MULTES : MUlti-Level Temporal-parallel Event-driven Simulation

Dusung Kim, *Member, IEEE,* Maciej Ciesielski, *Senior Member, IEEE,* Seiyang Yang, *Member, IEEE*

***Abstract*—**
Multi-level Temporal-parallel Event-driven Simulation (MULTES) is a new, radically different approach to simulation of designs described in Verilog HDL. It is based on a concept of time-parallel simulation applied to gate-level timing simulation. The simulation is performed in two steps: (1) fast *reference simulation* that runs on a higher, reference level design model (typically RTL) and saves the design state at predetermined checkpoints; and (2) *target simulation*, which runs on a lower, gate-level model and distributes the simulation run slices to individual simulators. The paper addresses a number of important issues that make this approach practical: finding initial state for each simulation slice; resolving initial state mismatches; and handling designs with multiple asynchronous clocks. Experimental results performed on industrial designs demonstrate the validity and efficiency of the method in terms of its performance and the debugging efficiency.

***Index Terms*—Verilog Simulation, Parallel Simulation, Timing Simulation, State Matching.**

I. INTRODUCTION

HARDWARE simulation remains the most widely used technique for functional and timing verification and will remain so for a foreseeable future [1]. However its performance for complex designs becomes prohibitively low, sometimes reaching the rate of less than one cycle per second. Static Timing Analysis (STA) is often used in timing verification, but it cannot replace simulation-based verification.

Several simulation techniques have been introduced over the last three decades in an attempt to improve simulation performance. One of them is hardware emulation, which in theory should provide orders of magnitude performance improvement. However, to achieve it, the testbench must also be synthesizable and implemented in hardware along with the design under test (DUT), which significantly lowers internal signal visibility and controllability. In this respect, the use of emulation for gate-level timing simulation is limited as the delay model of DUT needs to be adjusted with respect to the actual delay imposed by the emulation device. It is an error-prone and complicated task. Furthermore, the prohibitive cost of purchasing and maintaining the FPGA-based emulation platforms makes it an expensive proposition.

Another technique to improve simulation performance is Distributed Parallel Event-driven Simulation (PDES), which partitions the design into separate modules and performs concurrent simulation using multiple HDL simulators [2], [3]. Unfortunately, these methods have not been very successful due to inherent inter-dependence between the design modules, which imposes heavy communication and synchronization overhead, thus introducing significant runtime overhead for simulation. Partitioning of design into modules to minimize the inter-module communication is a known NP-hard problem. It is particularly severe in gate-level timing simulation because it involves many more event activities.

This paper presents a new approach to parallel simulation, specifically directed at efficient gate-level timing simulation that increases both the performance and debugging efficiency. The main goal of the simulation technique presented in this paper is to completely eliminate communication and synchronization overhead associated with spatially distributed parallel simulation. This is achieved by applying the concept of *time-parallel simulation* [4], which partitions the simulation run into shorter simulation slices, instead of partitioning the design. Such a simulation does not suffer from drawbacks of current distributed simulation, such as design partitioning, synchronization and communication overhead.

The proposed method consists of two steps:

1) Fast reference simulation, which runs on a higher level (reference) design model and stores the necessary information about the design state (i.e., register values and memory print) at predefined checkpoints; and
2) Target simulation, running on a lower level (target) design model, distributed to individual simulators.

The entire simulation run is divided into simulation *slices*, each to be executed on an independent simulator. For this reason, we refer to this technique as *Multi-level Temporal-Parallel Event-Driven Simulation* (MULTES). Including both reference and target simulation in a single simulation task dramatically increases debugging efficiency because comparison with the reference model is performed naturally, as part of the overall design process, at no significant additional cost.

The general framework of MULTES was introduced in [5]. This paper describes a complete solution, addressing a number of practical issues, such as state matching and handling multiple asynchronous clocks. It also presents the results of the prototype tool tested on industrial designs. The background work on distributed parallel simulation and time-parallel simulation is discussed in Section II. The basic idea of the proposed technique is explained in Section III, with the key concepts detailed in Section IV (state matching) and V (timing simulation issues). Experimental results are described in Section VI, and conclusions are presented in Section VII.

Dusung Kim is with Synopsys, Inc., Mountain View, CA 94043 USA

M. Ciesielski is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, 01002 USA (e-mail: ciesiel@ecs.umass.edu)

Seiyang Yang is with the Department of Computer Engineering, Pusan National University, Busan, Korea (e-mail: syyang@pusan.ac.kr)

Manuscript received Feb XX, 2012; revised XXX XX, XXXX.

II. Previous Work

A. Parallel Distributed Event-driven Simulation

Parallel distributed event-driven simulation is performed by running each portion of the design separately in a different processing unit, called a Logical Processor (LP). The correctness of the simulation is assured when the event order among LPs is always preserved. Therefore, event synchronization between the LPs is the key issue in this approach. There are two main techniques used in synchronization protocol, a *conservative* and *optimistic* protocol.

The key idea of *conservative protocol* is to avoid violating the local causality constraint: no LP should receive an event from other LPs whose time-stamp is earlier than its local simulation time. Therefore, each LP must maintain a pending event list. *Optimistic protocol* is a technique which allows for a tentative violation of the local causality constraint. In this approach simulation state is saved at predetermined *checkpoints* during each simulation run to save the state of the design for the purpose of data synchronization. Once a violation is detected, simulation system recovers the simulation state from violation by performing rollback to the latest checkpoint.

Several variations of distributed parallel simulation methods have been offered, differing in the way the inter-simulation synchronization is handled. The first algorithms for conservative protocol were shown by Chandy, Misra [6] and Bryant [7], known as a CMB algorithm. It uses simulation time lookahead to maintain the causality order of each processing. In addition, $null$ message sending protocol was used to avoid deadlock condition in [8].

The most popular optimistic method is Time Warp, introduced by Jefferson [9]. Time Warp is a framework of optimistic protocol which performs rollback in the case when a message having previous time stamp arrives. Manjikian et. al. [10] implemented a parallel gate-level simulator on a local area network of workstations. Simulations with circuits from the ISCAS-89 benchmark suite achieved speedups between 2 and 4.2 on seven processors. Bagrodia et. al. [11] developed a parallel gate-level circuit simulator in the MAISIE simulation language and implemented it on both distributed memory and shared memory parallel architectures, achieving speedup of 2-3 on 8 processors for circuits from the ISCAS-85 benchmark suite. Lungeanu et.al. [12] proposed a dynamic approach, which combines conservative and optimistic approaches by switching between the two protocols depending on the amount of roll-back, and developed Parallel Discrete Event Simulation (PDES) for VHDL simulation. They demonstrated speedup of up to 11 on 16 processors on a circuit with 14k gates.

Li et. al. [13] claim to have developed the first Verilog distributed simulator. However, their simulator couldn't achieve the desired performance improvement due to extensive message passing and rollback overhead. Zhu et. al. [14] attempted to tackle more realistic designs and achieved good event processing rate, with the number of events increasing linearly with the number of LPs. This indicates that the design was partitioned almost ideally, while maintaining good load balance among the LPs. However, such an approach is impractical for large, complex RTL or gate-level designs.

An approach to use GPUs for parallel gate-level simulation has been proposed in [15], but the method is confined to a zero-delay gate-level model. Furthermore, performance of such a simulator strongly depends on the type of the design. Zhu et. al. [16] adapted the CMB algorithm to handle arbitrary gate-level delays and created a distributed datastructure to make it suitable for GPU executions. A dynamic GPU memory management has been used for efficient utilization of the relatively limited GPU memory. The modified asynchronous CMB algorithm significantly lowers the overhead of synchronization in such a a fine-grained partition (but not the communication overhead, which remains strongly data dependent). The time-based method described in this paper offers an alternative approach to the spatial fine-grained approach.

Some attempts have been made to perform parallel simulation method at a system-level. In [17] a method has been proposed to manage synchronization and communication between the partitioned blocks by applying a synchronization wrapper. The method achieves some improvement in simulation speed at a cost of sacrificing the cycle-by-cycle accuracy. As such, the method cannot be applied to GL simulation where an event accuracy, rather than just cycle-by-cycle accuracy, is required. Most of the results in this area were demonstrated on small to moderate-size, single-clock designs that can be partitioned without incurring significant inter-module communication and synchronization. Only a few commercial products have been developed, including SimCluster [2] and MP-Sim [3], but they have not attracted much attention from designers due to limited performance and scalability issues.

B. Time-parallel Simulation

Time-parallel simulation is a technique which divides simulation run into independent intervals (simulation slices) in temporal domain and each interval is simulated in a different LP. In this approach, the key issue is identifying initial states for each simulation interval. To obtain the correct simulation result with respect to a stand-alone simulation, the signal values at the end of interval i-1 must match the initial state of interval i. We refer to this requirement as the *horizontal state matching* problem. Once the initial state for each interval is provided, one can expect a very high degree of parallelism because the intervals are independent of each other.

The first form of time-parallel simulation was introduced by Heidelberger and Stone [4], which targeted trace-driven simulations. During the simulation, trace-driven simulation records only a trace of those execution events which need to be simulated. If the trace is not too large it can be reused for the next simulation. Andradottir [18] applied this approach to simulation of an ATM multiplexer by using regeneration points. Greenberg [19] developed an algorithm using parallel prefix to simulation queuing system. Despite of its potential for massive parallelism, the research into time-parallel simulation has not been very active. This is dictated by the difficulty of finding the initial state of each simulation interval. As a result, the application of this approach for practical designs has been very limited.

III. Multi-level Temporal-parallel Event-driven Simulation

A. Basic Concept

In contrast to traditional distributed parallel simulation (spatial parallel simulation), which partitions the design in space into a set of interacting modules, MULTES partitions the simulation run in time. If does it by cutting the entire simulation run into a number of independent simulation slices, each to be executed on a separate simulator. The simulation consists of two major steps, illustrated in Fig. 1:

1) *Reference Simulation*. Simulation which provides essential information of temporal partitioning by storing initial states for each simulation slice.
2) *Target Simulation*. Simulation of each slice, performed in parallel, using initial states saved during reference simulation.

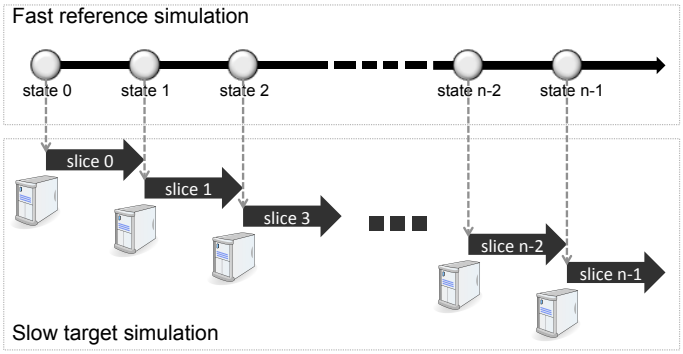


Fig. 1. Multi-level temporal-parallel event-driven simulation.

For this approach to work, the initial design state for each slice of the target simulation must first be captured and saved during the reference simulation run. This is done at predetermined *checkpoints*, determined by the number of processors available for parallel simulation. The *design state* consists of the state of all internal registers and memory print of the design. By saving and restoring the design states, simulation of each slice can be made independent of each other. As a result, target simulation can run concurrently and independently for each slice.

Theoretical performance of MULTES, measured in total simulation time T_M, can be expressed in its idealized form by equation (1).

$$T_M = \sum_{i=1}^{n} T_{Sts}(i) + T_{Rsim} + \max_{1 \le i \le n} [T_{Tsim}(i) + T_{Str}(i)] \quad (1)$$

The components of the expression (1) are defined as follows:

- T_M is the total simulation time in MULTES;
- $T_{Sts}(i)$ is the state saving time for slice i;
- T_{Rsim} is simulation time for the reference model;
- $T_{Tsim}(i)$ is simulation time of one slice for the target model;
- $T_{Str}(i)$ is state restoring time of one slice for the target model.

T_M includes the reference simulation time T_{Rsim} and the simulation time $T_{Tsim}(i)$ of the slowest target slice. In practice, state saving and restoring overhead parts, T_{Sts}, T_{Str} are very small and can be ignored. Hence, the smaller the reference simulation time T_{Rsim}, the greater the performance improvement that can be expected by this approach (asymptotically approaching linear speedup, if all but T_{Tsim} are ignored).

The main idea of MULTES is in line with the established design verification flow, which uses progressive refinement from high abstraction level to low abstraction level, as shown in Fig. 2. Notice the relative simulation speed of different design models shown in the figure. Using gate-level full timing simulation as a base line, RTL model offers about 100× speedup, while the most abstract, Electronic System Level (ESL), such as TLM or ISA level, gives another 100× speedup improvement. The proposed time-parallel simulation takes advantage of this progressive refinement, in which the fastest simulation (e.g. RTL) can serve as reference simulation for the simulation of the lower-level design (gate-level). There is also an intermediate, zero-delay gate-level model, which could be used as reference simulation for a full-timing gate-level simulation, although it only offers about 10× speedup with respect to the base line.

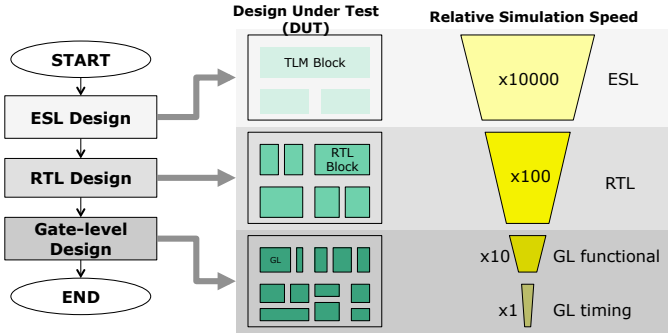


Fig. 2. Typical design implementation flow.

Therefore, any simulation at the lower abstraction level (target simulation) can be performed in a fully parallel fashion without incurring any communication or synchronization overhead, provided that the reference simulation at a higher abstraction level correctly stores initial states for the target simulation slices at a lower level. We should note that the simulation time of the reference simulation run need not to be counted towards the total simulation time if such a simulation is mandatory and is carried out at the higher abstraction level during the design refinement process. That is, if the simulation needs to be performed at a higher level as part of the overall design and verification process, it can serve as a reference simulation for temporal parallel simulation at a lower level without any additional overhead. Therefore, the T_{Rsim} term in equation in (1) can be removed, making it possible for MULTES to provide a very high degree of parallelism.

B. Simulation with Unsynthesizable Testbench

In addition to restoring the state of the DUT (saved at checkpoints during the reference simulation), we also need to restore the "state" of the testbench that corresponds to the saved DUT state. Since the software state of the testbench cannot be captured in registers or memory elements, we need

to run the testbench to the point which provided stimulus corresponding to that DUT state during reference simulation. To accomplish this, the testbench must be simulated (executed) from the simulation time 0 to the corresponding checkpoint. In our implementation it is accomplished by a technique called *testbench forwarding*, a fast, testbench-only simulation to reach the target testbench state. It is implemented as follows: The values of output ports of DUT, saved continuously during the reference simulation, serve as stimulus provider (a *dummy DUT*) for testbench simulation. The testbench is simulated with this stimulus from time 0 up to the starting point of the simulation slice in question. At this point the design state is restored form the data stored at the checkpoint, and the dummy DUT is replaced by the original DUT; each slice is then simulated normally and independently of the other slices.

It should be noted that testbench simulation affects the overall MULTES simulation time; hence the longest time for testbench forwarding for slice, $T_{TB}(i)$, must be added to the simulation time $T_M$in equation (1). Typically, however, testbench overhead is considerably smaller in complex large-scale gate-level designs, since testbenches are written as high-level descriptions with short execution time. Moreover, the size of testbench is not proportional to the DUT size; instead the number of test vectors in the testbench is more related to the complexity of the DUT. Therefore, the longer the testbench and the more complex the design, the lower the contribution of testbench to the overall simulation time is, and the greater performance improvement is achievable with the proposed method.

IV. STATE MATCHING

In order to make temporal-parallel simulation practical, one needs to compute the initial states of individual simulation slices. This process is referred to as *state matching*. Unlike the *horizontal state matching*, encountered in spatially distributed simulation ($c.f.$ Section II), the states to be matched in temporal simulation come from the designs at different abstraction levels. For this reason, one can think of it as *vertical state matching*, defined as follows: given a state in the reference design (in form of a register or a bit vector), find the corresponding *state* (in terms of register values) in the corresponding gate-level representation. In most cases (for designs obtained using standard combinational synthesis, without retiming), the initial states can be readily determined by simple structural analysis. In general, however, the problem is more difficult, since the notion of a state in gate level design is not as clear as in RTL. Finding a one-to-one correspondence between the registers in RTL and gate-level design is difficult.

This section describes a method for computing initial states when a gate level design is synthesized from RTL using sequential retiming and resynthesis techniques. In the following, we assume a certain level of structural similarity between the reference (RTL) and target (GL) design that makes this problem tractable. The method is based on finding register *values* in the target design by propagating the known *signal values* in the reference design, which is significantly simpler than proving sequential equivalence between the two designs.

A. Functional State Matching

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 can be obtained from state S_A through sequential synthesis (possibly including retiming). While the two states are sequentially equivalent, their respective register functions may not be the same, if state S_B was obtained from state S_A using retiming. Note, however, that in our work we are concerned with the *values* of registers corresponding to the two states, rather than with a general state equivalence. For this reason we use the term *state matching* to differentiate it from a state equivalence or a more general (and difficult) register equivalence.

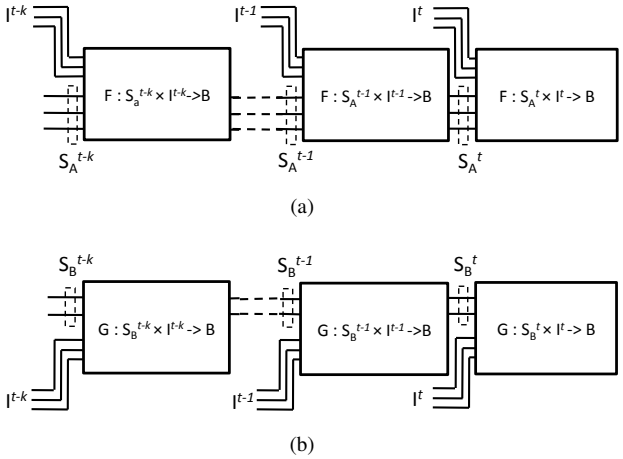


Fig. 3. Time-frame model of two sequential circuits unrolled over a fixed number of clock cycles: (a) Reference time frames; (b) Target time frames.

Fig. 3 represents a time-frame model of a general sequential circuit, with the reference design and target design shown in and Figs 3(a) and 3(b). Assume that the design in part (b) is a retimed and resynthesized version of the design in (a).

Given the original state ${S_A}^t$ at time frame t, the goal is to find a matching target state ${S_B}^t$. In our approach, the value of state ${S_A}^{t-k}$ (for some small constant k) is known from the reference simulation. In this case, values of all the internal signals between ${S_A}^{t-k}$ and ${S_A}^t$ are also known because these values can be computed by performing k cycles of simulation. Therefore, finding internal correspondence between the reference frames and the corresponding target frames allows one to compute target state ${S_B}^t$ by using the values in the reference frames between ${S_A}^{t-k}$ and ${S_A}^t$.

State matching is conceptually similar to the equivalence checking (EC) problem in formal verification. In this section, a combination of several formal techniques, used mainly for sequential equivalence checking (SEC), is applied to solve the problem. In particular, we use sequential equivalence techniques implemented in ABC system [20].

B. Signal Correspondence

To perform state matching, we rely on a sequential equivalence checking technique, called *signal correspondence* (SC)

[20]. It transforms a sequential equivalence problem into a combinational problem by unrolling the sequential circuit over a fixed number of times (k). It computes a set of classes of sequentially equivalent nodes using k-step induction [21], [22] and Boolean SAT. The procedure consists of the following two steps [20].

- *Base Case*: The equivalent classes hold for all inputs in the first k frames starting from the initial state; and
- *Inductive Case*: If the equivalence classes are assumed to be true in the first k frames starting from any state, they hold in the $(k+1)^{th}$ frame.

Initially, the algorithm starts with initial candidate equivalence classes. These initial candidate classes can be selected by performing a short random simulation (typically with 4,000 - 10,000 input vectors). The first node of each equivalence class is set as its *representative*.

For the base case of the induction, the algorithm checks if all the nodes in each candidate class are equivalent to their representative during the first k time frames. We assume that the initial states for both designs are given. If the equivalence classes cannot be proved, the candidate equivalence classes are split, so that each node from a non-equivalent pair is placed in a different class. At the same time, the generated counter-example from the disproval of equivalence is used to refine other equivalence classes by running a simulation with such a counter-example. A new representative node is then set for the split class. This process continues until the candidate classes are completely refined for the first k time frames.

For the inductive case, the candidate classes refined by base case are assumed to be equivalent for any consecutive k time-frames. This condition is established by merging all the nodes in the same candidate equivalent class with their representative node for the k time-frames. The merging procedure, called *speculative reduction* [23], plays an important role in improving the performance of SC because merging significantly simplifies SAT constraints.

Once the speculative reduction is finished, the candidate equivalence classes have to be proved or disproved by *SAT sweeping* [24] for $k+1$ time-frames. This process is similar to proving the base case, except that a symbolic state, instead of constant initial state, is used in the process. The final resulting classes are sequentially proven equivalent classes. Therefore, all the nodes in the same class are sequentially equivalent.

C. State Matching using Signal Correspondence

In order to use the idea of signal correspondence for the state matching problem, the reference design (e.g. RTL) is first transformed into a "canonical" gate-level circuit obtained using simple, trusted synthesis transformations with no (or minimum amount of) optimization involved. After that, both the transformed reference design and target design are converted into AIG (And-Inverter Graph), an efficient gate-level data structure used by ABC.

After merging the two AIG graphs, each unrolled over k cycles, functional simulation is performed on the unrolled AIG to propagate the value of reference registers to the internal nodes. After that, signal correspondence is solved in the merged AIG to find equivalence classes. If any of the proven equivalence classes have nodes with specific values that migrated from the reference registers, their values should be copied to all the nodes in the same equivalence class. Then, k cycles of simulation are performed again to propagate internal values to the target registers. If the propagation reaches all the target registers, the computed register values are used to establish the initial state of the corresponding target slice.

Fig. 4 illustrates the concept of state matching between two designs: the reference *Design 1* in Fig. 4(a), and target *Design 2* in Fig. 4(b). The goal is to find all the register values in *Design 2* at frame t in Fig. 4(c), based on the values of registers in *Design 1*, known from reference simulation. Assume that signals P and Q are found to be sequentially equivalent using the ABC software. The values for $P1$ and $P2$ in *Design 1* are computed from the register values at $t-2$ obtained from the reference simulation. By migrating the values into the corresponding nets ($Q1$ and $Q2$) of *Design 2*, all the register values of *Design 2* at t can be computed. As a result, the state of target design at t is matched with the corresponding state of the reference design.

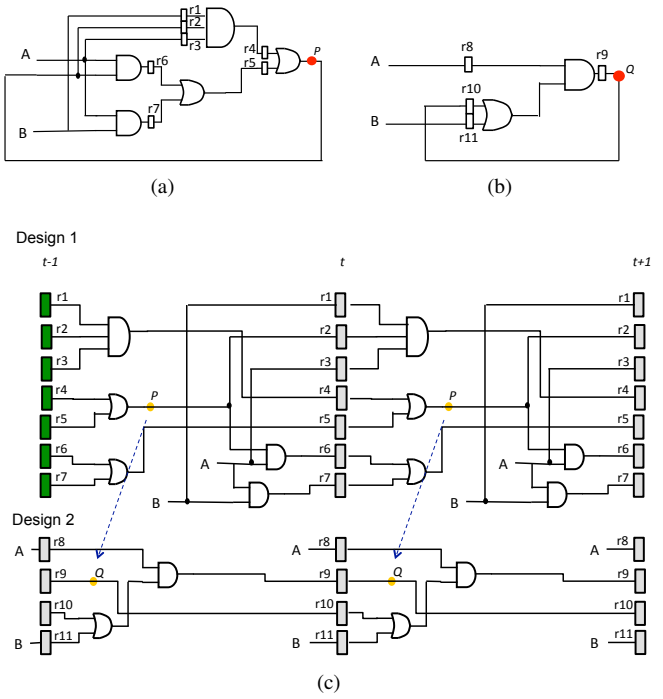


Fig. 4. Signal value migration from the reference design (a) to the retimed and re-synthesized target design (b).

As shown in Fig. 4, computing the target state may require unrolling the design over multiple cycles. In this example, at least two-cycle unrolling is necessary. The necessary amount of unrolling, needed to resolve the state matching, is not known because it depends on the SC values and their number. To address this problem, the state matching procedure can be extended by shifting the target state forward. By shifting the target state by j cycles forward, additional value migration is possible between the $k+1$ and $k+j$ frames. This additional migration increases a chance to compute the target state. This procedure initially tries to compute a target state within the k

time frames. If it fails, the target state is continuously shifted up by 1 cycle to extend the number of time frames to k+1. We illustrate this procedure with an example in Fig. 4. The state at time (t-1) of *Design 2* in Fig.4 is assumed to be the initial target state. Moreover, assume that only one time-frame unrolling was provided for the initial setup. Therefore, only $P2$ and $Q2$ are given as initial signal correspondence, which is insufficient to compute the target state at $(t-1)$. Specifically, values of registers $r9, r10$, shown as blank, cannot be determined. If this condition occurs, the target state is shifted by one time-frame, to frame t. After that, by running one more cycle of reference simulation, the value for $P1$ can be migrated to $Q1$ which allows computation of the new target state at time t. Note that the shifting target state does not cause performance drop if a long term simulation is considered.

D. Constrained Signal Correspondence (CSC)

Even though the procedure described above is able to compute state matching, we do not need a full blown sequential equivalence to solve the state matching problem. For the purpose of our work, we just need to find $values$ of the register signals in the target implementation based on the $value$ of the corresponding register in the reference (RTL) representation, rather than solving a complete functional equivalence problem. Since the value of reference registers is already known from reference simulation, it can be used as a constant invariant to simplify SAT constraints. We therefore modified the state matching procedure by using reference values to compute signal correspondence on the fly. As shown in Section VI, the modified approach dramatically improves the performance of state matching.

The modified state matching procedure is illustrated with an example in Fig. 5. In Fig. 5(a), the nodes $n1$ and $n2$ are nodes in the target design, and node $n3$ is a node from the reference design. At this point, both the target and the reference designs are already combined through a miter. Let nodes $n1$, $n2$ and $n3$ be proved equivalent by solving SAT. Therefore, during SAT sweeping $n2$ and $n3$ are merged into their representative node $n1$, as shown in Fig. 5(b). The merging process also migrates the constant value v from $n3$ to $n1$. In Fig. 5(c) the representative node $n1$ is replaced by v, inherited from the reference node $n3$. As a result, the entire sub-graph below node $n1$ is removed, and the procedure continues by merging the next set of nodes, $n4$ and $n5$. If $n4$ and $n5$ are proved to be equivalent (unSAT), they are considered to form an SC class. This, however, is a restrictive case of SC because it is based on the constant value v migrated from $n3$ to $n1$. We refer to this type of SC as *Constrained Signal Correspondence (CSC)*, since the nodes are equivalent under the restrictive condition $n1 = n2 = n3 = v$.

The main difference between this approach and the original state matching procedure is the time at which the reference values are applied in the procedure. In the original approach, the reference value is used after computing the signal correspondence, but in the new algorithm it is used while performing SAT sweeping. Solving SAT sweeping using CSC is much faster than with SC because all the constraints

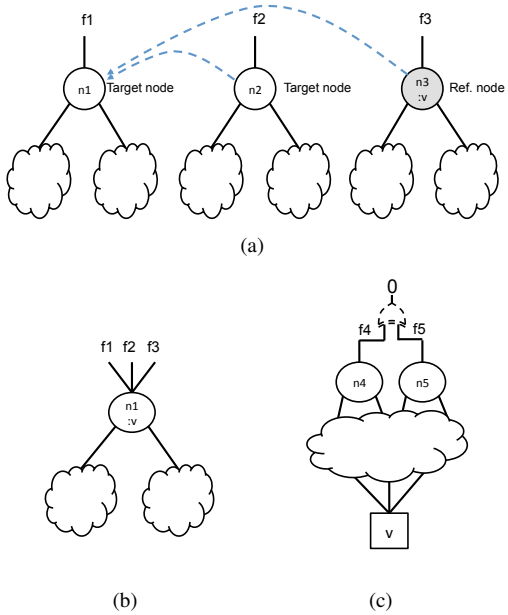


Fig. 5. SAT sweeping with constrained signal correspondence(CSC): (a) Equivalent signals $f1 = f2 = f3$; (b) Merging equivalent signals into their representative; (c) Finding CSC by replacing n1 with a constant v.

below the nodes (e.g., $n1$ in Fig. 5), which are replaced with reference constants, are removed. Moreover, the replaced constants can be propagated through the down-stream logic providing additional simplification.

E. State Matching with CSC

To implement state matching using CSC, we modified the original SC algorithm of ABC [20]. Fig. 6 shows the CSC-based state matching procedure, with the modified parts shown in bold. Our procedure starts by performing simulation of the reference design for k time frames of reference design. The initial state for the simulation comes from the reference design, so that the values of all the internal signals within the k time frames are known. Then, a miter (XOR) is added to the two designs to enable finding SC and CSC.

Generating accurate initial candidate equivalence classes dramatically increases the performance of the induction algorithm. In general, the classes are generated by a short period of random simulation. We use bounded model checking and speculative reduction implemented in ABC to establish the base case and the inductive case, respectively.

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

The procedure terminates when all the register values of the target frame are successfully computed by performing simulation from the migrated values at SC and CSC.

```
aig runStateMatching( aig ref, aig impl, r_state, int k, int kmax )
{
  //Find all signal values of reference design in k time frames
  Vref = performSimulation(ref, r_state, k)
  aig N = addMiter(ref,impl);
  set of node subsets Classes = randomSimulation( N );
  // refine equivalences by BMC from the initial state for depth k − 1
  // (this corresponds to the base case)
  refineClassesUsingBMC( N, k − 1, Classes );
  // perform iterative refinement of candidate equivalence classes
  // (this corresponds to the inductive case)
  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 in k-th frame
after merging
    SatSweepingIntoConstantWithConstraints(S, Classes, Vref);
    //try to compute target 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. 6. The procedure of state matching using CSC.

F. Assuring Correctness of MULTES Results

Even though the state matching technique uses formal methods based on CSC, there is no guarantee that target states can always be computed. This is because the number of extracted SCs may not be sufficient to reach the target state. As a result, computing target states may still fail. To address this issue, we developed the following failure-tolerant procedure. If computing the initial state for a particular simulation slice fails, the last slice having a successfully matched state is used to continue simulation beyond its slice boundary. This is shown in Fig. 7, where $S1_A{}^{-k}$ and $S2_A{}^{-k}$ of reference simulation are used to compute the initial states $S1_B$ and $S2_B$, for slice n and $n+1$, respectively. If the initial state $S2_B$ of slice $n+1$ fails to be computed, slice $n+1$ cannot be simulated in parallel because its initial state is not known. In order to achieve a fully continuous simulation, simulation that starts at state $S1_B$ must override slice $n+1$. This approach prevents simulation period with unknown initial state, at a cost of lengthening slice n to cover slice $n+1$.

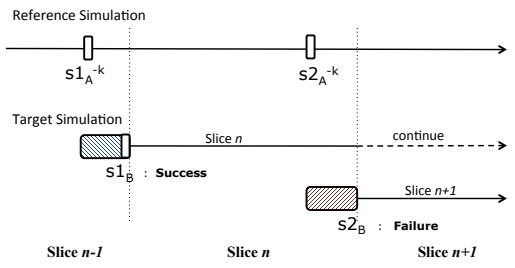


Fig. 7. Failure-tolerant initial state assignment in MULTES.

V. GATE-LEVEL TIMING SIMULATION

Simulation of timing-annotated gate-level designs is known to suffer from very low performance. In this section we discuss several issues related to timing simulation and propose solutions to improving simulation performance dramatically. To understand the issues related to consistency between the functional reference simulation and timing target simulation, we need to define the term *cycle-consistency*. Specifically, if simulation of two designs produces identical results within the required number of clock cycles, the two designs are considered *cycle-consistent*. With this definition in mind, we can now make the following observation:

Observation 1. *If a synchronous single-clock circuit does not have timing violation, the functional and timing simulations of the circuit must be cycle-consistent.*

A. Initial State Mismatches

Recall that in MULTES the design state is saved during the functional reference simulation (using RTL model) and restored for the target timing simulation (in the GL model). Depending on a position of the checkpoint, this may cause timing discrepancies to appear at the beginning of the target slice. This situation is illustrated in Fig. 8. In this example, q represents an output of a memory element.

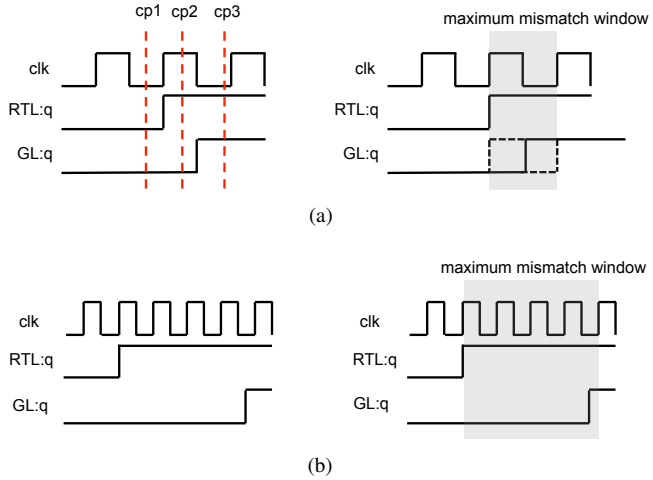


Fig. 8. Example of initial state mismatches caused by gate-level delay: (a) Single-cycle path case; (b) Multi-cycle path case.

In Fig. 8(a) there are three possible places to make a checkpoint: $cp1$, $cp2$ and $cp3$. It is safe to make a checkpoint at either $cp1$ or $cp3$ because at these points both reference and target simulations have identical value of q. However, if the checkpoint is made at $cp2$, value 1 reported by the reference simulation does not match value 0 obtained by gate-level timing simulation of the target slice.

An intuitive solution to this problem is to allow checkpointing only at the "safe" points ($cp1$ and $cp3$ in Fig. 8(a)). However, finding such points is difficult, as it requires a sophisticated delay analysis. Moreover, for a design with multiple-clocks, such points may not even exist.

To address this issue, we make use of Observation 1 (Cycle Consistency). Since the RTL and GL simulations must maintain cycle consistency, the mismatch cannot propagate to the next cycle. The maximum possible mismatch period, t_{mis}, caused by the delay, can be computed as follows:

$$t_{mis} = \gamma t_C; \qquad \gamma t_C > t_{MAX\_Delay} \tag{2}$$

where t_C is one clock cycle period, t_{MAX_Delay} is the maximum propagation delay in the target design, and γ is the minimum positive integer that satisfies Eq. (2).

This problem can be solved by providing an overlap between two consecutive slices for the $t_{mismatch}$ period as shown in Fig. 9. Here slice $n-1$ and n are allowed to share the simulation during the $t_{mismatch}$ period from the last checkpoint. Since mismatch occurs at the beginning of slice n, the corresponding period is discarded from the slice simulation, and correct simulation result for that period is obtained from slice $n-1$.

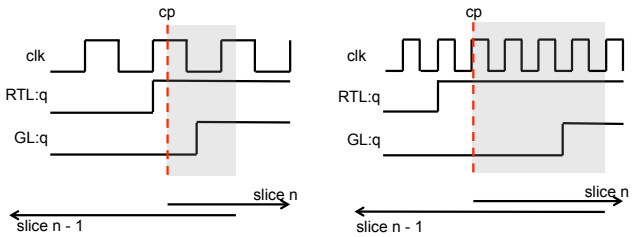


Fig. 9. (a) Slice overlap in (a) single-cycle path; and (b) multi-cycle path.

As studied earlier (Observation 1), cycle consistency between the reference and target simulation guarantees the correct initial state for each slice. Therefore, we need to adjust the reference simulation making it cycle consistent with respect to the target simulation. In this case, any mismatches between reference and target simulation, or between two consecutive target slices indicate a potential design bug.

The cycle consistency described here is naturally maintained in synchronous circuits with a single clock. However, if a design has multiple clocks, asynchronous with respect to each other, special care is required to provide the cycle consistency, as discussed in Section V-C.

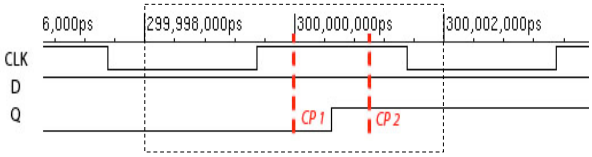
B. Checkpointing

In cycle-based simulation, the checkpoints can be assigned at the end of any cycle period without causing any discrepancy between the reference and target simulation. In an event-driven simulation, however, finding correct placement for checkpoints is more difficult because of non-deterministic delay between the event edges.

An example in Fig. 10 illustrates this situation. Let us consider a fragment of Verilog code in Fig. 10(a), which is a part of the reference RTL model used in MULTES. The #1 (one unit) delay at the right hand side of the non-blocking assignments models the clock-to-Q delay of the corresponding flip-flops. There are several reasons that designers use such delays in their Verilog code, e.g., for debugging convenience,

```
always @(posedge clk or posedge rst_n) begin
        if(!rst_n) q <= #1 0;
        else q<= #1 d;
end
```

(a)



(b)

Fig. 10. (a) RTL code for a flip-flop modeling clock-to-Q delay; and (b) Simulation waveform for the code with a checkpoint window.

for mixed RTL/gate-level simulation, etc. [25]. Fig. 10(b) shows a waveform from an actual simulation of the code. If the checkpoint is made at $CP2$ (300,001 nsec of reference simulation time), the *correct* value, 1, is saved and restored later for target simulation. This represents a correct behavior. However, if the checkpoint is made at $CP1$ (300,000 nsec), an incorrect value, 0, is saved instead. This wrong value is then restored at the corresponding flipflop in the target simulation, providing a wrong starting point for the target simulation. The problem becomes even more complicated for designs with multiple asynchronous clocks, in which the correct checkpointing for each clock domain is possible only at safe simulation times. In this case, we must find a safe simulation time common to all clock domains at each checkpoint, prior to the reference simulation.

One possible solution is to ignore any delays that appear in the high-level abstraction of the design used for reference simulation. But this may also result in an incorrect checkpointing, especially in designs with multiple asynchronous clocks. To address this checkpointing issue, we introduce the concept of a *checkpoint window*, an interval dedicated to saving and restoring design state. The size of the checkpoint window is one clock-cycle period in this example, but if Q belongs to a multi-cycle path, the window needs to be expanded accordingly. The dotted box in the middle of Fig. 10 represents one possible checkpoint window for this design.

C. Multiple Asynchronous Clocks

Contrary to a popular belief, gate level simulation for multiple-clock design may not maintain 100% cycle consistency with the RTL simulation, even if there is no timing violation. Therefore, a simple state saving and restoring may cause incorrect target simulation. Fig. 11 illustrates such a condition with a typical two-phase handshaking logic.

In Fig. 11(a), two synchronizers are used for Req and Ack signals. No synchronizer is used for $Data$ because the signal values in data bus are maintained at the same level for a sufficiently long time so that the receiving flip-flop will always sample stable values. Fig. 11(b) shows timing inconsistency between the RTL simulation and gate level timing simulation. In this case, flip-flop $Sync1$ in gate level simulation cannot

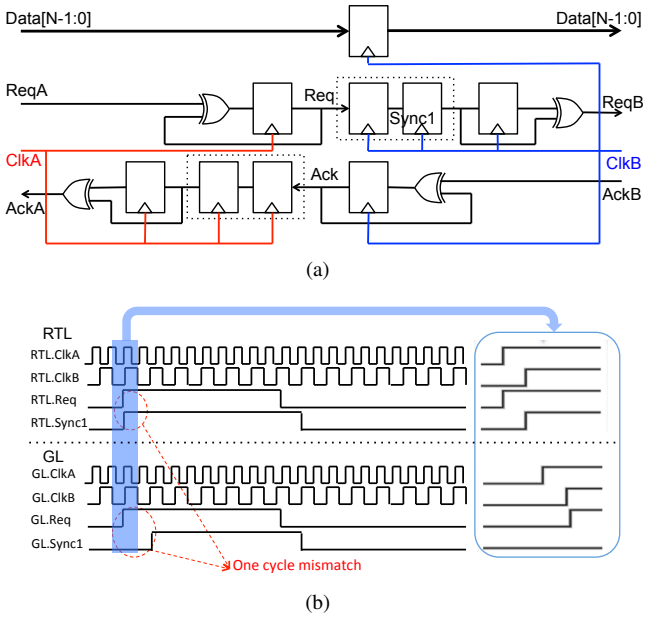


Fig. 11. Two-phase handshaking logic (a) and the simulation results (b).

sample value 1 on Req. Instead, its value is sampled in the next cycle in RTL simulation, because the delay of Req makes the value change from 0 to 1 to occur after the rising edge of $ClkB$. As a result, the sampling signal value of Req in gate-level design is delayed by one cycle. In conclusion, gate level timing simulation may not be 100% cycle consistent with RTL simulation, even if there is no timing violation. This inconsistency may cause our approach to produce, in general, different simulation results from the conventional gate level timing simulation. This results in timing mismatches between the reference and target simulation.

In order to handle the multiple-clock issue, the following method based on *abstract delay annotation* is proposed. Fig. 12(a) explains the main idea of this approach.

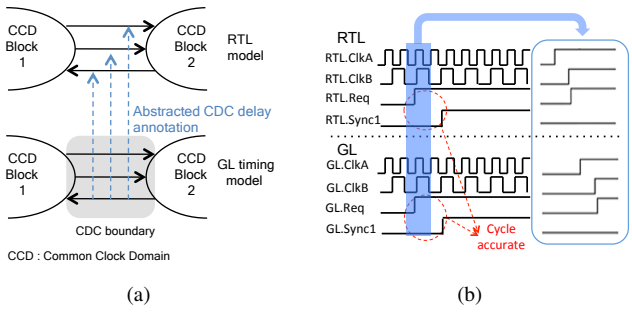


Fig. 12. Abstract delay annotation concept (a) and the result of the technique (b)

The method takes advantage of the delay information for CDC paths available from the Standard Delay Format (SDF) file. The propagation delay at the CDC boundary is extracted from SDF file and annotated (after the necessary simplification and normalization) onto the RTL model. The annotated abstract delay denoted $D_{ADA}(CDC)$ is a function of the propagation delay at the CDC boundary and the delay of two asynchronous clocks[1], as given by Eq. (3).

$$D_{ADA}(CDC) = D(Clk_{send}) - D(Clk_{recv}) + D(CDC) \quad (3)$$

where:

- $D_{ADA}(CDC)$ is the abstract delay for CDC path to be annotated in RTL design.
- $D(Clk_{send})$ is the delay of clock for the register in upstream clock domain obtained from SDF file.
- $D(Clk_{recv})$ is the delay of clock for the register in downstream clock domain obtained from SDF file.
- $D(CDC)$ is the total delay in a CDC path obtained from SDF file.

Fig. 12(b) shows that $RTL.Req$ signal is properly delayed after applying abstract delay annotation described above. After annotating abstract delay on the RTL design, the RTL reference simulation and gate level target simulation should maintain cycle consistency even in multiple-clock designs. By doing this, our approach produces identical simulation results as conventional simulator, unless there is a timing violation in the DUT.

In our work, we used Conformal CDC checker (Cadence) to identify CDC paths and implemented a tool to extract the corresponding delay information from those paths. We then applied Eq. (3) to modify the delay on CDC paths (and only on those paths). A new SDF file with the modified CDC delay information was then generated, and the gate-level design segments corresponding to the modified SDF file were applied to the RTL design. Due to some instability of our Verilog parser, this RTL-to-GL merging part was done manually, using simple Cross-Module Reference in Verilog.

The difficulty of state matching is one of the limiting factors in applying the temporal approach to RTL simulation, as this would require using ESL as reference. Establishing the correlation between ESL transactions and RTL cycles may not be possible. Also, it is not clear how to guarantee cycle consistency between two abstraction models, required for correct functioning of our technique.

VI. EXPERIMENTAL RESULTS

The temporal parallel simulator, MULTES, has been implemented in C/C++ as a plug-in to Cadence NC-Sim® simulator using PLI, and its performance compared to conventional simulation using NC-Sim.

The system is largely automatic, with only a few manual steps mandated by the use of third party software. Design is parsed using PLI of the NC-Sim simulator and CDC paths are extracted using Cadence Conformal tool. The system generates script to access and run the tool; the setup is straightforward and detecting CDC paths takes a few minutes even for large scale designs. Abstract delay annotation is performed automatically and in a matter of second(s) even for very large designs. Reference simulation of the annotated RTL design is fully automatic, with user providing basic parameters, such as the number of simulation slices, timescale, etc. Saving the design state and dumping stimulus (for testbench forwarding) are done automatically based on user configuration.

[1]Clock delay is defined as the propagation delay in clock path from clock source to destination register.

State matching requires some manual interaction and setup, with user identifying those parts of the design that required retiming. For those modules that did not involve retiming, MULTES automatically matches signals in the reference and target designs. If matching fails (which can happen when names in the GL design were changed during synthesis, or the GL design hierarchy is different than that in RTL), the system invokes Formality tool of Synopsys to complete the matching, using an automatically generated TCL script. This is a one-time task that typically takes several minutes even for a very large design. If the module under consideration involves retiming, the matching is solved using ABC. Solving the signal correspondence task is automatic, based on a fixed script. Some amount of human effort is required to guide the matching process, mainly because of the limitation of resources in developing academic tools. Finally, target simulation is completely automatic.

The simulation experiments were performed on a computer equipped with Intel® T7500 CPU. Target designs were synthesized by Synopsys Design Compiler® with TSMC 65nm technology library. The SDF files with delay information for gate-level design were generated by Design Compiler® (Synopsys). To measure the performance of target simulation, each simulation slice was run sequentially on a single computer, emulating a parallel simulation environment in which each slice is assigned to a separate physical machine. The run times reported in this section include all elements of MULTES, except for state matching, done with a commercial tool, when applicable. The state matching results are reported separately.

A. Zero-delay Gate-level Reference Simulation

The first set of experiments involved gate-level full timing simulation with *zero-delay* gate-level simulation as reference simulation. In this case, the netlist structures for the reference and target models are identical and hence state matching is not required.

1) S1-Core: In this experiment, an S1-Core design from OpenCORE [26] was used as benchmark. The design has 1.2 million gates and contains one 64 bit-SPARC Core.

	Full-timing simulation	Zero-delay simulation
Simulation Cycle	500,000 cycles	500,000 cycles
Simulation Time	11,860 s	223 s
Simulation Rate	42.16 cycles/sec	2242.15 cycles/sec

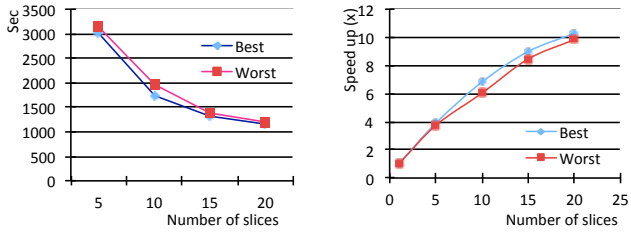
TABLE I
SIMULATION SPEED COMPARISON BETWEEN FULL-TIMING AND ZERO-DELAY SIMULATION.

Table I shows the speed of the traditional, single-processor simulation and the zero-delay simulation used as reference simulation. In this case, zero-delay simulation is 53 times faster than full timing simulation.

The results of the parallel simulation with MULTES are shown in Table II (all times are in seconds). The best simulation performance is attributed to the first slice: the initial condition for the first slice does not require the design state to be restored and the testbench does not need to be simulated. From the second slice on, the simulation is slower because of the testbench forwarding. However, the plots in Fig. 13 show that the overhead due to testbench forwarding is small, and the last slice is simulated almost as fast as the first one. Furthermore, the simulation overhead due to testbench forwarding is not a function of the number of slices but a function of the simulation time. Therefore, for gate-level simulation requiring reasonably long simulation runs, the total simulation overhead of temporal parallel simulation is maintained at a low level, regardless of the number of slices.

Slice number	Ref. Simul. (zero-delay) [sec]	Target slice simulation (timing) [sec]			Total simulation time (min, max) [sec]
		1st	2nd	Last	
5	361	2673	2756	2798	3034 - 3159
10	391	1342	1554	1567	1733 - 1958
15	429	887	951	969	1316 - 1398
20	457	648	732	745	1105 - 1202

TABLE II
PERFORMANCE OF GATE-LEVEL SIMULATION WITH 20 SIMULATION SLICES FOR S1-CORE DESIGN WITH 1.2M GATES.



(a) (b)

Fig. 13. Performance of MULTES for S1_Core design: (a) Simulation time in MULTES; (b) Speedup in MULTES.

2) 18M-gate Industrial Design: This experiment was carried out with the help of designers from a major semiconductor company on an industrial design with 18M gates. Mega-cells (such as RAM cache, etc.) are replaced with higher level simulation models. The design was simulated with Cadence NC-Verilog® simulators, running on SUN SPARC machines. Despite a small (10:1) speedup ratio between the zero-delay simulation and full-timing gate level simulation, the results showed an expected linear (6x) speedup with 10 simulators on a simulation run of 72,600,000 cycles.

B. RTL Reference Simulation

In these experiments, RTL design was used as the reference model for gate-level timing simulation. Formality® equivalence checker (Synopsys) was used to establish register matching between the RTL and gate-level design. The tool solves registers matching problem as part of equivalence checking, by detecting structural similarity between the two designs. This process is very fast even for multi-million gate designs.

1) JPEG Encoder: In this experiment, we used JPEG Encoder design from OpenCores [26]. Total gate count of the

GL design is 900k gates. Table III shows the performance of MULTES for this design. RTL reference simulation of JPEG Encoder is 256 times faster than conventional GL timing simulation. Under this condition, we were able to achieve speedup ranging from 6.35 to 110.78 times, depending on the number of simulation slices. The worst-case target simulation refers to the simulation of the last slice, as it includes the longest testbench forwarding period. Computation of the speedup rate was based on the worst-case simulation. As shown in Fig. 14(a), MULTES gives a linear speedup for the first 100 slices with a linear rate of 0.4, and continues at a lower rate up to 500 slices. Beyond this point, the improvement tends to saturate but is still noticeable at 1000 slices. This level of performance improvement is generally not possible with a conventional distributed parallel simulation. We expect that a longer total simulation period with the same number of slices will further delay the saturation point. This is because the reference simulation and testbench forwarding become dominating factors for target simulation simulation with 100 slices and beyond, as shown in Fig 14(b).

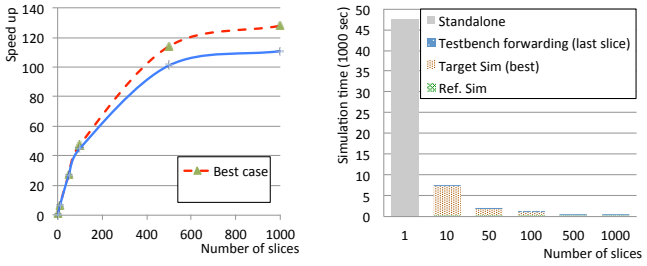
(a)

Design	Simulation run time (sec)	Ratio
RTL	184	1
GL timing	47192	× 256

(b)

# of slices	**10**		**50**		**100**		**500**		**1000**	
Reference Sim (sec)	246		249		255		269		291	
Best/Worst	**B**	**W**	**B**	**W**	**B**	**W**	**B**	**W**	**B**	**W**
Target Sim (sec)	7142	7191	1456	1508	737	791	145	198	78	135
Total (sec)	7388	7437	1705	1757	992	1046	414	467	369	426
Speed up	6.38	6.35	27.7	26.9	47.6	45.12	114	101.1	127.9	110.8
TB f/w (sec)			56		**State saving (sec)**				<0.5	

TABLE III
EXPERIMENTAL RESULTS OF MULTES FOR JPEG ENCODER: (A) PERFORMANCE GAP BETWEEN RTL AND GL TIMING SIMULATION; (B) SIMULATION PERFORMANCE OF MULTES.



(a) (b)

Fig. 14. Performance of MULTES for JPEG encoder: (a) Speedup in MULTES; (b) Simulation time in MULTES.

2) AES: Advanced Encryption Standard (AES) design from OpenCORES [26], is used as another benchmark for the experiment using RTL model for reference simulation. Total gate count of AES design is 25K.

(a)

Design	Simulation run time (sec)	Ratio
RTL	110	1
GL timing	18669	× 169

(b)

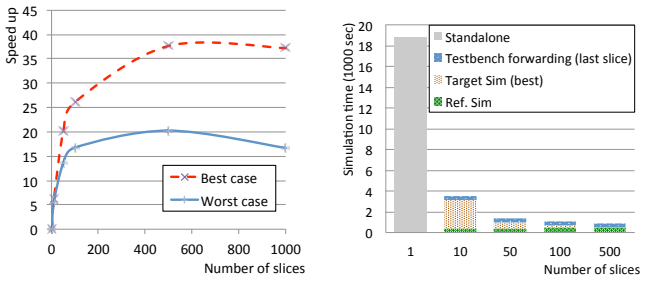
# of slices	**10**		**50**		**100**		**500**		**1000**	
Reference Sim (sec)	426		431		442		453		467	
Best/Worst	**B**	**W**	**B**	**W**	**B**	**W**	**B**	**W**	**B**	**W**
Target Sim (sec)	2634	2916	498	940	233	670	43	469	35	652
Total (sec)	3060	3342	929	1371	675	1112	496	922	502	1119
Speed up	6.1	5.59	20.1	13.6	27.7	16.8	37.6	20.3	37.2	16.7
TB f/w (sec)	**w/ I/O**		424							
	w/o I/O		81							
State saving (sec)	<0.5									

TABLE IV
EXPERIMENTAL RESULTS OF MULTES FOR AES: (A) PERFORMANCE GAP BETWEEN RTL AND GL TIMING SIMULATION; (B) SIMULATION PERFORMANCE OF MULTES.

Table IV shows that gate level timing simulation of this design is 169 times slower than RTL simulation. In this case, the speedup ranges from 5.59 to 20.25 times. Fig. 15 shows a sub-linear speedup up to 70 slices, and the performance improvement continues up to 500 slices. Considering a large gap between the RTL and GL simulation speed, the overall speedup is lower than one would expect. This is because the speed gap between RTL and GL simulation is small, due to the small size and low complexity of the DUT. As a result, testbench forwarding overhead becomes relatively high. Fig. 15(a) shows that such factors dominate the entire overhead for the simulation with 50 slices and the overall performance of MULTES eventually drops after 600 slices. The effect of the overhead imposed by reference simulation and testbench forwarding is shown in Fig. 15(b). The optimum number of parallel nodes during this simulation period is 50. Table IV(b) also shows that reducing disk I/O overhead, which can be obtained by compressing data and using faster storage devices, will contribute to better results. These two experiments demonstrate that MULTES offers higher performance improvement for designs having complex simulation data structure, longer simulation period, and a large amount of event activities. Therefore, we anticipate that our approach will have significant impact on dynamic verification of large-scale designs.

C. Performance of State Matching

The experiments in this section show the performance of state matching, discussed in Section IV, for AES design. In



(a) (b)

Fig. 15. Performance of MULTES for AES design: (a) Speedup in MULTES; (b) Simulation time in MULTES.

this experiment, the target gate-level design was obtained by resynthesizing the reference AES design using the "forward retiming with minimum delay" and "refactor" synthesis options in ABC tool.

Random reference state		Matching with SC (sec)	Matching with SC&CSC (sec)
1	retime	1.62	0.57
	retime&refactor	3.31	0.79
2	retime	1.52	0.51
	retime&refactor	3.29	0.73
2	retime	1.54	0.49
	retime&refactor	3.33	0.81

TABLE V
STATE MATCHING FOR AES DESIGN.

Table V shows the result of state matching. In this case, the number of time frames for the induction is fixed at 4. For this experiment, three different random reference states were first generated, and for each reference state, two different state matching approaches were employed. The first approach was to find all the possible SC (signal correspondence) classes first, and then to propagate reference state values to the target registers. These are shown in the second column of Table V. The second approach was to use CSC (constrained signal correspondence) and to propagate reference state values on the fly during SAT sweeping. These are listed in the third column of Table V. The table clearly demonstrates that using CSC significantly improves the state matching speed. Specifically, we observed that matching with both SC and CSC required much smaller running time than matching with SC only.

We should emphasize that solving the state matching problem, as described here, is simpler than a complete sequential equivalence problem. Recall that it is not our goal to prove that certain signals in the reference and target designs are sequentially equivalent; instead, we only need to find *register values* in the target design, based on *known signal values* in the reference design. In general, finding such a mapping is very fast for designs without retiming, using simple structural analysis employed by commercial combinational equivalence checkers. It is also fast for most of the tested designs, as shown in the above table.

VII. CONCLUSIONS

The time-parallel technique described in this paper is scalable and applicable to timing simulation of arbitrary, gate-level hardware descriptions. It naturally fits in a standard design synthesis flow, in which the design undergoes a progressive transformation from higher level abstraction to the lowest, gate-level implementation. By relying on a higher abstraction level as a reference model, this technique can automatically determine whether the simulated design is consistent with the model at the higher, reference model. Furthermore, the method handles designs with multi-cycle paths, multiple asynchronous clocks, and circuits synthesized with retiming, without sacrificing important verification features.

The question is how reliable is this technique for simulation of large industrial designs. Reliability of MULTES is dictated mostly by its ability to solve the matching problem. However, as mentioned in Section IV F, MULTES performs correct simulation by sacrificing performance even if the state matching failed. Except for retimed designs, for which the system may not be able to find state matching, the method can be considered reliable. Specifically, the system dynamically checks during target simulation if the computed state at the end of slice k matches the initial state used for slice k+1. In general, if the implementation is correct (i. e., if there is no functional difference between the reference and target designs) the simulation result must be correct. For a single-cycle design, a mismatch between the reference and GL design at the clock boundary indicates a bug. For a design with multiple asynchronous clocks, the mismatch requires more careful analysis, as it may indicate a design bug or an incorrectly annotated abstracted delay on CDC paths. If the abstracted delay on CDC paths is correctly annotated, any mismatches indicate that there is timing bug in the design. Even though the validation of a potential bug is user's responsibility, this feature simplifies the debugging process.

We should emphasize that our time-parallel simulation method is suitable for distributed computing environment, such as simulation farms, rather than for multi-core processor architecture. The main reason is large memory required by each processor to simulate the entire design (albeit over a shorter simulation slice than the single-thread simulation). This is in contrast to traditional distributed parallel simulation, where due to design partitioning each processor can benefit from the smaller size of the blocks. In the time-based approach, simulation of each slice requires a full simulation image, as in the single-processor simulation.

The method described in this work is scalable and applicable to gate level simulation of a large class of designs. To the best of our knowledge, no such temporal-parallel simulation have been successfully implemented or reported, in particular for designs synthesized with sequential optimization techniques, such as retiming.

ACKNOWLEDGMENT

This work has been supported by the National Science Foundation, award CCF-0702506 and CCF-1017530, and by a research grant from Samsung Electronics Co., Ltd.

REFERENCES

[1] W. Lam, *Hardware Design Verification: Simulation and Formal Method Based Approaches*. Prentice Hall PTR, 2005.

[2] "Avery Desgin Automation, SimCluster datasheet (http://www.avery-design.com)."

[3] "Axiom Design Automation (http://www.axiom-da.com)."

[4] P. Heidelberger and H. Stone, "Parallel trace-driven cache simulation by time partitioning." in *Proc. Winter Simulation Conference (WSC)*, 1990, pp. 734–737.

[5] D. Kim, M. Ciesielski, K. Shim, and S. Yang, "Temporal parallel simulation: A fast gate-level hdl simulation using higher level models." in *Proc. Design Automation and Test in Europe (DATE)*, 2011, pp. 1584–1589.

[6] M. Chandy and J. Misra, "Distributed simulation: A case study in design and verification of distributed programs." *IEEE Transactions on Software Engineering*, no. SE-5(5), pp. 440–452, 1977.

[7] R. Bryant, "Simulation of packet communication architecture computer systems." *Computer Science Laboratory. Cambridge, Massachusetts, Massachusetts Institute of Technology*, 1977.

[8] R. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, October 1990.

[9] D. Jefferson, "Virtual time." *ACM Transactions on Programming Languages and Systems*, no. 7(3), pp. 404–425, 1977.

[10] N. Manjikian and W. Loucks, "High performance parallel logic simulation on a network of workstations." in *Proc. Workshop on Parallel and Distributed Simulation (PADS)*, 1993, pp. 76–84.

[11] R. Bagrodia, Y. Chen, Y. Jha, and N. Sonpar, "Parallel gate-level circuit simulation on shared memory architectures." in *Proc. Computer Aided Design of High Performance Network Wireless Networked Systems*, 1995, pp. 170–174.

[12] D. Lungeanu and C. Shi, "Parallel and distributed VHDL simulation." in *Proc. Design Automation and Test in Europe (DATE)*, March 2000, pp. 658–662.

[13] L. Li, H. Huang, and C. Tropper, "DVS: An object-oriented framework for distributed verilog simulation." in *Proc. Workshop on Parallel and Distributed Simulation (PADS)*, 2003.

[14] L. Zhu, G. Chen, B. Szymanski, C. Tropper, and T. Zhang, "Parallel logic simulation of million-gate VLSI circuits," in *Proc. 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Sep. 2005, pp. 521–524.

[15] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-driven gate-level simulation with GP-GPUs." in *Proc. Automation Conference (DAC09)*, 2009, pp. 557–562.

[16] Y. Zhu, B. Wang, and Y. Deng, "Massively parallel logic simulation with GPUs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 3, p. 29, Jun. 2011.

[17] D. Yun, S. Kim, and S. Ha, "A parallel simulation technique for multicore embedded systems and its performance analysis." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 121–131, 2012.

[18] S. Andradottir and T. Ott, "Time-segmentation parallel simulation of networks of queues with loss or communication blocking." *ACM Transactions on Modeling and Computer Simulation*, no. 5(4), pp. 269–305, 1995.

[19] A. Greenberg, B. Lubachevsky, and I. Mitrani, "Algorithms for unboundedly parallel simulations," *ACM Transactions on Computer Systems*, vol. 9, pp. 201–221, Aug. 1991.

[20] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis." in *Proc. The International Conference on Computer-Aided Design (ICCAD)*, 2008, pp. 234 – 241.

[21] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs." in *Proc. Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, 1999, pp. 193–207.

[22] M. Sheeran, S. Singh, and G. Stalmarck, "Checking safety properties using induction and a SAT solver." in *Proc. Formal Methods in Computer-Aided Design(FMCAD)*, 2000, pp. 108–125.

[23] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, "Exploiting suspected redundancy without proving it." in *Proc. Design Automation Conference*, 2005, pp. 463–466.

[24] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking." in *Proc. The International Conference on Computer-Aided Design (ICCAD)*, 2006, pp. 836–843.

[25] C. Cummings, "Verilog nonblocking assignments with delays, myths and mysteries." in *Proc. Synopsys User Group Meeting (SNUG)*, 2002.

[26] "OpenCORES (www.opencores.org)."

Dusung Kim (M'12) received the B.S. and M.S. degree in computer engineering from Pusan National University, Busan, Korea, in 2004 and 2006 respectively, and the Ph.D. degree from University of Massachusetts (UMass), Amherst, in 2012. Between 2006 and 2012, he was a Research Assistant with the VLSI CAD Laboratory, Department of Electrical and Computer Engineering, UMass. His doctoral work focused on parallel simulation using multi-level design abstraction model. He is currently a senior research and development engineer with Synopsys, Inc., Mountain View, CA, developing static timing analysis techniques for transistor level design.

Maciej Ciesielski (SM95) is Professor in the Department of Electrical & Computer Engineering (ECE) at the University of Massachusetts, Amherst. He received M.S. in Electrical Engineering from Warsaw Technical University, Poland, in 1974 and Ph.D. in Electrical Engineering from the University of Rochester, N.Y. in 1983. From 1983 to 1986 he worked at GTE Laboratories on a silicon compilation project. He joined the University of Massachusetts, Amherst in 1987, where he teaches and conducts research in the area of electronic design automation, and specifically in synthesis, optimization and verification of digital systems. He is recipient of Doctorate Honoris Causa from the Universite de Bretagne Sud, Lorient, France. Dr. Ciesielski is the recipient of the Doctorate Honoris Causa from the Universit de Bretagne Sud, Lorient, France, for his contributions to the field of electronic design automation.

Seiyang Yang received the B.S. degree, and M.S. degree in electronic engineering from Korea University, Korea in 1981 and 1985 respectively, and the Ph.D. degree in computer engineering from University of Massachusetts, Amherst in 1990. From 1990 to 1991, he was a R&D staff with Microelectronics Center of North Carolina. In 1991, he joined the Pusan National University, Korea, where he is a Professor of the Department of Computer Engineering. In 1997, he founded a start-up, which specializes in FPGA-based debugging solutions. His research interest covers parallel HDL simulation, logic synthesis, and many aspects of design verification.