

Logic Equivalence Checking Has Arrived For FPGA Developers

William McDonald and Janny Liao

Semiconductor Test Division

Teradyne, Inc.

Boston, MA 02118-2238

Email: {william_mcdonald, janny_liao}@notes.teradyne.com

Abstract—Logic Equivalence Checking (LEC) long ago became a standard tool for developing Custom and Application Specific Integrated Circuits (ASICs). For ASICs, LEC tools have proven to be the best technology to exhaustively check for errors introduced by logic synthesis and physical implementation tools, and by netlist Engineering Change Order (ECO) edits. In contrast, due to the lack of viable LEC tools for Field Programmable Gate Arrays (FPGAs), FPGA development has remained in the dark ages, dependent on a combination of Register Transfer Level (RTL) Linting, Gate Level simulations and costly Lab Validation efforts to expose problems that could be introduced by the back-end physical implementation process. RTL Linting, Gate Level simulations and Lab Validation efforts are not exhaustive, leaving the potential for bugs to disrupt later stages of product development, or worse, to be shipped to the customer.

In addition, a recent trend has been for FPGA developers to reduce or skip gate level simulation for the same reasons that developers did for ASICs. Many product developers accept this tradeoff today because FPGAs offer re-programmability without suffering Non-Recurring Engineering (NRE) costs or fabrication delays. However, this still leaves their design verification blind to the back-end process and dependent on the reliability and maturity of the back-end tools, or on Lab Validation.

Teradyne has been working since 2003 with synthesis and verification EDA tool vendors and our key FPGA providers to deploy an RTL to post-synthesis LEC process for FPGAs. The results proved to be viable and valuable for production use in our FPGA development process.

This paper documents why LEC is as important for FPGAs as it is for ASICs, presents how it can be deployed in an existing development process, and provides real-life examples of how LEC has improved our confidence in our production FPGAs.

I. INTRODUCTION

Logic Equivalence Checking (LEC) is a formal verification tool that compares a reference design against a

derived design to prove equivalence or to report differences. LEC does not require test patterns. Instead, LEC uses Boolean arithmetic techniques to prove equivalence between two design netlists. Although LEC uses sophisticated formal algorithms to identify, map, and compare nodes in the netlists, the complexity is hidden from the user. In addition, the formal algorithm techniques are fast compared with simulation based tools. Today's tools, originally developed for Application Specific Integrated Circuits (ASICs), are faster and easier than ever to use with Field Programmable Gate Arrays (FPGAs).

This paper aims to increase momentum in this technology area by inspiring more companies to evaluate LEC in their own projects and by providing pointers through examples for others to use.

II. THE NEED FOR LEC TAKES SHAPE

The roots of today's LEC tools grew out of the demands of aggressive ASIC development schedules that required quick validation at each milestone and each large or incremental design change.

In the primitive days of ASIC development, designs were captured with schematics and verified by running compute intensive gate level simulations with labor intensive, hand-written tests. Around the late 1980's, Register Transfer Level (RTL) coding and synthesis revolutionized the industry by improving functional simulation speeds by an order of magnitude. Continual improvement in the speed of simulators and computers also contributed. Even with these significant advancements, the ASIC development process still required full gate level simulation at each incremental milestone in the development cycle. Gate Level simulations validated the synthesis results, and analyzed pre- and post-route timing. Gate level simulations also checked that the insertion of test features and late Engineering Change Order (ECO) changes did not alter the design intent. The ASIC design process remained simulation intensive and restricted by simulation throughput.

In addition, a big flaw with RTL and gate level simulation is the lack of verification coverage due to the difficulty of writing tests that exercise all the functional paths in the design. This problem increases exponentially with design size. As a result, the quality of verification is limited by the availability and quality of functional tests.

Over time, ASIC design teams became exasperated as they dealt with competitive schedule pressures and increasing gate counts. They coped with the problem by running only a subset of carefully chosen RTL tests in gate level simulation to reduce runtime. Developers were able to use static timing analysis, the older cousin of LEC, to replace the more limited timing checks that gate level simulation provided. However, the reduced set of tests run in gate level simulation still exposed projects to discrepancies in synthesis tool releases, errors from the insertion of test functions, and the ever tempting hand edits to meet timing or repair a bug.

LEC tools were introduced by companies like Chrysalis Symbolic Design, Inc. and Lucent Technologies' Bell Labs Design Automation group around 1995, but were slow to gain market acceptance. The technology, originally developed by mathematics research departments of universities as far back as the 1960's [1], suffered from a steep learning curve because the early tools had primitive mapping algorithms requiring a more complicated setup process. In addition, its radically orthogonal approach to the traditional design flow hampered adoption by the industry.

Today, LEC tools have since proven to be much faster than gate level simulations. Also, LEC verification coverage is inherently exhaustive because of the mathematical methods it uses. In the beginning, these major features were not trusted the way gate level simulations were.

Although taboo in the mid 1990's, by the late 1990s it became commonplace to skip gate level simulation as synthesis tools matured and gate counts increased [2]. Some teams relied on LEC to mitigate the risks, some tried hardware emulation in the same spirit of gate level simulation [3], and other teams just accepted the risks. Ultimately, with Non-Recurring Engineering (NRE) costs rising, ASIC and IC foundries required that LEC tools be run at each milestone, which cemented broad use of LEC for ASIC development.

Meanwhile, FPGA companies were slowly increasing design capacity and gaining market share by providing a lower entry point to the industry with no NRE cost. To lower the entry cost further, the major FPGA companies provided many of the required Electronic Design Automation (EDA) tools for free. As a result, the EDA industry largely ignored the FPGA market.

LEC tools were also not required for the majority of early FPGAs. Many FPGAs were targeted for either ASIC prototyping or other low volume applications, requiring small verification efforts. The major attraction of FPGAs, the ability to easily iterate when problems were found late in the lab or in the field, also reduced the need for LEC.

This has changed. FPGAs are facing the same pattern of escalating design complexity and schedule pressure as ASICs. FPGA teams are relying on RTL simulation, static timing analysis, and prototype validation efforts in the lab in favor of the less effective method of gate level simulation. As design complexity and size continue to expand, the next logical step for FPGA developers is to adopt LEC into their design methodology.

III. THE BENEFITS OF LEC

Now is a turning point in FPGA history where we predict FPGA developers will begin taking advantage of LEC the same way that ASIC developers did. The most compelling benefits of LEC are:

- Less reliance on gate level simulation.
- Boosted confidence in new tool revisions for synthesis and place & route.
- Watch-dog for poor RTL coding areas in the design.
- Nearly exhaustive proof of equivalence without writing test patterns.
- Decreased risk of missing a bug inserted by the back-end process.
- Identification of unreachable logic, leading to power and area reduction.

A. *Who Should Use LEC*

- Companies that can ill afford the high cost of lab iterations or delivery of bad FPGAs to customers.
- Military, medical, and aeronautical designs, or projects demanding high quality and customer satisfaction, fit naturally with this list of key benefits.
- Large FPGA projects that need to mitigate risk from running only a subset of tests in post place & route gate level simulation.
- Projects that wish to keep watch over the back end process and coding quality even when no bugs are found.
- FPGA companies that develop Intellectual Property (IP), as their customers expect the IP to be well coded and bug-free.

B. *Teradyne's Transition to LEC*

Teradyne develops a large number of FPGAs and experienced the ills from not having LEC capabilities many times. For example, while Teradyne's Tools and Technology group was working with our EDA and FPGA vendors testing alpha releases of LEC applications with real FPGA designs, a designer in the company came across a test that failed in gate level simulation but didn't fail in RTL simulation. The traditional debug approach lagged on for

weeks. Changing synthesis switches for state machine remapping seemed to correct the bug, directing blame to the synthesis tool. However, further study of the RTL code revealed ambiguous and redundant coding. Blindness of the synthesis process led to mistrust and prolonged frustration until the true problem was discovered. This initiated our company's final step with LEC, to bring it into production use on a real project; the project manager made a request to implement LEC on all chips in the project. On this project alone, we are reaping the benefits on three chips and are continuing to work on a fourth from a second FPGA supplier. Examples of issues already caught by the LEC tools on this project and on others are presented later in the paper.

IV. DEPLOYING LEC INTO AN EXISTING FPGA FLOW

Fitting naturally to the existing FPGA design flow, the LEC tool applies formal verification to the following places:

- **RTL Source Code** – This is the raw, pre-synthesized design code. Our company uses Verilog. We have not tried this process with VHDL.
- **Post Synthesis Netlist** – The synthesis software generates this gate-level netlist in the Verilog format. It is separate from the netlist generated for the place & route software, which can be in a different format such as EDIF.
- **Post Place & Route Netlist** – The place & route software generates this gate-level netlist in the Verilog format, which closely resembles the design layout as it appears in the FPGA device.

The current verification flow supports equivalence checking in two passes: the first pass is between the RTL code and the Post Synthesis netlist, the second pass is

between the Post Synthesis netlist and the Post Place & Route netlist. Figure 1 illustrates this methodology with additional details. Observe that this two pass approach does not upset the original flow for synthesis and place & route prior to including LEC.

It may be possible to make the leap of directly comparing RTL source code against the place & route netlist, but using two smaller steps as shown in Figure 1 reveals which of the steps could be flawed and reduces the complexity of changes to be resolved for each one.

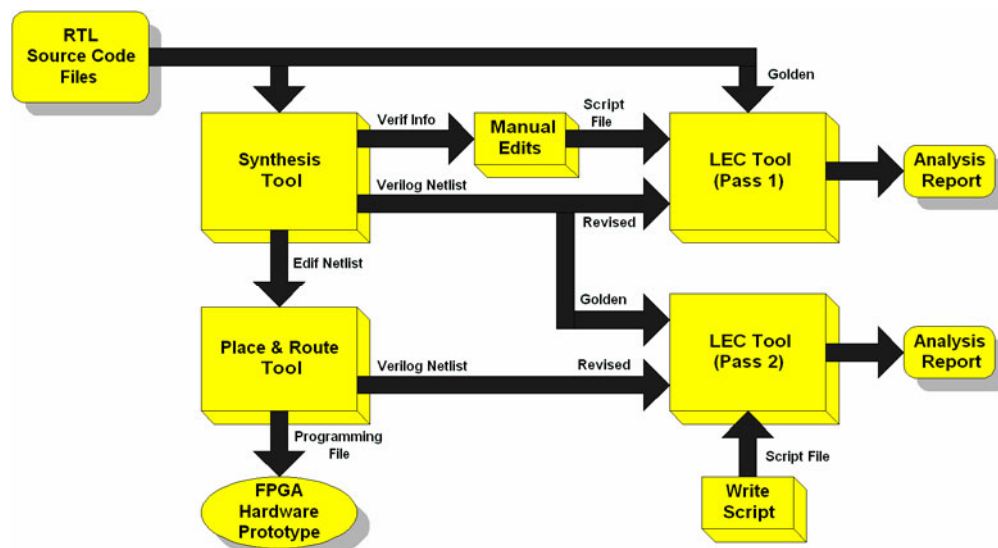
Along with the Verilog netlist, the synthesis tool also outputs files containing verification information about the optimizations done to the design. This information provides hints to the LEC tool about such things as signal and instance renaming, register merging and replication, state machine re-encoding, sequential optimization, etc. These optimizations are aggressive, and their incorporation into the LEC flow is critical to the automation of the mapping process. This points out that the LEC vendor must work cooperatively with the synthesis tool and FPGA vendors to ensure the right information is available for successful tool performance.

LEC tools perform proof of equivalence by using a three phase process: setup, mapping and comparing.

The source input, such as RTL Verilog code, is called the golden netlist. The transformed input, such as the synthesized design, is called the revised netlist. During the setup phase, the LEC tool loads both of these netlists.

In the mapping phase, LEC attempts to map known netlist differences using the synthesis hints along with additional inputs from the user. The tool builds a table of key points from primary inputs and outputs, internal flip flops and black box logic in the golden netlist.

Figure 1. Recommended LEC Verification Flow



In the comparison phase, the tool begins matching points in the revised netlist to points in the mapping table. Some points are easy to match because they have the same name or use between netlists, while others have been significantly renamed or combined within the hierarchy. With hints from the synthesis tool, use of renaming rules, and control of the hierarchy, these points all become matched. After matching, the comparison phase checks the cones of combinational logic connected to each pair of matched points to test whether they are logically equivalent.

V. MAKING LEC WORK

Although our LEC tool flow uses a specific synthesis engine and LEC application, we choose to avoid identifying them in an effort to present our material in a vendor-neutral format. Currently, there are several successful and popular FPGA tool flows, not to mention multiple FPGA suppliers. Without endorsing specific suppliers, here are some points to consider on selecting and setting up an LEC tool to fit into your existing design flow.

A. Choosing an LEC Tool

When choosing an LEC tool for FPGA work, it is essential to find one that works with your synthesis tool and supports your FPGA architecture. If your synthesis tool does not support any stable LEC tools on the market, then you will either need to change your synthesis tool, or explore a collaborative working arrangement with willing suppliers to help bring a new tool along, as Teradyne has. This can be challenging, rewarding, and educational for your company.

Once an LEC tool is chosen, we recommend evaluating it on a trial basis with one or more stable design snapshots until the evaluation is complete. This is also a good opportunity to assess the vendor's commitment to support. In the process of learning how to make the tool map the netlists correctly, you will have many opportunities to interact with them. LEC, being a new tool in the FPGA flow, will require time to become familiar with its capabilities. The tool suppliers can be an excellent source of help.

At this point, the new tool user will realize that it takes time and effort to get LEC to map correctly. The ASIC design community has complained before that LEC tools are difficult to use [3]. In fact, the new FPGA tools were derived from ASIC tools and must be learned through practice. Running the tool with a basic setup is easy, but discovering all details required to get it to map correctly requires experience with the tool, trial and error with the setup commands, and debug with the tool's user interface.

In the face of this, tool developers have made an effort to hide complexity by developing advanced mapping algorithms and passing information from the synthesis tool to the LEC run script. Although passing concealed information increases the risk that LEC will use the same assumptions as the synthesis tool, this approach is ideal for

the FPGA community, which has become comfortable with the highly automated and integrated nature of its synthesis and place & route tool suite. Also, synthesis algorithms for FPGAs have become so advanced in the last few years that passing along hints are required. A developer familiar with their RTL design can expect to spend about one to two weeks getting a new design to run through the LEC process.

B. Setting Up a Script to Run LEC

The first step in using the LEC flow is to set up the synthesis tool's run-time switches to generate a netlist in the Verilog format. We have been turning off re-pipelining and retiming optimizations in our synthesis. Turning these optimizations off may not be necessary in the future given the steady progress that LEC and synthesis tools are making, but we haven't strongly relied on these features anyway and this provides a simpler starting place.

The second step is to create a run script for the LEC tool that can be used reiteratively while the process is debugged. The synthesis tool we use has a unique interface to our LEC tool in that it generates a script intended to work directly with LEC. This script jumpstarts the process, getting results quickly. The script contains LEC commands which are hints from the synthesis step about the optimizations performed on the design. With this script, some of our designs have completed on the first try! This is a strong example of cooperation in the FPGA industry today, given that the tools were all developed at different companies.

The LEC run script has two sub steps. The first sub step reads in both netlists while applying some settings to them. These settings do such things as set up black boxes for Random Access Memories (RAMs) and multipliers, and flatten some of the hierarchy. The second sub step maps the two netlists, compares them, and reports the results. Figure 2 is a real script showing examples of some basic LEC commands.

C. Achieving Proof of Netlist Equivalence

The LEC tool reports both equivalent and nonequivalent nodes once the run-script finishes without setup errors. Inevitably, the first time through this process, LEC identifies points between the two netlists that do not map properly. Tables I and II show reformatted output from a real, albeit imperfect, LEC run.

These tables introduce the term *black box*, a common occurrence in LEC work for FPGAs. Black boxes identify FPGA logic that the LEC tool ignores. RAM, multiplier, and Phase Locked Loop (PLL) cores typically need to be black boxed. Other good examples include the generated FPGA IP cores that bypass the synthesis process and typically, but unfortunately, need to be ignored by LEC.

Table I states that LEC proved 5548 key points to be equivalent, but found problems with 466 other key points. Although these points in the golden and revised netlists

mapped with each other, the tool reported discrepancies between the cones of logic driving them.

Figure 2. Sample Script for a Synthesis to Place & Route LEC run.

```
// Set netlist parsing options. The "nottranslate" commands
// select the module names that are interpreted as black boxes.
//
set log file project1.log -replace
add nottranslate module RAMB* X_RAMB* -library -both
add nottranslate module \
project1_sr16x32 project1_sr29x16 project1_ram1kx18 \
project1_ram1kx25 project1_mul24x24 \
project1_mul10x8 -both
add nottranslate module project1_mul24x24_1 -revised
add nottranslate module project1_ram1kx25_1 -revised
//
// Read in golden and revised netlists.
//
read design -file longfin_top.vlc -verilog2k project1_top_bb.v \
project1_top.vm -verilog2k -golden -root project1_top
read design -file project1_top.vlc -verilog2k \
project1_top_par_cecn.v -verilog2k -revised -root project1_top
//
// Generate netlist parsing reports.
//
report messages
report black box
report design data
report floating signals
//
// Set mapping options.
//
set undriven signal 0 -revised
set flatten model -seq_constant
set mapping method -nobbox_name_match
add renaming rule r3 "\Q_r_e_g" "" -both
add renaming rule r4 "_Z[%d]\$" "[@1]" -type DFF \
-type DLAT -golden
add renaming rule r5 "_Z$" "" -type DFF -type DLAT -golden
//
// Run equivalence checker.
//
set sys mode lec
add compare point -all
compare
usage
exit
```

TABLE I. EQUIVALENCE ANALYSIS INFORMATION

	Primary Output	D Flip Flop	D Latch	Black Box	Total
Equivalent	51	5390	4	3	5548
Non-equivalent	0	463	0	3	466

TABLE II. NETLIST KEY POINT MAPPING ANALYSIS

Golden:	D Flip Flop	Black Box	Total
Unmapped:	10	0	10
Revised:			
Unmapped:	2317	30	2347
Unreachable:	474	0	474

Table II shows unmapped key points. Here, LEC tried to map 2317 D flip flops in the revised netlist to similar flip flops in the golden netlist, but could not find matches either by name or by function. There are also 30 black boxes that are not mapped. These problems are indicators of what went wrong. It is often easiest to start resolving these problems by working on the black box problems first, since these are usually indicating a fundamental problem with hierarchy.

In the run that produced these tables, an excessively flattened netlist caused the problems. RAM designs in the golden netlist contained multiple RAM cores stitched together and were black boxed as a whole unit. In the revised, flattened netlist, each RAM core was black boxed separately. This gave a larger number of black boxes in the revised netlist that had nothing to map to in the golden design, throwing off the counts. By working with the netlist generator in the FPGA place & route software, a hierarchical netlist was produced that gave the LEC tool a chance to black box the upper hierarchical module for this RAM design, leading to a realistic comparison of black boxes between the designs.

The LEC tool Teradyne uses has a Graphics User Interface (GUI) that provides side-by-side comparison of mapped points along with an indication of whether the points are equivalent. The GUI also offers a diagnosis window to show all signals influencing a key point along with an indication of which signals were likely to be the problem. From here it is easy to launch a schematic tool that shows the circuits for both the golden and revised designs. The GUI offers many clues for what to change in the setup process to make the tool run to completion. Table III summarizes four common symptoms and solutions that Teradyne observed across several FPGA designs.

VI. LESSONS LEARNED FROM REAL LIFE EXAMPLES

Over the last two years, during alpha and beta testing of the LEC flow, Teradyne ran about 10 FPGA designs from around the company through the LEC tool flow. These experiments contained design variations that often found bugs in the LEC tool, the place & route tool, or the synthesis information files that needed corrections from the tool developers. Over time, these problems decreased and the LEC runs succeeded. Through this experience, Teradyne learned from the examples that follow.

Surprisingly, LEC is an effective tool at screening designs for poor RTL coding style. Offering a second opinion to synthesis, it parses the code to decide what each statement means. One chip we tried gated an input signal with the asynchronous reset of an “always” block. The code implemented this gating inside an “if” block traditionally reserved for just one reset input. The synthesis tool responded by placing the gate before the asynchronous reset pin of the register while the LEC tool left the global reset signal on the register’s asynchronous reset pin and assumed a synchronous reset with the new input. There are more descriptive ways to code the RTL to consistently obtain

either outcome, so this example can be considered poor coding style.

Some designers are still misusing the synthesis directives, “//parallel_case” and “//full_case.” LEC exemplifies that these directives are not needed in modern RTL design work. If using these directives causes a change in the synthesis outcome, the synthesized netlist will not match the broader function of the RTL source code. This could be observed as a difference between gate level and RTL simulations, but it would require having a test to cover the conflicting condition. LEC catches this error automatically when there is a logical impact from the directives without depending on a test. Seeing LEC point out this problem can spark a useful debate on your team about the use of synthesis directives.

Another design had constructed a small first-in first-out (FIFO) memory from an array of registers. The designer routinely followed a team rule to connect an asynchronous reset to each register in the array. Our Verilog LINT (syntax checker) tool even enforced this rule. The synthesis tool inferred a RAM cell for this design that did not have an equivalent asynchronous reset. The synthesized design was not equivalent and it could have led to a functional problem. We immediately tried a later release of the synthesis tool, which did not infer the RAM, choosing instead to build the FIFO from an array of flip flops. This is probably the best response for the synthesis tool. If we want to infer the RAM, we must write code that is exactly compatible with the RAM IO interface. We can also be more direct and specify the RAM cell in the code, permitting us to drop the asynchronous reset without violating our register reset LINT rule.

On one chip, an LEC run indicated about 4 unconnected nets in the post place & route Verilog netlist. These went away in the next release of the FPGA place & route tool. We suspect these unconnected inputs only existed in the Verilog netlist and not in the programming file for the device. Similarly, a second chip indicated unconnected inputs to Double Data Rate PAD cells. For this design, the FPGA supplier quickly gave us a library update to solve this problem. For this example, the supplier confirmed that the unconnected inputs were not making their way into the programmed device.

Although these experiences have proven to Teradyne that the benefits of LEC are real, the tool flow still leaves two situations uncovered:

1. The final FPGA programming file is generated separately and thus deviates from the Verilog netlists used for LEC. The final comparison from the synthesis Verilog netlist to the post place & route Verilog netlist is, unfortunately, still slightly blind to the output file used to program the device. It is also possible to find errors in the LEC path that do not exist in the programmed device.
2. LEC does not verify or evaluate the behavioral models from the FPGA supplier for RAMs and multipliers that become black boxed. This is not really a problem LEC should be expected to solve, but many new LEC users fall into the trap of assuming LEC automatically covers these models. Although a verification problem area for the engineer, these models do not actually proceed into the synthesis flow. Teradyne has discovered blocking/non-blocking bugs in these models a number of times during lab debug and encourages FPGA suppliers to improve the IP they deliver.

VII. CONCLUSIONS

The methodology for FPGA development is following the historical progression of ASIC development as designs swell in size and complexity, leading to the adoption of LEC as the next logical step in the evolutionary process. In particular, schedule pressure and design complexity are driving the methodology to rely on static timing analysis, flexible and quick prototyping, and LEC in favor of gate level simulation, which can no longer be completed in time or with enough coverage to provide the trust that it once did.

The EDA industry has recently transitioned some popular ASIC LEC tools into a production FPGA flow. Teradyne participated in this transition by engaging in a cooperative relationship with EDA and FPGA suppliers. We look forward to further engagements between EDA and FPGA suppliers and encourage the industry to support EDA suppliers that invest in FPGA solutions.

TABLE III. SYMPTOMS AND SOLUTIONS TERADYNE OBSERVED THAT RESOLVED LEC CONVERGENCE

	Symptom	Likely Solution
1	LEC does not apply black box rules the same way on both netlists.	The hierarchy in one netlist does not show the upper hierarchical module for a RAM or a multiplier. Regenerate the revised netlist with hierarchy turned on.
2	Signals in the revised netlist do not map back to signals in the golden netlist.	Use the diagnosis GUI to diagnose signal name or instance name changes, then apply renaming rules and repeat.
3	One or two of the RAM or multiplier blocks are not black boxed properly, but hierarchy seems to be correct.	Find the block names that need to be black boxed with the diagnosis GUI and identify them by name in the “add notranslate” commands.
4	Use of the diagnosis GUI leads to suspicion that register instance merges or replications are not getting resolved in LEC mapping.	The flattening and mapping options have not been set up right. Review manual for use of these options. Request assistance from LEC vendor for recommended settings.

In using the new tool, Teradyne gained more confidence and improved design quality through LECs most compelling benefits: less reliance on gate level simulation, screening for poor coding style, watch-dog for tool release discrepancies, and monitoring of unreachable logic. Teradyne hopes the examples from this paper will inspire more companies to use this proven technology.

ACKNOWLEDGMENT

The authors thank David Bacon for his extensive editing feedback, Jason Mark, Steve Rideout, and Earl Shaw for their proofreading, and Mike Pearlman, Dominic Viens and Dominic Wong for their involvement in this LEC effort.

REFERENCES

- [1] E. M. Clark and R. P. Kurshan Eds, "Volume three: computer-aided verification '90," DIMACS series in discrete mathematics and theoretical computer science, American Mathematical Society, 1990, Preface, <http://dimacs.rutgers.edu/Volumes/Vol03.html>.
- [2] Gale Morrison, "Formal verification adoption quickens," Electronic News, 4/17/2000, www.reed-electronics.com/electronicnews/article/CA49273.html. The article pinpoints when LEC gained acceptance for ASICs.
- [3] Ralph Escherich, "Philips says Verplex was best buy," Electronic News, 10/30/2000, www.reed-electronics.com/electronicnews/article/CA50680.html. The article discusses the use of emulation with the company's ICs in combination with a popular LEC tool from the late 1990's.