Efficient Verification of Timed Automata using Dense and Discrete Time Semantics*

Marius Bozga, Oded Maler, Stavros Tripakis

VERIMAG, Centre Equation, 2, av. de Vignate, 38610 Gières, France
{**bozga,maler,tripakis**}@imag.fr

Abstract. In this paper we argue that the semantic issues of *discrete* vs. *dense* time should be separated as much as possible from the pragmatics of state-space representation. Contrary to some misconceptions, the discrete semantics is not inherently bound to use state-explosive techniques any more than the dense one. In fact, discrete timed automata can be analyzed using any representation scheme (such as DBM) used for dense time, and *in addition* can benefit from enumerative and symbolic techniques (such as BDDs) which are not naturally applicable to dense time. DBMs, on the other hand, can still be used more efficiently by taking into account the *activity* of clocks, to eliminate redundancy.
To support these claims we report experimental results obtained using an extension of Kronos with BDDs and variable-dimension DBMs where we verified the asynchronous chip STARI, a FIFO buffer which provides for skew-tolerant communication between two synchronous systems. Using discrete time and BDDs we were able to prove correctness of a STARI implementation with 18 stages (55 clocks), better than what has been achieved using other techniques. The verification results carry over to the dense semantics.
Using variable-dimension DBMs we have managed to verify STARI for up to 8 stages (27 clocks). In fact, our analysis shows that at most one third of the clocks are active at any reachable state, and about one fourth of the clocks are active in 90% of the reachable states.

1 Introduction

The analysis of discrete systems such as programs or digital circuits, while taking into account the temporal uncertainty associated with transition delays, is a very challenging and important task. In [MP95] and elsewhere, it has been demonstrated that reasonable models of digital circuits with uncertain delay bounds can be translated systematically into timed automata [AD94], which can then be analyzed using various verification tools. However, this remains a theoretical possibility as long as the performance bottleneck for timed verification remains (see the discussion in [BMPY97] as well as [TKB97]). During the last decade the

* This work was partially supported by the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid systems) and the French-Israeli collaboration project 970MAEFUT5 (Hybrid Models of Industrial Plants).

VERIMAG laboratory has been engaged in development of the timing analyzer KRONOS [DOTY96] and in attempts to improve its performance using various techniques.

Timed automata operating on the dense real time axis constitute an instance of *hybrid automata* and their analysis confronts researchers with problems usually not encountered in "classical" finite-state verification. These problems, such as Zeno's paradox (the possibility of making infinitely-many steps in a bounded interval) or the representation of an uncountable number of states, which are related to the foundations of mathematics, sometime give a false impression on the essense of timing analysis. We argue that this essence does not depend on the time domain, and that the difference between dense and discrete time semantics is "epsilon", so to speak.

We demonstrate these claims by applying discrete and dense techniques to a non-trivial case-study where we obtain the best performance results achieved so far for timed automata. More precisely, we report the application of two techniques: the BDD-based verification using the discrete time semantics [ABK+97] [BMPY97], and the "standard" DBM-based method using variable-sized matrices based on clock activity analysis [DY96], to a real hardware design, the STARI chip due to M. Greenstreet [Gre97]. This chip is an asynchronous realization of a FIFO buffer, composed of a sequence of stages, each consisting of two Muller C-elements and one NOR gate. According to the principles laid out in [MP95], and similarly to [TB97], each such stage is modeled as a product of 3 timed automata, each with 4 states and one clock. The (skewed) transmitter and receiver are modeled as two timed automata using a shared clock.

We have modeled the intended behavior of the FIFO buffer operationally as an automaton, and were able to verify that an 18-stage implementation (55 clocks) indeed realizes the specification. These are, to the best of our knowledge, among the *best performance results* for timed automata verification, and they show that some real circuits behave better than artificial examples of the kind we used in [BMPY97].

The rest of the paper is organized as follows. In section 2 we give a very informal survey of timed automata, their verification techniques and their discrete and dense semantics. In section 3 we describe STARI and its desired properties, which are then modeled using timed automata in section 4. The performance results are reported and analyzed in section 5, and future work is discussed at the end.

2 Verification using Timed Automata

2.1 Timed Automata

Timed automata can represent systems in which actions take some unknown, but bounded, amount of time to complete, in a rigorous and verifiable manner. They are essentially automata operating on the continuous time scale, employing auxiliary continuous variables called clocks. These clocks, while in a given state,

keep on increasing with time. Their values, when they cross certain thresholds can enable some transitions and also force the automaton to leave a state. Temporal uncertainty is modeled as the possibility to choose between staying in a state and taking a transition during an interval $[l, u]$.

Since their introduction in 1990 [AD94], Timed Automata (TA) attracted a lot of attention from the verification community, mainly for the following reasons:

1. They constitute a computational model in which one can faithfully represent many real-world situations where timing constraints interfere with discrete transitions.
2. In spite of the fact that their state-space is non-countable, their reachability problems (which are the essence of any verification problem) are decidable. Several verification tools such as Kronos [DOTY96], Timed Cospan [AK96] and Uppaal [LPY97] have been implemented, featuring various verification and synthesis algorithms which explore, this way or another, the state-space of TA.

The use of a non-countable state-space, $Q \times X$ where $|Q| = m$ and $X = [0, k]^d$, along with dense time, excludes immediately any verification method which is based on explicit enumeration of states and trajectories. All existing TA verification algorithms are based, either explicitly or implicitly, on the region graph construction [AD94]: an equivalence relation is defined on X, unifying clock configurations from which the future behaviors are essentially identical (i.e. two clock valuations $\mathbf{x}$ and $\mathbf{x}'$ are equivalent, $\mathbf{x} \sim \mathbf{x}'$, if the same sequences of discrete transitions are possible from $\mathbf{x}$ and from $\mathbf{x}'$). It turns out that in TA this relation is of finite index, and the quotient graph of a TA, modulo $\sim$, is a finite-state automaton with a combination of discrete transitions and abstract "time-passage" transitions, indicating the temporal evolution of clock values from one equivalence class to another.

Verification tools for TA, either construct first the region automaton (whose size is $O(mk^d d!)$ and then use standard discrete verification algorithms, or calculate the reachable configurations successively while representing them as unions of polyhedra of certain restricted form. These "symbolic states", which are generated by the equivalences classes of $\sim$, are polyhedra which can be represented by combinations of inequalities of the form $x_i \prec c$ or $x_i - x_j \prec c$ where x_i, x_j are clock variables, $\prec$ is either $<$ or $\leq$, and c an integer constant in $\{0, 1, \ldots, k-1\}$. For convex regions, there is a canonical form based on an irredundant set of inequalities, which can be efficiently represented using an $O(n^2)$-sized integer matrix, known as the *difference bounds matrix* (DBM). The main computational activity in TA verification is the storage and manipulation of sets of these matrices during fixed-point computations. The major bottleneck is due to the fact that the number and size of DBMs grows exponentially with the number of clocks and the size of k (roughly, the size of the largest constant in the TA after normalization). Moreover the representation of non-convex polyhedra as unions of convex ones is not unique. There have been many attempts to break the computational bottleneck associated with the manipulation of DBMs, such as [WD94,H93,B96,DY96,LLPY97,DT98], to mention a few, and to be able to

verify larger timed automata. One approach, [DY96], is based on the observation that not all clocks are active at any configuration (see also [SV96]). A clock x_i is inactive in a configuration $(q, \mathbf{x})$ if it is reset to zero before any future test of its value. In that case its value can be eliminated from the state description of the system. Consequently one can use variable-sized DBMs restricted to the relevant clocks in every region of the TA. In section 5 we will report the results of clock activity analysis of STARI and the performance of the variable-sized DBMs.

2.2 The Joy of Discrete Time

There is, however, an alternative semantics for TA based on discrete (and in fact, integer) time, which has already been discussed in early works about real-time logics (see the survey [AH92]). According to this view, time steps are multiples of a constant, and at every moment the automaton might choose between incrementing time or making a discrete transition. Consider the fragment of a 2-clock timed automaton depicted at the left of Figure 1. The automaton can stay in the state and let the time progress (i.e. let the values of x_1 and x_2 grow with derivative 1) as long as $x_1 \leq u$. As soon as x_1 reaches l (we assume $l \leq u$) it can take a transition to another state and reset x_1 to zero. By restricting the time domain to the integers, the staying conditions ("invariants") in every state are replaced by "idle" transitions as in the right of Figure 1.

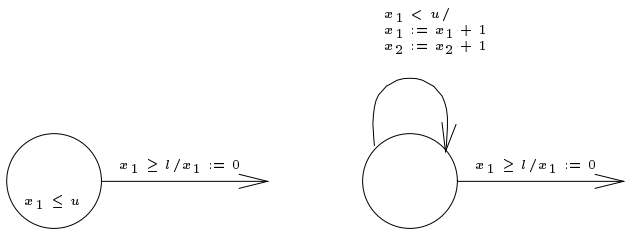


Fig. 1. A timed automaton and its discrete time interpretation.

Under this interpretation clocks are nothing but *bounded integer variables*, whose values are incremented simultaneously by time transitions and some of them are reset to zero by certain discrete transitions. Such systems are *finite-state*, but some components of the state-space, namely the clocks, have additional structure (addition and linear-ordering of clock values), which can be exploited by verification algorithms. In particular, any representation scheme for the dense semantics which is based on clock inequalities can be specialized for the discrete semantics. Since on discrete order, a strict inequality of the form $x_i < c$ can be written as the non-strict inequality $x_i \leq c - 1$, discrete regions can be expressed using exclusively non-strict inequalities. Hence even DBM-based methods can be tuned to work better on discrete time since the space of DBMs is smaller. A

typical step in the iterative calculation of reachable states is depicted in Figure 2 for the dense (left) and discrete (right) semantics.

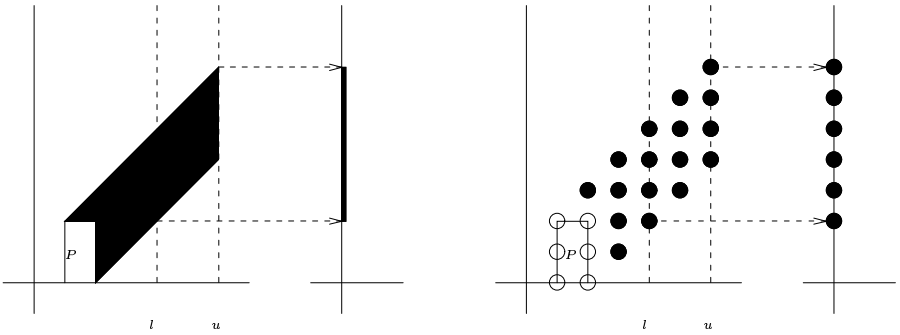


Fig. 2. Calculation of reachable configurations, starting from the initial set P, for the dense and discrete timed automata of Figure 1.

In addition to these methods one can take advantage of the finite-state nature of discrete TA and apply techniques which *cannot* be applied directly to dense time. One possibility is to push clocks values into states and transform the TA into a finite automaton (either off-line or on-the-fly). This provides for depth-first traversal of the state-space, as well as other search regimes. An alternative approach is the one advocated in [ABK+97,BMPY97] in which the clocks values are encoded in binary and subsets of them are written as BDDs. The advantage of this approach is that it gives a *canonical representation* for any subset (convex or not) of the state-space, and that it combines naturally with BDD-based representation of the control states. Most of this paper is a report of one success story of this approach, where a non-trivial system with 55 clocks has been verified. However, before that there is one little point to be clarified: although discrete time verification is inherently more efficient than dense time, it is certainly less expressive, and one might want to know what is sacrificed in order to improve performance. Our answer, based on the results in [HMP92,AMP98], which we explain below is: "not much".

2.3 Why Discrete Time Suffices

Consider two clock valuations $\mathbf{x}$ and $\mathbf{x}'$ sharing the same open unit square $S = (c_1, c_1+1, c_2, c_2+1)$ (Figure 3-(a)). Clearly, every inequality of the form $x_i \prec c$, $i \in \{1,2\}$ and $\prec \in \{<, \leq, >, \geq\}$ is satisfied by $\mathbf{x}$ iff it is satisfied by $\mathbf{x}'$. Hence a transition that can be taken at $\mathbf{x}$, leading to a new valuation $\mathbf{y}$, iff it can be taken at $\mathbf{x}'$ leading to a point $\mathbf{y}'$ on the same square as $\mathbf{y}$. For the same reasons time can progress at $\mathbf{x}$ iff it can do so at $\mathbf{x}'$, however unless the order among the fractional parts of x_1 and x_2 is the same in $\mathbf{x}$ and $\mathbf{x}'$ they might reach different squares as

time goes by. Only if they belong to the same triangular subset of X, namely a set of the form $\{\mathbf{x} : \langle x_1 \rangle \prec \langle x_2 \rangle\}$ where $\langle x_i \rangle$ denotes the fractional part of x_i, they will meet the same squares during time evolution (Figure 3-(b)). Combining these facts we obtain the equivalence relation on X which guarantees that all the members of an equivalence class can exhibit essentially the same behaviors.

This simple (and simplified) story becomes more complicated if transition guards and invariants are allowed to contain strict inequalities. In that case some transitions might be enabled in the interior of a region but not in its boundaries, and the region graph becomes a more involved mathematical object with elements of all dimensionalities from 0 to n. If, however, timing constraints are restricted to be closed (i.e. non-strict) every boundary point satisfies *all* the constraints satisfied by the regions in its neighborhood. In particular the set of integral points, the grid $\{0, 1, \ldots, k-1\}^n$ "covers" all X in the sense that it intersects the boundaries of all open full-dimensional regions and satisfies all the constraints that they satisfy (Figure 3-(c)). Hence these integral points can be taken as *representatives* and all the (qualitative) trajectories starting from them cover all the possible behaviors of the system.

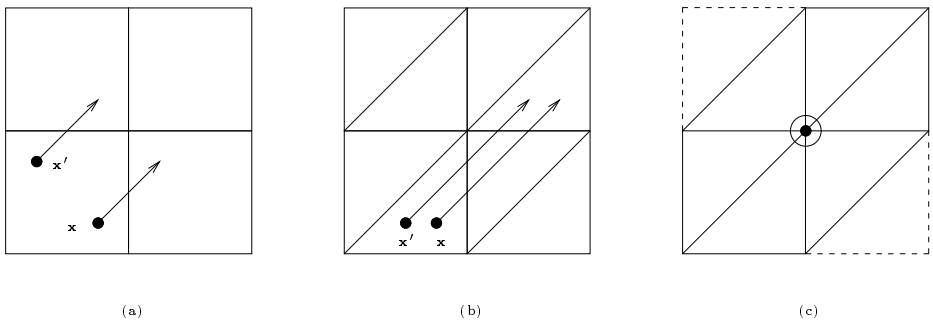


Fig. 3. a) Two points satisfying the same constraints; b) Two equivalent points; c) An integer point satisfying all the constraints satisfied by its six neighboring regions.

To be more precise, a discrete run might be a slight variation of some of the dense runs it represents: it may sometimes have few transitions taken simultaneously while in the dense run these transitions are separated by small amount of time. Nevertheless, the following results [HMP92,AMP98] underlie the soundness of discrete verification:

Theorem 1 (Emptiness of Closed TA). *The set of dense behaviors of a closed TA is non-empty iff it contains a discrete run.*

Combining this with the fact that untimed properties can be expressed as TAs in which all constraints are *true* (which is closed), we have:

Corollary 1 (Discrete Time Verification). *A closed TA satisfies an untimed property φ iff its discrete time version satisfies φ.*

If the TA is not closed, its closure is an over-approximation and a satisfaction of any linear-time property by the closure implies satisfiability by the original automaton.

3 STARI Description

STARI (Self-Timed At Receiver's Input) [Gre97] is a novel approach to high-bandwidth communication which combines synchronous and self-timed design techniques. Generally speaking, a transmitter communicates synchronously with a receiver through an asynchronous FIFO buffer (see Figure 4). The FIFO makes the system tolerant to time-varying skew between the transmitter and receiver clocks. An internal handshake protocol using acknowledgments prevents data loss or duplication inside the queue.

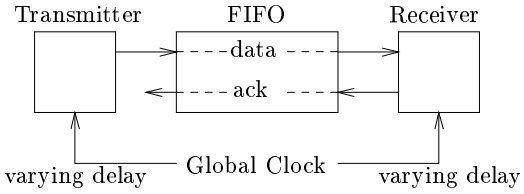


Fig. 4. The STARI overview.

The functioning of STARI is based on a rather intuitive idea. The FIFO must be initialized to be half-full. During each period of the clock one value is inserted to the FIFO by the transmitter and one value is removed by the receiver. Due to the complementary nature of these actions no control is required to prevent queue underflow or overflow. Short-term fluctuations in the clock rates of the transmitter and the receiver are handled by inserting or removing, more items to or from the queue.

Following the STARI model proposed by Tasiran and Brayton in [TB97], which differs slightly from the original description in [Gre97], we represent the boolean values *true* and *false* by *dual rail* encoding (see Figure 5). An auxiliary *empty* value is needed to distinguish between the case of two consecutive identical values and the case of one value maintained during more than one clock cycle. The transmitter is constrained to send sequences of *true* and *false* where each two occurrences of these values are separated by an occurrence of *empty*. The STARI chip consists of a linear array of n identical stages, each capable of storing a data value X.

The following two properties need to be proved to ensure the correct operation of the STARI circuit:

X	*true*	*false*	*empty*
X.t	1	0	0
X.f	0	1	0

Fig. 5. Dual rail encoding.

- Each data value output by the transmitter must be inserted in the FIFO before the next one.
- A new value must be output by the FIFO before each acknowledgment from the receiver

We specify the desired behavior of an n-stage STARI as an ideal FIFO buffer combined with a receiver and a transmitter respecting the abovementioned convention (see Figure 6). Note that in this specification, every transition is labeled with a pair of *put* and *get* actions, with the intended meaning that they can occur in *any order* including simultaneously. The goal of the verification is to show that if we hide the internal operations of STARI, the realizable sequences of *put*'s and *get*'s conform with this specification.

The operation principle of a stage k can be summarized as follows: *it may copy its predecessor value ($X_k := X_{k-1}$) when its successor has already copied (and acknowledged) its current value ($X_k = X_{k+1}$)*. Using the dual rail encoding of data values, such a behavior can be achieved using two Muller C-elements that hold the $X.t$ and $X.f$ components, and one NOR gate for computing the acknowledgment (see Figure 7).

A Muller C-element works as follows: when the two inputs become identical, after some delay the output takes on their value, otherwise the output maintains its previous value. Consider, for example, a situation where stages k and $k+1$ hold the *empty* value, stage $k-1$ the *true* value and $Ack_{k+1} = 0$. When Ack_{k+1} becomes 1, the C-element for $X_k.f$ remains unchanged at 0 because its inputs are different (i.e. $Ack_{k+1} = 1$, $X_{k-1}.f = 0$). However, both the inputs of the C-element for $X_k.t$ are equal to 1 ($Ack_{k+1} = X_{k-1}.t = 1$), and after some delay, it will switch to 1. This way the *true* value has been copied from stage $k-1$ to stage k.

4 Modeling STARI by Timed Automata

The correct functioning of STARI depends on the timing characteristics of the gates (the time it takes, say, for a C-element to switch) and its relation with the central clock period and the skew between the receiver and transmitter. We model the uncertainty concerning the delay associated with gates using the bi-bounded delay model, that is, we associate with every gate an interval $[l, u]$ indicating the lower and upper bounds for its switching delay (see [L89], [BS94], [MP95] and [AMP98] for the exact definitions).

Following [MP95] we can model any logical gate with a delay $[l, u]$ using a timed automaton with 4 states (0-stable, 0-excited, 1-stable and 1-excited) and

one clock. In particular, each stage of STARI is modeled by the three timed automata of Figures 8, 9 and 10.

Let us look at the automaton of Figure 8 which models the $X.t$ component of the k^{th} stage. Its state is characterized by two boolean variables $X_k.t$, $x_k.t$, the former stores the gate output and the latter stores the gate internal value, i.e. the value to which the gate "wants" to go after the delay. The stable states are those in which $X_k.t = x_k.t$. The conditions for staying and leaving stable states are complementary and do not depend on clock values: for example, the automaton leaves state $(0,0)$ and goes to the unstable state $(0,1)$ exactly when both its inputs are 1. During this transition the clock variable $C_k.t$ is reset to zero. The automaton can stay at $(0,1)$ as long as $C_k.t < u_C$ and can change its output and stabilize in $(1,1)$ as soon as $C_k.t \geq l_C$, where $[l_C, u_C]$ is the delay interval associated with a C-element. The automaton for the $X.f$ component (Figure 9) is exactly the same (with different inputs) and the automaton for the NOR gate (Figure 10) is similarly characterized by two boolean variables Ack_k, ack_k, a clock variable $C_k.a$ and a delay bounded by $[l_N, u_N]$. This means that an n-stage STARI can be translated into $3n$ automata with $3n$ clocks and $6n$ boolean variables.

In addition to the automata for modeling the stages, we need three other automata for modeling the transmitter, the reciever and their clock cycle. The global *clock cycle* is modeled by a simple timed automaton using one clock variable C. Whenever C reaches the cycle size p it is reset to zero. (see Figure 11).

The *transmitter* is modeled as a 3-state automaton (Figure 11). At each clock cycle it puts a value at the input ports of the first stage ($X_0.t$ and $X_0.f$), according to the convention that every pair of data items is separated by an *empty* item. Moreover, the transmission can be done with some skew with respect to the clock cycle, bounded by the s_T constant, that is, the actual time of transmission can be anywhere in the interval $[p - s_T, p]$.

The *receiver* is a 1-state automaton (see Figure 11) which reads the current output value (i.e. $X_n.t$ and $X_n.f$) and acknowledges the reception by modifying Ack_{n+1} according to whether or not X_n is empty. As in the transmitter, a skew bounded by s_R is allowed.

Note that the receiver and transmitter skews *cannot accumulate* during successive cycles. They always range in an interval depending on the (perfect) global clock cycle. However, each one can vary non-deterministically from one cycle to another. This is more general than assuming a fixed skew given in advance, or a fixed skew chosen at start-up from a given interval. The transitions of these automata are annotated by action names such as *put* and *get* whose role is explanatory – they have no effect on the functioning of the system.

5 Verification Results and Performance Analysis

5.1 Discrete Time and BDDs

We have modeled various instants of STARI, each with a different number of stages. For each instance we have composed the timed automata and then min-

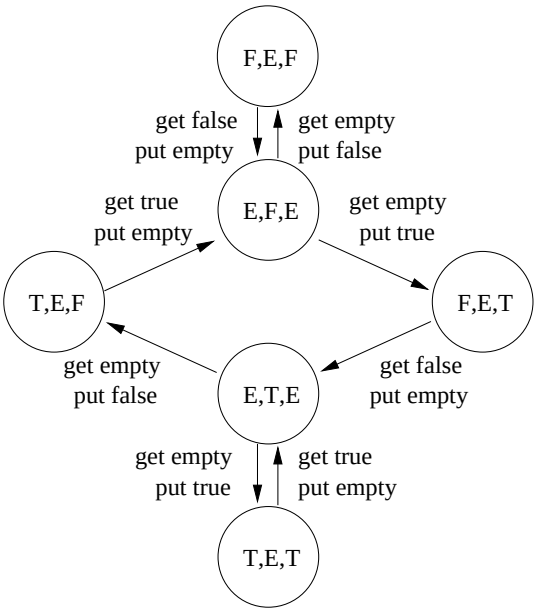


Fig. 6. The specification of an ideal 3-stage buffer. The states correspond to the buffer contents.

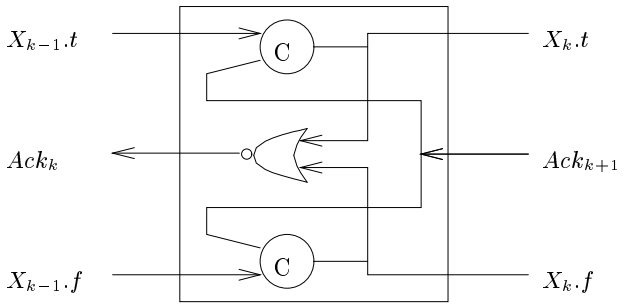


Fig. 7. Stage k of STARI.

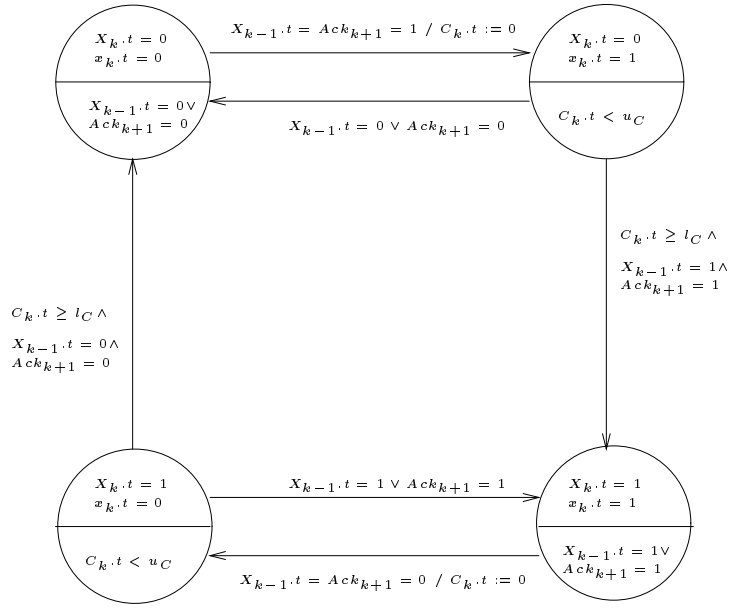


Fig. 8. The timed automaton for the C-element $X_k.t$.

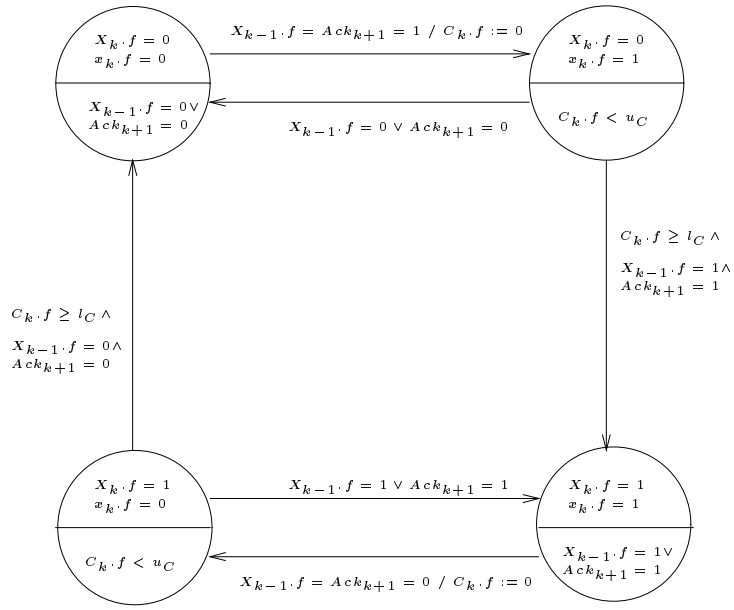


Fig. 9. The timed automaton for the C-element $X_k.f$.

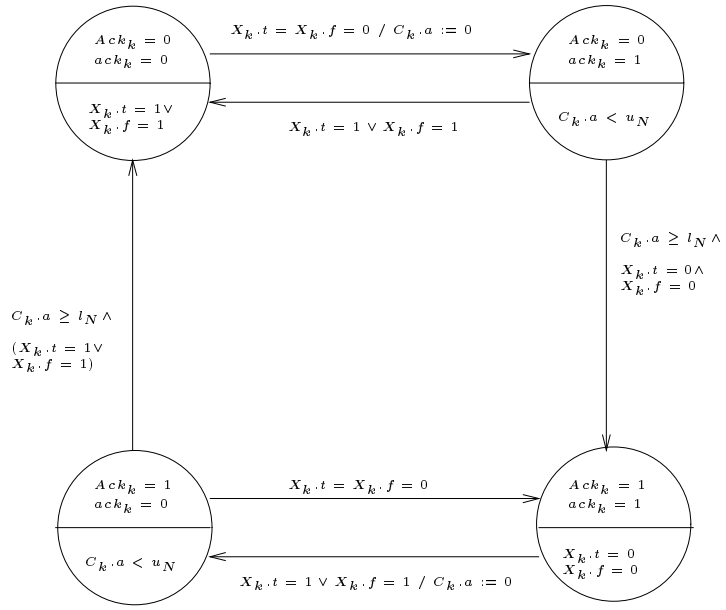


Fig. 10. The timed automaton for the NOR gate Ack_k.

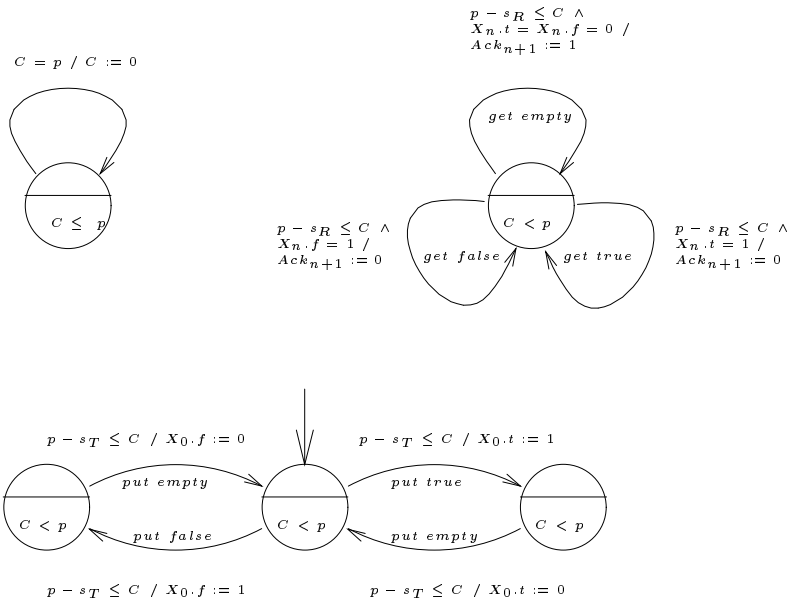


Fig. 11. A global clock with a period p, the receiver and the transmitter.

imized them by hiding the unobservable transitions. In Figure 12 one can see the automaton obtained for three stages, where in addition to the *put* and *get* actions, we left also the *tick* action which indicates the end of the global clock cycle. After hiding the *tick* we obtain a realization of the ideal FIFO as specified in Figure 6.

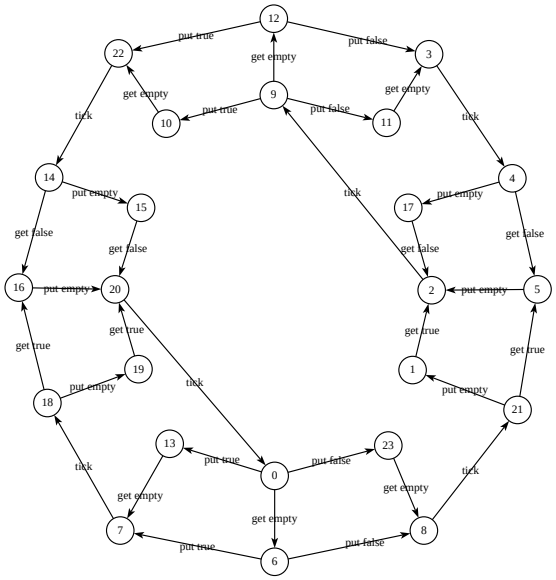


Fig. 12. A three-stage realizations of Stari with internal actions hidden.

We used the following concrete parameters to model the chip: $[l_C, u_C] = [l_N, u_N) = [2, 4]$ for gate delays, $p = 12$ for the clock cycle and $s_T = s_R = 2$ for the maximal allowed skew of the transmitter and the receiver.

The BDD implementation is based on the model-checker SMI [B97] and it uses the CUDD package [S95]. The 18-stage STARI is modeled by a network of

timed automata summing up to 55 clocks. It uses 286 BDD variables to encode the clocks and the states. The reachable state space is of order of 10^{15} states.

We were able to prove that each STARI model with $n \leq 18$ stages, right initialized with m distinct values ($m \sim n/2$) *simulates* an ideal buffer of size m. Moreover, we verified that the transition graphs of the implementation and the specification are equivalent with respect to the *branching* bisimulation [vGW89], if we consider only the reading and writing to be observable. The equivalence is verified symbolically using the method described in [FKM93]. The time and the memory needed[1] to perform this verification are presented in Figure 13.

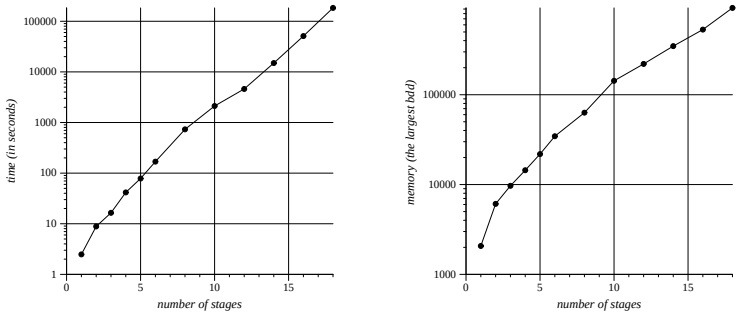


Fig. 13. Time and memory consumption for STARI verification using BDDs.

5.2 Variable-dimension DBMs

We have also verified STARI, interpreted over dense-time, using the DBM representation of Kronos and the forward-analysis technique of [DT98]. To overcome the explosion associated with the size and number of DMBs we have used the techniques of [DY96,DT98], based on the notion of active and inactive clocks.

As one can see in Figure 8, the basic building block which is used to model a timed gate is a four-state automaton with one clock which is active only in the unstable states. So a-priori, each clock is active in half of the global control states. However, in real designs, especially when there is some natural order in which information flows in the circuit, the average over the reachable states of the number of active clocks can be much smaller.

The information concerning clock activity has been extracted automatically from the TA description and,using the variable-dimension DBM library of KRONOS, we were able to verify STARI with up to 8 stages (27 clocks). The main reason for the relative inferiority compared to the BDD approach is the large size of the discrete state-space (2^{24}): using DBMs, discrete variables are enumerated, whereas using discrete time and BDDs all variables (including clocks) are

[1] All the results reported here were obtained on a Pentium II with 512 MB of memory.

handled uniformly, which results in more compact representation. Future techniques, combining BDDs for the state variables and DBMs for the clocks might improve performance significantly.

Figure 14 shows the time performance and the number of symbolic states (discrete variables plus DBM) generated for number of stages. We have also measured the number of active clocks in each symbolic state and the results confirm our expectations that only a small fraction of clocks are active at any time. For instance, in the case of 8 stages, out of 27 clocks at most 9 were active, and this in less than 4% of the total number of DBMs generated (see diagram on the right of Figure 15). In more than 85% of the symbolic states, only 6 to 8 clocks were active. The distributions have the same shape for other STARI configurations.

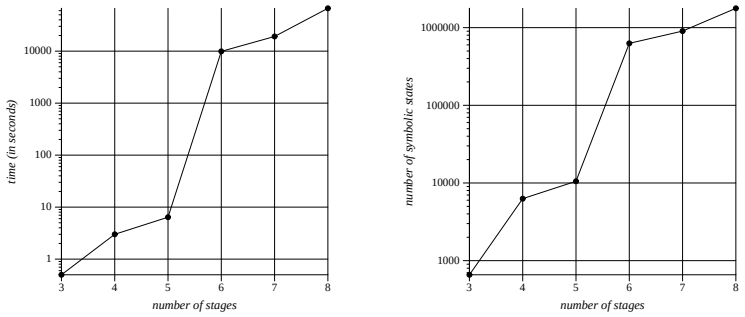


Fig. 14. Experimental results for STARI verification using DBMs.

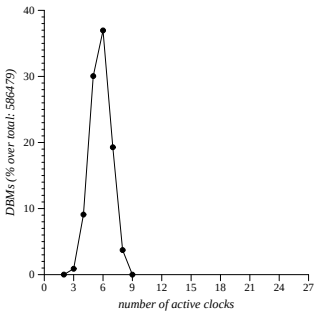


Fig. 15. The distribution of active clocks in an 8-stage STARI.

6 Discussion

Our performance resuts are significantly better than those reported by Tasiran and Brayton [TB97], from whom we have adopted the model. They prove, using techniques developed in [TAKB96], that every stage can be abstracted into a timed automaton having 5 states and only *one* clock. Using this abstract model and the tool Timed-Cospan they were able to verify an 8-stage buffer, while using the detailed model they could not verify more than 3 stages. Another attempt to verify STARI was reported by Belluomini and Myers [BM98] who model the circuit using a variant of timed Petri nets which they verify using the tool POSET which employs partial-order methods. The largest example they reported was of 10 stages. Yoneda and Ryu [YR99] improve these results significantly using circuit-specific heuristics.

We have demonstrated that a rather large example can be verified by tools based on timed automata, and we hope that this will contribute to the wide adoption of timed automata as a model for quantitative timing analysis. Our results indicate that in certain cases, discretized BDD-based approaches outperform other techniques. In the future we will try to characterize the class of systems for which this is the case. It is clear, however, that in examples where large constants (or equivalently, smaller time granularity) are involved, discrete time becomes less attractive.

References

[AD94] R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* 126, 183–235, 1994.

[AH92] R. Alur and T.A. Henzinger, Logics and Models for Real-Time: A survey, J.W. de Bakker et al (Eds.), *Real-Time: Theory in Practice*, LNCS 600, 74-106, Springer, 1992.

[AK96] R. Alur, and R.P. Kurshan, Timing Analysis in COSPAN, in R. Alur, T.A. Henzinger and E. Sontag (Eds.), *Hybrid Systems III*, LNCS 1066, 220-231, Springer, 1996.

[ABK+97] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli and A. Rasse, Data-Structures for the Verification of Timed Automata, in O. Maler (Ed.), *Proc. HART'97*, LNCS 1201, 346-360, Springer, 1997.

[AMP98] E. Asarin, O. Maler and A. Pnueli, On the Discretization of Delays in Timed Automata and Digital Circuits, in R. de Simone and D. Sangiorgi (Eds), *Proc. Concur'98*, LNCS 1466, 470-484, Springer, 1998.

[B96] F. Balarin, Approximate Reachability Analysis of Timed Automata, *Proc. RTSS'96*, 52-61, IEEE, 1996.

[B97] M. Bozga, SMI: An Open Toolbox for Symbolic Protocol Verification, Technical Report 97-10, Verimag, 1997.
`http://www.imag.fr/VERIMAG/DIST_SYS/SMI/`

[BM98] W. Belluomini and C.J. Myers, Verification of Timed Systems Using POSETs, in A.J. Hu and M.Y. Vardi (Eds.), *Proc. CAV'98*, 403-415, LNCS 1427, Springer, 1997.

[BMPY97] M. Bozga, O. Maler, A. Pnueli, S. Yovine, Some Progress in the Symbolic Verification of Timed Automata, in O. Grumberg (Ed.) *Proc. CAV'97*, 179-190, LNCS 1254, Springer, 1997.

[BS94] J.A. Brzozowski and C-J.H. Seger, *Asynchronous Circuits*, Springer, 1994.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, The Tool KRONOS, in R. Alur, T.A. Henzinger and E. Sontag (Eds.), *Hybrid Systems III*, LNCS 1066, 208-219, Springer, 1996.

[DT98] C. Daws and S. Tripakis, Model checking of Real-time Reachability Properties using Abstractions, *Proc. TACAS'98*, LNCS 1384, 1998.

[DY96] C. Daws and S. Yovine, Reducing the Number of Clock Variables of Timed Automata, *Proc. RTSS'96*, 73-81, IEEE, 1996.

[D89] D.L. Dill, Timing Assumptions and Verification of Finite-State Concurrent Systems, in J. Sifakis (Ed.), *Automatic Verification Methods for Finite State Systems*, LNCS 407, 197-212, Springer, 1989.

[FKM93] J.C. Fernandez, A. Kerbrat, and L. Mounier, Symbolic Equivalence Checking, In C. Courcoubetis (Ed.), *Proc. CAV'93*, LNCS 697, Springer, 1993.

[Gre97] M. R. Greenstreet, STARI: Skew Tolerant Communication, to appear in *IEEE Transactions on Computers*, 1997.

[H93] N. Halbwachs, Delay Analysis in Synchronous Programs, in C. Courcoubetis (Ed.), *Proc. CAV'93*, LNCS 697, 333-346, Springer, 1993.

[HMP92] T. Henzinger, Z. Manna, and A. Pnueli. What Good are Digital Clocks?, in W. Kuich (Ed.), *Proc. ICALP'92*, LNCS 623, 545-558, Springer, 1992.

[LLPY97] K. Larsen, F. Larsson, P. Pettersson and W. Yi, Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction, *Proc. RTSS'98*, 14-24, 1997.

[LPY97] K.G. Larsen, P. Pettersson and W. Yi, UPPAAL in a Nutshell, *Software Tools for Technology Transfer* 1/2, 1997.

[L89] H.R. Lewis, Finite-state Analysis of Asynchronous Circuits with Bounded Temporal Uncertainty, TR15-89, Harvard University, 1989.

[MP95] O. Maler and A. Pnueli, Timing Analysis of Asynchronous Circuits using Timed Automata, in P.E. Camurati, H. Eveking (Eds.), *Proc. CHARME'95*, LNCS 987, 189-205, Springer, 1995.

[S95] F. Somenzi, CUDD: CU Decision Diagram Package, 1995.

[SV96] J. Springintveld and F.W. Vaandrager, Minimizable Timed Automata, in B. Jonsson and J. Parrow (Eds.), *Proc. FTRTFT'96*, LNCS 1135, 130-147, Springer, 1996.

[TAKB96] S. Tasiran R. Alur, R.P. Kurshan and R. Brayton, Verifying Abstractions of Timed Systems, in *Proc. CONCUR'96*, 546-562, Springer, 1996.

[TB97] S. Tasiran and R.K. Brayton, STARI: A Case Study in Compositional and Hierarchical Timing Verification, in O. Grumberg (Ed.) *Proc. CAV'97*, 191-201, LNCS 1254, Springer, 1997.

[TKB97] S. Tasiran, Y. Kukimoto and R.K. Brayton, Computing Delay with Coupling using Timed Automata, *Proc. TAU'97*, 1997.

[vGW89] R. J. van Glabbeek and W. P. Weijland, Branching-Time and Abstraction in Bisimulation Semantics (extended abstract), CS R8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989.

[WD94] H. Wong-Toi and D.L. Dill, Approximations for Verifying Timing Properties, in T. Rus and C. Rattray (Eds.), *Theories and Experiences for Real-Time System Development*, World Scientific Publishing, 1994.

[YR99] T. Yoneda and H. Ryu, Timed Trace Theoretic Verification Using Partial Order Reductions, in *Proc. Async'99*, 108-121, IEEE Press, 1999.