Some Progress in the Symbolic Verification of Timed Automata*

Marius Bozga[1], Oded Maler[1], Amir Pnueli[2], Sergio Yovine[1]

[1] VERIMAG, Centre Equation, 2, av. de Vignate, 38610 Gières, France,
{bozga, maler, yovine} @imag.fr
[2] Dept. of Computer Science, Weizmann Inst. Rehovot 76100, Israel,
amir@wisdom.weizmann.ac.il

Abstract. In this paper we discuss the practical difficulty of analyzing the behavior of timed automata and report some results obtained using an experimental BDD-based extension of KRONOS. We have treated examples originating from timing analysis of asynchronous boolean networks and CMOS circuits with delay uncertainties and the results outperform those obtained by previous implementations of timed automata verification tools.

1 Introduction

The computational burden associated with the verification of discrete systems consists in representing and calculating the set of reachable states of a transition system, usually described as a product of small interacting systems. Timed systems were introduced in order to provide a more detailed level of modeling in which it is possible to refine a statement such as *"a is followed by b"* into *"a is followed by b within t time units"*. Timed formalisms for describing systems (timed automata [AD94], [D89], timed Petri nets [BD91], timed transition systems [HMP92a] or real-time process algebras [NS92]) and for specifying behaviors (real-time temporal logics [AH92], timed regular expressions [ACM96]) allow the intuitive expression of real-life phenomena. Among these is the hard fact that *it takes some time between the initiation and a completion of a change* and that quantitative timing information may matter in the future evolution of a system.

In fact, after playing with timed models for some time, one starts wondering about the underlying assumptions that make "classical" untimed reasoning valid and useful. What class of real-timed systems is hiding behind each and every untimed automaton? How are discrete transitions embedded in the real time axis? Without getting too much into the details one can suggest two kinds of answers:

Asynchronous answer: we assume that changes in various system components may take arbitrary amount of time to be accomplished. From this perspective

* This research was supported in part by the European Community projects HYBRID EC-US-043 and INTAS-94-697. VERIMAG is a joint laboratory of CNRS and UJF.

an untimed system can be viewed as a timed system with trivial $[0, \infty]$ bounds on the duration of a transition. Clearly, such an abstraction will create much more executions than a real system would.

Synchronous answer: certain assumptions are made and certain precautions are taken in order to ensure that most of the timing information can be ignored. This is the principle underlying clocked realization of sequential machines: we are not interested in the intermediate states of the next-state logic, nor whether one state variable has changed before the other. What is important is that everybody has stabilized until the next time their values are sampled.

When one is not satisfied with the type of answers suggested by the abstract untimed models or with the performance of clocked systems, timed models seem to be the next logical step (see also [BS94]). It has been shown elsewhere ([D89], [L89], [MP95]) how a very general model of non-clocked circuits with delays can be translated into timed automata, on which one can ask all sorts of interesting timing questions ([ACD93], [HNSY94] [AMP95]). The only problem with these models is the amount of time (and space) that might elapse between posing the question and obtaining the answer. Indeed, it is the performance bottleneck that prevents the transfer of timing verification technology from theory to practice. In this paper we describe some attempts to push forward the performance limitations of current timed automata verification tools by augmenting the tool KRONOS [DOTY96] with an additional BDD-based capability.

The rest of the paper is organized as follows: in Section 2 we discuss, via generic examples, the computational difficulty of timed automata analysis methods and present an alternative data-structure, NDD which is used to analyze the examples in this paper. NDDs are essentially nothing more than BDDs over the bits of discretized clocks. In Section 3 we show the performance of the NDD implementation on benchmark examples coming from asynchronous boolean networks and compare them with other implementations, while in Section 4 we apply NDDs to realistic (but small) examples of MOS circuits with up to 5 inputs and 16 transistors, in order to answer a question motivated by noise problems. We assume that the reader is familiar with the basic definitions of timed automata and with BDDs.

2 The Difficulty of Timing Analysis

Consider a system which can generate events out of a set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, such that every two consecutive occurrences of τ_i must be separated by l_i time units, while every occurrence of τ_i must be followed by another one within u_i units. Such a system can be modeled by the simple one-state timed automaton $\mathcal{A}$ depicted in Figure 1-a having n clocks and n transitions. Calculating the set of reachable clock configurations is needed in order to determine which $\mathcal{T}$-sequences are realizable by the system. An illustration of the calculation of the set of reachable clock configurations for $n = 2$ is given in Figure 2. At the beginning, Time progresses until it reaches the smallest lower-bound (in this case, l_1). Since then, until the first upper-bound is encountered (in this case, u_1) the transition

τ_1 can be taken while resetting C_1 to zero. After crossing the lower bound l_2, transition τ_2 can be, as well, taken, and so on. Although it might look simple for two clocks, this set can become rather complex in more dimensions!

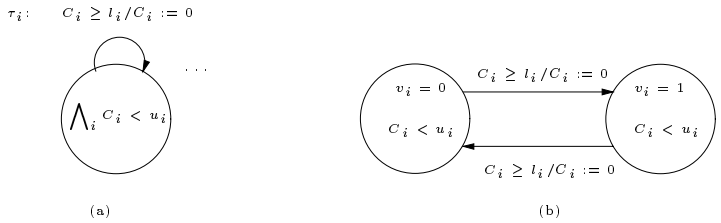

Fig. 1. (a) A one-state automaton $\mathcal{A}$ with n transitions and n clocks. (b) A two-state automaton $\mathcal{B}$ for representing a set of input signals satisfying upper and lower bounds on the distance between two switching points.

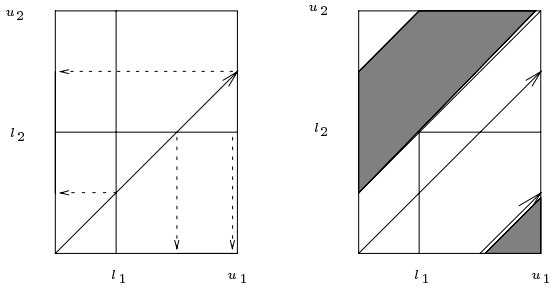

Fig. 2. The initial sets of reachable clock configurations of the automaton in Figure 1-(a) starting from $(0, 0)$.

In general, the sets of reachable clock configurations obtained this way can be expressed as a union of *zones*, that is, convex polyhedra generated by half-spaces of the form $C_i < k$ or $C_i - C_j < k$ for k in some finite subset of the integers.[3] Zones admit an efficient representation using difference-bounds matrices (DBM, [D89]) on which it is easy to calculate intersection and the progress of time. As it often happens in computational geometrical problems, the difficulty comes from the need to manipulate *non-convex* sets. In this case the representation is not canonical and a lot of work is needed in order to determine whether all the reachable states have already been encountered. It may turn out, for example,

[3] We ignore intentionally some technicalities concerning strictness of inequalities.

that a union of zones stored in memory is, in fact, convex and can be replaced by a single zone, but testing this possibility at every iteration is costly. Some authors ([H93], [AIKY95] [B96], [WD94]) try to use various sorts of approximations, e.g. to use convex hulls instead of unions, but these over-approximations often tend to become too large and hence not useful.

The problem aggravates when the untimed state-space is non-trivial. Consider the two-state automaton $\mathcal{B}$ of Figure 1-(b). Such automaton represents a boolean input signal whose only constraint is that every two changes in its value are separated by some time $t \in [l_i, u_i)$. An array of such automata is an unavoidable component in any model for analyzing the behavior of circuits under *all possible inputs*. When two such automata work in parallel, the reachable clock configurations are "distributed" among the discrete states $\{00, 01, 10, 11\}$ as shown in Figure 3. This raises several problems: there might be a lot of redundancy if we represent reachable configurations for every state separately because two states might share zones. In addition, if we use symbolic methods ([BCM+93], [McM93]) to overcome the discrete state-explosion problem, how should they be combined[4] with the DBM representation? Finally, the convergence of the set of reachable configurations into a convex zone is usually slower than in the case of a one-state automaton.

In order to overcome these problems we have devised and implemented an alternative representation scheme for sets of clock configurations, the *Numerical Decision Diagrams* (NDD, [ABK+97]) and tested its performance on these and other examples. This scheme has some major advantages over DBMs (canonicity, natural combination with discrete symbolic representations) but, of course, has its own disadvantages, most notably, the sensitivity to time granularity.

The idea behind NDDs is trivial. Suppose that each clock can take values in the range $[0, k)$, and consider a *discretization* of time such that the possible clock values are $K = \{0, \ldots, k-1\}$. Each clock can be treated as a bounded integer variable and any of its possible values can be encoded in binary using $\log k$ bits. Consequently, any subset of K^n can be viewed as a subset of $\{0,1\}^{n \log k}$ and represented by a BDD over $n \log k$ boolean variables. Given a fixed variable ordering, this representation is canonical regardless of convexity, and it offers BDD-based boolean operations as well as the calculation of the passage of time by simple arithmetical operations.

For dense time models, two discretization schemes has been proposed in [GPV94]. They are based on taking a rational constant Δ, depending on the number of clocks such that by cutting space and time into a Δ-grid, one obtains a discrete-time automaton which is equivalent (for all interesting purposes) to the given dense-time automaton. These two schemes require $\Delta = 1/(n+1)$ and $\Delta = (1/2n)$ respectively, and involve some distortion of the passage of time or of the reset operator in order to preserve the properties of the original dense-time system (a more detailed description appears in [GPV94] and [ABK+97]).

[4] This problem has been addressed by Wong-Toi and Dill [WD94], who combined DBMs and BDDs and recently by Balarin [B96] who encoded matrices using BDDs. Both used the representations for *approximate* reachability analysis.

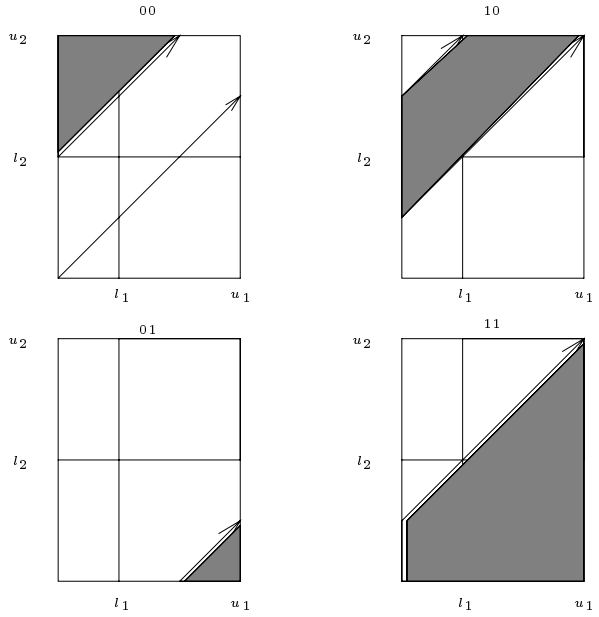

Fig. 3. The initial sets of reachable clock configurations of the automaton in Figure 1-(b) starting from the discrete state 00 and the clock configuration $(0, 0)$.

We have observed, however, that the special class of automata obtained from circuits ([MP95]), where all the clock conditions are of the form $C \geq l$ or $C < u$, admits a slightly simpler and coarser region graph (see also [HMP92]). For these automata, a discretization with $\Delta = 1/n$, where the passage of time is simply the addition of Δ to all the clocks, is sufficient.

Consequently, although all the reported experiments have been performed with respect to the discrete time interpretation, they can be viewed as if we used a dense time interpretation with all the constants divided by n. Approximations are used anyway in order to tackle the complexity of timing analysis ([AIKY95], [H93], [WD94], [B96]), and we believe that playing with the granularity of time might prove to be an alternative approximation strategy.

Note that the NDD-based method is different from calculating the region graph of the timed automaton and *then* trying to encode its transition relation using some choice of boolean state variables (see also [AK96] [CC95]). We build a *uniform discretized state-space* which happens to contain one or more concrete representative of every region, and on which the passage of time is calculated by adding a time unit Δ to every clock variable simultaneously.

We have implemented NDD-based verification algorithms for timed automata by using a system developed at VERIMAG for representing and manipulating communicating automata augmented with bounded integer variables [BFK96].

This system takes such automata and translates them into BDDs using one of several publicly-available BDD packages. We have used the CUDD package [S95] of Colorado University. The experimental results are reported in the following sections.

3 Asynchronous Boolean Circuits

With the NDD representation we were able to calculate within 12 hours all reachable states of the automaton $\mathcal{A}$ (Figure 1-a) with 18 clocks and transitions, while a DBM-based implementation could not treat more than 5 clocks. The relative weakness of DBM in this apparently-trivial example is due to the fact that the set of reachable configurations of this automaton converges finally to the whole clock space, by accumulating more and more zones. We were able to treat products of up to 9 $\mathcal{B}$ automata (Figure 1-b). The results[5] are illustrated in Figure 3. It should be noted, for the fairness of the comparison, that we have used the discrete time interpretation and have chosen clock values in the range {0,...,15} – NDDs are much more sensitive to the granularity of time than DBMs.

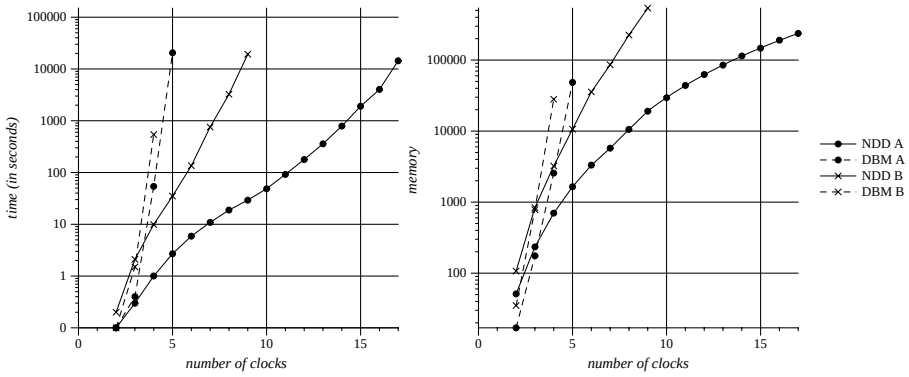

Fig. 4. Comparative performance of NDDs and DBMs for the automata $\mathcal{A}$ and $\mathcal{B}$.

A more complicated example is the family of circuits depicted in Figure 5. For every $i \in \{0, \ldots, n-1\}$ we let the XOR of x_i and x_{i-1} pass through a non-deterministic inertial delay buffer (the exact definitions and the translation procedure from circuits to timed automata are described in [MP95]). Every such gate is modeled by the four-state timed automaton appearing in Figure 6. The states are encoded using two Boolean variables $\mathbf{v}_i$ and v_i, the former denoting the value observed at the exit of the delay element while the latter represents the "hidden" value of the XOR. When both variables are equal we say that the state is *stable* and that it is *excited* otherwise.

[5] Unless otherwise stated, all the results reported here were obtained using a SUN Ultra-Sparc 1 with 256MB of memory.

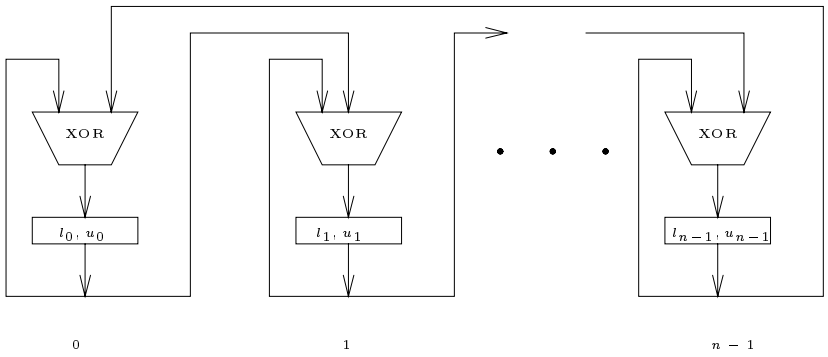

Fig. 5. A cascade of XOR gates with delays.

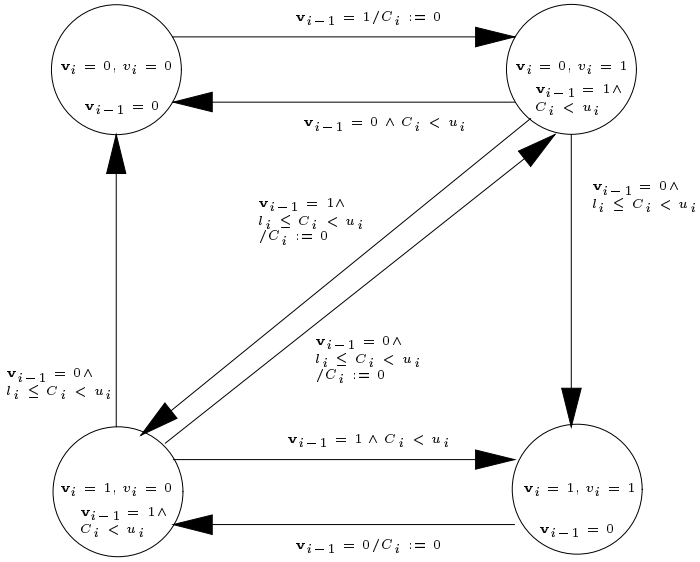

Fig. 6. The automaton for every XOR gate, $i \in \{0, \ldots, n-1\}$.

When n such automata are composed together we obtain a timed automaton $\mathcal{C}$ with 4^n discrete states and n clocks, which we let range in $\{0, \ldots, 7\}$. Note that the feed-back loops make this class of automata rather hard to analyze as all the variables depend on each other. We have managed to calculate all the states reachable from the unstable state $(1, 1, \ldots, 1)$ for a cascade of up to 10 components in less than 2 hours. These results outperformed those of the DBM implementation which could handle only up to 6 gates, using the clock

minimization techniques described in [DY96]. Note that in both implementations it was easier to treat the more logically-involved XOR network $\mathcal{C}$ than the n "independent" inputs of $\mathcal{B}$. This can be explained by the fact that in timed systems, independence of components is an illusion as there is a common shared variable, Time, observed and manipulated by *all* the components. This explains why the BDD results were more modest than initially expected. Nevertheless, the ability to analyze such a non-trivial circuit is remarkable and we could verify that under certain l and u parameters, the stable state $(0, 0, \ldots, 0)$ is never reached.

Concerning variable-ordering, we have found it most efficient to arrange the variables by component such that every discrete variable is followed by the bits of its associated clock with the most significant bit first.

4 MOS Circuits

The next example, motivated by problems related to noise and power consumption, illustrates some pragmatic trade-offs between accuracy and efficiency as well as the effect of other simplifying assumptions on verification performance.

Consider a 4-AND gate implemented by the MOS circuit of Figure 7. We assume that the system is governed by a clock with a period u_X and that the inputs are static, or more precisely: each of the inputs can change its value *at most once* in the sub-interval $[0, l_X)$ and remain constant in the sub-interval $[l_X, u_X)$. Concerning the transistors, we assume that they change their states t pico-seconds after the change of their inputs where $t \in [l_P, u_P)$ for the P-MOS elements (A,B,I and J) and $t \in [l_N, u_N)$ for the N-MOS elements (C,D,L and K). A 4-state timed automaton, similar to the one of Figure 6 can be constructed to model every such transistor.

Although such a circuit is supposed to work in a synchronous environment, some practical problems motivate us to look at what happens on a smaller time scale. A particular question one might want to ask is: "what is the maximal (over all legal input patterns) number of transitions that may take place *simultaneously*?" By a transition we mean the opening or closing of a transistor, which is the main energy consumer. When two many transitions occur simultaneously, it might create noise affecting the behavior of the chip.

While this question might be answered manually for a small circuit, it is not at all clear how to do it for a 8-AND made of 28 transistors, not to mention a 16-AND with 60 transistors, where the internal elements can "change their mind" several times within a clock cycle. It should be emphasized that unlike commonly-used SPICE simulations, where the simulation is done *once* for each input pattern, here the results of the calculations cover *all* possible legal input patterns and all delay uncertainties.

We have transformed the 4-AND circuit into 16 timed automata: 12 for the transistors and 4 for the inputs (the latter share the same clock in the range 0 to u_X), and attempted to calculate the set of reachable clock configurations.

We have kept $(l_N, u_N) = (8, 16)$ throughout the experiments. By taking $(l_P, u_P) = (10, 20)$ and dividing all the constants by $\text{lcm}\{10, 20, 8, 16\} = 2$ we

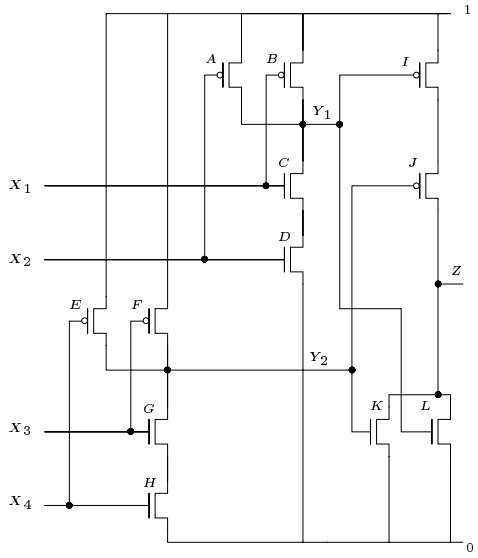

Fig. 7. A MOS realization of the 4-AND function.

had to code all the transistor clocks using 4 bits. Changing l_P from 10 to 8, the lcm becomes 4 and we could use only 3 bits for the clocks. Another factor which influenced performance was the partition of the central clock period into active and non-active phases. Not surprisingly, the results were much better for $(l_X, u_X) = (20, 60)$ than for $(l_X, u_X) = (40, 60)$.

We have constructed an auxiliary automaton for counting the number of transitions taking place at the same time and could test whether there is an input pattern generating more than a given number of simultaneous transitions. For example, concerning the 4-AND circuit, under the parameters $(l_P, u_P) = (8, 20)$ and $(l_X, u_X) = (40, 56)$ we asked whether 9 simultaneous transitions are possible starting from the initial stable state where all the inputs are 0. The system gave (in 1:15 hours) a positive answer and provided the following witness sequence:

$$(X_2 \uparrow, 0) \rightarrow (\{A \downarrow, D \uparrow\}, 8) \rightarrow (X_1 \uparrow, 88) \rightarrow (X_2 \downarrow, 96) \rightarrow (\{B \downarrow, C \uparrow\}, 100) \rightarrow$$
$$(D \downarrow, 104) \rightarrow (I \uparrow, 108) \rightarrow (A \uparrow, 112) \rightarrow (\{X_1 \downarrow, X_2 \uparrow, X_3 \uparrow, X_4 \uparrow\}, 120) \rightarrow$$
$$(\{A \downarrow, B \uparrow, C \downarrow, D \uparrow, E \downarrow, F \downarrow, G \uparrow, H \uparrow, I \downarrow\}, 128)$$

where each pair of the form (S, t) indicates the occurrence of the event (or set of events) S after t pico-seconds since the beginning. The results of the experiments with 3-AND, 4-AND and 5-AND circuits are given in Table 4.[6]

We have also detected the possibilities of short-cuts (a wire connected to both 0 and 1) as we did in [MY96] for a simpler example of a MOS circuit

[6] The results for the 5-AND circuit (17 clocks!) were obtained on a 200MHz PentiumPro with 512MB of memory.

#	test	(l_p,u_p)=(8,20) (l_n,u_n)=(8,16) (l_X,u_X)=(24,56)	(l_X,u_X)=(40,56)	(l_p,u_p)=(10,20) (l_n,u_n)=(8,16) (l_X,u_X)=(20,60)	(l_X,u_X)=(40,60)
3	reach	31.7	1:09.9	1:53.8	5:52.5
	seq#6	(*) 3.7		(*) 6.1	
	#7	1:09.8	(*) 2:21.0	3:23.4	(*) 8:57.7
	#8		2:32.2		10:37.1
4	reach	5:24.7	18:39.4	17:26.1	1:20:04.7
	seq#8	(*) 25.3		(*) 43.0	(*) 56.4
	#9	45:02.1	(*) 1:15:38.6	1:33:07.6	MO
	#10		1:43:02.8		
5	reach	28:24.5	1:45:36.2	1:08:41.7	MO
	seq#10	(*) 2:09.6		(*) 3:02.5	(*) 4:15.3
	#11	9:07:52.4	(*) 4:24:04.4	MO	MO
	#12		(*) 5:02:16.0		

Table 1. A summary of the MOS results. The lines denoted by "reach" correspond to the calculation of the reachable states. The lines of the form "seq#n" correspond to the time it takes to answer whether there exist a sequence of n simultaneous transitions – a positive answer is indicated by $(*)$ and MO denotes memory overflow.

using the DBM version of KRONOS. While some of the assumptions we made in the modeling of transistors deviate from the physical reality (for example, we have adopted a "lazy evaluation" approach concerning transistors whose input becomes "floating", that is, they maintain their previous status), we believe that the approach presented here can be integrated into the design methodology of MOS circuits. Once a suspicious input pattern has been detected by a tool like ours, a full-fledged SPICE simulation, focused around that pattern, can be invoked in order to determine whether or not the alarm is false.

5 Additional Examples

Other experimental results will be reported elsewhere due to lack of space. They include Fischer's mutual exclusion protocol which has become a traditional benchmark for timed automata verification tools ([DOY94], [WD94], [LPY95], [B96]). We managed to calculate the reachable states for 14 such processes. We have also verified (in few minutes) the manufacturing example due to A. Puri, described in [DY95], where timed automata are used as an abstraction of hybrid systems.

6 Conclusions

We have suggested, implemented and tested an alternative method for efficient verification of timed automata. The essence of this method is a canonical repre-

sentation of discretized sets of clocks configurations using BDDs. This method can take advantage of the symbolic representation of the untimed state-space. We were able to treat some examples that could not be treated by state-of-the-art DBM-based tools. Looking more closely at the "bit-structure" of the clock-space allows us to make an informed choice concerning the trade-off between model accuracy and computational hardness, as was demonstrated in the CMOS case-study.

Notwithstanding the achievements, this is still not *the* breakthrough in timed verification. The main reason, as mentioned in this paper, is the hidden dependency between "syntactically-independent" components, which makes the BDDs of the clock part of a system rather big.

Acknowledgement: We have benefitted from the CMOS know-how of Israel Wagner and Ken McMillan.

References

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

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill, Model Checking in Dense Real Time, *Information and Computation* 104, 2–34, 1993.

[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.

[AIKY95] R. Alur, A. Itai, R.P. Kurshan and M. Yanakakis, Timing Verification by Successive Approximation, *Information and Computation* 118, 142-157, 1995.

[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.

[ACM96] E. Asarin, P. Caspi and O. Maler, A Kleene Theorem for Timed Automata, *Proc. LICS'97*, 1997.

[AMP95] E. Asarin, O. Maler and A. Pnueli, Symbolic Controller Synthesis for Discrete and Timed Systems, in P. Antsaklis et al (Eds.), *Hybrid Systems II*, LNCS 999, 1-20, Springer, 1995.

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

[BD91] B. Berthomieu and M. Diaz, Modeling and Verification of Time Dependent Systems using Time Petri Nets, *IEEE Trans. on Software Engineering* 17, 259-273, 1991.

[BFK96] M. Bozga, J.-C. Fernandez and A. Kerbrat, *A Symbolic μ-calculus Model Checker for Automata with Variables*, Unpublished Manuscript, VERIMAG, 1996. `http://www.imag.fr/VERIMAG/DIST_SYS/SMI/`

[Bry86] R.E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, *IEEE Trans. on Computers* C-35, 677-691, 1986.

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

[BCM+93] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, Symbolic Model-Checking: 10^{20} States and Beyond, *Proc. LICS'90*, Philadelphia, 1990.

[CC95] S.V. Campos and E.M. Clarke, Real-time Symbolic Model Checking for Discrete Time Models, in T. Rus and C. Rattray (Eds.), *Theories and Experiences for Real-Time System Development*, World Scientific, 1995.

[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.

[DOY94] C. Daws, A. Olivero and S. Yovine, Verifying ET-LOTOS Programs with KRONOS, *Proc. FORTE'94*, Bern, 1994.

[DY95] C. Daws and S. Yovine, Two Examples of Verification of Multirate Timed Automata with KRONOS, *Proc. RTSS'95*, 66-75, IEEE, 1995.

[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.

[GPV94] A. Göllü, A. Puri and P. Varaiya, Discretization of Timed Automata, *Proc. 33rd CDC*, 1994.

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

[HMP92a] T. Henzinger, Z. Manna, and A. Pnueli, Timed Transition Systems, in J.W. de Bakker et al (Eds.), *Real-Time: Theory in Practice*, LNCS 600, 226-251, Springer, 1992.

[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.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* 111, 193–244, 1994.

[LPY95] K.G. Larsen, P. Pettersson and W. Yi, Compositional and Symbolic Model-Checking of Real-time Systems, *Proc. RTSS'95*, 76-87, IEEE, 1995.

[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.

[MY96] O. Maler and S. Yovine. Hardware Timing Verification using KRONOS, In *Proc. 7th Israeli Conference on Computer Systems and Software Engineering*, Herzliya, Israel, June 1996.

[McM93] K.L. McMillan, *Symbolic Model-Checking: an Approach to the State-Explosion problem*, Kluwer, 1993.

[NS92] X. Nicollin and J. Sifakis, An Overview and Synthesis of Timed Process Algebra, in J.W. de Bakker et al (Eds.), *Real-Time: Theory in Practice*, LNCS 600, 526-548, Springer, 1992.

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

[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.