Best known methods for using Cadence Conformal LEC at Intel

Erik Seligman (Erik.Seligman@intel.com) and Itai Yarom (Itai.Yarom@intel.com)

Intel Corporation

Abstract. In this paper we will explore how to use the Cadence Conformal LEC tool capabilities to verify different types of designs, based on the experiences of various projects at Intel. In particular, we will focus on the Conformal Ultra capability for verifying complex data-path synthesis and layout.. As an example, we will present a 1 Gigabit Ethernet chip with around 10 million standard cells. At first glance we thought that we would need to use the divide and conquer technique, and split the big design into smaller blocks in order to complete the verification. To our surprise, using the Conformal Ultra together with the effort "complete" we managed to verify the majority of the design in one flat run. In this presentation we will also discuss how to use the different Cadence Conformal LEC capabilities and what benefits they provide, describing numerous best-known-methods developed at Intel. The following learnings are based on using the tool on different types of designs. This kind of information can provide a significant saving of time and increase in user productivity. We will wrap up by presenting additional ideas for accelerating the design flow by further extensions to FEV tools like Cadence Conformal LEC.

1.0 Introduction

Formal Equivalence Verification (FEV) has become an essential part of the hardware design flow. FEV is being used in many different parts of the flow, from RTL to layout, supporting various representations, including RTL, netlist, and Spice. In this paper will present some of our experiences using Cadence Conformal LEC to perform FEV at Intel. We will present some of the key learnings for using LEC and will present their application on a high-performance, dual-port gigabit Ethernet controller. We will address questions like: How do we verify big and complex designs? How do we handle “abort points”, which are points that the FEV tool did not manage to verify? How can we handle different implementation structures, like those created when using retiming? And how can FEV help with ECO (Electronic Change Order) implementations?

Today FEV looks like a natural part of the flow. However, not too long ago we used simulation instead of FEV. What does the future hold for the FEV technology? Will FEV become the engine behind different technologies like ECO automation, power reduction and clock-domain crossing verification? Will FEV and formal property verification (FPV) merge one into each other? Although we can only guess

what the future holds for us, we will try to present some directions we think would be beneficial for future tool development.

The paper is organized as follows: first, we describe our general FEV methodology, present tips and tricks, and describe their application on our Gigabit Ethernet design. Then we discuss some future directions of FEV, where we would like to see increased support from later LEC releases.

2.0 Cadence Conformal LEC – Tips & Tricks

On ASIC design teams at Intel, LEC is our primary tool for FEV. Some of the major uses of FEV in Intel design flows for synthesized designs are:

- **Front-End (RTL-Synthesis) FEV:** This verifies that the synthesized cluster-level netlists, synthesized with another vendor's tool, logically match the original RTL. This can be done at fullchip level on smaller designs such as the Gigabit Ethernet chip described below. This is usually the most challenging of our FEV runs, and will be the emphasis of this paper.
- **Back-End (Synthesis-Postlayout) FEV:** This verifies that the synthesized cluster-level netlists match the final postlayout netlists, after all timing optimizations. Thus Front-End FEV + Back-End FEV ensures that the final postlayout netlist fully matches the RTL.
- **ECO FEV:** During the time-critical pre-tapeout period of a project, we have completed synthesis and are in a stable state, and want to quickly make late edits to both the RTL and the postlayout netlist if ECOs (Engineering Change Orders) are approved. Note that this run is an optimization—eventually we synthesize a "fake" intermediate netlist using the modified rtl, and complete both Front-End and Back-End FEV as before. That way, we can tolerate some Abort points in this quick ECO FEV run, and get a rapid verification that our ECOs are most likely valid.
- **Defeature FEV:** In some cases, especially as part of a late ECO, we will add a "chicken bit" to turn off the change just in case we overlooked some detail and it hurts the viability of the design. In these cases, we will do a special RTL-RTL FEV run with the new feature turned off, to verify that our RTL behaves the same as it did before.
- **Fullchip FEV:** In designs where we synthesized at cluster level, this is a run where we take all the synthesized logic of the entire chip, along with the top-level connectivity logic, and compare it to the final fullchip netlist, with only the custom pad/memory circuits blackboxed. This run is technically redundant if both Front-End FEV and Back-End FEV are complete, but is a good final end-to-end check that we really are taping out what the logic designers think they designed. A small number of Abort points in this run (typically <0.1%) is acceptable due to this redundancy.

2.1 An Example Design

The design that we use as our test case is a high-performance, dual-port gigabit Ethernet controller for servers and embedded system design [7]. The chip supports the PCI Express protocol and the Intel® Advance Management Technology (Intel® AMT). Figure 1 presents the architecture. The design has more than 1 million standard cells and around 200K flip-flops. In the runs that we present in this paper we used a model when part of the design is black-box, in order to focus on the abort-points. For those runs the number of flip-flops is around 36K.

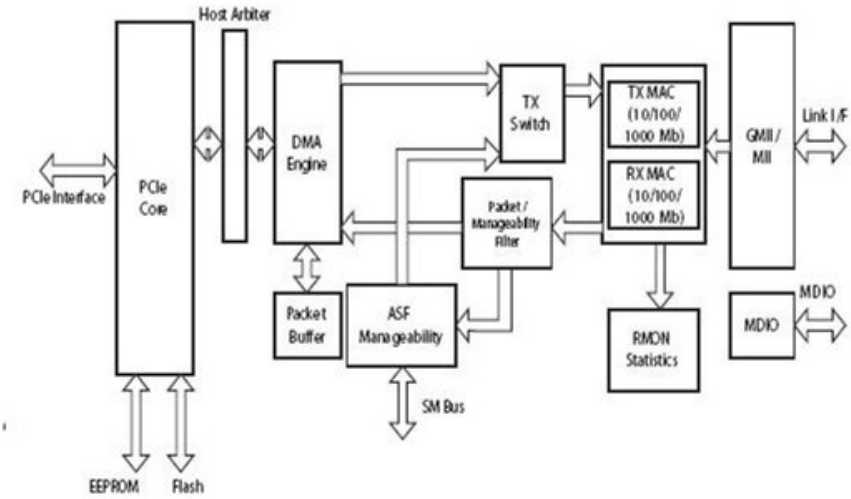

Fig. 1. The design block diagram

2.2 Dealing with Abort Points

A common issue that all LEC users must face is Abort points, cases where the tool is unable to complete verification of some points due to lack of computing power. Since the problem LEC is solving is exponential in principle, it is not surprising that such cases arise. But since Front End FEV + Back End FEV must form a complete RTL=tapeout equivalence chain, we must resolve all such points in order to confidently complete our designs. We have found a number of useful techniques to resolve these:

- **Compare Effort:** A simple way to address Abort points is to increase the compare effort ("set compare effort") in the LEC run. A subtle issue that users need to be aware of is that often it is better to start on the lowest compare effort, compare all the points that can be handled, and only then

increase it. Attempting to verify the whole design at maximum compare effort can cause a blowup in the runtime, as more effort is spent on essentially easy points. .

- **Library Cells**: If some standard cells are instantiated directly in parts of the RTL, you should run LEC using a single representation for the cell libraries in both the RTL and netlists. The cell libraries can be verified in a separate FEV run; there is no need to do library verification within your higher-level cluster run, which can increase its complexity. Also, experiment with different representations of the libraries (**.v, .lib**)—different ones may be more efficient for a given library depending on what cells are used and their local logic implementation.
- **Use Hierarchy to Divide and Conquer:** LEC's traditional "write hierarchical dofile" command causes LEC to try to split the design on hierarchy boundaries, dividing it into smaller subproblems. This can work well for designs that are synthesized with hierarchy intact and without boundary optimizations. However, in recent projects we have been making increasing use of advanced synthesis tool features, and are encountering many cases where this hierarchical breakdown results in false negatives due to changes near boundaries. Sometimes in such cases we can synthesize a low-effort hierarchical netlist as an intermediate verification point. But we have also seen some examples of smaller but logically complex designs where this hierarchical breakdown cannot help.
- **Partitioning:** Partitioning, or case splitting, means taking some small set of instances or inputs, and assigning constants to them. By verifying for all possible sets of constant values, we get full verification of the design. LEC allows this using "write partition dofile", or the new, cleaner "run partition_compare" command. Obviously, it is important to carefully choose the partition points; control inputs that mask out large portions of logic are ideal.
- **Analyze Datapath:** This new Conformal Ultra command helps the tool to remodel the design in a more FEV-friendly way, and can result in dramatic improvement. For example, one complex parity-type block on the recent Blackford project caused LEC (and competing FEV tools) to run for over a day and eventually Abort on most points, but after an "analyze datapath -merge", we were able to verify completely within five minutes. This feature was critical in enabling flat verification of the Gigabit Ethernet chip.
- **Parallel Compare:** The new "compare –parallel" command causes LEC to split the problem across multiple machines. Intuitively, we did not expect this to help with Abort points—but to our surprise, we have actually found running in parallel to significantly reduce Aborts on our designs. We suspect that this is because LEC is able to make more efficient use of the memory on remote machines by sending less data than is needed for the full model.

2.3 Ensuring Design Correctness

As we all know, our synthesis is not always perfect, and there are even occasions where LEC incorrectly reports non-equivalence. Of course, there are standard methods for debugging this: check that key point mappings are complete, bring up the schematic view, etc. But there are cases where these methods do not work, or the logic cones are simply too big to practically debug. Some of the other common issues we have found when debugging mismatches include:

- **Be careful about unsupported / unusual language constructs.** For example, one tricky issue that recently caused us headaches involved a Verilog expression "{a, {2-`WIDTH{1'b0}}, b}", when `WIDTH = 2. This 0-length replication is an unspecified corner case in the Verilog 2001 standard: LEC interpreted the expression as {a,b}, while our synthesis tool considered it {a,0,b}. This also underscores the importance of using good linting tools to check RTL, and watching synthesis logs for warnings, before attempting FEV and backend flows.
- **Be aware of model flattening options.** LEC provides remodeling through "set flatten model", for example allowing flops with clocks tied to 0 to be treated as latches. In most cases, if an inappropriate set of flattening options has been chosen, LEC will have mapping issues. However, we have found some cases where this causes non-equivalence: for example, in one model, we had a flop whose input fed back to itself in RTL, and it became a flop with a 0 clock input in the netlist. Once we turned on the flattening options "dff_to_dlat_feedback" and "dff_to_dlat_zero", we were able to correctly verify the models as equivalent.
- **Be aware of "don't-care" cases that must be excluded for clean verification.** For example, scan enable pins must often be tied to 0 to compare a non-scan-inserted RTL and a scan-inserted netlist, or a "$constraint" construct may be needed to tell LEC that a particular state encoding in a complex state machine is invalid.

The most dangerous cases overall are situations when LEC reports a design as clean, but it is actually broken! At least one major Intel project had dead A0 silicon due to an oversight where a last-minute ECO was not validated by any other method besides FEV, and incorrect constraints made the FEV result wrong. So we try to do a careful audit before tapeout, checking for major potential issues:

- **Are all constraints valid, and checked through assertion-based verification if possible?** As mentioned above, sometimes constraints are needed so LEC knows which RTL cases are truly don't-cares. But an invalid constraint can make the LEC run nonsensical. Constraints, $constraint verilog constructs, and similar LEC directives that create don't-care cases in logic must be carefully reviewed, and should be run in simulation and formal property verification whenever possible to check their accuracy.
- **Are 'unreachable' points handled correctly?** LEC excludes unreachable points, points which cannot affect any output, at several times

during the flow: on reading the design (if –keep_unreach is not used), during mapping (if –unreach is not in the mapping method), and during verification (where some Not-mapped points may be 'promoted' to Extra due to unreachability.) We use the –keep_unreach / -unreach options to retain and map as many unreachable points as possible, in case bonus logic or temporarily tied-off features are desired in our netlists. Then the design engineers must approve all remaining unreachables before tapeout. At least one Intel project found a real error this way—some logic had been incorrectly tied off before netlist generation, and it had been a rare corner case not covered in any simulation test.

- **Are all blackboxes and library cells accounted for?** It is important that if parts of the logic are blackboxed, we understand why and who verified them. In addition, for all library cells that have functional representations in our LEC runs, the library team must ensure they are verified. We have caught occasional cases where ownership confusion results in some piece of logic being blackboxed in all FEV flows, obviously a dangerous situation.

2.4. Applying the Techniques

In the previous subsections we presented tips and tricks that had proven themselves very useful resolving Cadence Conformal LEC verification issues. In this subsection we would like to take the dual-port gigabit Ethernet controller design as the target design for those techniques. Figure 2 presents the initial running results. Some of the hierarchies were verified in a different run using the divide and conquer principle. In the part of the design highlighted in this figure we have 36K flip-flops, while in the full design we have ~200K flip-flops. However, the 4 abort points are created at the top level, and therefore the divide and conquer technique cannot help us to resolve them.

```
================================================================================
Compared points      PO     DFF     DLAT   BBOX      Total
--------------------------------------------------------------------------------
Equivalent           177    36314   13     185       36689
--------------------------------------------------------------------------------
Abort                0      0       0      4         4
================================================================================
The busses din[67..0] & dft_din[67..0] are the inputs of each of the aborted points
```

Fig. 2. The initial report of the verification of the dual-port gigabit Ethernet controller design.

To resolve the 4 abort points we would like to enable LEC to work harder in order to resolve them. We usually prefer to first spend computing time before spending engineering time to resolve the abort points. In this case the abort points are in datapath logic, so we used the Cadence Conformal Ultra feature. They are driven by two 68 bit buses and figure 3 presents the analysis of bit 67 of one of the buses. As can be seen in figure 3, then the logic cone fanin of this bit has more than 7K signals and the logic in this logic cone is greater 30K cells. This information is illustrating the complexity of those abort points.

```
> analyze abort -compare
================================================================================
Analysis Results
================================================================================
Compared result is abort.
  (G) + 37034  BBOX /u_common/.../u_mem
  (R) + 37535  BBOX /u_common/.../u_mem
  (G) + 331158 BUF  /u_common/.../u_mem/din[67]
  (R) + 456191 BUF  /u_common/.../u_mem/din[67]
================================================================================
Region    Size     Support      ADD/SUB      MUL/DIV       DC     Corr
                                gates, mod  gates, mod
--------------------------------------------------------------------------------
Golden    30751      7161        304,  3       0,  0       150     2848
Revised   39672      7163          0,  0       0,  0       0       2945
Proved    14509      7161         14,  1       0,  0       0       -
================================================================================

Legends:
  Proved region: region proved to be equivalent.
  Golden region: region in golden design that is outside the proved region.
  Revised region: region in revised design that is outside the proved region.
  Size: the number of gates in the region.
  Support: the number of support points (logic cone fan-in).
  ADD/SUB: the number of gates and modules from adders and/or subtractors.
  MUL/DIV: the number of gates and modules from multipliers and/or dividers.
  DC: the number of don't care gates.
  Corr: the number of gates with possible correspondence gates.
```

Fig. 3. Analyze report of one of the inputs of the one of the abort points

In the Cadence Conformal Ultra flow we used the four commands that appear in figure 4. The command "analyze datapath" evaluated the success results of using those algorithms on the abort points. We used the flag '–qual 30' to address all the 4 abort points (the default value is 50%). You should note that Cadence is working to automate this flow and therefore improve the user experience. We want to enable Cadence Conformal LEC to work harder and therefore we use the complete effort. The–single flag in the compare program wasn't necessary in this case, but it had proved to be useful in similar scenarios: it causes points to be compared individually rather than in groups. In version 05.20-s220 (07-Mar-2006) the verification took 12 hours and removed the abort points (as can be seen if Fig. 5). In version 06.10-p100 (25-May-2006) the time had been reduced to 6 hours.

```
analyze datapath -merge -verbose -share -qual 30
analyze abort -compare
set compare effort complete
compare -single
```

Fig. 4. The commands used in the Cadence Conformal Ultra flow

```
==========================================================
Compared points      PO     DFF    DLAT   BBOX      Total
----------------------------------------------------------
Equivalent           177    36314  13     189       36693
==========================================================
    CPU time     : 45034.57 seconds (12.51 Hours)
    Memory usage : 795.77  M bytes
```

Fig. 5. The verification results using Cadence Conformal Ultra

3.0 The Future of Formal Equivalence Verification

Formal Equivalence Verification (FEV) has come a long way in the last decade. In the early 1990's, FEV was mainly used by research institutes and early adopters. You couldn't run FEV on the entire design and you needed to have an FEV expert to do it. Since then the FEV technology has progressed significantly, and today FEV is being used by most ASIC design teams. The FEV tools can run on entire designs and they are being used by engineers and not just FEV experts. Looking back on the past, can we guess what the future holds for FEV technology?

FEV technology can provide you with one logic representation for different structural abstractions (e.g., RTL, netlist). In fact, the Cadence Conformal product line is more than equivalence checking. Cadence Conformal can perform clock-domain-crossing (CDC)[8] checks and constraint verification of false and multi-cycle paths[9]. In [2], they show how FEV can be used for replacing design elements with the elements that have the same functionality but are more power efficient. In the rest of this section we will discuss in detail two opportunities for extending FEV capabilities into new areas. The first capability is that FEV can be used for ECO automation, and we will discuss some experiences at Intel. Cadence is working on providing this capability as part of the Cadence Conformal tools. The second capability will discuss the ability to enable FEV to compare designs that are logically equivalent but have different implementation of state-points (e.g., pipe-line, retiming). In addition to that, we will discuss the trend of merging capabilities of FEV and FPV (Formal Property Verification).

3.1. ECO Automation

How would you address late changes in a typical design, after synthesis is complete and backend layout and optimizations have been done? One approach is to apply the changes to the RTL code and then rerun the implementation flow. However, rerunning the implementation flow will take time and can influence the schedule. An alternative approach is to perform the changes both in the RTL and in the netlist. However, this approach is error-prone and tedious, since the designer needs to

implement each change twice. To address this, a solution is needed that will take advantage of the benefits of those two approaches. It will enable us to perform the changes only once, and then will automatically generate an ECO (Electronic Change Order) script. How could it work?

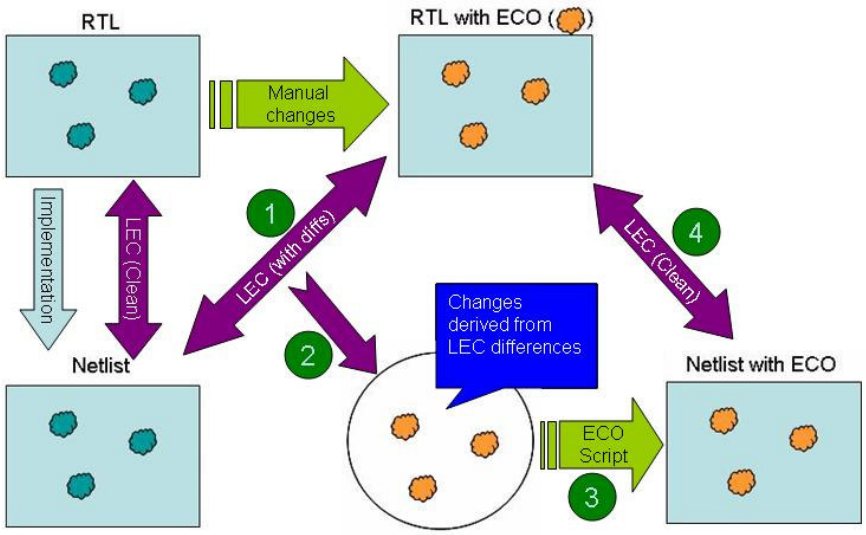

Fig. 6. The ECO Automation flow

The implementation flow takes the original RTL, and performs synthesis, place and route and adding clock-tree, scan-chains and other structures to the design. The design is passing several transformations but it keeps the same functionality. In particular, the original RTL and the generated netlist are FEV equal. Then we perform some changes to the original RTL and receive ECO-RTL. The original RTL and the ECO-RTL are non-equivalent, due to the changes that we made. Our goal is to change the original netlist in such way that it will have the same functionality as the ECO-RTL.

The differences between the ECO-RTL and the original netlist can be identified using LEC, as marked by (1) in Fig. 6 The next step (marked by (2) in Fig. 6) is to take the logic cones that are different and synthesize them. Then, we would like to update the logic-cones that were changed in the original netlist with the new logic-cones (marked by (3) in Fig. 6). Last, we compare the updated netlist with the ECO-RTL to verify the correctness of the change marked by (4) in Fig. 6). Note that the implementation of steps (2) and (3) can influence the efficiency of the ECO result. What part of the logic cone I want to replace? How do I reduce the timing and other violations that the change generates?

At Intel we have begun using several variations of this ECO automation flow. Cadence Conformal LEC is being used as the FEV engine[1] in some cases. The results of the ECO being done with this flow are good. The flow is faster than rerunning the implementation flow or performing it manually. Furthermore, the effect of the ECO on the implementation team is similar to manual ECOs.. The effectiveness of the ECO automation flow increases with the complexity of the ECO. However, we are currently limited due to the complexity of the scripts needed to implement this flow around LEC; as a result, not all projects have been able to make use of it. If Cadence succeeds in supplying this capability as a part of the Conformal tool suite, this could significantly increase our productivity.

3.2. Sequential Verification and FEV/FPV Convergence

One of the most difficult aspects of using current FEV tools has been the artificial restriction that designs must be state-matching: every latch in one design must have a corresponding latch in the other. But the design community's move towards high-level modeling in languages like System Verilog or System C means that the ability to compare designs that are not quite state-matching will become increasingly important. LEC has made baby steps towards relaxing this constraint, with the basic model flattening options and the datapath retiming capability, but is still very far from a true non-state-matching, or sequential, FEV engine. Other vendors and researchers have demonstrated the basic feasibility of such an approach ([4],[5],[6],[10]), so we will look forward to seeing Cadence's response in this area.

Another related issue is the convergence between FEV and FPV (Formal Property Verification) tools. FPV tools such as Cadence's Incisive, check properties, or assertions, in the RTL, to formally prove that they will be true for all cases. FPV and FEV differ mainly in that FPV analyzes large sequential cones, not stopping at state boundaries, and works on a single model. Once the state-matching constraint is relaxed from FEV, the tools start to look similar: a property verification can be thought of as an equivalence verification checking that some logical expression (the property) always matches a constant 1. It would be very nice if we had a single combined FEV-FPV platform, which could directly prove the validity of any constraints being used for FEV.

4.0 Summary

As we have discussed, Formal Equivalence Verification, through tools like Cadence's Conformal LEC, has become an essential part of design flows at Intel. We use this capability at many parts of the design flow including synthesis, backend netlist optimizations, ECO enabling, defeature testing, and final tapeout signoffs. Through our variety of experience with the tool we have developed a number of tips and tricks for fellow LEC users.

---

[1] There are ECO automation flows that use other FEV engines.

The main areas where we have shared our experiences in this paper are in resolving Abort point complexity issues, getting fully equivalent comparisons, and avoiding false positives before tapeout. To resolve Abort points, we use techniques like increasing compare effort, attention to cell libraries, hierarchical verification, partitioning, datapath analysis, and parallel comparison. Resolving nonequivalent points requires awareness of our Verilog usage style, understanding of model flattening options, and attention to the don't-care space. And most important of all, we carefully audit our logs to avoid false positives and holes in the verification flow.

Based on our experiences and our speculations about future needs, we have also suggested some extensions of LEC that might be useful: ECO automation support, to directly convert changes from an RTL model to changes in the netlist; sequential FEV, to allow direct comparisons between high-level models and RTL or Netlists; and FEV-FPV convergence, which might allow us to leverage multiple techniques in future flows. We have found Confromal LEC to be a great productivity enhancement, and hope to see its capabilities continue to increase in the future.

Acknowledgements

We would like to thank Michael Zuckerman, a LAD LEC expert that worked on the problem presented in the design and contributed from his time and knowledge to this paper. We would also like to thank to Zeev Yelin and Jerry Church, Cadence AEs, who supported us and helped us to identify useful LEC features, and Aviad Sokolver who worked on this as well. Furthermore, we would like to thank Chee Hak Teh, Han Yuen Ong, Hong Tin Goh and Tze Chun Ch'ng, who developed the LEC based ECO Automation flow at Intel. We would also like to acknowledge the many Intel design and validation engineers, too numerous to name here, that have assisted in implementing and running the various FEV flows on our designs.

About the authors

Erik Seligman
Erik Seligman has been at Intel for 12 years, working on design, valdation, and formal verification in teams including the Strategic CAD Lab, Circuit Research Lab, and the Desktop Platforms Group. Most recently he is a Formal Verification Architect in the Advanced Components Division (ACD), where he directs the design team's use of formal methods, including FEV and FPV, for next-generation chipset development. Before joining Intel, he received his M.S. in computer science from Carnegie Mellon University.

Itai Yarom
Itai Yarom joined Intel 8 years ago, driving the design and verification area in Intel LAN Access Division (LAD). Itai is the technical lead of ESL, Design, Verification and DFT in LAD. Itai drove new technologies to LAD and Intel, including formal equivalence checking, formal property verification, assertion based verification, SystemVerilog, DFT (Design-For-Test) and DFD (Design-For-Debug). Itai is the chair of the Intel SystemC User Group (ISCUG) and the Intel ESL Summit. Itai is a researcher in the center of rationality in the Hebrew University at Jerusalem, where he had received his MSc and BSc in computer science.

Reference

[1] S.A.Cook, “The complexity of theorem proving procedures”, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (1971), pp. 151-158.
[2] Z. Terem, G. Kamhi, M. Y. Vardi, A. Irron, “Pattern Search in Hierarchical High Level Designs”, ICECS 2004
[3] C. Pixley, "A theory and implementation of sequential hardware equivalence", IEEE Transactions on Computer-Aided Design, Dec. 1992, pp. 1469-1478.
[4] K.N. Lalgudi and M.C. Papaefthymiou, "Fixed-Phase Retiming for Low Power Design", Proceedings of the International Symposium on Low Power Electronics Design, Aug 1996.
[5] Z. Khasidashvili, M. Skaba, D. Kaiss and Z. Hanna, "Theoretical Framework for Compositional Sequential Hardware Equivalence in Presence of Design Constraints", ICCAD 2004, pp. 58-65.
[6] Calypto Design Systems website, http://www.calypto.com/.
[7] Intel Corporation, “Intel® 82571EB Gigabit Ethernet Controller”, Product Brief, http://www.intel.com.
[8] Cadence, “Clock Domain Crossing, Closing the Loop on Clock Domain Functional Implementation Problems”, http://www.cadence.com .
[9] Cadence, “Encounter Conformal Constraint Designer: Datasheet”, http://www.cadence.com.
[10] D. Kaiss, S. Goldenberg, and Z. Hanna, “Seqver: A Sequential Equivalence Verifier for Hardware Designs”, ICCD 2006, to appear.