Standard Data Representations for VLSI Algorithm Development

D. Hertweck, M. Nica, S. Park, and C. Purdy
(email: carla.purdy@uc.edu)
ECECS Department, University of Cincinnati, Cincinnati, OH 45221-0030

Abstract

Because so many important problems arising in VLSI design are NP-hard, VLSI algorithms must employ randomization techniques or heuristics. Thus the process of analyzing a new algorithm or of comparing two algorithms is at present an experimental one. Consequently, progress in VLSI algorithm development must be based on references to standard benchmarks. Yet examination of literature on specific problems, such as graph partitioning, shows that such standardization is not yet a reality. Here we describe a system, Circuitbase, which we are developing to address the standardization problem. Circuitbase will combine the extensive graph manipulation routines of Knuth's Stanford Graphbase package with actual circuit examples from the Benchmark Archives at CBL, standard routines for generating random examples of circuits, and standard methods for algorithm analysis. We describe Circuitbase versions of example behavioral, structural, and physical views of a VLSI circuit and discuss how Circuitbase can support modern VLSI design environments.

1. INTRODUCTION

For some time now progress in development of circuit and system design tools has lagged behind rapidly increasing circuit and system capabilities and complexity. Currently it is estimated that achievable hardware complexity is almost 100 times greater than it was in 1980, while tool capabilities are only about 7 times as great as in 1980 [16]. As million-device chips become commonplace, as feature sizes continue to decrease, and as the demand for multitechnology chips integrating digital, analog, optical, and mechanical devices continues to grow, narrowing this design gap or even keeping it constant will require more and more sophisticated tools for all stages of the design-simulate-test process.

In Figure 1 we have summarized the views and levels of abstraction typically present in a VLSI development environment [6] and the tasks which this environment must support. We usually identify three domains--behavioral (B), structural (S), and physical (P)--and several levels of abstraction within each domain. Producing a component or system amounts to traversing a path through some subset of these levels and views to complete all the necessary tasks--specification, design, simulation, fabrication, and testing (and ongoing maintenance). For example, a designer producing an FPGA (field programmable gate array) prototype might trace out a path beginning at B4, visiting B3 and B2, and ending at B0. The actual physical specifications for the part, which exist in the physical domain P, would be automatically generated by tools in the design environment as necessary. If a custom design is later desired for this component, then a new path which includes another view at P0 must be created by the designer.

To deal with increasing circuit complexity, modern VLSI design environments emphasize rapid prototyping, supported by high-level behavioral descriptions of components, extensive simulation before fabrication, and greater design reuse. Such strategies, which have been developed mainly for digital circuits, are now being extended to analog, mechanical, and other domains. To be effective, design environments which enable necessary path traversals must be supported by highly sophisticated design automation tools, based on better algorithmic strategies capable of efficiently solving more and more computationally intensive problems in ways which are transparent to the circuit or system designer Builders of these tools must therefore have available sufficient information about algorithm performance to make intelligent choices about which algorithms to implement in their systems.

```
====================================
              TASKS
   Specify-Design-Simulate-Fab-Test-
              Maintain
                |||
                \/
====================================
L:                VIEWS
E:
V:Behavioral   Structural    Physical
E:   (B)          (S)           (P)
L:
S:
 :___________________________________
4:Specs        CPU          Partition
 :-----------------------------------
3:Algorithm    Data           Cluster
 :             Structure
 :-----------------------------------
2:Register     ALU,             Floor
 :Transfer     Register          Plan
 :-----------------------------------
1:Boolean      Gate,             Cell
 :Equation,    Flip-flop
 :FSM
 :-----------------------------------
0:Transfer     Transistor,     Layout
 :Function     Wire
====================================
                /\
                |||
   Specify-Design-Simulate-Fab-Test-
              Maintain
               TASKS
====================================
```

Figure 1: Example VLSI Views

It is well known that most of the significant problems which must be solved in these environments are NP-hard, that is, they cannot be solved by an algorithm which finishes in a time which is polynomially related to the input size. Thus development of a usable algorithm amounts to experimentation, in which techniques such as randomization or problem-specific heuristics must be employed. In particular, the two major questions which an algorithm developer must answer, namely, (1) how well does my algorithm perform, and (2) how good is my algorithm compared to other known algorithms for this problem, can almost always only be answered by reference to a set of benchmarks.

2. BENCHMARK SETS--NECESSARY CHARACTERISTICS

To provide effective support for VLSI algorithm development, a benchmark set must do the following:

--allow for an experiment to be repeated, with the same outcomes, by independent researchers;

--contain a comprehensive set of benchmark examples, all of which will be used to evaluate a new algorithm;

--provide a well-defined i.d. number for each circuit or circuit primitive in the benchmark set so that results obtained by different researchers can be compared easily;

--provide standard "metrics" for the evaluation of algorithm performance on the data, along with easy-to-use routines for evaluating these metrics and for generating reports;

--contain data which is relevant to the problem at hand (e.g., data which relates to realizable circuits as opposed to abstract graph representations only);

--contain at a minimum the "best" and "worst" cases of example data for all algorithms developed so far;

--contain standardized routines, as easy to use as random number generators, for random generation of data, so that results based on random data can be replicated by independent researchers;

--be expandable so that important new examples can be included as they are found and so that algorithms previously developed can be tested on these new benchmarks;

--represent the data in some standard initial format and provide conversion routines to other standard formats (for graphs, for example, adjacency list and adjacency matrix representations should be provided or easy to generate);

--allow for useful subclasses of benchmarks (for example, "sparse" circuits") to be extracted from the total set;

--provide standard data manipulation routines for reading, writing, etc., so that an algorithm developer can concentrate on the algorithm under development and so that "housekeeping" tasks are simple to do and done in a standard manner;

--be easy to access and use.

The Circuitbase software is being developed to provide benchmark sets which meet all these criteria and which can support evaluation of algorithms for VLSI design environments.

3. EXAMPLE--GRAPH PARTITIONING

Currently methods of VLSI algorithm analysis are much more ad hoc, and thus it is often impossible to compare specific results or to choose an algorithm for a specific practical situation. As an example, we consider the problem of graph partitioning [17] This is a fundamental high-level "physical domain" problem on which many researchers have worked. As a specific example we can consider the situation where we have a circuit (combinational or sequential) represented as a graph G = (V,E) in which each vertex V is chosen from a fundamental set of gates (for example, NOT, 2-input AND, 2-input OR, 2-input XOR, D flipflop, JK flipflop, input and output pads) and each edge E represents a wire connecting the output of one gate with the input of one or more other gates. For each gate, a set of characteristics, such as area, function computed, and minimum and maximum switch times, may be defined. Edges may also have characterisitcs, such as length, defined for them. A simple version of the partitioning problem in this situation requires that we split the vertex set V into two subsets of equal cardinality, V' and V'', such that the sum of the edge weights over all edges with one end in V' and one end in V'' is minimized. This is well-known to be an NP-hard problem [11]. One common modification, in which we require the total area of vertices in V' to be about the same as the total area of vertices in V'', is also NP-hard. A partitioning problem of this type must be solved, for example, when FPGA design software splits a circuit into modules to be implemented in different devices.

The current state of the problem of standardizing data representations and algorithms in general is well explicated in [1] in the case of the partitioning problem. This description makes clear the need for a unified approach such as we are proposing here. For the partitioning problem alone, there are many possible choices, not only for the data representation of the input, and the benchmark circuits, but also for the exact problem statement itself. Thus it is very difficult to compare the results of two researchers, unless comparisons are explicitly stated in the research reports, and it is almost impossible for a system developer to be confident of making an appropriate choice for a basic algorithmic strategy to implement or an improvement that would improve system performance. Clearly this state of affairs can only hinder the development of the powerful VLSI design tools needed now and even more necessary in the near future.

4. CIRCUITBASE--REPRESENTING CIRCUITS AS GRAPHS

To support the development of VLSI design systems in general and the standardization of evaluation of VLSI algorithms in particular, we have begun development of a package which we call Circuitbase [12,13]. Circuitbase will enable more efficient system development by providing a data environment which is flexible, supports many different tasks, and can be used in a general setting or customized for particular tools and technologies. Circuitbase is designed to encourage much more extensive benchmarking of algorithms than is typically the case at the present time. Ongoing extensions of the work described here will eventually result in a comprehensive systems development tool which can be included in a development environment such as Ptolemy [15].

Circuitbase is built on top of the Stanford Graphbase (SGB), designed by Knuth [10]. It will combine the extensive graph manipulation routines of SGB with real-life examples from the MCNC benchmark sets [3], with the ability to generate random examples of circuits, and with routines for statistical analysis of output and report generation. In Circuitbase a particular circuit or network of circuits is represented by a modified adjacency list form of a graph. This allows all the power contained in the original SGB package to be applied to the problems we are attempting to solve. Circuitbase has particular applicability to nonnumeric problems such as partitioning, logic minimization, reconfigurability, and general routing problems. Circuitbase views will be most useful in the physical and structural domains, rather than in the behavioral domain.

The ideas behind Circuitbase are a unification of three main trends in combinatorial computing in general and in VLSI algorithm development in particular

--the development of powerful systems for manipulating and studying graph-based problems,

--the collection of benchmark sets of actual circuit data, and

--the progress in generating classes of random graphs which important properties with graph representations of actual circuits.

A comprehensive discussion of the current state of the art in combinatorial algorithms in general can be found at the Stony Brook Algorithm Repository [18], where a large collection of problems and algorithms for solving these problems is available. Many algorithm implementations are part of general graph manipulation packages, several of which are also described. Many of these packages have been developed to support the theoretical study of graphs and/or the visualization of graphs. For current VLSI problems, the graphs involved tend to be too large for visualization to be very effective. Also the graph generation routines attached to most of these packages do not support the generation of graphs with the specific real-life characteristics, which circuit-related graphs tend to possess [8]. From among the many packages available, we have chosen the Stanford Graphbase (SGB) as the basis for Circuitbase. SGB is compact (the source code for the SGB kernel, including extensive documentation, takes up only about 75 Mb) and comprehensive, and is rated highly by the developers of the Stony Brook Algorithm Repository. More important, it was developed originally to deal with "real-world" problems (specifically to manipulate data sets arising in compiler development and operations research [10]) . Thus its original purpose is in line with the project we are describing here. Finally, the original SGB already contains a module gb_gates which supports a structural view of circuits.

The most comprehensive collections of actual circuit data are probably those archived at [3]. These archives contain benchmark sets collected over the years from various sources and targeting many specific VLSI-related problems, including technology mapping, partitioning, timing, routing, logic synthesis, simulation, and testing. Each benchmark category has its own data format or formats. These sets are an invaluable resource, but the wide variety of sets and formats available means that two researchers choosing benchmarks for evaluating an algorithm for a specific problem may not choose the same benchmark sets or the same format for a given set. Thus they may not actually be using the same test data. Another source of benchmarks is [14]. Other sources of actual circuits, primarily industrial, have also been employed (as, for example, in [5]), but they have not been used consistently by a wide range of authors.

A promising relatively recent development has been the generation of "circuit-like" random graphs [4,7,8,9]. Consistent use of random graphs generated according to one of these schemes is a valuable addition to the use of actual circuit data. Care must be taken, however, to ensure that circuit interpretations are uniform from one algorithm analysis to another and that parameters for generating the graphs are clearly given if these circuits are used to determine the performance of a particular algorithm or algorithms.

5. SGB--A BRIEF DESCRIPTION

The complete SGB package is in the public domain. SGB is written in the language CWEB, which is a mix of C and TEX. This language is an example of what Knuth calls "literate programming", since the actual program files are self-documenting. The source files which make up SGB should not be modified. Circuitbase uses the built-in SGB "change file" mechanism which allows local customization without changes to the master files. The SGB files can be compiled to C or used to produce TEX documents by using the programs CTANGLE and CWEAVE, which are included in the SGB package. Currently most of our modifications and additions to SGB, which make up Circuitbase, are written in standard ANSI C and have been run on SUN workstations and on PC's. Some of our extensions are written in C++. As the Circuitbase user interface is developed further, more extensions will be in C++ rather than in C.

The kernel of SGB consists of four modules:

--gb_graph defines standard vertex, arc, and graph data structures and includes routines for storage allocation. The basic graph data structure is a linked list adjacency representation of the graph adjacency list;

--gb_io performs input and output;

--gb_flip is a portable random number generator;

--gb_sort performs sorts on 32-bit keys.

SGB also includes a mechanism for comparison of algorithms called "mem-counting." This reports running time of an algorithm in terms of references to memory and thus provides a machine-independent measure of efficiency.

SGB elements contain user-modifiable "utility fields", each of which can hold a pointer to a vertex, a pointer to an arc, a pointer to a graph, a string, or an integer. This allows graphs, vertices, and arcs to be customized to represent problem-specific data. The size of an SGB graph is limited only by the resource constraints of the system on which SGB is running.

The basic data structure in SGB is a graph, G, defined as a C structure. A graph G consists of vertices and arcs, each of which is also a structure. The vertices of a graph are stored in an array, with the arcs coincident with a given vertex stored in a linked list (adjacency list representation). The basic SGB graph is directed, but undirected edges can also be represented.

For our purposes, the most interesting special purpose module in SGB is gb_gates, which contains logic gates (OR, AND, NOT, XOR, and a latch) as primitive vertices and from which various circuits can be constructed and functionally simulated using a built-in function gatè_eval. The module gb_gates comes with built-in routines prod (m,n), for constructing a parallel multiplication circuit with m-bit multiplicand and n-bit multiplier, and risc(r), for constructing a reduced instruction set computer with r registers. The gb_gates module essentially provides an S1 (structural, level 1) view of VLSI components and supports the creation of higher level structural views, including "prod" and "risc".

6. CIRCUITBASE EXTENSIONS TO SGB

Circuitbase, which contains SGB and our extensions to it, currently supports a physical view, a structural view, and a behavioral view of a digital circuit.

6.1. PHYSICAL VIEW

The physical view which we support is based on the Partitioning93 benchmark from CBL [3]. This benchmark consists of a library of 28 distinct combinational gates, with associated physical information. and a collection of 31 circuits which have been built by defining nodes consisting of one of the 28 gates and connections, or edges, between the nodes. The circuits range in size from 10-20 nodes up to about 100,000 nodes. Most of the circuits are "sparse", i.e., the number of edges in each circuit is no more than two or three times the number of nodes in the circuit. Since the specifications of the 28 combinational gates include physical information, the circuits can be viewed as abstract graphs or as partial physical instantiations of the actual circuits (where gate area, gate functionality, gate delays, and connectivity are defined, but placement of each node is undefined, e.g.).

In our Circuitbase representation, cell functionality is stored in an array of truth tables, one for each primitive gate, while physical information is stored in a separate, corresponding, array. Thus it is easy to replace specific physical information about the set of primitive gates with another set of values. In fact, we can define as many different (physical) versions of a cell set as we desire. We can also ignore the physical information and view the circuits as abstract "lookup tables" (LUT's) if we wish to concentrate only on the abstract graph theoretic properties of the circuits we are studying.

Currently we are using this physical view to develop a protocol for design and analysis of partitioning algorithms. We have identified four classes of partitioning algorithms which are common in the literature--Kernighan-Lin type strategies, simulated annealing, genetic algorithms, and straightforward greedy methods [11,17]. We have chosen one representative of each of these classes of partitioning algorithms for extensive study. Using the Circuitbase representations of the Partitioning93 benchmark circuits, we are concentrating not on developing better algorithms but on developing comprehensive and robust methods for comparing these four basic algorithms and modifications of these four algorithms. With the support of SGB and the routines we have added, an algorithm designer can easily parameters the different inputs to a partitioning algorithm and determine how changes in these parameters will affect algorithm performance. We are considering two versions of the partitioning problem. In the simplest, each circuit is treated as a graph with unweighted nodes and edges and our objective is simply to partition the vertex set V into two halves, V' and V'', with a minimal number of edges between V' and V''. In the second version we are also attempting to balance the total area of the vertices in V' with the total area of the vertices in V''. For each version of the problem we are interested in studying, we define a cost function on the Circuitbase graphs. The cost function can be very general, taking into account, for example, vertex area, edge length, number of edges between the partitions, or any other measure deemed reasonable. By relying on the SGB housekeeping routines which have been incorporated into Circuitbase, the algorithm developer can spend less time on messy graph manipulation and more time experimenting with various cost functions and choices for the algorithm parameters.

6.2. DEFINING STRUCTURAL AND BEHAVIORAL VIEWS

We have also implemented structural and behavioral views in Circuitbase.

The structural view is based on the LGSynth91 benchmark set [3], for which extensive documentation is available Since the primitive gates in this benchmark set include latches, our Circuitbase implementation of these gates must include data fields which store both "previous" state and "current" state. Logic gates in this benchmark set which have exact counterparts in the partitioning benchmark mentioned above can be linked.

The behavioral view which we have implemented initially is based on the basic SGB module, gb_gates and the associated risc generator module risc(r). We have translated this risc architecture description into AHDL, a high level description language supported by the Altera toolset [2]. Thus we can take a behavioral AHDL description of a component and map it either to a corresponding structural or physical view, based on its gate-level implementation. The Complex Programmable Logic Devices (CPLD's) parts supported by the Altera tools are structurally and physically very different from the primitive gates included in the partitioning and logic synthesis sets described above. They represent yet another circuit view which must be accommodated in a comprehensive design system. Our work so far has begun the process of defining reliable pathways between these views.

A much lower level behavioral description of a circuit is the SPICE simulation view. Currently this is the level at which designers of analog circuits, optical interconnect, and MEMS (microelectromechanical systems) are most likely to be working. We have begun the work of including a SPICE-level view in our system. Currently we are evaluating the necessity for generalizing our circuit representation to include hypergraphs to support this level, since we want to be able to include subcircuits efficiently in our SPICE descriptions.

7. CONCLUSIONS

Circuitbase, the system we have described here, extends the concept of a well-supported tool for combinatorial computing to the field of VLSI circuit and system design. Circuitbase currently includes several examples of benchmark data from actual technologies and circuits, as well as the primitive gates and routines from the gb_gates module of SGB. At present we are implementing several extensions. These include:

--addition of randomly generated circuit graphs. This may involve developing new techniques or providing routines to convert data available from other sources (e.g., [4,7,8,9]) into Circuitbase format;

--addition of a view which includes SPICE information. As noted above, this may eventually lead to generalizing the basic Circuitbase structure to include hypergraphs. A standardized lower-level behavioral view could become extremely important for maintenance and for circuits which include nondigital components;

--further development of object-oriented concepts to support system modularity; inclusion of object-oriented concepts could improve standardization of algorithm comparisons.

Future work planned also includes exploring the addition of Knuth's "literate programming" concept to our system, development of new VLSI algorithms, particularly for partitioning, and comparison of these with existing algorithms.

REFERENCES

1. C. J. Alpert and A.B. Khang. Recent directions in netlist partitioning: a survey. *Integeration* 19, Aug. 1995, 1-81.
2. *Altera Max+Plus II AHDL.* Altera Corporation, 1995.
3. The Benchmark Archives at CBL. http://www.cbl.ncsu.edu/pub/Benchmark_dirs/
4. J. Darnauer and W. Dai. A method for generating random circuits and its application to routability measurement. *Fourth ACM.SiGDA International Symposium on FPGAs, FPGA96*, 66-72.
5. A. Dasdan and C. Aykanat, Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* 16 (2), Feb. 1997, 169-178.
6. D.D. Gajski, ed., *Silicon Compilation*, Addison-Wesley, 1988.
7. D. Ghosh, Circuit benchmarks as random mutations generated from a compact directed bipartite graph signature. http://www.cbl.ncsu.edu/publications/1997-Talk-05-Ghosh/1997-Talk-05-Ghosh.bib.
8. M. Hutton, J.P. Grossman, J.S. Rose, and D.G. Corneil. Characterization and parameterized random generation of digital circuits. *33rd ACM/SIGDA Design Automation Conference*, June 1996, 94-99.
9. M. Hutton, J. Rose, and D. Corneil, The circuit characterization and generation project at the University of Toronto. http://www.eecg.toronto.edu/~mdhutton/gen/.
10. D. Knuth. *The Stanford Graphbase*. ACM Press, 1994.
11. T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, 1990.
12. M. Nica. Extensions of Stanford Graphbase to support computer systems development. M.S. Thesis, University of Cincinnati, 1997.
13. M.Nica and C. Purdy, Generating benchmark data using Stanford Graphbase. Presented at First CERC Workshop, Ohio State University, 1996.
14. Programmable Electronics Performance Corporation, PREP PLD, *Benchmark Suite #1, V1.2*, Los Gatos, CA, 1993.
15. Ptolemyhomepage. http://www.ptolemy.eecs.berkeley.edu/
16. RASSP Workshop: Digital Design 2000, University of Cincinnati, March 1997.
17. S. Sait and H. Youssef, *VLSI Physical Design Automation*, McGraw-Hill, 1995.
18. The Stony Brook Algorithm Repository, J.R. Bradley and S.S. Skiena. http://www.cs.sunysb.edu/~algorith/index.html