Reference Model Based RTL Verification: An Integrated Approach

William N. N. Hung*
Synplicity Inc.
Sunnyvale, CA 94086
whung@synplicity.com

Naren Narasimhan
Intel Corporation
Hillsboro, OR 97124
naren.narasimhan@intel.com

Abstract

We present an approach that makes reference model based formal verification both complete and practical in an industrial setting. This paper describes a novel approach to conduct this exercise, by seamlessly integrating formal equivalence verification (FEV) techniques within a verification flow suited to formal property verification (FPV). This enables us to take full advantage of the rich expressive power of temporal specification languages and help guide the FEV tools so as to enable reference model verification to an extent that was never attempted before. We have successfully applied our approach to challenging verification problems at Intel®.

1 Introduction

Formal property verification (FPV) is assuming growing importance as digital designs get more complex and traditional validation techniques struggle to keep pace. Formal methods introduce mathematical rigor in their analysis of digital designs thereby guaranteeing exhaustive state space coverage. Within this framework, one can employ sound design abstraction techniques to rein in complexity. We will describe one such usage model in this paper and present a technique that facilitates its use in an industrial setting.

Reference model based verification attempts to address the two classes of problems that we routinely face when performing formal property verification on RTL designs: design complexity and verification re-use. It is common knowledge that most RTL designs are rendered complicated, primarily because they model complex functional behavior while accommodating tight performance constraints. Formal property verification is concerned with establishing the correctness of design functionality with respect to a high-level temporal specification. By developing a reference model that purely embodies the functional aspect of the original design without the performance considerations, it is often possible to mitigate the complexity issue.

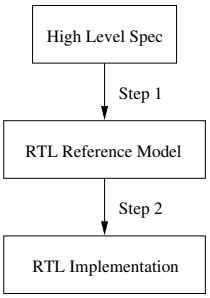

Figure 1. reference model for RTL verification

The exercise of reference model based verification may be viewed as a two step process and is illustrated in Figure 1. The first step is to verify that the reference model satisfies a higher-level specification. This problem is addressed by mainstream FV and is documented in a rich collection of papers both from academia and industry [1, 2, 7, 8, 12, 13, 15, 16]. The second step is to verify that the reference model, in the presence of environmental constraints, does imply the RTL design implementation. An important aspect of this step in the verification process is that it is directed towards a specific goal - to validate the result from Step 1 in Figure 1. Typically the reference model tends to be very close to the actual RTL and is simplified in areas that actually matter to the correctness of the high-level specification in Step 1. The differences in both the models therefore, tend to be localized and therefore far more amenable to automatated verification.

In the context of environmental assumptions E and temporal specification S, establishing the implication relation in step 2 is similar to the sequential equivalence verification problem. FEV tools are particularly adept at exploiting symmetry between models when establishing equivalence. Unfortunately FEV tool infrastructures provide little support for specifying E and S, especially when they are tem-

*This work was originally performed while at Intel Corporation.

poral and arbitrarily complicated. FPV tools on the other hand, provide good support for specifying protocol with complex timing, by way of incorporating expressive temporal specification languages within their flow. However, FEV tools and FPV tools have historically worked in isolation, never trying to exploit each other's strengths to further their impact on a given verification problem.

The Integrated Verification (IntVer) approach attempts to synergize the strengths of FEV tools with the rich expressiveness of FPV specification languages to engineer a system that enables reference model based verification. There has been some work done in academia in attempting to integrate equivalence checking within a more expressive specification framework like process algebra [5]. But our approach is probably the first attempt to effect such an integration so as to enable verification of large and complex RTL design systems.

Furthermore, reference model based verification supports verification re-use. In an industrial setting, designs are continually refined during the life of a project, either to accommodate functional changes or more likely performance constraints. In the case of the latter, the cost of re-proving the new design against the high-level specification becomes prohibitive. In a reference model based approach, since functionality is preserved, the re-proving task involves verifying the second verification step which is usually tractable and a more automatable problem.

2 IntVer: An Overview

IntVer combines the specification power of temporal logic with a state-of-the-art sequential formal equivalence verification tool suite to address the problem of performing reference model verification. This integrated approach offers certain important advantages over a strictly FEV or FPV approach.

As described in Section 1, we view reference model verification as a two step process. The first step involves verifying that the RTL reference model R satisfies the high-level specification S in the presence of temporal environmental assumptions E and a set of basic constraints B,

$$\psi,\ R \models S \tag{1}$$

Here ψ is a conjunction of the constraints B and E. We associate B with simple environmental constraints like clock and reset behavior and signal stability assumptions. The temporal constraints E tend to be more involved and specify constraints on protocol. Although semantically they are handled the same, the reason for making a distinction between the two types of assumptions will become clear soon. Establishing the relation in Formula 1 is a classic model-checking problem and can be presented to a decision procedure. The second step of the verification problem is to verify the abstraction relation between the reference model R and the implementation I. This involves verifying the relationship specified by the following:

$$B \wedge E \wedge S \Rightarrow I \tag{2}$$

Establishing a relationship between an abstract (reference) model and its implementation, using an FEV approach should make intuitive sense. In the absence of E and S, establishing the above relation reduces to a implication verification problem. FEV tools are quite adept at detecting symmetry in the model and using this information to help simplify their verification task. However, E and S tend to be fairly complicated and are best described by a specification language that supports linear or branching-time temporal logic. Typically, FEV tools do not directly support the use of such an expressive temporal specification language. The IntVer framework engineers a solution that enables such a support and re-constructs the problem in such a way so as to facilitate the use of FEV technology to prove the relation in Formula 2.

IntVer constructs a model combining the reference model R and the implementation I in such a way so as to restrict their behavior by the environment restrictions. The specification S and the temporal assumptions E are modeled as circuit representations and then used to restrict the behavior of the combined R-I model. This approach has the added advantage of re-using the same restrictions and specifications used in Step 1 of the verification when constructing the models for verifying Step 2.

The IntVer framework is independent of the FEV technology. The implementation outlined in this paper uses a sequential equivalence checker (with dual rail modeling) similar to the one described in [10], with satisfiability [13], and a rich specification logic that is LTL based. Like many sequential FEV predecessors [14], the checker exploits the symmetry between the two models to simplify the verification task. It uses a 3-valued logic (which is implemented through dual rail) to encode 3 possible values (X, 0, 1) for each circuit signal.

Verification of present day digital designs routinely stretch the FV technology to their limits. As a means to reining in complexity, typically an abstracted view of the design is presented to the verification tools and environmental assumptions (presented formally as a part of the overall specification) serve to constrain the design to all legal behaviors of interest. This has the net effect of restricting the initial state space of the design thereby leading to fewer state transitions and consequently a restricted reachable state space. Most verification approaches are fairly poor in integrating the benefits from a smaller initial state space with the actual verification effort. In order to remedy this, A class of techniques at Intel [9, 10] attempts to de-

couple the initialization process from the actual verification flow. The environmental assumptions originally formulated in temporal logic have to be restated in a form amenable to the initialization tools. It is often a non-trivial task to re-specify complex assumptions as waveform relationships. Furthermore, it raises the question of maintenance, since changes made to the temporal formulas describing the assumptions have to be reflected in the inputs to the initialization tools as well.

In contrast, IntVer's initialization technique imposes no such requirement on the translation process. IntVer uses mainstream formal verification tools in a novel way to help determine the initial states of the design, and can therefore take direct advantage of their capability to handle formally specified environmental assumptions. IntVer automatically extracts the relevant information from these constraints and uses this information to guide the verification flow to determine an initial state that satisfies these constraints. Once complete, the formal specification along with the initial state information is then passed on to the rest of the verification tool flow.

We present an efficient and automatic approach to initializing digital systems that takes full advantage of the design topology and the environmental restrictions. This dramatically reduces verification run times, lowers tool complexity, and opens an avenue to address verification of designs larger than what was possible in the past. Our approach improves on existing initialization approaches that are limited by the degree to which they can handle formally stated temporal environmental constraints. Furthermore, out technique presents a flexible and extensible approach that can be easily incorporated within diverse formal verification flows - be it formal property verification, formal equivalence verification or dynamic simulation exercises.

As a first step, IntVer analyzes the combined RTL model and determines its initial state. In our experience, this has a significant effect on improving the performance of the actual verification performed in Step 2 (Figure 1).

Our initialization technique, determines the legal set of initial states under the constraints specified by B and E. The problem of determining the initial state of the combined model is formulated as a satisfiability constraint and we present it to a bounded model checker (BMC) [3, 4, 6]. We specify a pair of mutually contradicting invariant constraints for each state element in the combined model and attempt to falsify each of them at the end of a certain predetermined bound,b. If a state element is indeed initializable to a constant value, one of the invariants for this piece of logic will hold while the other will fail.

The invariant pair for each state element i in the model is constructed in the following form:

Inv_T	Inv_F	Rule
fail	fail	Latch is uninitialized
pass	fail	Initialized to TRUE
fail	pass	Initialized to FALSE
pass	pass	Conflict detected

Table 1. BMC pass/fail outcomes per latch

$$\text{Inv\_T:} \quad \text{Signal } i \equiv T$$
$$\text{Inv\_F:} \quad \text{Signal } i \equiv F$$

Thus initialization approach allows us to take a snapshot of the initial machine state by determining all initializable state elements in the model using the resolution scheme shown in Table 1.

These properties are created for every state element i in the machine. There are four possible outcomes for every invariant pair associated with a state element. Table 1 describes the resolution to deduce the initialization values. A "pass"' in Table 1 denotes that the invariant holds within the user supplied bound. Clearly both properties cannot "pass" (last entry of Table 1), and that would indicate the presence of contradicting environmental assumptions.

Our technique lends itself naturally to coarse-grained parallelism. Initialization of a model with n state elements requires running BMC to falsify $2n$ properties. Our initialization tool partitions and groups these properties into separate BMC sessions through a distributed computing network.

IntVer, after running through the initialization step, presents a composite model of the hardware that has been correctly initialized with the basic and temporal constraints. This model is packaged in an appropriate form to the FEV tool flow. This generalized property verification problem is then analyzed by the FEV tool flow to determine the correctness of the reference model abstraction.

3 Case Study

We have used IntVer to solve quite a few challenging verification problems pertaining to designs of next generation Intel processors. We will briefly present a case study on the register renaming logic to illustrate the effectiveness of this integrated approach.

The purpose of the RAT (Register Alias Table) is to rename logical registers (visible in the instruction architecture) to physical registers (in the micro-architecture) as described in [11]. It can be viewed as a big array (table) of data storage. At every cycle, we are given an index (address). Using this index, the machine should first read the data that was stored in the indexed array entry, and then write new

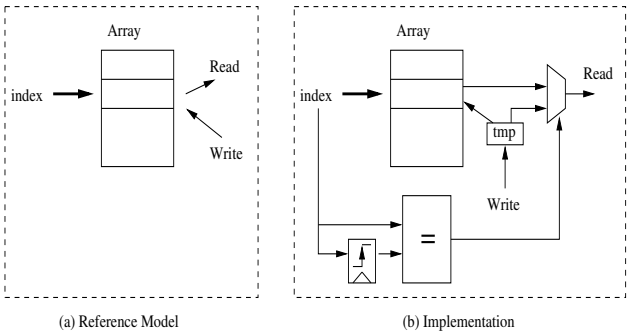

Figure 2. RAT models

data to that entry. We have developed a STE proof that every array entry is being updated (or not updated) based on the index (address) and the write port, and also checks that read port is getting the correct data from the indexed array entry. The proof also handles microarchitectural features such as pipelining, superscalar and hyper-threading.

We inherited the verification strategy for the register allocation logic, from the work developed for an earlier Intel microprocessor. The micro-architectural design of this logic had changed in certain key areas to improve overall performance. As a result, the original proof strategy could not be applied directly to the new design. For example, the designers had introduced bypass logic to improve array access throughput (see Figure 2(b)). Data from the write port of the array was now stored in the bypass logic. In the succeeding cycle, the temporary data is written into the entry originally pointed to by the index (address). This temporary register introduced added complexity for the read logic since since data that came in the previous cycle was not present in the array entry in the current cycle, it was put in the temporary register. So for the read logic, it was required that we compare the indices of both in the current cycle and the previous cycle. If they were the same, the logic read from the bypass logic. Otherwise, data was indexed from the array.

The changes in no way affected the functionality of the logic. It just made array reads faster with the introduction of the bypass logic. However, the original verification strategy cannot be used in the face of these design changes.

There were many complications introduced by this simple logic change, primarily because of the superscalar microarchitecture and special IA-32 instructions that cause concurrent read/write operations. To handle these changes, we needed to introduce an environmental constraint - an invariant that relates the temporary register with the lookup table. At a high level, this invariant ensures that the data in the current cycle's temporary register is same as the data stored in the current cycle's array entry pointed to by last instruction's index.

Design	State Elem	IntVer (CPU sec)
R1	3512	82966
R2	2834	22333
R3	1849	1748
S1	331	92
S2	233	360
S3	244	969

Table 2. Performance Results

$$entry(index(t-1), t) = tmp(t)$$

The above invariant was then used to constrain the verification flow. We were then left with the obligation to prove this invariant separately. However, a careful look at this invariant reveals a bigger problem: The index can point to any array entry. Therefore the $index$ is purely symbolic and, on account of the superscalar nature of the design, can potentially assume values from multiple sources. All these make it extremely hard to incorporate such an invariant within a verification strategy, be it BDD or SAT based.

As a first step to address this problem, we create an appropriate reference model for the RTL, that consists just the right amount of detail to preserve functionality but to abstract away the microarchitectural details that contribute solely to performance issues. In this case, the reference model is designed without the temporary register. It uses a direct writing and reading scheme as shown in Figure 2(a). We model-checked the reference model and ensured that it satisfies the high-level specification, without imposing the invariant restriction. We then used IntVer to verify that the reference model is a sound abstraction of the optimized implementation.

Furthermore, due to some environmental constraints (e.g. microcode restrictions), some primary inputs may contain the same information as the index (input to lookup table). The original RTL uses these inputs to update the temporary register under certain conditions. By removing the temporary register, our simplified RTL does not use these alternative primary inputs anymore. We specified an assumption (based on the microcode restriction) in formal specification to enable IntVer to prove the implication relation between the two RTL models. The behavior of the original RTL was undefined for invalid indices. Therefore, output values from the lookup table for invalid indices are ignored by the downstream logic since they do not perform any useful operation in that case. The reference model only captures the behavior of the RTL when instructions are valid. We were able to thus use these don't care conditions to further simplify the model. Furthermore, the relation holds only for the valid state-space of the RTL behavior.

Table 2 presents some results on performing verification on a collection of challenging RTL test cases from a

leading-edge Intel microprocessor design. We first used the the state-of-the-art model checkers to verify the high level specifications directly on the complicated RTL design. None of the verification runs completed. Using the two step approach, IntVer helped improve the capacity and the verification time taken to complete the verification . The number of state elements gives an indication of the size of the design state space.

4 Summary

We have powerful FPV and FEV tools at our disposal but they do not interact well. In this paper, we presented a novel approach that harnesses the strengths of existing formal verification techniques to enable verification of designs previously considered too large/complex for any one FPV technology. The approach described in this paper enables us to specify design abstractions in order to enable FPV. We then use the IntVer approach to bridge the abstraction gap between the reference model and the actual RTL design. The two step verification approach described in the paper could be easily generalized to a multi-step approach, should the reference model and the implementation be separated by an abstraction not directly amenable to be bridged by the FEV tools.

Since the FV proofs are on the reference model and not the actual RTL design, our approach is also fairly stable under the face of micro-architecture changes. Often these changes are made for performance reasons and since our reference model is a functional embodiment of the logic they remain stable even if the actual RTL implementation is modified. As a result, there is no need to redo the actual FPV proof. Verifying that the modified RTL design continues to match the reference model is sufficient and this can be easily done using the IntVer approach described in this paper.

Our approach is practical and less prone to design complexity problems, particularly because we are not using the FEV engine to do full abstraction verification. The verification is always performed in the context of a particular temporal behavior of interest. Therefore, only a small subset of output signals participate in the verification effort. This makes the overall exercise both practical and relatively less complicated than solving a generic FEV problem.

5 Acknowledgment

The authors would particularly like to thank team members Tom Schubert and Roope Kaivola for their guidance in developing IntVer, and folks at the Intel Israel CAD team for their support in debugging issues with the FEV tool flow.

References

[1] I. Beer et al. Rulebase: Model Checking at IBM. In *Computer Aided Verification*, pages 480–483, 1997.

[2] B. Bentley and R. Gray. Validating the Intel®Pentium®4 Processor. *Intel Technology Journal*, 5(1), February 2001.

[3] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. In *Proc. Design Automation Conf.*, 1999.

[4] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proc. Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99)*, volume 1579 of *LNCS*. Springer-Verlag, 1999.

[5] R. Cleaveland and S. Sim. The NCSU Concurrency Workbench. In R. Alur and T. Henzinger, editors, *Computer Aided Verification*, volume 1102, pages 394–397. Lecture Notes in Computer Science, Springer-Verlag, 1996.

[6] F. Copty et al. Benefits of Bounded Model Checking at an Industrial Setting. In *Proc. Computer Aided Verification*, 2001.

[7] R. Fraer et al. Prioritized Traversal: Efficient Reachability Analysis for Verification and Falsification. In *Computer Aided Verification*, pages 389–402, 2000.

[8] S. Hazelhurst and C.-J. H. Seger. Symbolic trajectory evaluation. In T. Kropf, editor, *Formal Hardware Verification: Methods and Systems in Comparison*, volume 1287 of *LNCS*. Springer-Verlag, 1997.

[9] S. Hazelhurst, O. Weissberg, G. Kamhi, and L. Fix. A hybrid verification approach: Getting deep into the design. In *Design Automation Conference*, June 2002.

[10] Z. Khasidashvili and Z. Hanna. SAT-based methods for sequential hardware equivalence verification without synchronization. In *Proc. International Workshop on Bounded Model Checking*, 2003.

[11] D. T. Marr et al. Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Journal*, 6(1), February 2002.

[12] K. L. McMillan. *Symbolic model checking - an approach to the state explosion problem*. PhD thesis, Carnegie Mellon University, 1992.

[13] M. Sheeran, S. Singh, and G. Stlmarck. Checking safety properties using induction and a SAT-solver. In *Proc. Formal Methods in Computer Aided Design*, 2000.

[14] C. A. J. van Eijk. Sequential equivalence checking based on structural similarities. *IEEE Trans. CAD*, 19(7):814–819, July 2000.

[15] G. Yang, J. Yang, W. N. N. Hung, and X. Song. Implication of Assertion Graphs in GSTE. In *Proc. IEEE/ACM Asia South Pacific Design Automation Conference (ASP-DAC)*, January 2005. Accepted for publication.

[16] J. Yang and C.-J. Seger. Introduction to Generalized Symbolic Trajectory Evaluation. In *Proc. International Conf. Computer Design (ICCD)*, September 2001.