Computing Timed Transition Relations for sequential cycle–based simulation

Gianpiero Cabodi† Paolo Camurati† Claudio Passerone‡ Stefano Quer†

† Politecnico di Torino
Dip. di Automatica e Informatica
Turin, ITALY

‡ Politecnico di Torino
Dip. di Elettronica
Turin, ITALY

Abstract

In this paper we address the problem of computing silent paths in an Finite State Machine (FSM). These paths are characterized by no observable activity under constant inputs, and can be used for a variety of applications, from verification, to synthesis, to simulation.

First, we describe a new approach to compute the Timed Transition Relation of an FSM. Then, we concentrate on applying the methodology to simulation of reactive behaviors. In this field, we automatically extract a BDD–based behavioral model from the RT or Gate Level description. The behavioral model is able to "jump" in time and to avoid the simulation of internal events. Finally, we discuss a set of promising experimental results in a simulation environment under the Ptolemy simulator.

1 Introduction

Finite State Machines (FSMs) are a convenient formal model for specification, analysis and synthesis of electronic systems. State traversal techniques have been used to characterize the sequential behavior of an FSM, for the purpose of proving equivalence between FSMs, proving properties about an FSM, determining don't cares to be used in synthesis, and so on.

In this paper, we deal specifically with one class of FSMs, in which long sequences of states have no observable external effect, or no external effect relevant to the property being checked. In particular, we are interested in sequences traversed when the primary inputs of the FSM do not change, causing the FSM to "sleep" for a while. We call such state sequences *silent paths*, by analogy with τ, the silent event of process algebras. However, in this case we are concerned with the actual length of such silent sequences, as a rudimentary mechanism to measure time.

This sort of behavior can be used to model a variety of real-life situations, from timing analysis of embedded hardware and software, to verification and synthesis of counter–based circuits (e.g., micro-controller timing units), to cycled–based simulation. For example, a counter coupled with an FSM network can be used to check that a bus protocol modeled by the FSM will serve every request that is being held for a specified amount of time (clock cycles). Similarly, an FSM including a counter, modeling a microcontroller timing unit such as Intel 8251, can be verified to be equivalent (under a given initial programming mode) to a delay for a specified number of clock cycles.

Within the field of formal verification of FSMs, based on state space exploration and the concept of *product machine*, no optimization dealing with time has been proposed. Within the field of model checking, delay issues in real-time problems have been dealt with by resorting to various kinds of timed temporal logics. Recent works concentrate on discrete time expressed by natural numbers and adopt Quantitative Computational Tree Logic (QCTL) for timed formulas [1, 2, 3]. In [1] authors extend standard CTL operators with *quantitative constraints*, i.e., placing bounds on response time. Extended operators are implemented resorting to standard ones. Authors also introduce the *Timed Transition Relations* (TTR) in which state transitions of the original Transition Relation are labeled with time. In [2] the authors use vectors of transition relations, where each function deals with a single time value. In [3] time is represented by terminal nodes of Multi Terminal BDDs. There are advantages and disadvantages in all solutions, a common drawback being represented by compositional problems, where temporal structures require expansion to unit delay structures [3].

The main idea of this paper is the following. Let us suppose to deal with an FSM that "sleeps" for a predefined number of cycles given a set of "constant" values on its primary inputs. Starting from the original Transition Relation of the FSM we compose it with a free–running counter, that measures time. The new object is a Timed Transition Relation TTR, i.e., a TR augmented with a *time input* τ. A sequence of $\hat{\tau}$ *equal input symbols* $\hat{x}$ ($\hat{x}, \hat{x}, \ldots, \hat{x}$, $\hat{\tau}$ times) for the original FSM is represented as the input pair $(\hat{\tau}, \hat{x})$ for the Timed Transition Relation. This TTR allows us to "jump" in time in different situation, e.g., simulation, verification and synthesis, where time plays an important role.

Similar problems are quite often solved with hand–written behavioral models whereas the techniques presented in this paper allow us to derive *automatically* the TTR from a given synchronous Register Transfer or Gate Level specification. If the TTR can be constructed, it can dramatically speed up simulation, because it transforms a sequence of internal events into a single application of a Boolean map, which contains pre-computed *sleep times* of some components, that interact with the rest of the specification only sporadically. Decoupling component simulation can be an effective mechanism, both for multi-processor and for single-processor cycled–based simulation environments. Timed Transition Relations are a means to perform symbolic manipulations *once and for all*, before addressing several simulation tasks. We particularly target circuits containing embedded counters, the knowledge of whom as sub–components may be either given or not. In the former case we simply derive the TTR of the counter, in the latter we don't extract the counter, but we use TTRs to model counting sub–behaviors of the whole circuit.

As far as we know this is the first time such an approach has been proposed. It differs from previous ones, in [1, 2, 3], both in terms of construction technique and of application domain.

The remainder of the paper is organized as follows. Section 2 summarizes some useful concepts on the FSM model, the transition relation and iterative squaring. The Timed Transition Relations model is introduced in Section 3. Section 4 describes how to efficiently compute Timed Transition Relations and Section 5 how to use them in simulation. Section 6 shows some experimental results and Section 7 closes the paper with conclusions and some indications on possible future work.

2 Background

We assume that the reader is familiar with the concept of BDDs and basic operations using BDDs.

A Finite State Machine (FSM) M is defined as

$$M = (I, O, S, \delta, \lambda, S_0)$$

where I is the input alphabet, O is the output alphabet, S is the state space, δ is the next state function, λ is the output function and $S_0 \subseteq S$ is the initial state set. We denote with s current state variables, with x primary inputs, and with y next state variables.

The behavior of a FSM is described using its *Transition Relation*. We indicate a transition relation with $\mathsf{TR}(s, x, y)$. Abstracting primary inputs from TR, we obtain $\mathsf{T}(s, y) = \exists_x \mathsf{TR}(s, x, y)$, usually called *non–deterministic transition relation*.

Given a transition relation, the image (pre-image) of a set of states described by its characteristic function $\mathsf{C}(s)$ ($\mathsf{C}(y)$) according to TR is defined as:

$$\mathsf{Img}(\mathsf{TR}(x, s, y), \mathsf{C}(s)) = \exists_{x,s}\ (\mathsf{TR}(x, s, y) \cdot \mathsf{C}(s))$$
$$\mathsf{PreImg}(\mathsf{TR}(x, s, y), \mathsf{C}(y)) = \exists_{x,y}\ (\mathsf{TR}(x, s, y) \cdot \mathsf{C}(y))$$

For very high sequential depth *iterative squaring* results in an exponential reduction of the number of iterations necessary to reach fixed points. This method relies on computing the transitive closure $\widehat{\mathsf{T}}$ of T, i.e., the least fixed–point of the following recurrence equation:

$$\mathsf{T}^{n+1}(s, y) = \mathsf{T}(s, y) + \exists_z (\mathsf{T}^n(s, z) \cdot \mathsf{T}^n(z, y)) \qquad (1)$$

setting $\mathsf{T}^1(s, y)$ to $\mathsf{T}(s, y)$.

3 Timed Transition Relations

A Timed Transition Relation $\mathsf{TTR}(x, s, y, \tau)$ can be thought of as a $\mathsf{TR}(x, s, y)$ in which the additional input τ denotes *time*. In our usage, primary inputs x are supposed to be held constant (they may or may not have been abstracted away) and under that hypothesis, the time input τ determines the state y (or set of states) reached from state s in τ clock cycles. This object is a compact representation of the temporal behavior of the FSM, and can be used to perform *repeated "timing analysis"* operations *without the need to perform state traversal every time*. In this way, the effort required to compute the TTR from the TR is amortized over a set of operations using it.

Starting from $\mathsf{TR}(x, s, y)$, we represent time by adding to it a special transition relation representing the behavior of an n–bit free–running counter. The behavior of this device can be functionally represented as $y_c = s_c + 1$[1]. Keeping in mind that the least significant bit toggles at each clock cycle, whereas the other bits toggle only with the falling transitions of the previous bits, its Transition Relation has the following form:

$$\mathsf{TR_c}(s_c, y_c) = (s_{c0} \neq y_{c0}) \cdot \prod_{i=1}^{n-1} (s_{c_{i-1}} \cdot \overline{y_{c_{i-1}}} \cdot (s_{c_i} \neq y_{c_i}) + (\overline{s_{c_{i-1}}} + y_{c_{i-1}}) \cdot (s_{c_i} = y_{c_i})) \qquad (2)$$

The free–running counter is used to measure cycles along state paths. The resulting circuit can be represented by a transition relation $\mathsf{TR_g}$[2] expressed as follows:

$$\mathsf{TR_g}(x, s, s_c, y, y_c) = \mathsf{TR}(x, s, y) \cdot \mathsf{TR_c}(s_c, y_c) \qquad (3)$$

in which every state transition has the property of incrementing the free–running counter.

Once we have $\mathsf{TR_g}$ we proceed taking into account the following considerations:

- As we want to represent silent paths, i.e., path with constant primary inputs, we augment TR with edges representing those transitions. We call this augmented transition relation $\mathsf{TR_g^+}$. It is a subset of the general transitive closure of $\mathsf{TR_g}$, $\widehat{\mathsf{TR_g}}$, derived by *closing* only silent paths. Each new transition represents a path of length $\tau = y_c - s_c$.
- We are not interested in the full $\mathsf{TR_g^+}$, but on a subset characterized by a property (e.g., a counting sequence with terminal count at 0 for all steps except the *end–of–count* state). Given a Boolean property $P(x, s)$, which is required to hold only on the last state of the silent paths we observe, we restrict our construction to paths in TR and $\mathsf{TR_g}$ where $\overline{P(x, s)}$ holds for all transitions but the last one. We call that subset $\mathsf{TR_g^+}|_P$.
- Since we are interested only in the difference between the initial and final states of the free–running counter along silent paths (the number of elapsed clock cycles τ), we may consider only $\mathsf{TR_g^+}$ transitions leaving states

[1] From here on, subscript c indicates the variables and the Transition Relation of the free–running counter.

[2] From here on, subscript g indicates the *global* transition relation.

with $s_c = 0$. In this particular case $\tau = y_c$ represents the length of a transition: A relevant simplification of TR_g^+ thus stems from removing all other transitions with $y_c - s_c = \tau$.

The final result actually represents the target *Timed Transition Relation* (whose edges represent *timed transitions*), i.e.:

$$\mathsf{TTR}|_P(x, s, y, \tau) = \mathsf{TR}_\mathsf{g}^+|_{P, y_c \to \tau}(x, s, 0, y, y_c)$$

Where, again, the P subscript represents the property holding only on the last step of the represented transitions, and $y_c \to \tau$ denotes the substitution of variable y_c with variable τ.

4 Computing Timed Transition Relations

A naive approach to compute TR_g^+, or its overestimation $\widehat{\mathsf{TR}_\mathsf{g}}$, would be iteratively squaring TR_g. Silent paths might be produced by avoiding the existential abstraction of primary input variables in Equation 1 (this is equivalent to composing only state transitions with equal inputs). Moreover, computing TR_g^+ would require constraining TR_g with $\overline{P(x, s)}$ before squaring, then selecting only in-going transitions to states where $P(x, y)$ holds. Again squaring would suffer from overestimating the target, since it would consider all paths in $\overline{P}$, even those not leading to P.

The approach is logarithmic in the maximum state path length, but squaring TR_g (with the additional free–running counter) is even more expensive than squaring TR (yet a difficult problem in many cases).

Following [4] we prefer a *linear iteration method* to compute the transitive closure to the *square methodology*. In fact, forward or backward traversals are good ways to find the right balance between step complexity (much lower in linear traversals than in squaring) and number of steps (linear against logarithmic). In our analysis backward traversal has proved to be more efficient, because it may be better focused on the property under observation, than forward traversal (that blindly follows any path from $s_c = 0$, characterized by $\overline{P}$, even if it never reaches P). We find mutual reachability between $s_c = 0$ (free–running counter at 0), and P (property holding at the last step of silent paths) using the following recurrence equation:

$$\begin{aligned}
&\mathsf{TR}_\mathsf{g}|_P^1(x, s, s_c, y, y_c) = \overline{P}(x, s) \cdot \mathsf{TR}_\mathsf{g}(x, s, s_c, z, z_c) \cdot P(x, y) \\
&\mathsf{TR}_\mathsf{g}|_P^{i+1}(x, s, s_c, y, y_c) = \mathsf{TR}_\mathsf{g}|_P^i(x, s, s_c, y, y_c) + \overline{P}(x, s) \cdot \\
&\qquad \exists_{z, z_c}(\mathsf{TR}_\mathsf{g}|_P^i(x, s, s_c, z, z_c) \cdot \mathsf{TR}_\mathsf{g}|_P(x, z, z_c, y, y_c))
\end{aligned}$$

where $\mathsf{TR}_\mathsf{g}|_P^1$ is the set of one step transitions from $\overline{P}$ to P, and all other $\mathsf{TR}_\mathsf{g}|_P^i$ (with $i > 1$) are computed as preimages within the $\overline{P}$ state subspace.

Given the least fixed point $\mathsf{TR}_\mathsf{g}|_P^+$, we compute $\mathsf{TTR}|_P$ by restriction and variable relabeling:

$$\mathsf{TTR}|_P(x, s, y, \tau) = \mathsf{TR}_\mathsf{g}^+|_P(x, s, 0, y, y_c)_{y_c \to \tau}$$

When memory or time efficiency, to completely evaluate a TTR, turn out to be still too high, we scale down the problem to a tractable one by bounding the length of the silent paths considered. We call BTTR (Bounded TTR) a TTR restricted to silent paths shorter than a given length. The use of BTTR in simulation requires more than a single Boolean operation to cope with the full range of time durations (except for times within the bounded range).

Let us define Timed Image ($\mathsf{Img_t}$) an image computation considering only paths of length *exactly* equal to t[3]. For any given t, we express and compute $\mathsf{Img_t}$ as a logarithmic number of image steps, where only transitions whose length is a power of 2 are involved. A recursive definition of $\mathsf{Img_t}$ is the followings:

$$\mathsf{Img_t}(\mathsf{TTR}, \mathsf{From}) = \\
= \begin{cases}
\mathsf{Img}(\mathsf{TTR}(x, s, y, t), \mathsf{From}) & \text{if } t \text{ is a power of } 2 \\
\mathsf{Img_{t-l}}(\mathsf{TTR}, \mathsf{Img}(\mathsf{TTR}(x, s, y, l), \mathsf{From})) \text{ with } (t/2 < l = 2^k < t) & \text{otherwise}
\end{cases}$$

Any path of length t is decomposed as a chain of subpaths whose length is a power of 2. This allows us to pre-compute not the full TTR, but a much simpler relation, that we represent as a vector of n relations (one for each 2^k length, with $0 < k < n$)

$$\begin{aligned}
&\mathsf{PTTR}^0(x, s, y) = \mathsf{TTR}(x, s, y, 1) = TR(x, s, y) \\
&\mathsf{PTTR}^k(x, s, y) = \mathsf{TTR}(x, s, y, 2^k) = \\
&\qquad \exists_z(\mathsf{TTR}^{k-1}(x, s, z) \cdot \mathsf{TTR}^{k-1}(x, z, y))
\end{aligned} \tag{4}$$

We call this PTTR (Power TTR)[4], and we use it in combination with BTTRs to extend their temporal range. Since the latter only include silent paths shorter than a given bound, or the last part of longer silent paths, PTTRs for a P property are restricted to paths in $\overline{P}$.

The technique (vector of relations, one for each time value) is similar to the one presented in [2], but here we use only a logarithmic number of functions/times. A similar approach is also used in [5] to compute a transitive closure of given bounded length, but with the need to perform iterative squaring for any new $\mathsf{Img_t}$ problem.

5 Using TTRs in Simulation

We now consider more in detail the application of TTRs to common cases of silent paths occurring while simulating FSMs. A TTR is able to compute the next evaluation time for a sequential module, within a cycled–based simulator that incorporates some top–level event–driven mechanism (e.g., to partially evaluate a sub-system, or to distribute the simulation over a multi-processor network). We want to compute the next time an output event will occur. Hence the P property is a change of the outputs on the transition between states s and y, being x a constant value.

$$P(x, s) = \exists_y(TR(x, s, y) \cdot (\lambda(x, s) \neq \lambda(x, y)))$$

Let us consider simulation time t. Given $\mathsf{TTR}|_P(x, s, y, \tau)$, the present state s_t and the module input value x_t at t, we

[3] Notice that t iterations of a standard breadth–first traversal consider all paths of length up to t.

[4] The additional free–running counter coding time is not required by a PTTR.

are able to compute *immediately* the time $t+\tau$ and the corresponding state $y_{t+\tau}$ at which next output change has to be scheduled, assuming no further input change (*reactive* behavior).

Of course the scheduled event must be deleted and the module state updated if a new input event occurs at $t+t_1$ with $(t_1 < \tau)$. The correct evaluation of the module state at time $t+t_1$ is done resorting to the Timed Image described in the previous section:

$$s_{t+t_1} = \mathsf{Img_t}(\mathsf{TTR}(x, s, y, t_1), (x_t, s_t))$$

A similar procedure is activated in the case of Bounded TTR (BTTR). Whenever a new input event occurs at time t, with input x_t and present state s_t, two cases are possible: (a) A τ time interval for next output change is found in BTTR, matching with the couple (x_t, s_t), this is the general case previously described. (b) No match exists in BTTR for (x_t, s_t), meaning that a silent path possibly starts, but longer than the time bound; Power TTRs are now used to find a next evaluation time $t+t_1$ and the corresponding state y_{t+t_1} where $t_1 = 2^k$, and k is the largest index for a $\mathsf{PTTR}^k(x_t, s_t, y) \neq 0$, and y_{t+t_1} is the corresponding next state.

6 Experimental Results

We present in this section data concerning the construction of TTRs (BTTRs) and its application in simulation.

We present experimental data on a few benchmarks and a few counters based circuits. s838_1 is an ISCAS'89, while s1512 is an ISCAS'89–addendum'93 circuit. oc_* are circuits [5] originated from the output compare functions of the timing unit of the Motorola 68HC11 micro-controller. Briefly, we have chosen to model and traverse the following functions:

- frc is a 16 bit free–runnning counter with a pre-scaler.
- oc_self a self-triggered fixed period counter.
- oc_fix a fixed period counter.
- oc_prog_abs a programmable absolute counter.
- oc_prog_rel a programmable relative counter.

Observe that frc is also a sub–component of all the other timers. All circuits except s1512 have a single output, associated with the end–of–count condition. In the case of s1512 we have done experiments with the eoc output.

6.1 Building the Timed Transition Relation

For the experiments presented in this section we use a program built on top of the Colorado University Decision Diagram (CUDD) package. The experiments run on a 266 MHz PentiumII processor with 256 Mbytes of main memory, and dynamic reordering (as supported by the CUDD package) active.

Table 1 presents data about building the full TTR with a free–running counter of 12 bits. # L and # I represent number of latches and inputs, respectively. # Max-τ is the maximum length of a silent path. # Nodes is the number

Circuit	#L	#I	Max-τ	#Nodes		#Arcs	Time
				$\mathsf{TR_g}$	TTR_P		[sec]
s838_1	32	34	4095	309	2384	$9.22 \cdot 10^{18}$	87
$\mathsf{s1512}_{eoc}$	57	29	255	17465	26923	$1.29 \cdot 10^{25}$	829
frc	32	1	4095	434	2135	4095	44
oc_fix	59	3	4095	3881	13071	$2.82 \cdot 10^{14}$	237
oc_self	58	2	4095	4963	24008	$5.63 \cdot 10^{14}$	467
oc_prog_rel	59	19	4095	4178	14305	$1.89 \cdot 10^{18}$	255
oc_prog_abs	59	19	4095	3719	19486	$9.22 \cdot 10^{19}$	449

Table 1: Building Bounded TTRs.

of nodes of the BDDs. # Arcs is the number of timed transitions in TTR_P. Memory requirement is not reported but, in all the cases, is limited to a few Mbytes of main memory.

In all cases we were not able to evaluate the full TTR by means of iterative squaring (due to BDD node explosion). Using the technique presented in Section 4, all the experiments required a fairly low CPU time and reasonable sizes for the TTRs/BTTRs.

Table 2 shows a similar set of data of Table 1 for circuit oc_self with different sizes of the free–running counter. # Bit is the number of bits of the free–running counter added to the circuit and limiting the length of the observed silent paths. Mem. indicates the required main memory (in Mbytes) with a cache table of $1,000,000$ entries for the CUDD package. As the table shows, scalability of the problem is good, i.e., we are able to jump in time up to 2^{16} clock cycles.

# Bit	Max-τ	# Nodes		# Arcs	Mem.	Time
		$\mathsf{TR_g}$	TTR_P		[Mbyte]	[sec]
8	255	4943	6573	$3.09 \cdot 10^{13}$	2.5	28
10	1023	4953	7852	$1.41 \cdot 10^{14}$	2.9	248
12	4095	4963	24008	$5.63 \cdot 10^{14}$	5.0	467
14	16383	4973	88756	$2.25 \cdot 10^{15}$	17.0	2584
16	65535	4983	399637	$9.01 \cdot 10^{15}$	48.0	24485

Table 2: Building a Bounded TTR for circuit oc_self with different free–running counter sizes.

Table 3 shows data on PTTRs to be used in connection with BTTRs. n is the number of relations computed by means of simplified squaring (Equation 4). We show BDD size for: $\mathsf{TR} \cdot \overline{P}$, the initial Transition Relation and-ed with $\overline{P}$; PTTR, the array of n relations, where the k-th element (PTTR^k) represents subsets of silent paths of length $l = 2^k$; PTTR^k_{max}, the maximum size relation within the previous set. The last column shows execution times.

Circuit	n	# Nodes			Time
		$\mathsf{TR} \cdot \overline{P}$	PTTR	PTTR^k_{max}	[sec]
s838_1	33	467	5593	469	0
frc	15	436	2584	216	2
oc_fix	20	3044	58463	8098	9
oc_self	20	3261	53348	6814	6
oc_prog_rel	20	2985	94985	11992	31
oc_prog_abs	20	2423	101036	10198	13

Table 3: TTR Approach.

BDD size and execution time are largely within an acceptable range for state-of-the-art BDD packages. This is a good starting point for the experimental results described in Section 6.2, because TTRs, BTTRs and PTTRs are computed once and forall and used for several simulations.

6.2 Simulation in the Polis–Ptolemy environment

To evaluate, in terms of simulation speed, the technique presented so far we decided to work in the *Polis* [6] environment with the *Ptolemy* [7] simulator.

Polis is centered around a single FSM–like representation, which is well suited to our target class of control–dominated systems. Ptolemy treats the system to be designed as a hierarchical collection of objects, described at different levels of abstraction and using different semantic models to communicate with each other. In particular, we used the Discrete Event domain of Ptolemy to implement the event-driven communication mechanism. This domain is event–driven, rather than data-driven as most other domains in Ptolemy, and hence seems the most appropriate for our purposes. In fact BDDs representing the TTR can be loaded into the Polis environment and used for simulation purposes. Polis already provides an RTL library for the counters, and we derived from them TTR–based behavioral models in which next output change is immediately computed when counting is started.

The results obtained are summarized in Table 4. We used a Sun Enterprise 450 server, with 128 Mbyte of RAM, running at 250 MHz.

We concentrate on the free–running counter with prescaler frc and on the oc_* timers. The complex experiment is produced by using four oc_fix in parallel and the RTL description is optimized with the four timers sharing the same frc counter.

The second column (RTL) indicates the CPU time for the RTL simulation and the third one the CPU time for the TTR–based behavioral simulation. The last column shows the ratio between the two.

To have meaningful timing data, we always simulated 500, 000 clock cycles. As we can reasonably compute TTR up to a depth of 2^{14} we perform experiments in which counting phases of different length are repeated several times (e.g., counting phases up to 1000 repeated 500 times). Experimental data do not vary too much if the depth of counting is changed. Therefore we decided to report only average values over a certain number of runs.

Simulation times are very similar for all the circuits under test; this is due to the fact that the function of the circuits does not change too much. The speedup has a minimum value of 34.9 for the complex circuit. In all cases we are able to substantially decrease simulation time, without affecting the behavior of the simulation.

Circuit	RTL [sec]	Behavioral [sec]	Ratio
frc	41.6	0.6	69.3
oc_fix	77.7	0.8	97.1
oc_self	84.0	1.0	84.0
oc_prog_rel	74.4	1.6	46.5
oc_prog_abs	73.8	1.2	61.5
complex	195.4	5.6	34.9

Table 4: Simulation CPU times in the Ptolemy environment.

7 Conclusions

In this paper we address the problem of dealing with *silent paths* of FSMs, i.e., paths that are characterized by no observable activity under constant inputs. In particular, we describe how to solve these problems, using Timed Transition Relations. We first examine how to efficiently compute Timed Transition Relations. Then, we apply them on logic simulation. In this field, we are able to *automatically* extract a BDD–based behavioral model (from the RT or Gate Level description) able to "jump" in time and to avoid the simulation of internal events. Experimental results regarding the construction of the relation and its use in the Ptolemy simulation environment (under Polis) are reported. They show the potential usefulness of the approach.

References

[1] S. V. Campos and E. Clarke. Real–Time Symbolic Model Checking for Discrete Time Models. *T. Rus and C. Rattray Editors, Theories and Experiences for Real–Time System Develpment, AMAST Series in Computing*, May 1994.

[2] J. Froβl, J. Gerlach, and T. Kropf. An Efficient Algorithm for Real-Time Model Checking. In *Proc. EDAA/ACM/IEEE ED&TC'96*, pages 15–21, Paris, France, March 1996.

[3] T. Kropf and J. Ruf. Using MTBDDs for Discrete Timed Symbolic Model Checking. In *Proc. EDAA/ACM/IEEE ED&TC'97*, pages 182–187, Paris, France, March 1997.

[4] Y. Matsunaga, P. .C. McGeer, and R. K. Brayton. On Computing the Transitive Closure of a State Transition Relation. In *Proc. ACM/IEEE DAC'93*, pages 260–265, Dallas, Texas, June 1993.

[5] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Verification and Synthesis of Counters based on Symbolic Techniques. In *Proc. EDAA/ACM/IEEE ED&TC'97*, pages 176–181, Paris, France, March 1997.

[6] F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli. *Hardware-Software Co-design of Embedded Systems – The POLIS experience.* Kluwer Academic Publishers, 1997.

[7] J. Buck, S. Ha, E. A. Lee, and D. G. Masserschmitt. Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems. *International Journal of Computer Simulation*, special issue on Simulation Software Development, January 1994.