Timed SystemC Waiting-State Automata

Nesrine Harrath & Bruno Monsuez
UEI, ENSTA, 32 Bd Victor, 75739 Paris cedex 15, France
nesrine.harrath, bruno.monsuez@ensta.fr

Keywords: SystemC, time analysis, compositional verification, automata, model-checking

Abstract

System-Level Modeling using system-level languages like SystemC or SystemVerilog is gaining more and more popularity. They are supposed to provide the guarantee of critical functional properties about the interaction between concurrent processes like determinism or liveness up to a basic unit, the delta-cycle. Additionally to this functional correctness, system level models should also provide valuable information about important non-functional properties like time constraints. Since timing properties (execution times, delays, periods, etc.) are especially important in performance verification of multiprocessing real-time embedded systems [1], we propose a formal model based on SystemC waiting-state automata [2] that conforms to the SystemC scheduler up to delta-cycles (1) and that also conforms to the provided time constraints (2).

1. INTRODUCTION

With the increasing complexity of today's systems, there is a growing need for System-Level Modeling Languages that can be used to describe systems at a high level of abstraction. It would be helpful if a single high-level model of the system could be used to implement both the hardware and the software components of the system.

SystemC is one such System-Level Modeling language based on C++ that is intended to enable system level design in response to the need of a very fast executable specification to validate and verify system concepts. It is a SoC language representing functionality, communication, software and hardware at various levels of abstraction, it provides structural description features including modules and ports, it also introduces channels, interfaces, and events to enable communication and synchronization between modules or processes.

SystemC simulation engine introduces the important notion of delta-cycle as the fundamental simulation unit. In fact, a simulation procedure can be seen as a sequence of delta cycles and all the interactions within a delta cycle are abstracted from the modeling perspective. Such an abstraction is supposed to provide a guarantee that all these interactions should work correctly so that higher-level analysis or verification can be done without considering what goes on inside delta-cycles. However, this is probably the most error-prone part of SystemC modeling, as the final process space within a delta-cycle can be very large and the interaction in between might be extremely complicated. A typical problem is the causality waiting cycles among processes, *i.e.* a process triggers another process that in response triggers the first process. In such case, the scheduler will never generate the next clock tick. Another typical issue is the initialization of shared resources. When starting a simulation process, all delta events as well as all runnable processes are triggered by different clocks. Depending on the order of the triggered signals, the initial values or states of the shared resources may be different. This may result in non-deterministic behavior at the delta-cycle level. Both behaviors are certainly not desired in system level design. However, the designer should first ensure that no causality cycles are present in his design and then that no race condition occurs between the initial delta cycle when starting the processes and that the processes get initialized in the right sequence.

To address the previous issues and ensures correct design, we introduce a formal model for verifying SystemC [2] system design up to delta-cycle that take into account information about execution time. We first abstract the concrete SystemC design in order to extract a representation that only includes the process related information (execution and activation events). Semantics of SystemC scheduler is preserved in our model since it is based on how processes get into and get out from a waiting state. As time management is required in simulations to ensure temporal aspects of the system [6], the formal model is annotated with time information. This extension of SystemC waiting state automaton with timing properties may be used to predict the timed behavior of system level design of embedded systems (1) and to verify if the timing constraints are guaranteed through the direct analysis of processes waiting states (2). We then define the composition and reduction algorithms to compose and reduce timed SystemC waiting-state automaton and to reduce the resulting automaton. We finally present an extension of this model that consists in combining timing properties and counters that counts the number of time an edge has been activated. Those automata can finally be used to ensure that real-time constraints of the whole system are verified [3] as well as to determine the most appropriate scheduling policy for each process and to estimate the whole time execution.

The paper is structured as follows: Section 2 is an introductory section on SystemC, through a simple example: a clocked FIFO model. We describe in particular how can we model time in SystemC and how important it is. Then we introduce in Section 3 an extension of the waiting-state automata, by annotating transitions with time and we determine temporal properties of the obtained automata, in Section 4 we present another extension of the waiting-state automata, by adding both time and counters and we conclude the paper in section 5.

2. SEMANTICS OF SYSTEMC

The power of SystemC lies in extending C++ with hardware primitives, together with a simulation kernel. The difference between a system-level modeling language like SystemC and C++ is that the system-level modeling primitives are used to model a system, but not to implement it.

With such features, SystemC is able to support multiple abstraction levels and refinement capabilities ranging from high-level functional models to low level timed, cycle-accurate, and RTL models. Various means are provided to represent communication protocols and channels in high-level modeling. In addition, a help library is made available to assist users with signal and timing details either through high level transactions or low-level communications.

Some of the distinctive characteristics of SystemC are informally summarized below:

- A process is in the ready state when either the SystemC program starts or there is an event that the process is waiting for. A process is in the waiting state when it is waiting for an event. A process is unblocked when it is notified by an event. Events can be notified immediately, or the notification can be delayed until all processes are waiting.
- Time is modeled through macro-time in some pre-defined quantifiable unit; a process waits for a given amount of time, expiration of which is notified through an event.
- Processes communicate through reading and writing signals. During the execution of a SystemC program, all signal values are stable until all processes reach the waiting state. When all processes are waiting, signals are updated with the new values.
- Transaction-level communications are through channels, which are accessed using an interface defined by a set of methods. The transaction-level model (TLM) interface can be put and get methods, to connect to channels like FIFO buffers etc, or a custom set of methods to connect to specific channels.

SystemC, respecting all requirements above, has come into prominence as the most appropriate solution for raising the abstraction level to accommodate system level design. A system conceived by SystemC demonstrates particular characteristics in concurrency, reactivity, distributiveness, timing, and data types. But if we focus on timing properties, we find that the major difficulty of implementing the global time accuracy in the waiting state automata's is the task management

Program	{modules, channels, signals, events, variables}
Module	{ports, variables, process-decl, process-body, methods}
Process-decl	$\langle \text{processname} \rangle \langle \text{sensitivity} \rangle \langle \text{reset} - \text{condition} \rangle$
Process-body	$\langle \text{event} - \text{comm} \text{signal} - \text{comm} \text{chan} - \text{comm} \text{control} - \text{flow} \text{arithmetic} \rangle$
Event-comm	$\text{wait}(\text{event}), \text{wait}(\text{event}, \text{time}), \text{wait}(\text{time}), \text{wait}(\text{eventlist}), \text{notify}(\text{event}), \text{notify} - \text{delayed}(\text{event})$
Signal-comm	$\text{signal.read} \text{signal.write}$
Chan-comm	$\text{tlm_port} \rightarrow \text{put}(\text{value}) \text{tlm_port} \rightarrow \text{get}(\text{var}) \text{tlm_port} \rightarrow \text{method}(\text{parameters})$
Control-flow	$\langle C++\text{controlflow} \rangle$
Arithmetic	$\langle C++\text{arithmetic} \rangle$

TABLE 1: Simplified abstract syntax for SystemC.

based on timing but not on event, for this reason this paper gives solutions and algorithms about how to express temporal information in systemC.

2.1. Time and SystemC

One of the most important components in the SystemC library is the model of time. However, this is hidden to the programmer through the data type (class) *sc_time*. Therefore, in SystemC there is a minimum representable time quantum, called the *time resolution*. This can be set by the user. Note that this time resolution limits the maximum representable time, thus, any time value smaller than the time resolution will be rounded to zero. Today, SystemC can deal with the important notion of continuous time, not only discrete time. Our model exploit this approach in order to add continuous timing properties to SystemC models within delta-cycles.

2.2. A simple annotated FIFO model

We start with a simple SystemC model: an implementation of FIFO with clocks. The model contains a First-In-First-Out buffer and two modules cooperating through the buffer:

- the producer randomly generates data that is immediately sent to the consumer.
- the consumer requests data at randomly-chosen points of time.

FIFO: Timed FIFO

write,t: w_event at t time.
read,t: r_event at t time.

FIGURE 1: A Timed FIFO

The necessary synchronization is performed by the FIFO that takes and delivers the generated data on demand. The two processes are triggered by two individual clocks: *p_clock* and *c_clock*. When a *p_clock* signal arrives, the producer starts producing and tries to write the product into

the buffer. Similarly, the *c_clock* signals control the consumer. The two clocks are independent, hence the producing and the consuming can be at different paces. Modeling is usually done with the perspective of performing analysis. Modeling for analysis requires building a model of the *system* **and** of the *environment*. In the context of real time systems the environment includes time properties and constraints. However, this model (the FIFO model) is deprived of temporal annotation, e.g when a producer starts producing and writing the product into the buffer and for how much time? besides there is no information about the duration of each event like *w_event* or *r_event*, about communication delays and the execution time. As a result the satisfaction of real time constraints cannot be verified.

Modeling the time environment of systems means imposing constraints on time spent in transitions, in communications and also in all other implicitly defined activities. To overcome these limitations, we annotate the FIFO model with time information, so that we eliminate ambiguity about time execution.

We take into account the synchronization between the two processes producer/consumer when releasing/filling buffer. It is not only the occurrence of an event that determines which process will be executed but also the time at which this event occurs, i.e the consumer can not start to read before the consumer has finished to write. We have two possible execution models, if the execution model is a sequential model, the time annotations are ordered, if the execution model is a concurrent model, we introduce time annotations as partially ordered.

2.3. Events and time

SystemC is a modeling language based on C++ that is intended to enable system level design in response to the need of a very fast executable specification to validate and verify system concepts [7]. It provides structural description features including modules and ports that can be used in systems design. In addition, there exist different data types to enable modeling hardware systems, processes to express concurrency, channels, interfaces, and events to enable communication and synchronization between modules or processes. Despite of the diversity in syntax, SystemC is essentially an event-driven model and all communications in SystemC models are implemented using events and the associated wait/notify mechanism. The only possible action on an event is to trigger it. This is done with the notify function. A SystemC event is the result of such a notification, and has no value and no duration, but only happens at a single point in time and possibly triggers one or several processes. For instance, the FIFO module is actually defined as a channel *sc_channel*, and the mutual access to the buffer is implemented through the two events *r_event* and *w_event*.

SystemC events can be roughly classified into the following three sorts:

User-defined events These are events defined by SystemC programmers in source code. For instance, the *w_event* and the *r_event*. Such events are usually triggered by the command `notify`;

Channel events These are pre-defined SystemC events and they are triggered when something occurs on channels. For instance, an event denoting the arrival of a new value will be triggered whenever some value is written to a *sc_buffer* channel. Channel events at different types of channels have different semantics;

Clock events Clock signals are also seen as events and they are usually defined as *sc_clock*'s in the SystemC main program. SystemC core engine is in charge of generating *sc_clock* events at proper time. We never notify a clock event in the program.

We shall not distinguish between the first two sorts of events since both of them can be dynamically notified. Clock events are rather seen as the input events of SystemC models. In fact, we prefer not considering any pre-defined channels or signals in our modeling, but rather taking into account their event-driven implementation.

An *sc_event* can be notified using *immediate* or *timed* notification. Immediate notification of *e* is achieved with *e.notify()*; The statement *e.notify(t)*; triggers the event *e* at least at *t* time of simulated time (it overrides any previous timed notification that did not yet occur).

A set of timing constraints is generated for each occurrence of an event or exactly when a transition is triggered. The above procedure formalize the concept of annotation or modeling time for high level synthesis:

```

current_time = 0;
while(Simulation is not achieved)
{
    wait till the event E occurs;
    current_time = get the time of the event E;
    executes the process;
    annotates the transition with the current time,
                                and the execution time (duration) of the transition
}

```

2.4. SystemC simulator

Most HDLs, VHDL for example, use a simulation kernel. The purpose of the kernel is to ensure that parallel activities (concurrency) are correctly modeled. The behavior of the simulation should not depend on the order in which the processes are executed at each step in simulation time. The SystemC simulation kernel introduces the notion of delta cycles. A delta cycle consists of an evaluation phase and an update phase. This is typically used for modeling primitive channels that cannot change instantaneously. By separating the two phases of evaluation and update, it is possible to guarantee deterministic behaviors.

Here is a brief description of a delta-cycle: a delta-cycle starts with a non-empty runnable process table. The scheduler executes these processes one by one, in a pre-defined order; every runnable processes executes until it ends or it is pended again (by a wait command for instance); if any immediate event (e.g. the *w_event* generated by the producer process in the FIFO example) is notified during the execution of runnable processes, it will add processes that are currently sensitive to this event into the runnable process table; delta-events and timed events that are generated during the execution of a process are stored in other tables. The process table is emptied when all runnable processes are executed and the procedure of executing all the runnable processes is called an evaluation phase. The scheduler then checks those delta-events notified in the evaluation phase: if there are processes that are sensitive to these events, then add them into the process table. This procedure is called a delta notification phase. If the process table is non-empty, the scheduler enters the next delta-cycle and executes the evaluation phase again; otherwise, it checks the timed events notified in the evaluation phase and adds processes that sensitive to these timed events into the process table. This is called a timed notification phase. The scheduler then advances the simulation clock and enters the next delta-cycle. SystemC simulation kernel contains a scheduler to determine the order of execution of processes within the design based on the event sensitivity of the processes and the event notifications which occur. It supports both software and hardware-oriented modeling.

3. EXTENDING SYSTEMC WAITING STATE AUTOMATA WITH TIME

Unlike some other formal models such Petri nets [5] and Finite-State Automata that are used to model concurrent distributed systems, the WSA [2] is a formal model for verifying SystemC components that is based on the waiting states of processes within a delta cycle and that conforms to the SystemC scheduler up to delta cycle. The basic idea of Waiting-State Automata's is to consider only the states where processes are hung up, i.e. waiting for some events. In fact, a delta-cycle always starts from a state where all the processes are hung up and the whole cycle can be seen as a sequence of transitions between these waiting states. A main drawback of this model is that all information about time properties get lost. First, refining SystemC code with

respect to the delta-cycle semantics abstract delta-cycles to untimed atomic transitions. Secondly, during SystemC simulation, precise information about execution time are not available.

To overcome those limitations, we annotate each transition with some additional information about time properties and time constraints. More precisely, transitions are timed; ie. a transition can only be activated if a given time condition is verified and it also defines how long it takes to be executed.

Let us illustrate this on the FIFO example and consider the two small automata for the processes of the producer and the consumer. Figure 2 shows both automata annotated with times. For

FIGURE 2: The timed automata for the producer and the consumer

instance, t_r^1 represents the time when the transition from c_wait_p state to c_wait_clk state starts, it is actually the time when the consumer is hung up waiting for the producer to write on the buffer. Besides, d_r^1 presents the period of time during which the consumer read from the buffer. These two values provide meaningful information about the execution time and the behavior of the whole system. For instance, a *read* can only occur if a *write* occurs before it, this property can be represented using durations, i.e., the starting time of transition representing an r_event after a w_event is always more or equal to the sum of the starting times and durations representing a w_event . Formally, it should always hold that:

$$\begin{cases} t_r^1 \geq t_w^1 + d_w^1 \\ t_r^1 \geq t_w^2 + d_w^2 \end{cases}$$

And it is the same for a starting time of transition representing a w_event .

$$\begin{cases} t_w^1 \geq t_r^1 + d_r^1 \\ t_w^1 \geq t_r^2 + d_r^2 \end{cases}$$

As you see a transition is characterized by:

- Its functional triggering conditions we are talking about *entry* and *exit* conditions for example the condition to entering c_wait_p state is $num_elem=0$ and the condition for exiting the c_wait_clk state is a c_clk ,
- Events and transition function or relation (predicat), in the fifo example we're talking about conditions on num_elem ,
- A time dependent enabling conditions, expressing at which time the transition is possible,
- An attribute: delay, giving information about the duration that may take each process to change its state and an event to be notified,

- Finally a function that modifies variables for example *dec* to decrement *num_elem* and *inc* to increment it.

Definition A timed SystemC waiting-state automaton over a set V of variables, is a 5-tuples $A(V) = (S; E; T; \Upsilon; D)$, where S is a finite set of states, E is a finite set of events, Υ is a finite set of transition starting times, D is a finite set of transition duration times and T is a finite set of transitions where every transition is a 8-tuples $(s; e_{in}; p; t; e_{out}; f; d; s')$:

- s and s' are two states in S , representing respectively the initial state and the end state of the transition;
- e_{in} and e_{out} are two sets of events : $e_{in} \subseteq E; e_{out} \subseteq E$;
- p is a predicate defined over variables in V , T and D , i.e., $FV(p) \subseteq V \cup T \cup D$, where $FV(p)$ denotes the set of free variables de in the predicate p ;
- t is the starting time associated to the transition that increments each time it is executed;
- f is an effect function over V ;
- d is the duration of the transition.

Our strategy of verifying SystemC models using timed waiting-state automata requires first building a minimal-step automaton for every process, inferring relations over transitions timers (starting time and duration) and finally composing them together to build a larger automaton that can be passed to a model-checker. The symbolic composition of two automata $A(V)$ and $A'(V)$ works similarly as in the case of standard waiting-state automata. What makes the composition a bit more complicated is that values of times of the automata $A(V)$ and $A'(V)$ do not correspond to the values of times of the composed automaton $A(V) \times A'(V)$.

A transition τ of the automata $A(V)$ may be composed with many transitions $\{\tau'_1, \dots, \tau'_n\}$ of the automaton $A'(V)$, and the values of t and d of the transition τ in $A(V)$ should be represented in the values of times for $(\tau \times \tau'_0), \dots, (\tau \times \tau'_n)$. If t_τ^k and d_τ^k denote respectively the starting and the duration time associated to the transition $(\tau \times \tau^k)$ in the composed automata, then $t = \min_k t_\tau^k$ and $d > \sum_k d_\tau^k$. As the transition predicates (i.e., guard conditions) in timed waiting-state automata are defined over the timers and system variables, the composition should ensure that these predicates of component automata are properly translated in the composed automaton. In particular, we must replace all the occurrence of a transition starting time and duration from component automata, with the values t and d recently presented.

We consider a timed automaton as deterministic when it does not contain any non-deterministic transition. In the untimed case a deterministic transition has a single start state and from each state, given the next *sc_event*, the next state is uniquely determined. We want similar criterion for determinism for the timed automata: given an extended state and the next input *sc_event* along with its time of occurrence, the extended state after the next transition should be uniquely determined.

Algorithm 1 (Symbolic composition) Given two timed SystemC waiting-state automata $A(V) = (S; E; T; \Upsilon; D)$ and $A'(V) = (S'; E'; T'; \Upsilon'; D')$, over a set V of variables, the combination of the two timed SystemC waiting-state automata is still a timed SystemC waiting-state automaton $(S \times S'; E \cup E'; T''; \Upsilon''; D'')$ written as $A(V) \times A'(V)$ where T'' is the smallest set of transitions, Υ'' the associated set of the starting times and D'' is the corresponding set of durations, $\Pi(s; e_{in}; p; t; e_{out}; f; d; s')$ is the set of times of Υ'' and D'' associated to a transition in $T \times T'$ and M_t a morphism that maps starting times in $\Upsilon \times \Upsilon'$ to Υ'' , M_d a morphism that maps duration times in $D \times D'$ to D'' such that:

$$\begin{aligned} \bullet \quad & \Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2) \quad := \quad \{t^*, d^*\} \cup \Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2) \quad \text{and} \\ & (s_1, s'_1) \xrightarrow[e_{out}, f', d^*]{e_{in}, M_t(p), M_d(p), t^*} (s_2, s'_1) \in T'' \text{ for every transition } s_1 \xrightarrow[e_{out}, f, d]{e_{in}, p, t} s_2 \in T \text{ and for} \end{aligned}$$

- every state $s'_1 \in S'$ such that for every transition $s'_1 \xrightarrow[e'_{out}, f', d', s'_2]{e'_{in}, p', t'} s'_2 \in T'$, either $e'_{in} \not\subseteq e_{in}$ or
- $p \not\Rightarrow p'$;
 $\Pi(s'_1; e_{in}; p'; t'; e'_{out}; f; d'; s'_2) := \{t^*, d^*\} \cup \Pi(s'_1; e'_{in}; p'; t'; e'_{out}; f; d'; s'_2)$ and
 $(s_1, s'_1) \xrightarrow[e'_{out}, f', d^*]{e'_{in}, M_t(p'), M_d(p'), t^*} (s_1, s'_2) \in T''$ for every transition $s'_1 \xrightarrow[e'_{out}, f', d']{e'_{in}, p', t'} s'_2 \in T'$ and
 for every state $s_1 \in S$ such that for every transition $s_1 \xrightarrow[e_{out}, f, d]{e_{in}, p, t} s_2 \in T$, either $e_{in} \not\subseteq e'_{in}$ or
 $p \not\Rightarrow p'$;
 $\Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2) := \{t^*, d^*\} \cup \Pi(s_1; e'_{in}; p; t; e_{out}; f; d; s_2)$ and
 $\Pi(s'_1; e'_{in}; p'; t'; e'_{out}; f; d'; s'_2) := \{t^*, d^*\} \cup \Pi(s'_1; e'_{in}; p'; t'; e'_{out}; f; d'; s'_2)$ and
 $(s_1, s'_1) \xrightarrow[e_{out} \cup e'_{out}, f \cup f', d^*]{e_{in} \cup e'_{in}, M_t(p \wedge p'), M_d(p \wedge p'), t^*} (s_2, s'_2) \in T''$, for every pair of transitions
 $s_1 \xrightarrow[e_{out}, f, d]{e_{in}, p, t} s_2 \in T$ and $s'_1 \xrightarrow[e'_{out}, f', d']{e'_{in}, p', t'} s'_2 \in T'$.
 - According to the transition $(s_1; e_{in}; p; t_{s_1}^{s_2}; e_{out}; f; d_{s_1}^{s_2}; s_2)$, the morphism M_t maps the starting t to the *min* of transition starting times defined in $\Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2)$

$$M(t) \rightarrow \min_{t^* \in \Pi(s_1, e_{in}, p, t, e_{out}, f, d, s_2)} t^*$$
 - According to the transition $(s_1; e_{in}; p; t_{s_1}^{s_2}; e_{out}; f; d_{s_1}^{s_2}; s_2)$, the morphism M_d maps the the duration d to the *sum* of durations defined in $\Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2)$

$$M(d) \rightarrow d \geq \sum_{d^* \in \Pi(s_1, e_{in}, p, t, e_{out}, f, d, s_2)} d^*.$$

Algorithm 2 (Symbolic reduction) Given a timed SystemC minimal-step waiting-state automaton $A(V) = (S; E; T; \Upsilon; D)$, where T has reducible transitions, let $T_0 := T$, $\Upsilon_0 := \Upsilon$, $D_0 := D$, let $T_{remove}; T_{new} := \{\}; \Upsilon_{remove}; \Upsilon_{new}; D_{remove}; D_{new} := \{\}$.

The following steps define an algorithm to remove all the reducible transitions:

- for every reducible pair $(t_1; t_2)$ and its contractum t_3 , where $t_1, t_2 \in T_0$, let: $T_{remove} := T_{remove} \cup \{t_1\}$ and $T_{new} := T_{new} \cup \{t_3\}$; $\Upsilon_{remove} := \Upsilon_{remove} \cup \{ \text{the starting times associated to } t_1 \text{ and } t_2 \}$; $\Upsilon_{new} := \Upsilon_{new} \cup \{ \text{the starting time associated to } t_3 \}$; $D_{remove} := D_{remove} \cup \{ \text{the durations associated to } t_1 \text{ and } t_2 \}$; $D_{new} := D_{new} \cup \{ \text{the durations associated to } t_3 \}$;
 replaces the starting time and the duration associated to the removed transitions t_1 and t_2 that appear in all the pre-conditions and post-conditions defined in T_0 with the new couple of time associated to the transition t_3 .
- repeat the above step until all reducible pairs in T_0 have been manipulated;
- Let $T_0 := (T_0 / T_{remove}) \cup T_{new}$, let $\Upsilon_0 := (\Upsilon_0 / \Upsilon_{remove}) \cup \Upsilon_{new}$, let $D_0 := (D_0 / D_{remove}) \cup D_{new}$;
- let $T_{remove}; T_{new}; \Upsilon_{remove}; D_{remove}; \Upsilon_{new} := \{\}; D_{new} := \{\}$.
- if there are still reducible pairs in T_0 , go to the step 1 and repeat the above procedure; otherwise, let $T' := T_0$; $\Upsilon' := \Upsilon_0$; $D' := D_0$. The reduced automaton is $(S; E; T'; \Upsilon'; D')$.

4. EXTENDING TIMED SYSTEMC WAITING STATE AUTOMATA WITH COUNTERS

In the previous section, transitions were annotated with temporal information: the starting time and the delay, which makes the automaton a nice framework for the verification of reactive systems. We add a new timescale to the whole system within a delta cycle considering only moments when a process get into and get out of a waiting state.

Moreover we can extend in a nice way the previous model with counters that counts the number of times a transition has been activated. Combining the both information, the time at which a

transition is activated, the execution of the transition as well as the number of time a transition has been activated allow to infer precise information about the system time execution; i.e if we take each process separately and we study how many times it gets into a waiting state and for how much duration, we can compute the lower bound of its execution time. When we compose processes automata together to get a bigger one using the same parameters time and counters, we obtain more information about the model and its execution time and consequently, we are able to verify additional relevant properties and infer new time constraints.

4.1. The automata for the producer and the consumer annotated with time and counters

Again, let us look at the FIFO example and consider the two small automata's for the processes of the producer and the consumer. In this section we will try to define new relations between both times and counters considering in addition to that the total execution time. We have usually the same definitions for our parameters, e.g γ_w^3 counts the times of executing the transition from p_wait_clk to p_wait_c, and t_r^1 represents the time the consumer starts reading. Usually we have

FIGURE 3: The extended timed automaton for the producer and the consumer

this relation: a *read* can only occur if a *write* occurs before it, so when we talk about counters we have these relations:

$$\begin{cases} \gamma_w^1 + \gamma_w^2 \geq \gamma_r^1 + \gamma_r^2 \\ (\gamma_w^1 + \gamma_w^2) - (\gamma_r^1 + \gamma_r^2) \leq num_{max} \end{cases}$$

Our strategy of verifying temporal constraints in SystemC waiting-state automata is similar as before, we have also these properties that are mentioned before: We can furthermore formulate some properties that may require more precise information, considering both times and counters, i.e we have to verify that the whole reading and writing duration is less than all time execution, so we have the following property:

$$\left\{ t_0 + (\gamma_w^1 d_w^1 + \gamma_w^2 d_w^2) + (\gamma_r^1 d_r^1 + \gamma_r^2 d_r^2) \leq T_{exec_tot} \right.$$

t_0 : is the starting time for simulation. As we know the first step for verifying SystemC models is to define the minimal step-automaton for every process, and then compose them together so as to get a big automaton for the whole system that will be followed by a reduction procedure. Algorithms for composing and reducing automata, but we consider three essential sets: the set of starting times, the set of durations and the set of counters.

$$\begin{cases} t_r^1 \geq t_w^1 + d_w^1 \\ t_r^1 \geq t_w^2 + d_w^2 \end{cases} \quad \begin{cases} t_w^1 \geq t_r^1 + d_r^1 \\ t_w^1 \geq t_r^2 + d_r^2 \end{cases}$$

4.2. Algorithms for composing and reducing WSA annotated with Time and Counters

Our strategy of verifying SystemC waiting-state automata is similar as before: we define first a minimal automaton based on waiting states of process, we annotate transitions with timing constraint and counters and then we compose automata's so as to obtain a bigger automaton for the whole system model, what makes the composition more complicated is that time properties and counters of the composed automata $A()$ and $A'()$ do not correspond to time properties and counters of the composed automaton $A() \times A'()$. Symbolic composition is followed by a reduction procedure, taking into account critical properties for real-time systems like determinism and liveness.

Definition An extended timed SystemC waiting-state automaton over a set V of variables, is a 5-tuples $A(V) = (S; E; T; \Upsilon; D; \Gamma)$, where S is a finite set of states, E is a finite set of events, Υ is a set of transition starting times, Γ is a finite set of counters, D is a set of transition duration times and T is a finite set of transitions where every transition is a 9-tuples $(s; e_{in}; p; t; e_{out}; f; d; s'; \gamma)$:

- s and s' are two states in S , representing respectively the initial state and the final state of the transition;
- e_{in} and e_{out} are two sets of events : $e_{in} \subseteq E; e_{out} \subseteq E$;
- p is a predicate defined over variables in V , T and D , i.e, $FV(p) \subseteq V \cup \Upsilon \cup D \cup \Gamma$;
- t is the starting time associated to the transition that increments each time it is executed;
- f is an effect function over V ;
- d is the duration of the transition.
- $\gamma \in \Gamma$ is the counter associated to the transition that increments each time it is executed.

A transition τ of the automata $A(V)$ may be composed with transitions $\{\tau'_1, \dots, \tau'_n\}$ of the automaton $A'()$, and the values of t , d and γ of the transition τ in $A(V)$ should be represented in the values of times and counters for $(\tau \times \tau'_0), \dots, (\tau \times \tau'_n)$. If t_τ^k , d_τ^k and γ_τ^k denote respectively the starting time, the duration and the counter associated to the transition $(\tau \times \tau^k)$ in the composed automata, then $t = \min_k t_\tau^k$, $d > \sum_k d_\tau^k$ and $\gamma = \sum_k \gamma_\tau^k$. As the transition predicates (i.e., guard conditions) in the extended timed waiting-state automata are defined over the timers, counters and system variables, the composition should ensure that these predicates of component automata are properly translated in the composed automaton. In particular, we must replace all the occurrence of a transition starting time, duration and counter from component automata, with the values t , d and γ recently presented.

Symbolic composition Given two timed SystemC waiting-state automata $A(V) = (S; E; T; \Upsilon; D; \Gamma)$ and $A'(V) = (S'; E'; T'; \Upsilon'; D'; \Gamma')$, over the same set V of variables, the combination of the two SystemC waiting state automata is still a SystemC waiting-state automaton $(S \times S'; E \cup E'; T''; \Upsilon''; D''; \Gamma'')$ written as $A(V) \times A'(V)$ where T'' is the smallest set of transitions, Υ'' is the associated set of the starting times, D'' is the corresponding set of durations and Γ'' is the associated set of counters, $\Pi(s; e_{in}; p; t; e_{out}; f; d; s'; \gamma)$ is the set of times and counters of Υ'' , D'' and Γ'' associated to a transition in $T \times T'$ and M_c a morphism that maps counters in $\Gamma \times \Gamma'$ to Γ'' , M_t a morphism that maps times in $\Upsilon \times \Upsilon'$ to Υ'' and M_d a morphism that maps times in $D \times D'$ to D'' such that:

- $\Pi(s_1; e_{in}; p; t; \gamma; e_{out}; f; d; s_2) := \{t^*, d^*, \gamma^*\} \cup \Pi(s_1; e_{in}; p; t; \gamma; e_{out}; f; d; s_2)$ and $(s_1, s'_1) \xrightarrow[e_{out}, f', d^*]{e_{in}, M_t(p), M_c(p), M_d(p), t^*, \gamma^*} (s_2, s'_1) \in T''$ for every transition $s_1 \xrightarrow[e_{out}, f, d]{e_{in}, p, t, \gamma} s_2 \in T$

and for every state $s'_1 \in S'$ such that for every transition $s'_1 \xrightarrow[e'_{out}, f', d', s'_2]{e'_{in}, p', t', \gamma'} s'_2 \in T'$, either

- $e'_{in} \not\subseteq e_{in}$ or $p \not\models p'$;
- $\Pi(s_1; e'_{in}; p'; t'; \gamma'; e'_{out}; f'; d'; s'_2) := \{t^*, d^*, \gamma^*\} \cup \Pi(s'_1; e'_{in}; p'; t'; \gamma'; e'_{out}; f'; d'; s'_2)$ and $(s_1, s'_1) \xrightarrow[(e'_{out}, f', d^*)]{e_{in}, M_t(p'), M_c(p'), M_d(p'), t^*, \gamma^*} (s_1, s'_2) \in T''$ for every transition $s'_1 \xrightarrow[e'_{out}, f', d']{e'_{in}, p', t', \gamma'} s'_2 \in T'$

and for every state $s_1 \in S$ such that for every transition $s_1 \xrightarrow[e_{out}, f, d]{e_{in}, p, t, \gamma} s_2 \in T$, either $e_{in} \not\subseteq e'_{in}$

or $p' \not\Rightarrow p$;

- $\Pi(s_1; e_{in}; p; t; \gamma; e_{out}; f; d; s_2) := \{t^*, d^*, \gamma^*\} \cup \Pi(s_1; e_{in}; p; t; \gamma; e_{out}; f; d; s_2)$ and $\Pi(s_1; e_{in}; p'; t'; \gamma'; e_{out}; f'; d'; s_2) := \{t^*, d^*, \gamma^*\} \cup \Pi(s_1; e_{in}; p'; t'; \gamma'; e_{out}; f'; d'; s_2)$ and $(s_1, s'_1) \xrightarrow[e_{out} \cup e'_{out}, f \cup f', d^*]{e_{in} \cup e'_{in}, M_t(p \wedge p'), M_c(p \wedge p'), M_d(p \wedge p'), t^*, \gamma^*} (s_2, s'_2) \in T''$, for every pair of transitions

$s_1 \xrightarrow[e_{out}, f, d]{e_{in}, p, t, \gamma} s_2 \in T$ and $s'_1 \xrightarrow[e'_{out}, f', d']{e'_{in}, p', t', \gamma'} s'_2 \in T'$.

- According to the transition $(s_1; e_{in}; p; t_{s_1}^{s_2}; e_{out}; f; d_{s_1}^{s_2}; s_2; \gamma_{s_1}^{s_2})$, the morphism M_t maps the starting t to the *min* of transition starting times defined in $\Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2; \gamma)$

$$M(t) \rightarrow \min_{t^* \in \Pi(s_1, e_{in}, p, t_{s_1}^{s_2}, e_{out}, f, d_{s_1}^{s_2}, s_2, \gamma_{s_1}^{s_2})} t^*$$

- The morphism M_d maps the the duration d to the *sum* of durations defined in $\Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2; \gamma)$

$$M(d) \rightarrow d \geq \sum_{d^* \in \Pi(s_1, e_{in}, p, t_{s_1}^{s_2}, e_{out}, f, d_{s_1}^{s_2}, s_2, \gamma_{s_1}^{s_2})} d^*.$$

- The morphism M_c maps the counters γ to the *sum* of counters defined in $\Pi(s_1; e_{in}; p; t; e_{out}; f; d; s_2; \gamma)$

$$M(\gamma) \rightarrow \sum_{\gamma^* \in \Pi(s_1, e_{in}, p, t_{s_1}^{s_2}, e_{out}, f, d_{s_1}^{s_2}, s_2, \gamma_{s_1}^{s_2})} \gamma^*.$$

Symbolic reduction Given a SystemC minimal-step waiting-state automaton $A(V) = (S; E; T; \Upsilon; D; \Gamma)$, where T has reducible transitions, let $T_0 := T, D_0 := D, \Upsilon_0 := \Upsilon, \Gamma_0 := \Gamma$, let $T_{remove}; T_{new}; D_{remove}; \Upsilon_{remove}; \Gamma_{remove}; D_{new} := \{\}; \Upsilon_{new} := \{\}; \Gamma_{new} := \{\}$. The following steps define an algorithm of removing the reducible transitions:

1. For every reducible pair $(t_1; t_2)$ and its contractum t_3 , where $t_1, t_2 \in T_0$, let: $T_{remove} := T_{remove} \cup \{t_1\}$ and $T_{new} := T_{new} \cup \{t_3\}$; $\Gamma_{remove} = \Gamma_{remove} \cup \{ \text{the counters associated to } t_1 \text{ and } t_2 \}$; $\Gamma_{new} = \Gamma_{new} \cup \{ \text{the counters associated to } t_3 \}$; $\Upsilon_{remove} = \Upsilon_{remove} \cup \{ \text{the starting times associated to } t_1 \text{ and } t_2 \}$; $\Upsilon_{new} = \Upsilon_{new} \cup \{ \text{the starting time associated to } t_3 \}$; $D_{remove} = D_{remove} \cup \{ \text{the durations associated to } t_1 \text{ and } t_2 \}$; $D_{new} = D_{new} \cup \{ \text{the durations associated to } t_3 \}$; replaces the starting time, the duration and the counters associated to the removed transitions t_1 and t_2 that appear in all the pre-conditions and post-conditions defined in T_0 with the new values associated to starting time, duration and the counter associated to the transition t_3 .
2. Repeat the above step until all reducible pairs in T_0 have been manipulated;
3. Let $T_0 := (T_0/T_{remove}) \cup T_{new}$, let $\Upsilon_0 := (\Upsilon_0/\Upsilon_{remove}) \cup \Upsilon_{new}$, let $D_0 := (D_0/D_{remove}) \cup D_{new}$ and let $\Gamma_0 := (\Gamma_0/\Gamma_{remove}) \cup \Gamma_{new}$
4. let $T_{remove}; T_{new}; \Upsilon_{remove}; D_{remove}; \Gamma_{remove}; \Upsilon_{new} := \{\}; D_{new} := \{\}$ and $\Gamma_{new} := \{\}$.
5. If there are still reducible pairs in T_0 , go to the step 1 and repeat the above procedure; otherwise, let $T' := T_0; \Upsilon' := \Upsilon_0; D' := D_0$ and $\Gamma' := \Gamma_0$. The reduced automaton is $(S; E; T'; \Upsilon'; D'; \Gamma')$.

5. CONCLUSION

Verification of reactive systems, critical systems or embedded systems is a very important issue today. The increasing complexity requires more design efforts and choosing the right architecture to guarantee system performance and reliability requires large design space exploration.

In this paper, we propose a new formal model to verify real-time systems described in a high level language: *SystemC*. This model provides the capacity to verify timed SystemC designs, to infer or

verify time constraints as well as to estimate partial or global system execution time, at the level of delta cycles.

The process of the verification of timed SystemC design is clear; we first translate SystemC processes into timed SystemC waiting-state automata. We then combine those automata together. During all the combination, we may infer affine relations regarding transition counts as well as the time at which transitions can be activated. We finally reduce the resulting automaton so that the final automata recognize the abstraction of the whole system at the level of delta-cycles. Timed SystemC waiting state automata may also be a valuable representation for timed co-simulation [9,10].

REFERENCES

- [1] K. Richter, M. Jersak, R. Ernst, A formal approach to MpSoC performance verification, Computer, April 2003.
- [2] Yu Zhang, Franck Védryne, Bruno Monsuez, SystemC Waiting-State Automata, 2008
http://www.bcs.org/upload/pdf/ewic_ve07_s2paper2.pdf
- [3] F. Balarin, L. Lavagno, P. Murthy, A. SanGiovanni-Vicentelli, Scheduling for embedded real-time systems, IEEE Design & Test of Computers, January-March 1998.
- [4] SystemC homepage. <http://www.systemc.org>.
- [5] J. L. Peterson, Petri Nets, 1977
<http://www.cs.olemiss.edu/~dwilkins/Seminar/S05/PetersonPetri.pdf>.
- [6] Richard M. Fujimoto, Time Management in the High Level Architecture, 1998
http://www.cc.gatech.edu/computing/pads/PAPERS/Time_mgmt_High_Level_Arch.ps.
- [7] H. Posadas, F. Herrera, P. Sánchez, E. Villar & F. Blasco, System-Level Performance Analysis in SystemC, 2004
http://date.eda-online.co.uk/proceedings/papers/2004/date04/pdf/files/04b_2.pdf.
- [8] S. Yoo, G. Nicolescu, L. Gauthier, A. Jarraya, Automatic generation of fast timed simulation models for operating systems in SoC design, Proc. of DATE, IEEE, 2002.
- [9] J.Y. Lee, I. Park, Timed Compiled-Code Simulation of Embedded Software for Performance Analysis of SoC
- [10] E. Clarke, O. Grumberg, D. Peled, Model-Checking, The MIT Press, Cambridge, Massachusetts, 1999.
- [11] Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, A. Petit, Systems and Software Verification - Model-Checking Techniques and Tools, Springer, 2001.
- [12] Amjed Gawanmeh, Ali Habibi, Sofène Tahar, Enabling SystemC Verification using Abstract State Machines, 2004
http://hvg.ece.concordia.ca/Publications/TECH_REP/SYSC_TR04/SYSC_TR04.pdf.

A. THE TIMED SYSTEMC WAITING STATE OF THE FIFO MODEL

FIGURE 4: The reduced Timed SystemC Waiting-State Automaton for the FIFO