Magic: An Industrial-Strength Logic Optimization, Technology Mapping, and Formal Verification Tool

Alan Mishchenko Niklas Een Robert Brayton
Department of EECS
University of California, Berkeley
{alanmi, een, brayton}@eecs.berkeley.edu

Stephen Jang Maciej Ciesielski
LogicMill Technology
San Jose, CA / Amherst, MA
{sjang, mciesielski}@logic-mill.com

Thomas Daniel
Abound Logic
Santa Clara, CA
tdaniel@aboundlogic.com

ABSTRACT

This paper presents an industrial-strength CAD system for logic optimization, technology mapping, and formal verification of synchronous designs. The new system, Magic, is based on the code of ABC that has been improved by adding industrial requirements. Distinctive features include: global-view optimizations for area and delay, scalable sequential synthesis, the use of white-boxes for instances that should not be mapped, and a built-in formal verification framework to run combinational and sequential equivalence checking. Comparison against a reference industrial flow shows that Magic is capable of reducing both area and delay. Experiments on a suite of industrial FPGA designs show that LUT count is reduced by 12.7%, flip-flop (FF) count is reduced by 9.4%, FF-to-FF level is reduced by 22.3%, and fMAX is improved by 11.8%. A remarkable consequence of these reductions is that, although Magic itself takes time to run, the total runtime of the design flow is reduced.

1. INTRODUCTION

Although manual design efforts are still used, several decades of academic research in electronic design automation (EDA) have produced efficient methods for synthesis and verification of large industrial designs. In most cases, the progress in academia is paralleled by that in industry; engineers build commercial CAD tools by utilizing the results of academic research while making substantial contributions of their own.

The "technology transfer" between academia and industry takes many forms. A typical situation is, a researcher publishes a paper describing a new idea; an engineer reads the paper, implements the idea, and improves a commercial tool. Another example is that of a graduate student who interns at a company and later joins as an employee, bringing along knowledge gained in graduate school. Yet another scenario is an academic research group develops a tool, which is then evaluated by industry and adopted to fill the gap in an existing design flow.

This paper presents an example of the latter type involving ABC, a public domain tool. ABC represents about eight years of active research and software development and improves on previous generations of logic synthesis and verification solutions, exemplified by Espresso, SIS, MVSIS, and VIS.

ABC has piqued the interest of several companies who either learned from it and implemented comparable solutions, or re-used it in their own tools with varying degrees of success. However, what is often missing in these industrial efforts, is the sharing of information: which parts of ABC were found helpful, how were these parts integrated into an existing flow, what QoR, memory, runtime improvement were achieved on large designs, etc.

The present paper outlines the result of integrating ABC into one commercial flow and shows the results produced.

We named the result of this integration "Magic" to distinguish it from ABC as a public-domain system. The two are closely related but not the same: ABC is a store-house of implementations called application packages, most of which are experimental, incomplete, or have known bugs, while Magic integrates and extends only those features that create a robust optimization flow.

Magic features an all-new design database developed within ABC to meet industrial requirements. The database was developed from scratch, based on our experience gained while applying ABC to industrial designs. It reduces the memory requirements and runtime of the integration and addresses some known limitations of ABC, such as the inability to work with multiple clock-domains, flops with complex controls, and persistent instances that should be skipped by the mapper (white and black boxes).

Magic represents a whole greater than the sum of its parts:

- scalable sequential synthesis
- fast local transformations
- iterative computations
- mapping with structural choices
- global view optimizations
- the use of white-boxes

Each of these aspects can bring an improvement independently in a synthesis tool. However, in combination they allow Magic to make the most of each aspect and their synergy with the others.

For example, the use of white-boxes allows for tracing of functional relationships through the logic cones, which is used by sequential synthesis to detect sequential equivalences across the entire design. Another example: iterative local computations may not lead to substantial improvements if they are not supplemented by a global-view aspect of the computation, such as a global hash-table of shared logic structures, and the use of (global) delay-optimal technology mapping. Another goal of the paper is to list application packages used in Magic and explain how they helped produce good experimental results.

In summary, the contributions of this paper are three-fold:

- provide an example of technology transfer from academia to industry,
- describe Magic, an integration of ABC into an industrial tool, and
- outline the most useful optimizations performed by Magic while tracing their synergies.

The rest of this paper is organized as follows. Section 2 describes the background. Section 3 describes the optimization flow implemented in Magic. Section 4 outlines the industrial design environment where Magic was applied. Section 5 concludes the paper and outlines future work.

2. BACKGROUND

A *Boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. The terms Boolean network, netlist, and circuit are used interchangeably in this paper. If the network is sequential, the memory elements are assumed to be D-flip-flops with initial states.

A node *n* has zero or more *fanins,* i.e. nodes that are driving *n*, and zero or more *fanouts*, i.e. nodes driven by *n*. The *primary inputs* (PIs) are nodes without fanins in the current network. The *primary outputs* (POs) are a subset of nodes of the network. A *fanin* (*fanout*) *cone* of node *n* is a subset of all nodes of the network, reachable through the fanin (fanout) edges of the node.

A *node* is a logic component having a propagation delay. An *edge*, also called *wire*, is the connection between two adjacent nodes. The delay of a path includes *logic delays* and *wire delays*. The *logic delay* occurs in a logic component, such as a LUT. The *wire delay* occurs on the edges. In modern FPGAs, the delay for each pin in a LUT is different, so a variable-pin-delay model is used in this paper. Wire delays are usually not known until placement and routing. To approximate wire delays, a fixed delay is added to the delay of all pins of the LUTs in the library.

3. OPTIMIZATION FLOW

This section describes typical optimizations applied by Magic.

3.1 Algorithms

The optimizations of ABC integrated into Magic where modified when needed to interface with the new design data-base. The algorithms are divided into three categories: (a) synthesis, (b) mapping, (c) verification. They are not discussed in detail here because they are presented in the past publications, listed below. These algorithms were selected because they constitute a practical subset of efficient and scalable features of ABC.

3.1.1 Synthesis

Scalable sequential synthesis [9] and retiming [12].
Combinational synthesis using AIG rewriting [6].
Combinational restructuring for delay optimization [10].

3.1.2 Mapping

Mapping with structural choices [3].
Mapping with global view and priority cuts [8].
Mapping to optimize application-specific metrics [4][5].

3.1.3 Verification

Fast sequential simulation [5].
Improved combinational equivalence checking [7].
Improved sequential equivalence checking [9][11].

3.2 Integration

This subsection describes the integration of components inside Magic, illustrated in Figure 1. The design database is shown as the circle in the figure. It is the center of Magic, storing the design and interfacing the packages.

The design entry into Magic is performed through a file or using programmable APIs.

Shown on the right of Figure 1, is sequential synthesis based on detecting, proving, and merging sequentially equivalent nodes. This transformation can be applied at the beginning of the flow, before combinational synthesis and mapping. Another optional transform is retiming to reduce the total number of logic levels in the AIG or in the mapped network. Reducing logic level correlates but does not always lead to an improvement in the clock frequency after place-and-route. The sequential transforms can be verified by sequential simulation and sequential equivalence checking, shown at the bottom of Figure 1.

Shown on the left, is the combinational synthesis flow, which includes AIG rewriting, computing structural choices, and LUT mapping. Computation of structural choices can be skipped if fast low-effort synthesis is desired. The result of mapping is returned to the design database or passed for delay optimization. Correctness of combinational synthesis and mapping can be verified using combinational equivalence checking.

Finally, the box in the bottom right corner of Figure 1 stands for post-placement resynthesis, which includes restructuring and retiming with wire-delay information. Detailed description of these transformations is beyond the scope of the paper.

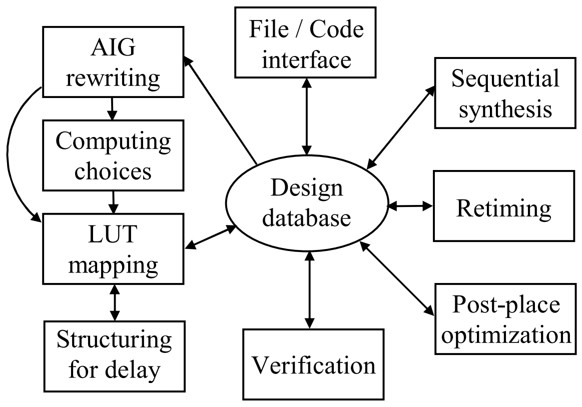

Figure 1. Interaction of application packages in Magic.

4. EXPERIMENTAL ENVIRONMENT

Magic enhances the Raptor Development Environment (RDE) from Abound Logic, to perform sequential and combinational synthesis, LUT-based technology mapping, and verification.

For completeness, we summarize the features of RDE. It is a complete FPGA design flow targeting Raptor devices. A detailed description can be found in [1]. Highlights relevant to this integration are summarized below.

The RDE provides a complete RTL-to-bitstream design flow, including synthesis, Magic-based netlist optimization and mapping to Raptor architecture, timing-driven placement and routing, bitstream generation, as well as a set of tools for design debugging, IP generation, and package planning.

The flow is able to handle half-million LUT class designs in under two hours in a completely automated approach, where the only required user inputs are source VHDL, Verilog, SystemVerilog RTL files.

RDE includes Synopsys Primetime-compatible static timing analysis engine and accepts standard SDC timing constraints, providing support for input and output delays, clock and generated clock definitions, false and multi-cycle paths, as well as netlist exploration and manipulation commands.

Unlike many other FPGA or ASIC design flows, RDE provides fine-grained control over placement and routing and includes incremental engines capable of re-using previous results as is and place and route only the differences.

This last feature is particularly needed in post-placement and post-routing netlist reoptmizations. In that use model, a design is first placed and routed normally. Using either exact (after routing) or estimated (after placement) delays, an optimization tool such as Magic can then further optimize the netlist to improve delays on critical paths. The new netlist is then sent back for place and route. Since RDE supports incremental placement and routing, it only needs to process the differences between the two netlists. The primary benefit of that approach is that it ensures convergence of the flow. If a full placement of the new netlist has to be performed, it often results in completely different paths being critical and the iteration doesn't improve delays. The secondary benefit of the incremental approach is that is significantly reduces the runtime.

RDE incremental flows were originally developed to address ECO requirements, when designers find last minute functionality problems. Correcting these problems through RTL changes and rerunning the whole flow not only costs time, but often makes significant perturbation to an already optimized design. Instead, it is often easier to make a small localized change in a netlist and have new design available in minutes instead of hours or days.

A unique feature of RDE, compared to other FPGA tool flows is that it allows to retain not only the existing placement, but also existing routing information.

Real-life designs are rarely all done from scratch. Typically, existing IP blocks or blocks from previous designs are reused. In typical FPGA designs flows, such re-use can occur only at the RTL level. Raptor Development Environment supports design reuse at the netlist, placement or even routing level. For example, a high-speed, timing critical logic may be heavily optimized, placed adjacent to I/O pads and routed. When such design is verified, it may be then copy-pasted over the chip or to other designs, similar to GDS instantiation in an ASIC design.

In addition to the top-down, synthesis and place and route design flow, RDE also provides robust design debugging capabilities. This includes ability to automatically insert real-time observability into the source RTL, whereas pre-defined trigger events cause signals being stored in on-chip memory resources as well as offline (i.e. when clocks are stopped) ability to read and write all chip registers and memories.

5. EXPERIMENTAL RESULTS

The experiments in RDE were performed by adding a call to Magic between design entry and global placement. No other part of RDE was changed. The resulting designs were verified using sequential simulation and formal verification in Magic.

The experiments were performed on 20 industrial designs ranging in size from 175K to 648K LUT4. The results are shown in Table 5.1, which contains two experimental runs:

- Section “Reference” stands for the typical RDE flow.
- Section “Magic” stands for the new RDE flow with Magic.

Columns “PI” and “PO” in Table 5.1 shows the number of primary inputs and primary outputs, respectively. Columns “LUT” and “FF” shows the LUT count and the register count, respectively. Column “Lev” shows the maximum logic level between registers. Column “fMAX” shows the maximum frequency, computed using static timing analysis after place-and-route. Column “Time” shows the total runtime without initial synthesis. Row “Geomean” shows the geometric mean for each column. Row “Ratio” shows the ratio of values in sections “Magic” vs. “Baseline” for each metric.

The experiments can be summarized as follows:

- fMAX is improved by 11.8%
- FF-to-FF level is reduced by 22.3%
- LUT count is reduced by 12.7%
- Register count is reduced by 9.4%
- The total runtime is reduced by 3.1%

It was observed that Magic optimization leads to better results in terms of all metrics, including fMAX, LUT count, and register count. It is also interesting that using Magic as part of the design flow reduces the total runtime of the flow.

To understand the reason for the runtime reduction, the runtime distribution and peak memory usage were analyzed. Although not included in Table 5.1, it was found that the runtime and memory used by the routing tool was reduced by 54% and 8%, respectively, when Magic was used. This is probably the reason that the total runtime of the design flow was reduced, even though the non-negligible runtime of Magic is included in the total. Routing probably became faster because the LUT count was reduced significantly, freeing routing resources in congested regions and simplifying the routing task.

6. CONCLUSIONS

Magic is a system developed on top of ABC [2] and deployed in a commercial design flow from Abound Logic [1]. Magic features an all-new memory-efficient design database which integrates the most robust application packages in ABC to perform synthesis, FPGA mapping, and verification for large industrial designs. Experimental results produced with Magic show improvements in all metrics, including area, delay, memory, and even runtime of the complete design flow.

Future work will include:

- Continuing to improve the underlying application packages in ABC, such as sequential synthesis, mapping with structural choices, and formal verification.
- Exploring optimization that targets tighter integration of logic synthesis and physical design, for example, integrated retiming and logic restructuring to reduce delay.
- Extending the optimization flow in Magic to work on standard cell designs and standard cell libraries represented using Verilog and Synopsys Liberty format, respectively.

7. REFERENCES

[1] Abound Logic. *Raptor Development Environment*. Product brief. PB003 (V1.0), Dec 2009. http://www.aboundlogic.com/

[2] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. Release 00127p. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[3] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, “Reducing structural bias in technology mapping”, *Proc. ICCAD '05*, pp. 519-526.

[4] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "WireMap: FGPA technology mapping for improved routability". *Proc. FPGA '08*, pp. 47-55.

[5] S. Jang, K. Chung, A. Mishchenko, and R. Brayton, "A power optimization toolbox for logic synthesis and mapping", *Proc. IWLS '09*, pp. 1-8.

[6] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", *Proc. DAC '06*, pp. 532-536.

[7] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking", *Proc. ICCAD '06*, pp. 836-843.

[8] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, “Combinational and sequential mapping with priority cuts”, *Proc. ICCAD ’07*, pp. 354-361.

[9] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis", *Proc. ICCAD'08*, pp. 234-241.

[10] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices", *Proc. FPGA'10*, pp. 181-184.

[11] H. Mony, J. Baumgartner, A. Mishchenko, and R. Brayton, "Speculative reduction-based scalable redundancy identification", *Proc. DATE'09*, pp. 1674-1679.

[12] S. Ray, A. Mishchenko, R. K. Brayton, S. Jang, and T. Daniel. “Minimum-perturbation retiming for delay optimization”. *Proc. IWLS’10*.

Table 5.1. Experimental evaluation of Magic against the reference flow on industrial circuits after place-and-route.

Circuits	Profile		Reference					Magic				
	PI	PO	LUT	FF	Lev	fMAX	Time	LUT	FF	Lev	fMAX	Time
C1	736	369	174972	113157	12	128.53	1.05	173561	100398	10	133.87	0.70
C2	150	67	187037	112991	18	91.32	0.53	161303	93930	16	95.69	0.67
C3	4	80	199097	53954	27	68.49	0.69	137126	36190	20	75.59	0.77
C4	517	253	206725	132416	11	105.37	1.31	197029	114745	8	129.20	0.67
C5	4	280	212124	64120	26	68.82	0.65	152799	49513	19	77.70	0.74
C6	803	258	255415	166644	11	113.25	2.08	255026	148445	8	123.00	1.00
C7	24	10	296152	133704	17	89.93	0.72	246908	114002	14	120.48	0.90
C8	124	58	323818	86712	32	40.68	1.99	346516	86662	25	47.08	1.94
C9	268	132	413017	195150	18	81.50	1.40	375481	174306	15	79.81	1.61
C10	205	94	439963	134139	20	63.17	3.55	445950	133575	15	69.06	2.64
C11	148	456	455429	160450	96	27.53	2.23	398428	149126	56	33.11	1.90
C12	4	3	455630	20277	6	66.67	0.78	152414	19446	6	100.40	0.41
C13	4	240	470436	230811	28	53.59	3.30	462010	225676	18	57.34	6.18
C14	218	69	522988	311436	17	68.78	1.83	448426	257996	15	69.40	2.19
C15	377	183	575355	351911	10	136.05	2.59	575672	349715	8	136.99	2.95
C16	73	33	599413	216051	4	202.02	1.07	599413	216051	4	209.21	1.79
C17	136	66	618377	259844	56	47.66	2.75	562367	243084	34	53.53	2.61
C18	136	66	621875	249327	27	45.68	4.60	606135	247825	27	52.58	4.03
C19	146	391	630918	275871	55	46.36	2.50	572834	259336	36	50.76	2.51
C20	135	32	648849	353940	7	127.71	2.45	645501	353616	5	136.43	2.91
Geomean			377883	150015	18.54	74.768	1.591	329751	135972	14.40	83.572	1.541
Ratio			1	1	1	1	1	**0.873**	**0.906**	**0.777**	**1.118**	**0.969**