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

Alan Mishchenko  Niklas Een  Robert Brayton  Stephen Jang  Maciej Ciesielski  Thomas Daniel

Department of EECS  LogicMill Technology  Abound Logic
University of California, Berkeley  San Jose, CA / Amherst, MA  Santa Clara, CA
{alanmi, een, brayton}@eecs.berkeley.edu  {sjang, mciesielski}@logic-mill.com  tdaniel@aboundlogic.com

ABSTRACT

This paper presents an example of 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 for 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 IMAX 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 as well as by making substantial contributions of their own.

The "technology transfer" between academic 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 in a commercial tool. Another instance is a graduate student interns at a company and later joins an employee, bringing along knowledge gained in graduate school. 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, in which a public domain tool, ABC, is used. 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 useful, how were these parts integrated into an existing flow, what QoR, memory, runtime improvement were achieved on large industrial designs, etc. The present paper outlines the result of integrating ABC into a particular commercial flow and shows the results produced.

We named the result of this particular 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), some of which are experimental, incomplete, or have known bugs, while Magic integrated and extended only those packages that are needed to create a robust optimization flow. Magic features an all-new design database developed within ABC to meet the requirements of this integration. It was developed from scratch, based on experience with ABC applied 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 which is greater than the sum of its parts. The parts in this case consisted of:
- 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 the 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 hashable of shared logic structures, and the use of (global) delay-optimal technology mapping. Another goal of the paper is to list the application packages used in Magic to achieve the synergism represented by the 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 and trace 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.

Fanin 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 pin-to-pin 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.

3.1.1 Synthesis

Scalable sequential synthesis [9] and retiming [12].

Combination synthesis using AIG rewriting [6].

Combination 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, shown 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 on the left of Figure 1, 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 on to restructuring for delay optimization. After combinational synthesis, the design 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 this 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 reoptimizations. 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 it 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 tools 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 R&D 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. 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 “GeoCount” 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 in 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: area, delay, memory, and even runtime of the complete design flow.

Future work will include:

- Continuing the improvement of 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


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

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Profile</th>
<th>Reference</th>
<th>Magic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PI</td>
<td>PO</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>736</td>
<td>369</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>150</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>4</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>517</td>
<td>253</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>4</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>803</td>
<td>258</td>
<td></td>
</tr>
<tr>
<td>C7</td>
<td>24</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>124</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>C9</td>
<td>268</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>C10</td>
<td>205</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>C11</td>
<td>148</td>
<td>456</td>
<td></td>
</tr>
<tr>
<td>C12</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C13</td>
<td>4</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>C14</td>
<td>218</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>C15</td>
<td>377</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>C16</td>
<td>73</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>C17</td>
<td>136</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>C18</td>
<td>136</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>C19</td>
<td>146</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td>C20</td>
<td>135</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Geomean</td>
<td>377883</td>
<td>150015</td>
<td></td>
</tr>
<tr>
<td>Ratio</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>