Scalable Asynchronous Hardware Protocol Verification for Compositions with Relative Timing

Krishnaji Desai Kenneth S. Stevens
University of Utah
Electrical and Computer Engineering Department

ABSTRACT

Correct interaction of asynchronous hardware protocols requires verification. Performance and power of asynchronous hardware circuits and protocols can be vastly improved by modifying them with judicious application of timing constraints. A methodology is presented for verifying larger asynchronous protocols through compositional model checking with symbolic methods. This approach uses Relative timing constraints for modeling timed asynchronous hardware protocols and hence is generic for timed and untimed protocols. This paper illustrates the modeling technique for interleaving and simultaneous models with relative timing, which helps in pruning state space. Properties which are verified for ensuring correctness of composed models are explained. Case studies of a linear pipeline controller and C-Element are discussed by applying BDD and SAT methods. The simultaneous model yields better execution times when compared to interleaving for large number of compositions. In the simultaneous model, SAT methods tend to find a counter example in lesser time than BDD methods. Automatic model and property generation, leading to push button solution will enable verification of larger control planes.

1. INTRODUCTION

Design and verification of asynchronous hardware protocols is a challenge which must be formally addressed. This verification is required for the correct design of individual protocols implemented with logic gates, as well as the composition of protocols and controllers into a system.

There are primarily two methods of verification. Firstly, conformance checking [5], where an implementation is checked for conformance against a specification. This approach generally works well for leaf level components. A specification for these blocks is usually readily available because it was used for design and synthesis. Secondly, model checking can be employed to verify deadlock, liveness, safety and other required correctness properties. This approach generally works well for protocols that have been composed into systems where a global specification may not be available. This work attempts to extend the ability to validate asynchronous systems by studying circuit models and different algorithms to compositions of asynchronous specifications.

Formal verification ensures correctness of a system given a required set of properties. Scalability due to state explosion has been a primary challenge in the verification community. State explosion is worse in the asynchronous paradigm due to the extra concurrency produced by unbounded delay models. To avoid state explosion problem, efficient modeling techniques, symbolic methods, abstraction and compositional reasoning have been explored. Our focus in this paper is to apply Symbolic model checking [8] on the composition of minimized controller state graphs which are in conformance with their specifications.

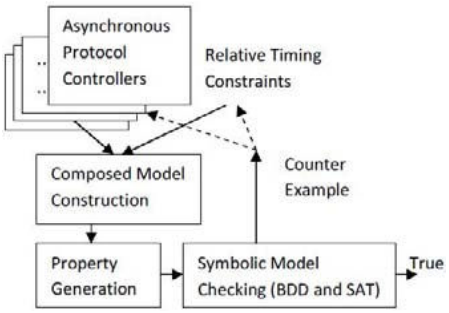


Figure 1: Illustrating the Model checking flow.

Performance and power of asynchronous hardware circuits and protocols can be vastly improved by modifying them with judicious application of timing constraints [10]. A new method of representing timing has been developed called *relative timing* [11]. The logical effect of timing on a system results in signals that arrive at certain nodes in a design in a fixed order. For instance, data arrives at a flip-flop before the clock due to system timing. This relationship is represented in relative timing (RT) as a point-of-convergence (POC) timing constraint that enforces signal ordering. The common timing reference of the two race paths is represented as a point-of-divergence (POD).

A methodology for synthesizing creating asynchronous designs with timing information for performance targets, power delay optimizations, and post layout timing validation using commercial ASIC CAD tools is being developed [12]. However, the system level verification has not yet been addressed. The verification problem becomes even more challenging when the protocols are timed. In this work, a methodology is developed for verifying larger asynchronous hardware protocols with timing through compositional symbolic model checking. An automated CAD tool is being built to provide a push button solution for symbolic model checking based on this paper.

A key aspect of this work is to enable the verification of *timed* asynchronous protocols. Designs will be checked for incorrect behavior when signal ordering is not logically enforced. When signal ordering can be enforced by system timing a relative timing constraint can be used. Relative timing constraints will be enforced within our model to prove

timed behavioral correctness. This approach enables hierarchical Symbolic Model checking of composed modules for both timed and untimed asynchronous hardware protocols, as well as clocked protocols.

The basic flow is illustrated as shown in Fig. 1. Inputs are the protocol controllers to be composed in the form of a state graph. Relative timing constraints are provided for timed asynchronous hardware protocols at each level of hierarchy. The composed model is constructed by translators for both interleaving and simultaneous [8] models with relative timing. Properties to be verified for correctness are generated. The composed model with properties are model checked for correctness with the industry-strength symbolic engine NuSMV [2]. If all properties are verified to be true then the system composed of the controllers has no timing or protocol interaction failures. In case of any counter example, one needs to investigate the system of controllers to determine if additional timing constraints or a different protocol is required for correctness.

An asynchronous linear pipeline controller (LC) circuit and specification are introduced. This circuit implements a timed (burst-mode [3, 15]) asynchronous protocol. Modeling techniques with relative timing for interleaving and simultaneous models are then illustrated for this example. The properties to be verified for correctness are explained. This approach is applied to the design of some other circuits and compositions of protocols. This approach is shown to be promising for hierarchical model checking of larger timed and untimed asynchronous hardware protocols. However, compositional reasoning aspects must be applied for scaling this approach to very large examples.

2. BACKGROUND

2.1 Asynchronous Hardware Protocols

Asynchronous systems do not have the global synchronization that a clock signal provides in synchronous systems, and therefore must rely on formal handshake protocols for correct sequencing and behavior. The modularity of asynchronous systems is derived from these formal handshake protocols. For timed asynchronous hardware protocols, the timing constraints must be enforced to avoid any computational interference errors where input changes occur in unaccepting states to a protocol [14]. These timing constraints ensure compositional correctness of the composed modules. Fig. 2 shows the petri net and Calculus of Communicating Systems (CCS) [9] specification for the LC example [4]. This specifies a timed handshake protocol with `lr` (left request) and `ra` (right acknowledge) as inputs and `rr` (right request) and `la` (left acknowledge) as the outputs. LEFT and RIGHT controllers in the CCS specification are barrier synchronized with causal arcs `c1` and `c2`. Dotted arcs `r1` and `r2` are the relative timing arcs in the petri net which explicitly enforce timing by constraining the input arrival times. Without the relative timing constraints `r1` and `r2`, the protocol is not burst-mode and becomes a different untimed protocol.

2.2 Relative timing

Relative timing (RT) is a method that explicitly represents the signal-ordering effect timing provides to a system. The ordering is logically related to a pair of signals. The common timing reference in a sequential protocol can be found by backtracking to a common causal reference signal

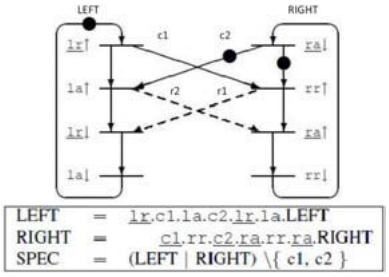


Figure 2: Petri net and CCS specification of LC.

called a point-of-divergence (POD). The signal events that must be ordered are listed in a point-of-convergence (POC) with a timing relationship. These path-based timing constraints are represented in the equation pod $\mapsto$ $\text{poc}_0 \prec \text{poc}_1$. The semantics are such that when the event `pod` occurs, $\texttt{poc}_0$ must strictly occur before $\texttt{poc}_1$. The equation can be annotated by signal logic levels, logical guards, etc. Relative timing constraints help to achieve behavioral correctness topographically by satisfying a required temporal property. When the constraints are enforced, there is pruning of state space because of sub graph eliminations [14].

2.3 Symbolic Model Checking

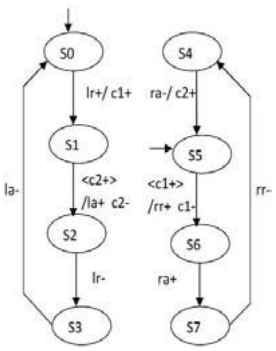
Model checking is verifying certain properties being true which behaviorally and temporally describe the system. Explicit state exploration techniques which express the transition relation in an explicit manner are more prone to state explosion problems. To avoid this problem, engines such as NuSMV apply the symbolic methods of Binary Decision Diagrams (BDD) and SATisfiability (SAT) [6]. Here, the state space is reduced by exploring state and transition sets rather than individual states or transitions. Model checking for BDD is based on checking the reachability with image and pre-image computations. In the case of SAT based Bounded model checkers, transition relations are unrolled for a bounded length. Unrolled transition boolean formulae in conjunction with negated properties is converted into Conjunctive Normal Form (CNF) or And Invert Graphs (AIG) boolean formulas. SAT Solvers verify for satisfiability on the combined set of boolean clauses. A failure occurs when the clauses find a solution, otherwise the property is verified to be true.

2.4 Property Specification Languages

Model checking of a particular property can be broadly classified as linear time (Linear Temporal Logic (LTL)) and branching time (Computational Tree Logic (CTL)) [7]. In case of LTL, There are no quantifiers in LTL, hence linear time path evaluation occurs by using proposition logic for describing the properties. In case of CTL, there is quantification of the paths with ForAll (A) and ThereExists (E) quantifiers. There are several other variants of temporal logic which include both LTL and CTL. The Property Specification Language (PSL)/Sugar [1] is a standardized property specification language which is an IEEE 1850 standard. It is sufficiently generalized and has all the features of LTL and CTL.

3. MODELING WITH RELATIVE TIMING

Compositions of asynchronous hardware protocols are required to perform system level verifications. Each proto-



(a)

Extended State Graph is a seven tuple: $EG(P, A, S, Init, L, [C])$ *where*
P is the finite set of atomic propositions
A is a finite set of actions
S is a finite set of states
$Init$ is the initial state
$R \subseteq S \times A \times S$ is the set of state transitions.
$L : s \rightarrow 2^P$ is the state labeling function.
$[C]$ represents any conditional event variables which are set/reset and probed for transitions R.

(b)

Figure 3: LC Extended State Graph (a) and Formal Definition of Extended State Graph (b).

col specification is converted into an extended state graph. These are then converted into interleaving and simultaneous models used by the verification engines as described below.

3.1 Extended State Graph Formalism

The extended state graph (ESG) is a boolean representation of a transition system with transition actions and states. Every state is assigned a binary vector of signal values for both inputs and outputs. The ESG has optional input conditions [C] for input and output transitions as defined in Fig. 3(b). Conditions are event variables which are set and reset based on the input/output behavior of the state graph. These may be required for modeling the causal arcs of a state transition graph or a petri net. This is shown for the LC example in Fig. 3(a). The minimized specification of an ESG can also be used as a standard state graph without the optional input condition. In that case, there may be improvement in performance based on modeling techniques.

3.2 Modeling Relative timing

Timed asynchronous protocols are modeled with relative timing constraints. The relative ordering of events can be tracked with auxiliary relative timing (RT) variables. These variables are set and reset based on the occurrence of events. One RT variable is required per RT constraint. The RT variable for constraint pod $\mapsto$ poc$_0$ $\prec$ poc$_1$ is set upon event `pod` and reset upon event `poc`$_0$ as shown in Fig. 4. The event `poc`$_1$ is constrained by the status of the relative timing variable (`r1` in the figure) and can only fire after `r1` is reset. RT variables are probed at the level of hierarchy in a design where the POC and POD signals reside.

One protocol timing constraint that must hold for the composition of specifications in Fig. 2 is lr+ $\mapsto$ rr+ $\prec$ lr-. Here, the RT variable `rt1` (correlating to arc `r1` the figure) is asserted with `lr+` and reset with `rr+`. Signal `lr-` is constrained to fire only when variable `rt1` is not asserted.

Relative timing constraints can be shared when they use the same set and reset conditions. For example, if we had two constraints lr+ $\mapsto$ rr+ $\prec$ lr- and lr+ $\mapsto$ rr+ $\prec$ lr+, they could share the same RT constraint. In addition, as timing is mapped into the logic domain, constraints can be further optimized based on dependencies between RT constraints and events.

3.3 Interleaving Model

The interleaved concurrency model only allows a single process to evolve at any time. This successfully models asynchronous hardware protocols. The transition relation R can be formally described as below.

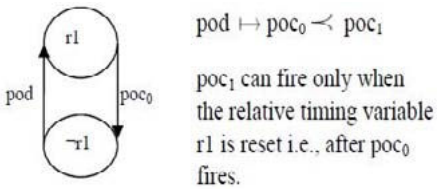


Figure 4: Modeling Relative timing constraint.

$R = \vee_i R_i$ where $R_i = (v_i' \Leftrightarrow f_i) \wedge (\wedge_{j \neq i}(v_j' \Leftrightarrow v_j))$

In any of the transitions, the new modified value v_i' is evaluated with function f_i for one of the state variable and the rest of the variables remain unchanged. This disjunction of component relations allows only one of them to change.

NuSMV code for the LC `left` and `right` modules is shown below. The left and right modules are composed to create the specification for the module lc with the lproc and rproc instantiations. The outputs of the left and right modules are la and rr respectively, with lr and ra being inputs. The protocol controller specifications are modeled as extended state graphs. Each module has four states and the state machine is modeled with init() and next() operators to represent its sequential behavior. The FAIRNESS running constraint forces each of the module instances to execute infinitely often. Outputs `la` and `rr` are assigned values based on the controller specification. Similarly, the conditional auxiliary variables are set and reset.

LC Interleaving model Code - 'left' and 'right' modules

```
MODULE left(lr,c1,c2,rt1,rt2)MODULE right(ra,c1,c2,rt1,rt2)
VAR                          VAR
state: {s0,s1,s2,s3};        state: {s4,s5,s6,s7};
la:boolean;--output          rr:boolean;--output
ASSIGN                       ASSIGN
init(state) := s0;           init(state) := s5;
next(state) := case          next(state) := case
(state = s0) & (lr=1): s1;   (state = s5) & (c1=1) : s6;
...                          ...
1: state;                    esac;
esac;                        --Also rr, c1, c2 are reset/set
ASSIGN                       MODULE lc(li,ri)
init(la) := 0;               VAR
next(la) := case             c1:boolean; c2:boolean;
state=s1 & (c2 = 1) : 1;     rt1:boolean; rt2:boolean;
--Also c1, c2 are set/reset  lproc:process left(li,c1,c2,rt1,rt2);
esac;                        rproc:process right(ri,c1,c2,rt1,rt2);
FAIRNESS running             FAIRNESS running
```

Signal ordering imposed by RT constraints can be modeled without changing the internal behavior of a protocol. Ordering is achieved by modifying the behavior of an input to a protocol from its environment. Two LCs composed with two RT constraints modeled at the interface are pictured in Fig. 5. Signal lc0.rproc.rr of two composed LC protocols is constrained in the figure by modifying its behavior before supplying it as signal `c_lr` to lc1.lr. Without this constraint, input changes will occur in unaccepting states resulting in an error. In an unconstrained composition, lc0.rproc.rr can go low before lc1.rproc.rr goes high, resulting in the error on signal lc1.lr. This error is avoided by the constraint lr+ $\mapsto$ rr+ $\prec$ lr- modeled with RT variable rt1.

Code for the composed system of Fig. 5 is shown below. Keyword process is necessary for the module instances lc0 and lc1 during interleaving composition. This makes process execution non-deterministic with unbounded delay, correctly

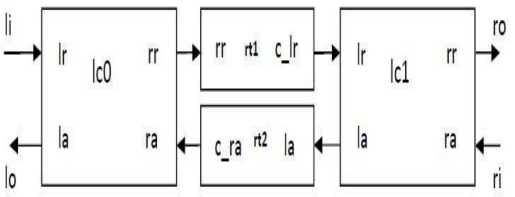


Figure 5: Composition of two Linear controllers with Relative timing constraints.

modeling asynchronous protocols. Because each process is executed independently, the setting and resetting of the RT constraints must occur within a process. Boolean variable rt1 is used for constraint lr+ $\mapsto$ rr+ $\prec$ lr- on process lc1. (The code for constraint rt2 is not shown for brevity.) Variable rt1 is set in the left and reset in the right process of lc. The RT variable is sensed at the top level of hierarchy to constrain signal ordering. This is performed in the final ASSIGN statement in the code below, ensuring that `lr-` can happen only when the rt1 is reset. Similarly, lc1.lproc.la is constrained and applied as c_ra to the lc0.ra module.

Composed and RT Constrained Interleaving LC model code

```
--Setting RT variable in 'left' --Reset RT variable in 'right'
ASSIGN                          ASSIGN
init(rt1) := 0;                 next(rt1) := case
next(rt1) := case                (state=s5) & (c1 = 1) : 0;
 (state=s0) & (lr = 1) : 1;      1                     : rt1;
 1                     : rt1;   esac;
esac;
                                ASSIGN
MODULE main                     init(c_lr) := 0;
VAR                             next(c_lr) := case
li:boolean;ri:boolean;           (lc0.rproc.rr=1) : 1;
c_lr:boolean;c_ra:boolean;       (lc0.rproc.rr=0)
....                               & (lc1.rt1=0) : 0;
lc0:process lc(li,c_ra);         1               : c_lr;
lc1:process lc(c_lr,ri);        esac;
```

3.4 Simultaneous Model

Simultaneous model allows execution of multiple processes at the same time. Within each process, there is independent choice to undergo a state transition or remain in the same state non-deterministically. the required reachability is explored similarly to the interleaving model. This approach also correctly models asynchronous hardware protocols. Transition relation R is formally defined as below:

$R = \wedge_{1 \leq i \leq n} R_i$ where $R_i = (v_i' \Leftrightarrow f_i) \vee (v_i' \Leftrightarrow v_i)$

For any transition, either the modified variable value v_i' is evaluated by f_i or the old value holds for any state variable v_i. This modeling of simultaneous execution differs with respect to interleaving model in three ways: (i) The union keyword needs to be used for modeling non-deterministic nature of next state transitions [8]. (ii) While setting and resetting outputs and auxiliary variables based on the events, one has to make sure whether the next state is the one which is desired using next() operator. (iii) There is no process keyword to be used for module instances. Since all process are allowed to run every time no fairness constraints are needed. Code for the LC example is shown below for the left and right modules.

In the simultaneous model we can set and reset the RT variables at the top level of hierarchy by probing the signal variables that cause the change. (This can be performed internal if all signals are in the same module.) Like the interleaving model, the signals are also constrained at this level of hierarchy. Modules are composed without a process keyword as shown in LC example code below.

LC Simultaneous model - 'left' and 'right' modules

```
MODULE left(lr,c2)              MODULE right(ra,c1)
VAR                             VAR
state: {s0,s1,s2,s3};           state: {s4,s5,s6,s7};
la:boolean;--output             rr:boolean;--output
ASSIGN                          ASSIGN
init(state) := s0;              init(state) := s5;
next(state) := case             next(state) := case
 (state=s0)&(lr=1) : s1          (state=s5)&(c1=1)  : s6
  union state;  ...               union state;  ...
 1                : state;       1                  : state;
esac;                           esac;
ASSIGN                          ASSIGN
init(la):=0;                    init(rr):=0;
next(la) := case                next(rr) := case
  state=s1&next(state)=s2&       (state=s5)&next(state)=s6&
   (c2=1)            : 1;          (c1=1)              : 1;
  1                  : la;        1                   : rr;
esac;                           esac;
```

LC Simultaneous composed model - RT Constraining

```
MODULE lc(li,ri)                ASSIGN
VAR                             init(rt1) := 0;
c1:boolean;                     next(rt1) := case
c2:boolean;                      (lc0.rproc.rr=0) &
lproc :left(li,c2);               (next(lc0.rproc.rr)=1) &
rproc :right(ri,c1);              (rt1=0)                : 1;
....                             (lc1.rproc.rr=0) &
--c1,c2 are set/reset here        (next(lc1.rproc.rr)=1) &
....                              (rt1=1)                : 0;
                                esac;
MODULE main                     ASSIGN
VAR                             init(c_lr) := 0;
c_lr:boolean;c_ra:boolean;      next(c_lr) := case
rt1:Boolean;                     (lc0.rproc.rr=1)&(c_lr=0) : 1;
....                             (lc0.rproc.rr=0) & (rt1=0)
lc0: lc(li,c_ra);                 & (c_lr=1)              : 0;
lc1: lc(c_lr,ri);                1                       : c_lr;
                                esac;
```

RT variable rt1 is set and reset as shown. The variable is set after the POD event i.e, lc0.rproc.rr+, and reset after lc1.rproc.rr+. The constrained event can only occur after the reset of the RT variable. The composition of the modules is the same as the interleaving model shown in Fig. 5.

4. PROPERTIES VERIFIED

4.1 Deadlock and Livelock freedom

A composition of asynchronous protocols must be free from deadlocks and livelocks. This can be verified by checking the reachability of two distinct states in possible paths [13] with CTL properties as shown below for the LC example. Distinct states were verified for reachability in LC example and there were no deadlocks detected. With regards to automation, initial states of the individual controllers can be used to check reachability.

```
SPEC AG EF lc1.rproc.state = s5;
SPEC AG EF lc1.lproc.state = s0;
```

4.2 Semi-Modularity Check

The protocol in this paper is a timed burst-mode protocol. Therefore a relative ordering of events at the interface is required. The following PSL property verifies the ordering constraint lr+ $\mapsto$ rr+ $\prec$ lr- of the protocol composition in Fig. 5. The first sequence of `c_lr` within { } i.e., `c_lr+`, sequentially implies `rr+` should occur before `c_lr-`. This property was verified to always be true with the relative timed model. Without relative timing, this property fails to satisfy and generates a counter example.

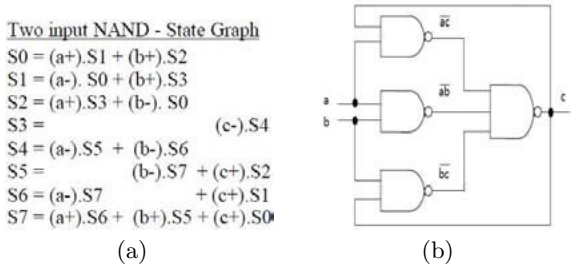


Figure 6: Semi-modular state graph of NAND Gate and C-Element Implementation with NAND gates

```
PSLSPEC always ({c_lr=0;c_lr=1}
               ((lc1.rproc.rr=0 & next(lc1.rproc.rr=1))
                before (c_lr=1 & next(c_lr=0))));
```

4.3 Failure reachability

Another way to verify correct timed behavior is to include a transition to a failure state on the occurrence of an unaccepted input. If the failure state is reachable, an error has occurred. Such transitions must be avoided through RT constraints for the protocol to be correct. This requires a change to the ESG as shown below. In this model, lr lowering in state s2 if rt1 is asserted is an illegal input condition which leads to the failure state f.

LC Interleaving model - Failure state modeling

```
MODULE left(lr,c1,c2,rt1,rt2)
VAR
state: {s0,s1,s2,s3,f};
la:boolean;--output

ASSIGN
init(state) := s0;
next(state) := case
 (state = s0) & (lr=1) : s1;
 ...
 (state = s2) & (rt1=1)
   & (lr=0)          : f;
 1                   : state;
esac;
FAIRNESS running
```

```
/*Properties verified with and
without relative timing.
With input constrained by RT,
failure state is not reachable.
Without RT constraint,
Failure state is reachable.*/
--true with RT and false without
CTLSPEC !EF
  (lc1.lproc.state = f);

--true with RT and false without
PSLSPEC never (eventually!
           (lc1.lproc.state = f));

--false with RT and true without
SPEC AG EF lc1.lproc.state = f;
```

4.4 Eventuality and Liveness

Behavioral properties of a protocol and its composition are specific to the design. These will not be automatically generated by the CAD tool we are developing. However, the tool will be designed such that the user can provide additional properties to verify. For example, a user could provide the following PSL property verifying that always eventually, whenever there is left most request `li+`, there will always eventually be the right most request `rr+`.

```
PSLSPEC always eventually!({li=0;li=1}(always eventually!
(lc1.rproc.rr=0 & next(lc1.rproc.rr=1))));
```

5. CASE STUDIES AND RESULTS

5.1 C-Element

An implementation of a C-Element protocol with gates is shown in Fig. 6(b). This C-Element protocol can be verified as a composition of asynchronous NAND gate protocols. The semi-modular state graph of a two input NAND gate is shown. Failure occurs if inputs change when they are not enabled, indicated by blank spaces in the state graph.

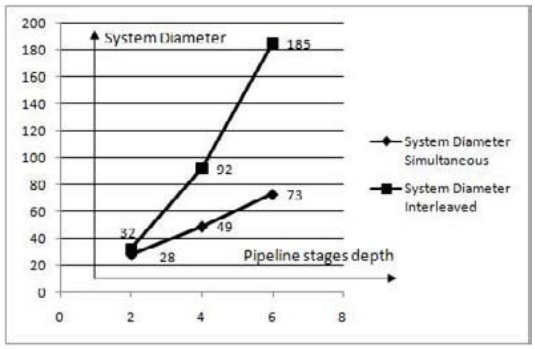


Figure 7: LC example: System diameter for Simultaneous and Interleaving models with different stages.

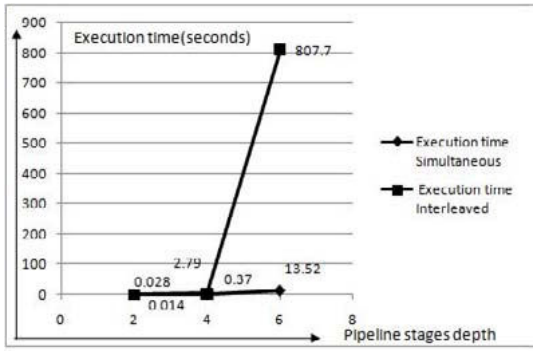


Figure 8: LC example: Execution times for semi-modular property being true of Interleaving and Simultaneous models with different stages.

The state graph for a two input NAND gate can be modeled equivalently in NuSMV model. Similarly, the three input NAND gate is modeled and all the modules are composed in both interleaving and simultaneous models. Deadlock properties were verified. Relative timing constraints modeled and verified include:

$c+ \mapsto ac+ \prec b-$ $\quad$ $c+ \mapsto ac+ \prec a-$

$c+ \mapsto bc+ \prec a-$ $\quad$ $c+ \mapsto bc+ \prec b-$

5.2 Linear Controller

The composition of linear controllers shown in Fig. 5 is verified. The two RT constraints are applied.

$lr+ \mapsto rr+ \prec lr-$

$ra- \mapsto la+ \prec ra+$

Reachable states were same for both interleaving and simultaneous models without applying the RT constraints. Performance evaluation of the LC composition is measured with different modeling techniques, properties, symbolic methods, and different number of pipeline stages. A workstation with AMD Athlon^{TM} 64 X2 Dual core processor 5600, 2.9GHz and 2GB memory was used. The NuSMV engine was employed using its default parameters. Fig. 7 compares the system diameter [2] for interleaving and simultaneous models with different pipeline stages. Fig. 8 shows the execution times in seconds for interleaving and simultaneous models. Fig. 9 compares BDD and SAT methods for the simultaneous model in terms of execution times required for finding a counter example/failure. This also shows the com-

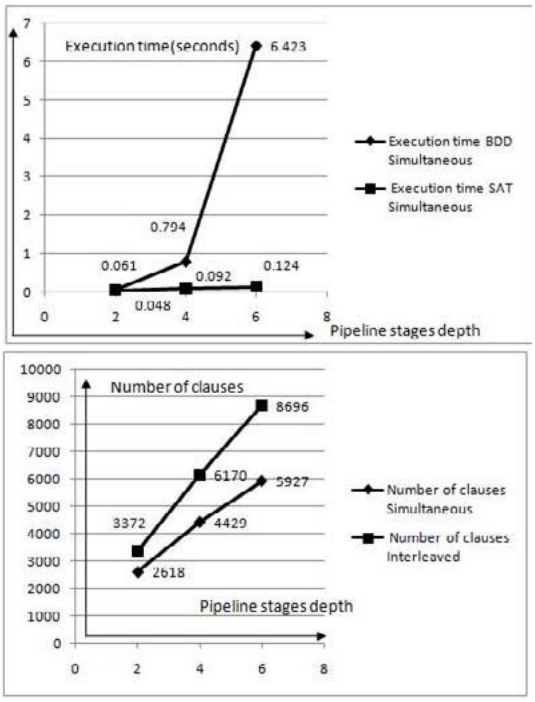


Figure 9: LC example:Execution times of BDD and SAT methods for a semi-modular failure property and clauses for SAT method with different stages.

parison in number of clauses with different pipeline stages. The simultaneous model is more efficient with regards to execution time and system diameter. SAT based methods have reduced execution times for finding a counter example when compared to BDD. SAT based methods are dependent on the unroll bound of the failures.

6. CONCLUSIONS

A symbolic model checking flow for verifying systems of timed asynchronous protocols is presented. Timing failures are manifest by the occurrence of unaccepted inputs in a system composed of asynchronous protocols. A method has been developed to integrate RT constraints into the protocols in order to prevent timing failures. Timing is efficiently modeled in a hierarchical fashion by reducing the reachable space of the composed protocols, with overhead of the added RT variables. This method does not modify the behavior of the initial protocols composed in a system. The approach has been applied to both interleaving and simultaneous execution models. In general the simultaneous models show better performance. The industry-strength NuSMV engine was employed. A number of properties can be automatically generated to verify liveness and timing correctness. The flow was validated on two examples using interleaving and simultaneous models. For the linear pipeline controller example, in the simultaneous model, SAT methods tend to find a counter example in lesser time than BDD methods. This work has laid the foundation for the development of a CAD tool to automate push-button verification of large timed asynchronous designs.

7. ACKNOWLEDGMENTS

This material is based upon work supported by Semiconductor Research Corporation task 1817.001 and the National Science Foundation under Grant No. 0702539.

8. REFERENCES

[1] Accellera. PSL Reference Manual. `http://www.eda.org/vfv/docs/PSL-v1.1.pdf`.

[2] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev. Nusmv 2.4 user manual. `http://nusmv.irst.itc.it`.

[3] W. S. Coates, A. L. Davis, and K. S. Stevens. Automatic Synthesis of Fast Compact Self-Timed Control Circuits. In *IFIP Working Conference on Design Methodologies*, pages 193–208, April 1993.

[4] J. Cortadella, M. Kishinevsky, S. M. Burns, A. Kondratyev, L. Lavagno, K. S. Stevens, A. Taubin, and A. Yakovlev. Lazy transition systems and asynchronous circuit synthesis with relative timing assumptions. *IEEE Transactions on Computer-Aided Design*, 21(2):109–130, Feb 2002.

[5] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

[6] M. Ganai and A. Gupta. *SAT-Based Scalable Formal Verification Solutions*. Springer Publishers, 2007.

[7] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive Systems: specification*. Springer-Verlag, New York, 1992.

[8] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[9] R. Milner. *Communication and Concurrency*. Computer Science. Prentice Hall, 1989.

[10] K. S. Stevens, R. Ginosar, and S. Rotem. Relative Timing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(11):129–140, Feb. 2003.

[11] K. S. Stevens, S. Rotem, R. Ginosar, P. Beerel, C. J. Myers, K. Y. Yun, R. Kol, C. Dike, and M. Roncken. An Asynchronous Instruction Length Decoder. *IEEE Journal of Solid State Circuits*, 36(2):217–228, Feb. 2001.

[12] K. S. Stevens, Y. Xu, and V. Vij. Characterization of Asynchronous Templates for Integration into Clocked CAD Flows. In *15th International Symposium on Asynchronous Circuits and Systems*, pages 151–161. IEEE, May 2009.

[13] V. Vakilotojar and P. A. Beerel. RTL verification of timed asynchronous and heterogeneous systems using symbolic model checking. *Integration, the VLSI Journal.*, 24(1):19–35, December 1997.

[14] Y. Xu and K. S. Stevens. Automatic Synthesis of Computation Interference Constraints for Relative Timing. In *26th International Conference on Computer Design*. IEEE, Oct. 2009.

[15] K. Y. Yun and D. L. Dill. Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-Free Implementation). *IEEE Transactions on Computer-Aided Design*, 18(2):101–117, Feb 1999.