew rate.
- **Full timing mode**: OA Gear Timer has two modes of operation. *Full* timing analysis computes and stores the arrival and required arrival times and slew rates for *all* nodes in the design. Timing queries simply return the stored values for these figures. Under full analysis, if the netlist or delays are changed, the timing for all nodes is fully recomputed.
- **Incremental timing mode**: In contrast, *incremental* timing uses *lazy evaluation*, and only computes timing for a minimal subset of nodes of the design in order to satisfy any timing queries made. Techniques for preventing unnecessary recomputation in incremental timing analysis are well known. We choose a simple approach using *invalid flags* to

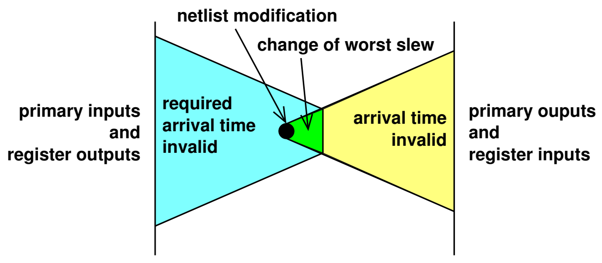

netlist modification
change of worst slew
primary inputs and register outputs
required arrival time invalid
arrival time invalid
primary ouputs and register inputs

Figure 1: Propagation of invalid flags.

indicate when any particular data item is potentially incorrect. When timing for a node is computed, it is cached and the corresponding invalid flag is cleared to mark the validity of the stored value (i.e. a form of *memoization*).

Whenever a modification is made to the netlist, the arrival times in the transitive fanout of the change become invalid and the appropriate flag is set. The slew is propagated forward from the modification and updated immediately. The required arrival time is marked invalid in the transitive fanin of the nodes right after the modification and of every node in the fanout for which the slew was changed.

Figure 1 shows the parts of the netlist for which the slew is updated, for which the arrival time becomes invalid, and for which the required arrival time becomes invalid. Our approach is similar to that of [17] except that we do not attempt to minimize the size of the change region. Invalid flags are simply propagated throughout the entire transitive fanin and fanout of changed nodes.

- **Wire Delay Modeling**: Both the ability to model delays due to wires and estimation of capacitive wire loading on drivers of nets are critical for timing-driven physical design. Currently there are two wire delay/load models in OA Gear Timer. One simply ignores wire delays, and the other estimates delay and load using the half-perimeter bounding-box as an estimate of routed wirelength. More sophisticated wire delay models require integration with the OA database and can be defined by users through a function callback mechanism. Such user-defined models are then automatically invoked during timing analysis. This flexibility allows arbitrary non-linear models to be added to OA Gear.

- **Standard File Formats**: It is vital not to underestimate the frustrations that "yet another file format" create in most academic research efforts. Thus, OA Gear Timer supports the standard timing library formats offered by Cadence (`.tlf`) and Synopsys (`.lib`, "Liberty"). The number of features found in these file formats is large, and we cannot support them all completely. However, there is sufficient support for basic timing analysis, such that useful experiments in physical design (such as timing-driven placement) can be easily performed. For timing constraints, a useful subset of the `.sdc` file format is supported, sufficient for use in timing-based physical design. `.sdc` commands which set the clock period, create external delays on primary inputs and outputs, set the driving cell for inputs and set load capacitance on outputs are all available.

- **Reporting**: The timing engine can generate human-readable timing reports, in addition to annotating the OA database with timing information. Several different reports can be created, including reporting the path having the worst slack in the entire design, reporting the path having the worst slack starting from, ending with, or passing through any given node, and reporting the slacks at all timing endpoints.

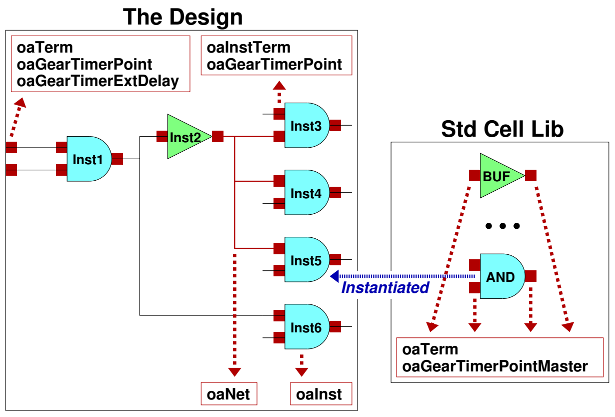


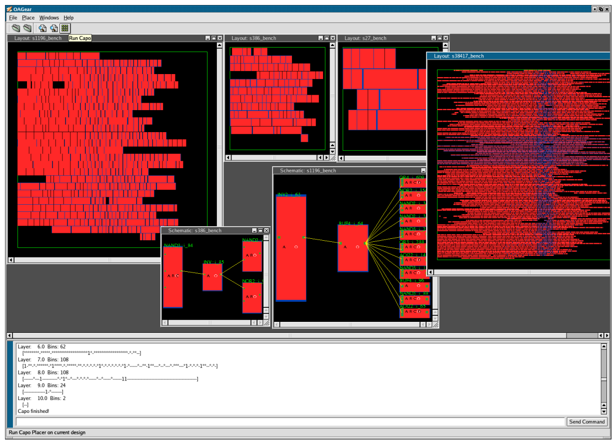
Figure 2: OA Gear model and interfaces for static timing.

- **Timer-Database Integration**: The OA database does not currently include direct support for timing, so we rely on annotating the database using the OA *extensions* (`appDef`) mechanism. Extensions allow objects in the database to be annotated with arbitrary data, so we use this to store the timing information. The instance terminals on all instances in a design are each given an `appDef` storing a unique *timerPoint*, a data structure which contains the arrival and required arrival times and slew rate associated with the corresponding instance terminal. To handle storage of internal gate timing arcs, the terminals on all master cells in the standard cell library are each given an `appDef` storing a *timerPointMaster*, which is a data structure containing the internal timing arcs associated with the corresponding terminal. See Figure 2.

  OA Gear Timer registers *callbacks* with the OA database so that, when an element (instance or net) of a design ever changes in the database, OA Gear Timer will automatically be notified, and can then set the invalid flags for the changed nodes and propagate these flags through their fanin and fanout cones as appropriate. This ensures that the timing information/invalid flags are *always* completely synchronized with the database itself.

The OA Gear Timer has been validated on a variety of public and proprietary benchmarks, and performs well. For example, a full timing analysis of a 50k cell design took less than 1 minute on a 2.0GHz Pentium 4, and only a few seconds to incrementally update timing for a typical single cell change. This is with the simple bounding-box wire delay model; of course, more sophisticated delay models will add to these runtimes. The timer output validates within 1% of Cadence's commercial RTL signoff timing flow across our initial benchmarking experiments.

There are currently a number of shortcomings of OA Gear Timer which may limit its use in certain cases. For instance, absent are capabilities for analysis across multiple clock domains, accounting for false paths and multi-cycle paths, and handling of transparent latches are absent. We intend to address some of these missing features in future releases of OA Gear, but for now we expect the



```
Layer:   6.0  Bins: 62
  [illegible]
Layer:   7.0  Bins: 108
  [illegible]
Layer:   8.0  Bins: 108
  [illegible]
Layer:   9.0  Bins: 24
  [-------------1-*-------]
Layer:   10.0  Bins: 2
  [--]
Capo finished!
```

Run Capo Placer on current design

Figure 3: OA Gear Bazaar screenshot showing layout (top) and netlist (center) views.

current tool to be sufficient to deal with many of the ordinary designs which can be found in academic settings.

2.3 OA Gear Bazaar: User Interfaces

Visualization is essential for debugging physical design algorithms. In placement, bugs can show themselves in obvious ways when the output is displayed as shapes rather than as lists of numbers. The graphical user interface (GUI) — called *Bazaar* — is thus an important part of OA Gear.

Two visualization tools are included with OA Gear. One is a *layout viewer*, intended for visualization of placement results. This tool displays a geometrically correct physical layout of a design, where instances (gates) are shown with their proper shapes and with their assigned locations. The second tool is a *netlist browser*, which displays simple shapes for the instances, not necessarily corresponding to their actual sizes, and with locations chosen by the netlist browser. This tool gives the designer the ability to easily view and verify netlist connectivity.

The architecture of the GUI tools was designed to be flexible and easily customizable by the end developer. All GUI components are programmed using QT, a popular cross-platform graphics toolkit [3]. QT is freely available for non-commercial use of OA Gear. The GUI also makes extensive use of OpenGL, a standard API for graphics acceleration, for improved performance on modern graphics hardware [4]. A sample visualization with several Capo placements and netlist views appears in Figure 3.

OA Gear also includes a command line interface component. This is intended to allow support for scripting, to provide for batch jobs and similar tasks which are amenable to text-based interaction.

2.4 OA Gear Benchmarks

Proper algorithm design hinges on having benchmarks available for testing quality of results. However, in many areas of research, especially timing-driven layout, the current sets of public benchmarks are incomplete and often lack useful scale, detailed sizing or pinout information, timing views, and real logical structure/intent. As the usefulness of OA Gear depends on having benchmarks in native OA format, it is critical that benchmarks be a fundamental component of OA Gear.

We have collected some benchmarks which we have divided in two categories. One group contains designs and libraries which are freely available for all uses, while the other group contains benchmarks which are restricted for use in non-commercial settings only. This distinction is necessary in order to allow OA Gear to be freely distributable.

Type	Name	PIs	POs	Instances	Registers
Free	s13207	32	121	2680	466
	s15850	15	87	4565	540
	s35932	36	320	11587	1728
	s38417	29	106	14762	1463
	s38584	13	278	12221	1292
Restricted	DMA	661	262	24942	2073
	DSP	575	269	24306	3550
	RISC	276	351	45455	7590

Table 1: OA Gear Benchmarks: Largest ISCAS89 designs (free) and Faraday designs (restricted).

- **Freely distributable benchmarks**: This benchmark suite is included as a part of the OA Gear distribution; it includes a standard cell library along with the ISCAS89 sequential logic benchmarks. The standard cell library is hypothetical: it does not correspond to any real library or technology process. However, the timing and electrical parameters have been chosen to resemble a typical 250nm process. The ISCAS89 benchmark designs are provided in technology mapped form using the given standard cell library. SIS [20] was used to map the 30 designs in the suite. The characteristics of the largest circuits from the benchmark suite are shown at the top of Table 1.

  These designs are relatively small, yet serve two important purposes. First, they allow new OA Gear users to start working immediately with the toolkit, without having to find suitable benchmarks elsewhere. Second, these designs are used as part of the regression test suite for OA Gear itself.

- **Restricted benchmarks**: A second set of benchmarks which carry restrictions regarding commercial use is also available in OA format. Because of these restrictions, these benchmarks cannot be included in OA Gear directly, but instead are available for download from a separate web site [5]. Table 1 (bottom) shows the characteristics for these designs. The restricted benchmarks also include the Generic Standard Cell Library (GSCLib), which is based on a hypothetical 180nm process. The designs for this benchmark suite come from the Faraday Structured ASIC test cases [6].

We are currently seeking to expand the set of benchmarks, technology libraries and electrical models available for OA Gear, especially in the direction of smaller technology nodes (i.e. 90nm and below). Contributions from the physical design community toward this goal would be most welcome.

3. EXPERIMENTAL RESULTS USING OA GEAR INFRASTRUCTURE

To validate and to illustrate the various components of the system and their potential utility across a range of research projects at different levels of maturity, we briefly describe here three different experiments, across projects at two different universities.

3.1 Simple Buffer Insertion

As a simple design exercise demonstrating some of the capabilities of OA Gear Timer, we look at the problem of *buffer insertion* for timing improvement. The goal here is to reduce the capacitive

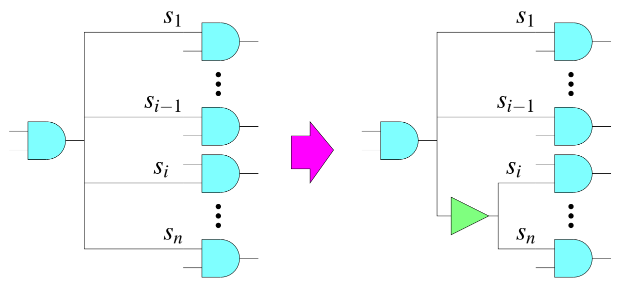


Figure 4: Simple buffer insertion example.

Name	Full (s)	Incr (s)	Full/Incr
s13207	59.50	1.31	45.42
s15850	121.76	3.72	32.73
s35932	2033.80	208.02	9.78
s38417	458.44	5.64	81.28
s38584	437.14	3.42	127.82
DMA	698.85	13.14	53.18
RISC	22189.07	2664.22	8.33
Avg			51.22

Table 2: Simple buffer insertion runtimes.

loading on gates which lie on the critical path by inserting buffers on the non-critical fanouts of such gates. If the buffer offers a smaller capacitive load than those fanouts which it isolates, then the timing on the critical path may be improved. Of course, the new buffer must not degrade timing on the non-critical paths to the point where some other path becomes critical instead.

Consider the following naive algorithm to find the optimum position for a single buffer:

1. Find the most critical path in the design by evaluating the slack at the primary inputs and registers and traversing the netlist from these points along the timing arcs, following the pins with the worst slack.

2. For each net on the critical path do the following:

   (a) Sort the sink pins of the net according to the slack in increasing order. Let the sink pins be $s_1, \ldots, s_n$ in this order. See Figure 4.

   (b) For each i, $1 \leq i \leq n$, insert a buffer which drives the sinks $\{s_i, \ldots, s_n\}$ and which is in turn driven by the original driver for the net. Evaluate the change in the slack at the driver. Remove the buffer and reconnect the sink pins.

3. Finally, insert the buffer at the position which showed the greatest timing improvement.

This is brute force to be sure, but does give a clear sense of the capabilities of the tools to allow *quick* experimentation. We implemented this algorithm in two different ways: first, using full timing analysis, so that timing information in the network is completely recomputed for each change to the netlist; second, using the incremental timing capability of the OA Gear Timer.

Table 2 compares the runtimes between these implementations; execution was on a 2.0GHz Pentium 4. For this experiment, using incremental timing was on average about 51 times faster than full timing.

3.2 Retrofitting Capo Placer into OA Gear

Placement forms a cornerstone of EDA. Because of its importance in the overall design process, including an interface to a mature placer in OA Gear was a top priority. We chose Capo [8, 10] as the initial placement tool for OA Gear, due to its open source nature and reputation for producing high-quality results.

However, Capo also has another virtue for this work: as an example of mature, nontrivial code base, we face the problem of how to *retrofit* the tool into the environment. This is a significant "barrier to entry" for many mature academic projects. In a perfect world, for optimal performance and ease of future integration, Capo would ideally be rewritten to run *natively* on OA data structures. However, this would be a difficult and time-consuming task. Hence, we chose to instead construct a *wrapper* around Capo. The wrapper reads placement information from the OA database, generates the corresponding Capo internal data structures, and invokes Capo. The placement results obtained are then written back into the OA database. All wrapper interaction is done in memory, avoiding slow file accesses. Including the wrapper code in the OA Gear serves the useful purpose of illustrating how one can re-engineer existing academic tools to take advantage of the OA environment.

Another advantage of using a wrapper-based approach to integrate placement in OA Gear is that the interface for placement can be easily standardized. Given two different placers, they each can be encased in a wrapper with the same interface. This allows for *mixing* usage of different placers as demanded by the task at hand. For example, it is relatively difficult to integrate individual timing constraints within a typical min-cut placement tool [12, 15]. However, there are various ways to incorporate such constraints in an analytic placement framework, for example [16]. It might therefore be advantageous to use min-cut placement on most of a design, which yields routability improvements, and switch to an analytic formulation for the timing-critical portions. Such integration of heterogeneous placement techniques would normally be *enormously* difficult, but with a common interface wrapper, two different placement tools can work smoothly together on different portions of the same design.

3.3 Integrating a Timing-Driven Flow Natively in the Warp Placer

The Capo placer represents one end of the integration/research spectrum: a large, mature tool with a significant existing codebase. The concern there is to be able to integrate with OA Gear expeditiously, to quickly take advantage of new features. The Warp placer, on the other hand, represents a different type of integration problem. Warp is still a very new, evolving placer. The wirelength-only version of Warp was introduced in 2004 [22], and extensions to handle timing, macroblocks, etc., are still under active investigation. Thus, Warp is at the other end of the spectrum of "integration complexity" as Capo. In particular, to enhance our ability to do future research on Warp, we are willing to abandon our *ad hoc* database infrastructure and replace it completely with OA and OA Gear components at a native level. In other words, we made the decision to integrate Warp placer tightly with OA and OA Gear. To illustrate this concretely, we undertook to add the first version of a timing-driven component to Warp using the OA Gear Timer and database infrastructure. Any such effort requires a significant investment to set the overall timing-flow in which the placer must function; we summarize our experience with OA Gear in this regard.

The Warp placer is based on a novel strategy called *grid-warping*. Grid-warping relies on a strikingly simple idea: rather than move the gates to optimize their location, we elastically deform a model

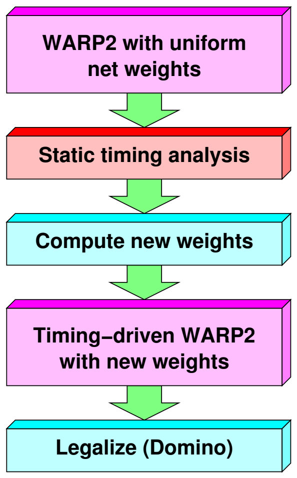

WARP2 with uniform net weights
Static timing analysis
Compute new weights
Timing–driven WARP2 with new weights
Legalize (Domino)

Figure 5: Basic flow for timing-driven Warp placer.

of the 2-D chip surface on which the gates have been crudely and quickly placed, "stretching" it until the gates arrange themselves to our liking. Put simply: *we move the grid, not the gates*. Deforming the elastic grid is a relatively simple, low-dimensional nonlinear optimization, and augments a traditional quadratic formulation. Details of the wirelength-only formulation appear in [22]. We decided to use OA Gear as the basis of our timing driven version.

Since our focus in this paper is on the OA Gear integration, we omit discussion about the mechanics of extending a grid-warping placer to include timing; see [23] for details. Roughly speaking, we use net-based delay budgeting ideas from the slack sensitivity model of [19], create a set of net weights for timing critical paths, and augment the following three steps to use these weights: (1) the initial quadratic placement, which locates gates on the "elastic sheet"; (2) the nonlinear elastic distortion step which drives the next phase of layout improvement; and (3) the min-cut-based partition improvement step by which warping recursively descends to finer and finer layouts.

As is common for such flows, we run placement iteratively, first without timing-based net weights, then again after invocation of static timing provides the slack data we need to update the weights. The initial version of our timing driven flow (see Figure 5) can be summarized as:

1. Run wirelength-only Warp placer with uniform net weights.
2. Run OA Gear Timer to obtain the slack for each net. Note that we use the generic 250nm cell library, with the simple bounding-box length-proportional model for wire delay.
3. Compute a new weight for each net using slack sensitivity.
4. Run timing-driven version of Warp with new weights.
5. Legalize final placement using Domino [11] as backend.

This flow is still rather simple, but preliminary results are encouraging [23]. We compared the results from the wirelength-only version of Warp with the timing-driven version, using the ISCAS89 sequential logic benchmarks. On average, the timing-driven version of Warp can improve the worst-case negative slack by about 36%, with only moderate increase in wirelength.

Perhaps more interesting is our rough accounting of the effort to execute this integration:

- **Native integration**: We spent about 1 week to replace Warp's original database with OA. We were, at this point, very familiar with the OA and OA Gear APIs. Our experience is that "running up" this learning curve takes about 4-6 weeks.
- **Timing integration**: We spent about 3 weeks to put in place the first rough version of the timing flow from Figure 5, at which point we could productively focus on the actual algorithmic and tuning details of adding delay budgeting and net weighting to our placer.
- **Code size**: We added only $\sim$ 1000 lines of code for the native integration and the rough timing flow, which is approximately 8% of the total placer package.

Our experience of putting in place this flow, from scratch, in roughly 1 month using the OA Gear components, suggest that we have at least partially achieved our stated goals of reducing the time and effort to put academic tools in more realistic flows. We estimate this probably would take 3-4 months to do from scratch in a typical proprietary commercial flow.

4. CONCLUSIONS

Physical design EDA research in academia has historically been based on infrastructure developed independently by individual contributors. This has led to fragmentation in the community, where interaction, data interchange and comparison of results between tools are difficult. We discussed some early experiences with the *OpenAccess Gear* system, an open source software initiative intended to provide pieces of the critical integration and analysis infrastructure that are taken for granted in proprietary tools, but often wholly absent in research tools. Built on top of the widely available OpenAccess database, OA Gear provides components such as industrial-strength static timing analysis and extensible layout and netlist visualization. By reducing some of the historically nontrivial "barriers to entry", we hope OA Gear promotes adoption of a standardized database infrastructure, unifying the fragmented landscape of physical design research. Preliminary results from on-going research efforts that have adopted OA Gear as their infrastructure — the Capo group at Michigan and the Warp group at Carnegie Mellon — have been extremely positive.

Looking forward, in addition to improving the existing components, we hope to extend the capabilities of OA Gear with other tools which EDA researchers will find useful, for example global and detailed routing, parasitic extraction, noise analysis, clock tree generation, and design for manufacturability. Contributions from the community in these and other areas are welcome.

Downloads of OA Gear are available through the project home page at [2].

5. ACKNOWLEDGMENTS

Zhong Xiu wrote OA Gear Timer, and David Papa wrote OA Gear Bazaar during internships at Cadence Berkeley Labs. We are grateful to Cadence and Lavi Lev for their support of the OpenAccess Gear project. Thanks also go to the OpenAccess development team, whose efforts form the base of our project. Work on the Capo placer at Michigan is supported by the Gigascale Systems Research Center. Work on the Warp placer and recent timing extensions at CMU is supported by the Pittsburgh Digital Greenhouse.

6. REFERENCES

[1] `http://openeda.si2.org/`.
[2] `http://openedatools.si2.org/oagear/`.

[3] `http://www.trolltech.no/`.

[4] `http://www.opengl.org/`.

[5] `http://crete.cadence.com/`.

[6] `http://www.faraday-tech.com/`.

[7] S. N. Adya et al. Benchmarking for large-scale VLSI placement and beyond. In *IEEE Transactions on Computer-Aided Design*, volume 23, pages 472–488, Apr. 2004.

[8] S. N. Adya et al. Unification of partitioning, placement and floorplanning. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 550–557, 2004.

[9] T. Blanchard, R. Ferreri, and J. Wilmore. The OpenAccess Coalition: The drive to an open industry standard information model, API, and reference implementation for IC design data. In *International Symposium on Quality Electronic Design*, pages 69–74, 2002.

[10] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection alone produce routable placements? In *Design Automation Conference*, pages 477–482, 2000.

[11] K. Doll, F. Johannes, and K. Antreich. Iterative placement improvement by network flow methods. *IEEE Transactions on Computer-Aided Design*, 13(10):1189–1200, Oct. 1994.

[12] B. Halpin, C. R. Chen, and N. Sehgal. Timing driven placement using physical net constraints. In *Design Automation Conference*, pages 780–783, 2001.

[13] A. P. Hurst, P. Chong, and A. Kuehlmann. Physical placement driven by sequential timing analysis. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 379–386, 2004.

[14] A. B. Kahng, S. Mantik, and I. L. Markov. Min-max placement for large-scale timing optimization. In *ACM International Symposium on Physical Design*, pages 143–148, 2002.

[15] A. B. Kahng, I. L. Markov, and S. Reda. Boosting: Min-cut placement with improved signal delay. In *Design, Automation and Test in Europe*, pages 1098–1103, 2004.

[16] A. B. Kahng and Q. Wang. An analytic placer for mixed-size placement and timing-driven placement. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 565–572, 2004.

[17] J.-F. Lee and D. T. Tang. An algorithm for incremental timing analysis. In *Design Automation Conference*, pages 696–701, 1995.

[18] P. H. Madden. Reporting of standard cell placement results. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2):240–247, Feb. 2002.

[19] H. Ren, D. Pan, and D. Kung. Sensitivity guided net weighting for placement driven synthesis. In *ACM International Symposium on Physical Design*, pages 10–17, 2004.

[20] E. M. Sentovich et al. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, University of California Berkeley Electronics Research Laboratory, May 1992.

[21] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *International Conference on Computer-Aided Design*, pages 260–263, 2000.

[22] Z. Xiu, J. D. Ma, S. M. Fowler, and R. A. Rutenbar. Large-scale placement by grid-warping. In *Design Automation Conference*, pages 351–356, 2004.

[23] Z. Xiu and R. A. Rutenbar. Timing-driven placement by grid-warping. To appear in *Design Automation Conference*, 2005.

[24] X. Yang, B.-K. Choi, and M. Sarrafzadeh. Timing-driven placement using design hierarchy guided constraint generation. In *International Conference on Computer-Aided Design*, pages 177–180, 2002.

[25] M. Yildiz and P. H. Madden. Global objectives for standard cell placement. In *Great Lakes Symposium on VLSI*, pages 68–72, 2001.