Computing State Matching in Sequential Circuits in Application to Temporal Parallel Simulation

Dusung Kim[1] Daniel Gomez-Prado[1] Seiyang Yang[2] Maciej Ciesielski[1]

[1]*Department of Electrical and Computer Engineering*
University of Massachusetts, Amherst, MA-01003
{dukim, dgomezpr, ciesiel}@ecs.umass.edu
[2]*Department of Computer Engineering*
Pusan National University, Busan, Korea, 609-735
syyang@pusan.ac.kr

ABSTRACT

This paper addresses the problem of computing relationship between the states of two designs, specification and implementation. The problem is considered here in the context of temporal parallel simulation, where state matching is required to determine the initial values of registers used as starting points for individual simulation runs. This problem is particularly challenging if the implementation design is obtained from the specification design by a series of retiming and re-synthesis transformations. We show that the problem can be solved efficiently by modifying inductive techniques of ABC for computing signal correspondence.

General Terms: Design Validation, Verification.

Keywords: Parallel Simulation, Equivalence Checking, State Matching.

1. Introduction

The need for finding relationship between the states of different designs arises in several applications. Typically it is a part of Sequential Equivalence Checking (SEC), which attempts to prove equivalence of two designs by matching internal equivalence points, such as internal signals and registers. The problem is particularly hard if one design has been obtained from the other by sequential synthesis that involves retiming [1] and re-synthesis. Such a process is known to destroy a one-to-one register correspondence in the designs. All SEC methods rely on establishing such register correspondence, unless the transformation history generated by synthesis is provided. To the best of our knowledge, there is no general solution to the SEC problem due to its high computational complexity.

In this paper, the state matching problem is considered in the context of Temporal Parallel Simulation (TPSim) [2], which combines formal methods with simulation-based validation. Unlike other hybrid validation techniques [3, 4], which concentrate on increasing the simulation coverage, this approach significantly improves efficiency and the overall performances of both functional and timing simulation. The next section briefly reviews the basic concept of temporal parallel simulation.

1.1 Temporal Parallel Simulation

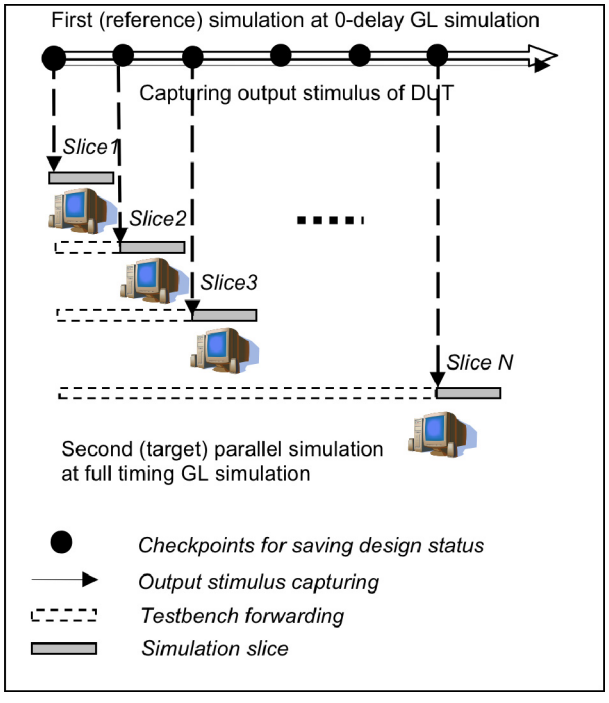

Fig 1. Concept of Temporal Parallel Simulation

Temporal Parallel Simulation, TPSim [2], is a technique that parallelizes gate-level simulation into temporal domain by using snapshots of states captured during the reference (e.g., RTL) simulation, as shown in Figure 1. In the following we assume that both the reference and target designs are given as gate level netlists. TPSim consists of two basic simulation steps:

1. Fast, functional zero-delay reference simulation on a single processor that collects design states and other meta information at predetermined checkpoints.
2. Second, full-timing target simulation, distributed to the individual processors.

It is known that functional gate-level simulation is usually 10~50 times faster than full-timing gate level simulation. Therefore, the functional simulation can serve as an abstract reference of timing annotated simulation because two simulations should be cycle-by-cycle consistent.

In this approach, the design state at each checkpoint is obtained from the functional reference simulation. These states are restored for the target full-timing simulation so that each slice between the consecutive states can be simulated in parallel. Note that testbench must be simulated from the beginning for each slice because the testbench is unsynthesizable and does not have explicit states.

1.2 Initializing States at Checkpoints

One of the essential elements of Temporal Parallel Simulation is that it requires knowledge of the initial states of individual simulation segments [2]. Rather than finding functional relationship between the states of two designs, we focus on finding register values of the transformed (implementation) design assuming the knowledge of register values of the original (specification) design. In our case the values of registers in the original design are obtained by initial (reference) simulation of that design. While this problem is easier than a general problem of proving sequential equivalence of a sets of registers from the two designs, it remains difficult if sequential transformations (retiming and re-synthesis) are involved in design generation. In this work the register values of the matched states in the implementation design are computed using induction based sequential equivalence checking techniques (SEC) of ABC software [5] with much smaller computational cost.

Sequential Equivalence Checking (SEC) of ABC relies on induction-based techniques and does not require structural matching points. It is very effective in finding functional relationships between internal combinational signals and registers. However, it is not sufficiently scalable to deal with complex designs. In our work, rather than proving circuit level equivalence or finding relationship among the registers, we consider a simpler problem, *state matching*, which computes a target design state from a reference design state.

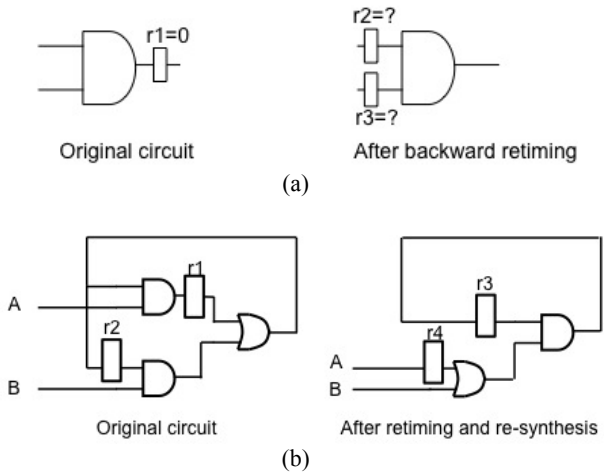

Fig 2. Retimed and re-synthesized designs.

One of the difficulties is proper computation of initial states. This point is illustrated in Figure 2(a), which shows a simple backward retiming. While finding the value of *r1* from *r2* and *r3* is trivial because $r1 = r2 \cdot r3$, finding *r2* and *r3* from *r1* is not always possible. In this case, unrolling the design over multiple time-frames is required to compute the values of these registers.

Figure 2(b) shows an example of retiming and re-synthesis. Since a one-to-one correspondence does not exist among registers, finding state matching or relationship between the states is not trivial.

Our approach takes advantage of Signal Correspondence (SC) algorithm implemented in ABC [5]. We modified the original algorithm so that it can compute a matched state at a lower computational cost compared to the original algorithm. This reduction in complexity is possible because constant values of the registers in the original design are known from the reference simulation, which dramatically reduces constraints imposed on the SAT solver used by the algorithm.

The general problem statement is presented in Section 2 and the technique of signal correspondence needed to solve state matching is given in Section 3. Our approach to state matching and its application to parallel simulation are explained in Section 4. Experimental results and conclusions are given in Sections 5 and 6, respectively.

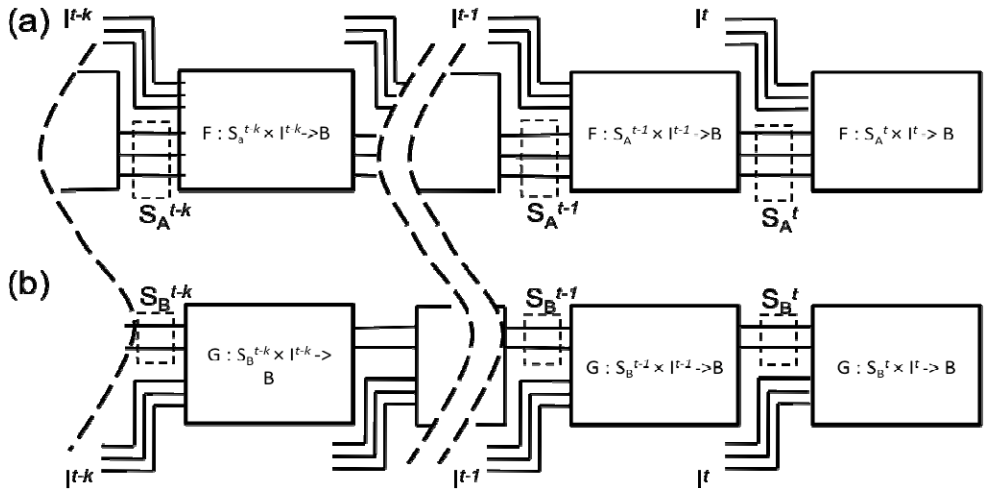

Fig 3. Time-frame model of two sequential circuits unrolled over a fixed number of clock cycles.

2. Problem Statement

Consider a pair of states, one (S_A) from the original design and the other (S_B) from the target design. The two states are considered matched if state S_B is obtained from state S_A through sequential synthesis (including retiming). While the two states are considered equivalent, their respective register functions may not be equivalent, if state S_B was obtained from state S_A using retiming. In our work we are concerned with registers *values* corresponding to the two states, rather than with a general state equivalence. For this reason we use the term *state matching*.

Figure 3 represents a time-frame model of a general sequential circuit. Fig. 3(a) represents a reference design and Fig. 3(b) the target design. Assume that the design in part (b) is a retimed and re-synthesized version of design (a).

Given the original state S_A^t at time frame t we want to find a matching target state S_B^t. In our approach, we assume that S_A^{t-k} (for a small constant k) is known from the reference simulation.

3. Signal Correspondence

Signal correspondence (SC) is a computation of a set of classes of sequentially equivalent nodes using *k-step* induction [5] based on SAT solving. The *k*-step induction is a gradual process that refines candidate equivalence classes. If SC exists for all the registers in the two designs, functional state relationship can be established. However, finding such a correspondence is not always possible because retiming and re-synthesis breaks the initial one-to-one correspondence between the registers.

The SC technique relies on structural similarity [6] of two designs. Since retiming-based sequential synthesis maintains a great degree of structural similarity (as it moves registers across logic gates), the technique is effective for sequential circuit verification.

SC can be computed by using inductive proof technique [7, 8]. Van Eijk [6] applies the method in equivalence checking by taking advantage of structural similarity. Recent implementations of SC [5, 9] combine other techniques such as SAT sweeping [10], speculative reduction [11] and invariant identification [9].

K-step induction involves multiple (*k*) frames, which allows it to identify more invariants than a one-step induction. The k-step-induction consists of the following two steps [5]:

- Step 1: Base Case – The equivalent classes hold for all inputs in the first *k*-1 frames starting from the initial stage, and
- Step 2: Inductive Case – if, assuming to be true in the first *k*-1 frames starting from *any* state, they hold in the k^{th} frame.

One of the most advanced implementations of the *k*-step induction for the purpose of SEC has been done in ABC. Base case contributes the initial refinement of equivalence classes by using Bounded Model Checking (BMC) from initial state. Inductive step is an iterative process to make further refinement by applying speculative reduction and SAT sweeping, which merges all nodes of an equivalence class in each of the first *k* time-frames onto its representative and proves or disproves equivalence of nodes. All the nodes that belong to the same equivalence class are merged. If the merging process results in constant 0,

the two circuits are sequentially equivalent. Detailed algorithm is described in [5].

4. State Matching

The basic idea of state matching is based on the following steps: (1) compute signal correspondence between the internal signals of the two designs; (2) migrate the value of a given reference signal to the corresponding signal in the target design; and (3) propagate it in the target design to obtain other register values.

While in principle we can use the original SC algorithm of ABC, it is overkill because we don't need to prove equivalence of signals for *all* possible value assignments. Instead, we need to find a specific assignment for all the register *values* in the target design based on the *known* value of the corresponding reference state. For this reason we introduce a concept of Constrained Signal Correspondence (CSC), which is an SC valid in certain time-frames that satisfy certain constraints. Our algorithm iteratively detects CSCs by using reference state values as constant invariants. Assigning values to those CSCs gradually disregards unmatched time-frames as well as computes the target state.

Figure 4 summarizes the main concept of our approach. Assume that S_A and S_B are the matching states. An initial state S_A^{-2} of the reference design (at two time frames prior to some reference point) is given in Fig 4(a). All the reference signal values between S_A^{-2} and S_A are computed by simulation.

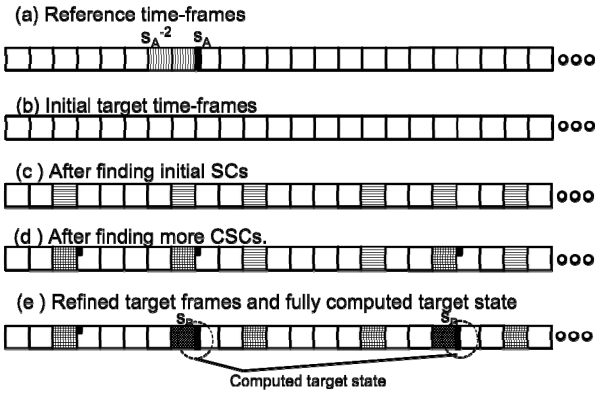

Fig 4. Progressive state matching.

Initially, as shown in Fig. 4(b), there is no candidate in the target design that matches with the state in the reference design in the required time-frame. Finding the initial SCs allows us to migrate the corresponding values from the reference frame (S_A) to target. Performing SC algorithm with the migrated values detects CSCs, which allows further migration of signal values. Recursive process of finding CSCs and value migration successively filters out the unmatched time-frames, see Fig. 4 (c),(d),(e). By iterating the process, target state is gradually computed (see Fig. 4(d), (e)). Note that our algorithm doesn't care to which time-frame the computed state belongs (Fig 4(e)).

4.1 State Matching Algorithm

We modified the original SC algorithm of ABC to enable efficient state matching. Figure 5 shows the algorithm, with the modified parts shown in bold.

Our algorithm starts by performing simulation of the reference design for *k* time-frames of the reference design (Figure 3.) The initial state for the simulation comes from the reference design. Therefore, the values of all the internal signals within the *k* time-frames are known (from the reference simulation). Then, a miter is added to the two designs to enable finding SCs.

```
aig runStateMatching( aig ref, aig impl, state r_state, int k,
int kmax ) {
  //Find all signal values of reference design
  V_ref = performSimulation(ref, r_state, k)
  aig N = addMiter(ref,impl);
  set of node subsets Classes = randomSimulation( N );
  //guided simulation if TB exists
  performGuidedSimulation(N,TB)
  //refine equivalences by BMC from the initial state
  refineClassesUsingBMC( N, k-1, Classes );
  //perform iterative refinement of candidate classes
  do {
    //do speculative reduction of k-1 uninitialized frames
    network NR = speculativeReduction( N, k-1);
    //derive SAT solver containing CNF
    //of the reduced frames
    solver S = transformAIGintoCNF( NR );
    //check equivalences and mark them with constants
    SatSweepingWithConstant (S, Classes, V_ref);
    //try to compute target state
    state tg_state = simulate(N,k);
    if(tg_state is computed) return tg_state;
  }
  while ( Classes are refined during SAT sweeping );
  //try additional value propagation up to kmax frame
  tg_state = simulate(N,kmax);
  if(tg_state is computed) return tg_state;
  return failure;
}
```

Fig 5. State matching using *k*-step induction

Generating accurate initial candidate equivalence classes dramatically increases performance of the

induction algorithm. In general, the classes are generated by a short period of random simulation. In this paper, a guided simulation is also performed because it provides a better chance to obtain initial classes. For example, many sequential circuits require special long term warm up process that needs special input sequences. In this case, random simulation easily leads to reset state. We use original BMC and speculative reduction to establish base case and assume stage of inductive case, respectively.

SatSweepingWithConstant method is used for proving stage of inductive case. It proves or disproves the equivalence classes in k^{th} time-frame. Unlike original SatSweeping algorithm of ABC, the nodes subjected to sweeping are not only merged into their representative but are also marked with a constant value if one of the nodes belongs to the reference design. The migration of values from reference frame is done implicitly. The marked constant values reduce constraints for proving the equivalence of other nodes.

All the newly detected equivalences based on migrated values are not SCs but CSCs. In other words, the detected equivalences are valid only for the time-frames having the same migrated value assignments. Therefore, as CSCs are detected iteratively, unnecessary time-frames are filtered out.

The algorithm terminates when all the register values of the target frame are computed successfully. The computation is done by simulation with the values of CSCs. In the case when the induction algorithm cannot find a target state, we try to find an alternate matching between k and k_{max} by performing additional simulation with the CSCs found by induction.

In conclusion, our approach is optimized for state matching problem in the following sense:

- Assigning constants during SAT sweeping reduces constraints by propagating constants
- The number of detected CSCs is much larger compared to SCs. Therefore the refinement process is faster.

4.2 TPSim with State Matching

Combining state matching technique with TPSim makes it possible to handle sequential transformation between reference design and target design. In this case the values of state registers in the reference design are provided by reference simulation. Our state matching approach guarantees that the computed state exists at the same distance from the initial state as in the reference state. Therefore, restoring the matched state for target simulation produces the same simulation result as stand-alone target simulation.

The following are a few applications of TPSim with state matching.

- Functional regression simulation for sequential transformations

  EC, especially SEC, is relatively less scalable technique compared to simulation. Therefore, simulation based regression test is still actively used. Especially for repeated synthesis tasks, quick simulation-based verification is more realistic than EC. With our approach, once reference design states are saved, regression simulations can be parallelized.

- Fast dynamic timing verification of sequentially optimized circuits.

  In many practical cases retiming is used for fine tuning of critical paths after delay annotation. This type of try-and-fix process usually involves trivial sequential transformation, yet it must assure the correctness of functional as well as timing verification.

The minimum value k to successfully compute a matched state depends on the values of CSC. Therefore, it may be possible that the initial state for some simulation slice may not be computed. To address this issue, we use *k-tolerant* model for TPSim to remove such a block-out period. If k is not sufficient to find a matching state for particular simulation slice, the earliest slice having successfully matched state provides backup by continuing its simulation beyond its slice boundary (see Figure 6). $S1_A^{-k}$ and $S2_A^{-k}$ are used to compute $S1_B$ and $S2_B$, respectively. Assuming that $S2_B$ fails to be computed, slice $n+1$ cannot be simulated in parallel because its initial state is not known. In order to achieve fully continuous simulation, simulation from $S1_B$ must override slice $n+1$.

This approach prevents block-out periods caused by failure to compute a matched state. It is a trade-off between the cost of the slice simulation and computation of the matching state.

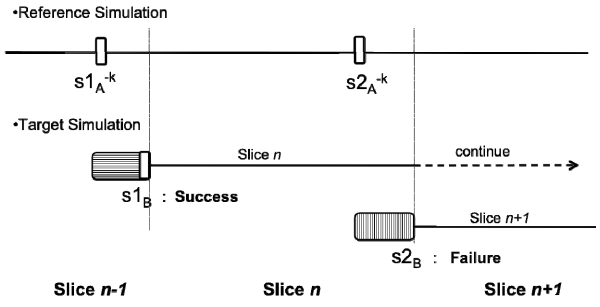

Fig. 6 TPSim with state matching.

5. Experimental Results

We used *aes_core* (IWLS 2005 benchmarks) as a benchmark for the experiments. The testbench for the design was obtained from opencores [12].

	Matching with SC	Matching with SC&CSC	Refining SC	Refining SC&CSC
retime	0.62s	0.47s	1.97s	1.21s
retime & refactor	1.41s	0.92s	5.42s	1.43s

Table 1. State Matching results.

Number of Cycles	1M
Conventional timing-annotated simulation	**1226s**
Functional reference simulation	329s
Timing-annotated Target simulation (1 out of 10 slice)	132s
Reference + Target simulation	**461s**

Table 2. Pre-layout timing simulation.

Table 1 shows the result for state matching. In this case, the number of time frames for the induction is fixed at 4. Both retiming (forward retiming with minimum delay) and refactoring are done by ABC.

The first and second columns give the time to obtain target state by using SCs and CSCs. The third and forth columns represent the time to get final refinement for both SC and CSC in the given four frames. The result shows that refining CSC requires less computational effort than SC. The performance of state matching is also improved by taking advantage of CSC. Finding matched state can be completed before finishing the refinement process if enough migrated values are generated to compute a target state.

Table 2 represents the performance of TPSim. The original version of *aes_core* design is used for reference simulation. Retimed and re-factored design with the necessary pre-layout timing information was used for target simulation. We used standard NC-Verilog simulator as simulation engine; interfacing with TPSim module was provided by PLI (Programmable Language Interface). The initial state for target slices was obtained by state matching. Since each slice can be simulated in parallel, we obtained a 3x speedup compared to conventional standalone simulation, while maintaining the same signal visibility. Note that the overhead due to state matching is included in the simulation time of target simulation and such an overhead is small. Most of the overhead in these experiments comes from the PLI implementation to connect simulation core and the TPSim module. It can be further improved by optimizing the implementation.

More experimental results for TPSim are available in [2].

6. Conclusions

Finding state relationship between two designs is in general a hard problem. This paper introduces a simpler, yet practical problem of *state matching* between a design and its retimed and re-synthesized version.

To compute a matched target state from a known reference state, we modified the original SC algorithm so that it requires much lower computational overhead. During the iteration of equivalence class refinement, the values of SCs and CSCs from the reference design frame are progressively migrated to the corresponding target design frame. These migrated values are used as a seed values to compute target state.

We showed the application of signal correspondence and state matching to temporal parallel simulation (TPSim), a practical solution to design validation. It is useful for fast dynamic timing verification as well as functional regression simulation for verification of sequential transformations. *K*-tolerant simulation approach assures preventing block-out even in the case when target state for some slices cannot be computed.

For future work, we are planning to investigate state matching between RTL and gate level design, which is of great practical interest. Under this model, TPSim is expected to increase simulation performance dramatically.

7. Acknowledgements

This work was supported in part by the US National Science Foundation, award no. CCF-0702506.

8. References

[1] C, Leiserson and J. Saxe. “Retiming synchronous circuitry.”, Algorithmica Vol 6, pp. 5-35, 1991.

[2] D. Kim, M. Ciesielski, K. Shim and S. Yang, “ Temporal parallel gate-level timing simulation”, High Level Design Validation and Test Workshop (HLDVT), 2008, pp. 111 – 116.

[3] Pei Hsin Ho, Thomas Shiple, Kevin Harer, James Kukula, Robert Damiano, Valeria Bertacco, Jerry Taylor, and Jiang Long. Smart simulation using collaborative formal and simulation engines. in Proc of ICCAD ’00, pp. 120–126, Piscataway, NJ, USA, 2000. IEEE Press.

[4] S. Shyam and V. Bertacco. “Distance-guided hybrid verification with guide.”, in Proc. of DATE ’06, pages 1211–1216, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.
[5] A. Mishchenko et al. “Scalable and scalably-verifiable sequential synthesis.”, in Proc. of The International Conference on Computer-Aided Design (ICCAD), pp. 234 – 241, 2008
[6] C. A. J. van Eijk. Sequential equivalence checking based on structural similarities, IEEE TCAD, 19(7), July 2000, pp. 814-819.
[7] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in Proceedings of TACAS, vol. 1579, LNCS, 1999.
[8] M. Sheeran, S. Singh, and G. Stalmarck, "Checking Safety Properties using Induction and a SAT Solver," in Proc. of FMCAD, 2000.
[9] F.Luand, T.Cheng. “IChecker : An efficient checker for inductive invariants”. in Proc. HLDVT ’06, pp. 176-180.
[10] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking", in Proc. ICCAD '06, pp. 836-843.
[11] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman. “Exploiting suspected redundancy without proving it”. in Proc. DAC’05.
[12] www.opencores.org