Exploiting Retiming in a Guided Simulation Based Validation Methodology

Aarti Gupta[1], Pranav Ashar[1], and Sharad Malik[2]

[1] CCRL, NEC USA, Princeton, NJ
[2] Princeton University, Princeton, NJ

1 Introduction

There has been much interest recently in combining the strengths of formal verification techniques and simulation for functional validation of large designs [6]. Typically, a formal *test model* is first obtained from the design. Then, test sequences which satisfy certain coverage criteria are generated from the test model, which are simulated on the design for functional validation. In this paper, we focus on automatic abstractions for obtaining the test model from the design for simulation vector generation under the transition tour coverage model. Since most efforts using guided simulation have concentrated only on state/transition coverage, without relating these to error coverage of the original design, there is hardly any notion of preserving correctness, which has made it hard to use abstraction effectively.

We introduce one possible notion of abstraction correctness in this context– *transition tour error coverage completeness* of the abstract test model with respect to the original design. Intuitively, the abstraction is correct if it preserves coverage of those errors that can be captured by a transition tour. In other words, the test sequences generated from a transition tour on the abstract test model should cover the complete set of those design errors that are covered by test sequences generated from any transition tour on the original design. We have focused on the transition tour methodology in this paper, since it seems to be the most prevalent mode of generating test sequences. The notion can be potentially extended to other modes as well.

In particular, we propose the use of *Maximal Peripheral Retiming (MPR)* [8], where the number of internal (non-peripheral) latches is minimized via retiming. Subsequently all latches that can be retimed to the periphery are automatically abstracted in the test model, thereby reducing its complexity. We prove the correctness of such an abstraction under certain conditions, where correctness is regarded as preserving design errors that can be detected by using a transition tour on the original design.

2 Retiming for Abstraction

An *input peripheral latch* p_i is a latch whose fanin q is either a primary input or the output of another input peripheral latch, such that q fans out only to p_i.

L. Pierre and T. Kropf (Eds.): CHARME'99, LNCS 1703, pp. 350–355, 1999.

An *output peripheral latch* p_o is a latch which fans out only to either primary outputs or other output peripheral latches. An *internal latch* l_{int} is a latch which is neither an input peripheral latch, nor an output peripheral latch. The application of primary inputs is merely delayed by input latches – they do not affect reachability of rest of the state space. Similarly, output latches merely delay the availability of primary outputs. Internal latches determine the reachability of a circuit's state space.

A design implementation is considered to have an *error* with respect to its specification if the implementation produces an incorrect output for some input sequence. In general, a transition tour cannot capture all errors, since it covers all single transitions only, and not all transition sequences [3]. This is a basic limitation of using transition tours – an error may get detected several transitions after it is excited, and only along a specific path in the state transition graph. If this path is not selected in the transition tour, the error will not be covered. Furthermore, depending on what particular paths are selected, different transition tours may cover different sets of errors.

In order to not tie our analysis to a particular choice of a transition tour, we focus on those errors that can be covered by *any* transition tour of a given design implementation. Such an error can be detected as a difference in the observed output on a transition from a state, regardless of how that state was reached. We also do not restrict our analysis to systems with special properties that allow *all* errors to be covered by a transition tour [3]. We propose the following definitions as the criteria for correctness of an abstraction in this context:

Definition 1: Transition tour error coverage completeness: A model T has transition tour error coverage completeness with respect to another model D, if all errors in D which can be covered by any transition tour on D, are also errors in T and can be covered by some transition tour on T.

Definition 2: An abstraction $\mathcal{A}$ is correct if the abstract test model $T = \mathcal{A}(D)$ has transition tour error capture completeness with respect to the original design implementation D.

In particular, we propose the use of *Maximal Peripheral Retiming (MPR)* as an abstraction which satisfies this correctness property under certain conditions. In general terms, an MPR is a retiming where as many state elements as possible are moved to the periphery of the circuit. Consequently, there are as few internal state elements as possible. In the rest of this paper use the term "latches" to refer to all forms of state elements (latches/registers/flipflops). Once MPR is done on the design, consider the following two abstractions:

- $\mathcal{A}_{eq}$: Removal of all p_o, and removal of an *equal* number of p_i across all inputs.
- $\mathcal{A}_{neq}$: Removal of all p_o, and removal of *all* p_i across all inputs.

First note that the I/O-preserving nature of a retiming transformation itself guarantees that an error observed on a transition from a reachable state s in the original circuit will be preserved as a corresponding error from an equivalent state s' in the retimed circuit. It follows that the removal of a peripheral latch does not affect the presence of such an error, provided the latch was *correctly positioned* to start with. The position of a peripheral latch implies that it can

not affect the value of an output (only its timing). It follows that a transition tour on the abstract circuit will eventually reach s' (given the initial conditions) and cover the erroneous transition. We say that a peripheral latch is *correctly positioned*, if it has been separately validated that this position of the latch at the circuit periphery is consistent with intended behavior. In a sense, we are decomposing the burden of checking design correctness into (i) detecting errors in the peripheral latches, and (ii) detecting errors in rest of the circuit. It is our belief that the former task can be handled separately, and in some cases more efficiently, than the latter. In the typical case, a comparison with a higher level description of the design, or designer input may be called for to establish the correcness of the peripheral latches.

2.1 Proofs of Correctness

Theorem 1. *Removal of correctly positioned output peripheral latches preserves error detection for the circuit.*

Proof. The only purpose served by correctly positioned output peripheral latches is to buffer the primary outputs. It is clear that removal of these latches has no impact on either the state space visited during a transition tour, or on its input sequences, thereby preserving detection of errors. ∎

Theorem 2. *Removal of an equal number of correctly positioned input peripheral latches across all inputs preserves error detection for the circuit.*

Proof. For the abstract test model, let n_{pi} be the number of input peripheral latches removed from all inputs. Given m inputs, let $l_{i,j}, 1 \leq i \leq m, 1 \leq j \leq n_{pi}$ denote the initial value on the j^{th} input peripheral latch for the i^{th} input. Let state s be reachable by an input sequence Σ in the original design. Then, there exists an equivalent state s' reachable in the abstract model by the input sequence $\Sigma' = \sigma_1, \sigma_2, \ldots, \sigma_{n_{pi}}, \Sigma$, where σ_j is the input vector $l_{1,j} l_{2,j} \ldots l_{m,j}$. Since s and s' are equivalent, if there is an error from state s on input a, there will be an error from state s' on input a. Furthermore, since all reachable states and all transitions from those states are covered by a transition tour, this error will be detected by any transition tour with $\sigma_1, \sigma_2, \ldots, \sigma_{n_p i}$ as the initial prefix. Finally, during simulation on the original design, Σ' should be padded by $n_p i$ dummy inputs at the end, in order to account for the delay due to the original input peripheral latches with respect to the observed outputs. ∎

Theorem 3. *Removal of all correctly positioned input peripheral latches preserves error detection for the circuit.*

Proof. For the abstract test model, let n_i be the number of input peripheral latches removed from the i^{th} input. Given m inputs, let $l_{i,j}, 1 \leq i \leq m, 1 \leq j \leq n_i$ denote the initial value on the j^{th} input peripheral latch for the i^{th} input. Now consider a state s reachable by an input sequence $\Sigma = \sigma_1, \sigma_2, \ldots, \sigma_r$ in the original design, where each $\sigma_k = \sigma_{1,k} \sigma_{2,k} \ldots \sigma_{m,k}$ denotes the vector of

inputs 1 through m. Due to the equivalence of blocks C and C', there exists an equivalent state s' reachable in the abstract model by the input sequence $\Sigma' = \sigma'_1, \sigma'_2, \ldots, \sigma_{r+max(n_i)}$, where $\sigma'_k = \sigma'_{1,k}\sigma'_{2,k}\ldots\sigma'_{m,k}$ is the input vector constructed in such a way that:

$$\begin{aligned} \sigma'_{i,k} &= l_{i,k} && \text{if } k \leq n_i \\ &= \sigma_{i,k-n_i} && \text{if } n_i < k \leq r + n_i \\ &= - \text{ (don't care)} && \text{if } k > r + n_i \end{aligned}$$

Since s and s' are equivalent, same reasoning as for Theorem 2 follows. ∎

Note that in practice, Σ is not known *a priori* since we want to avoid performing a transition tour on the entire design implementation. Instead, during the generation of the transition tour Σ' on the abstract test model, the initial prefix captures the constraints imposed by the initial values of those input peripheral latches that are removed by the abstraction. Also note that though our analysis has been presented in terms of abstraction from the original design implementation to a test model, it also holds for abstraction from any concrete test model to an abstract one.

3 Algorithms for Maximal Peripheral Retiming (MPR)

Given a circuit C, we would like to derive a circuit C' by a retiming such that the number of latches at the periphery of C' is maximized. We refer to this problem as *maximal peripheral retiming* (MPR). We will consider two flavors of this, MPR_{eq} and MPR_{neq} corresponding to the abstractions $\mathcal{A}_{eq}$ and $\mathcal{A}_{neq}$ in Section 2. In MPR_{eq}, only an equal number of latches at each primary input will eventually be abstracted, so the remainder must be counted as internal latches. In MPR_{neq} there is no such restriction. In the worst case, if none of the latches move to the periphery, then the final circuit is the same as the initial circuit. Similar ideas are used in automatic test pattern generation for sequential circuits [1,4].

Let us consider the algorithm for MPR_{neq} first. The conventional retiming algorithm [7] for latch minimization with the cost of peripheral latches assigned to zero will suffuce to minimize the number of internal latches by attempting to push as many as possible to the periphery.

The algorithm for MPR_{eq} is handled with a minor modification to the circuit. A dummy node d_1 is added to the circuit with fanout to each of the primary inputs i_n. Another dummy node d_0 is added with no fanin and a single fanout to d_1. The cost of the edges is as follows: (1) for the (d_0, d_1) edge it is 0 (2) for the (d_1, i_n) edges it is infinity (3) for the output periphery edges it is 0 (4) for all other edges, it is the same as in the original circuit. This will force a minimization of internal latches in the circuit by trying to push as many latches to the output periphery and on the (d_0, d_1) edge. We can also maximize the number of latches at the primary outputs, by biasing the peripheral retiming in that direction by picking appropriate bus widths, e.g. 0 for output peripheral edges, 1 for input

peripheral edges, and some large number (greater than the maximum fanin of any gate) for internal edges.

Handling Enables and Multiple Clocks/Phases: To maximize the ability to move latches to the periphery, paths with unbalanced phases are balanced by adding dummy latches in appropriate locations. We can do this since the dummy latches do not affect the logical behavior. Only the clock period requirement is affected – which is immaterial to our application.

4 Case Study: The DLX Processor

We used a Verilog RTL implementation (without floating-point and exception-handling instructions) [2] of the popular DLX processor [5]. It uses a standard 5-stage pipeline consisting of the fetch, decode, execute, memory and write-back stages. The model after manual abstraction consisted of 92 latches. Upon applying MPR, we could abstract 32 latches that were already at the input periphery, 31 latches that were already at the output periphery, and 8 latches that could be retimed to the periphery. It was also interesting to examine the 21 remaining latches which could not be peripherally retimed. There are two main structures that prevent such retiming – (i) self-loops (ii) reconvergent paths with different number of latches. Typical examples of these for the DLX design are the self-loop for dumping the fetched instruction in case of a taken branch, and the reconvergent structure used to select the ALU sources. To summarize, of the 92 original latches, 71 were abstracted out by use of MPR, resulting in a final test model of 21 latches, 25 primary inputs and 31 primary outputs. The Verilog code was converted to an FSM description, and the implicit transition relation representation of the final model was obtained in about 10 seconds on an Ultrasparc (166 MHz.) workstation with 64Mb. main memory.

References

1. A. Balakrishnan and S. T. Chakradhar. Software transformations for sequential test generation. In *Fourth Asian Test Symposium*, 1995. 353
2. P. Franzon. Digital computer technology and design: Fall 1994 project. Private Communication, 1996. 354
3. A. Gupta, S. Malik, and P. Ashar. Toward formalizing a validation methodology using simulation coverage. In *Proc. 34th Design Automation Conf.*, pages 740–745, June 1997. 351
4. R. Gupta, R. Gupta, and M. A. Breuer. BALLAST: A methodology for partial scan design. In *Proceedings of the International Symposium on Fault Tolerant Computing*, pages 118–125, June 1989. 353
5. J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1990. 354
6. R. C. Ho, C. H. Yang, M. A. Horowitz, and D. L. Dill. Architecture validation for processors. In *Proc. 22nd Annual International Symposium on Computer Architecture*, June 1995. 350

7. Charles E. Leiserson and James B. Saxe. Retiming Synchronous Circuitry. *Algorithmica*, 6(1):5–36, 1991. 353
8. S. Malik, E. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. *IEEE Tran. on CAD of Integrated Circ. and Sys.*, 10(1):74–84, Jan. 1991. 350