Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I
IMECS 2008, 19-21 March, 2008, Hong Kong

$SystemC^{\mathbb{FL}}_{\text{tlm}}$: Motivation and Development

K.L. Man, M. Mercaldi, F. Garberoglio, A. Trischitta, H.Y. Lai and C.M. Ho *

Abstract— **This paper introduces an algebraic theory $SystemC^{\mathbb{FL}}_{\text{tlm}}$ that can be used to specify and analyse the behaviour of SystemC designs. This theory/language is the successor of the $SystemC^{\mathbb{FL}}$ language. It extends $SystemC^{\mathbb{FL}}$ with the possibility to define process term instantiations and the use of SystemC positional connections/named connection and Transaction Level Modelling (TLM). We illustrate the practical use of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ by means of a TLM example.**

Keywords: formal semantics, $SystemC^{\mathbb{FL}}$ and $SystemC^{\mathbb{FL}}_{\text{tlm}}$, process algebras, formal specification and analysis, transaction level modelling (TLM)

1 Introduction

In an attempt to give a formal semantics of a reasonable subset of SystemC [8] based on process algebras that could be used for the formal specification and analysis of SystemC designs, the formal language $SystemC^{\mathbb{FL}}$ (SCFL in ASCII format) [5] was first defined in [9] (2004); and subsequently extended with some features in [10] (2005). $SystemC^{\mathbb{FL}}$ maybe regarded as the formalisation of a reasonable subset of SystemC based on the classical process algebras *Algebra of Communicating Processes* (ACP) [2] and *A Timed Process Algebra for Specifying Real-Time Systems* (ATP) [18]. The semantics of $SystemC^{\mathbb{FL}}$ is defined by means of deduction rules in a *Structured Operational Semantics* (SOS) [21] style that associates a time transition system (TTS) with a $SystemC^{\mathbb{FL}}$ process. A set of properties is presented for a notion of bisimilarity. More precisely, $SystemC^{\mathbb{FL}}$ is aimed at giving formal specifications of SystemC designs and to perform formal analysis of SystemC processes. Furthermore, $SystemC^{\mathbb{FL}}$ is a single formalism that can be used for specifying concurrent systems, finite state systems and real-time systems (as in SystemC).

Nowadays, *Transaction Level Modelling* (TLM) is indispensable to solve a variety of practical problems (e.g. providing an early platform for software development and system level design architecture analysis) during the design and development of complex electronic systems. Also, TLM has been widely propagated and used for System-on-a-Chip (SoC) and embedded system design. The interested reader may refer to [7] for excellent surveys on the topic of TLM. SystemC has supported TLM since the version 2.0. In the past few years, SystemC has proven to be suitable for TLM and has also becoming the defacto standard for TLM in the electronic design community. However, TLM is still a relatively young kind of approach meant to ease the handling of the constantly growing complexity of electronic systems; by raising the level of abstraction it allows system architects, embedded software engineers, and system developers, to explore architectural alternatives, to start software development, and to produce raw performance estimation at a much earlier stage than it would be possible if a RTL description of the system were used as platform reference.

With alternative exploration in mind, the main advantage of TLM is the simulation speed-up that it offers w.r.t. cycle-accurate representation, essentially due to the different abstraction level, which turns into a much smaller amount of information to be handled. The main disadvantage shown by TLM, so far, is the lack of a formal semantics, that could be used both for consistency checking during description refinement, and for property checking on untimed descriptions, mainly aimed at checking functional correctness on an abstraction of the final system. Various attempts to give TLM a formal background have already been made, but none of those has proposed a framework to allow checking on specific aspects of the component being designed with the most suited formal checking tool. To reach the goal of formal verification of SystemC designs (with a focus on SystemC TLM), as reported in [14], we have focused our attention on SystemC as a language for TLM, and selected $SystemC^{\mathbb{FL}}$ as the language to formally represent SystemC designs.

In the frame of a tight collaboration between researchers/engineers from industrial entities and research institutes, several tools for $SystemC^{\mathbb{FL}}$ have been being developed. These tools enable automatic translations from SystemC codes to $SystemC^{\mathbb{FL}}$ specifications and from $SystemC^{\mathbb{FL}}$ specifications to various formalisms that are the input languages of some existing formal verification tools. Using $SystemC^{\mathbb{FL}}$ tools in combination with some formal verification tools yields automatic verifications of SystemC designs via $SystemC^{\mathbb{FL}}$ specifications (for different verification purposes). Our first goal of the research in these directions is to develop an automatic translation tool which converts untimed SystemC codes into the corresponding $SystemC^{\mathbb{FL}}$ specifications that can be further mapped to the input languages of several formal verification tools (e.g. SPIN and NuSMV [19]). Recently, such an automatic translation tool SC2SCFL has been developed in the Java language (JDK

*K.L. Man is with the Centre for Efficiency-Oriented Languages (CEOL), Department of Computer Science, University College Cork (UCC), Cork, Ireland, email: systemcfl@gmail.com; M. Mercaldi is with the M.O.S.T., Turin, Italy, email: michele.mercaldi@most.it; F. Garberoglio is with the Magneti Marelli S.P.A., Turin, Italy, email: flaviano.garberoglio@mmarelli-se.com; A. Trischitta, H.Y. Lai and C.M. Ho are freelance informatics consultants.

1.5.0) using JavaCC 4.0 as a parser generator. Although the current release of SC2SCFL can be used to translate some SystemC designs (e.g. counter & test-bench and scalable synchronous bus arbiter as shown in [13, 16]) to the corresponding specifications in SystemC, it is not applicable in practise to deal with the translation of industrial SystemC designs. Our experience with SC2SCFL tells us that, based on the current semantics of $SystemC^{\mathbb{FL}}$, it is impossible to build a translator in such a way that it can be used to translate complex SystemC designs, because $SystemC^{\mathbb{FL}}$ is not expressive enough to formally represent the current version of SystemC (2.2). For instance, $SystemC^{\mathbb{FL}}$ (developed in 2004) has no well-defined semantics for TLM and cannot deal with SystemC process instantiations and positional connections modelling features.

After having several attempts, by means of defining new semantics and new operators, to extend $SystemC^{\mathbb{FL}}$ to cope with the features such as SystemC TLM and process instantiation; it turned out to be very difficult to show that $SystemC^{\mathbb{FL}}$ with new operators could be an operational conservative extension [28] of $SystemC^{\mathbb{FL}}$ as defined in [9, 10]. Furthermore, we had several ideas to improve $SystemC^{\mathbb{FL}}$ in such a way that the semantics of $SystemC^{\mathbb{FL}}$ would be more intuitive, simpler and elegant. Hence, we made a decision to redesign $SystemC^{\mathbb{FL}}$. Recently, the successor of $SystemC^{\mathbb{FL}}$, called $SystemC^{\mathbb{FL}}_{\text{tlm}}$ (SCFL2 in ASCII format), has been developed.

The aim of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ is to serve as a formalism to formally represent SystemC (current version) including SystemC TLM features. In this paper, we sketch the newly developed formal language $SystemC^{\mathbb{FL}}_{\text{tlm}}$. The $SystemC^{\mathbb{FL}}_{\text{tlm}}$ language extends $SystemC^{\mathbb{FL}}$ with the possibility to define process term instantiations and for the use of SystemC positional connections/named connection and SystemC TLM. For the reason of space limitation, overviews of SystemC and $SystemC^{\mathbb{FL}}$ are not given in this paper. Some familiarities with SystemC and $SystemC^{\mathbb{FL}}$ are required. The desirable backgrounds can, for example, be found in [8, 26, 9, 10]. The reminder of the paper is organised as follows.

The motivations and outlines of the development of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ are given in Section 2 and Section 3 presents the $SystemC^{\mathbb{FL}}_{\text{tlm}}$ language including the syntax and the formal semantics. By means of a TLM example, the practical use of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ is illustrated in Section 4 and Section 5 discusses the related work of $SystemC^{\mathbb{FL}}$. Finally, concluding remarks are made in Section 6 and the direction of future work is pointed out in the same section.

2 From $SystemC^{\mathbb{FL}}$ to $SystemC^{\mathbb{FL}}_{\text{tlm}}$

As we already mentioned in Section 1, it turned out to be impossible to use $SystemC^{\mathbb{FL}}$ to formally represent the current version of SystemC. The main clauses are the following:

- Expressivity and TLM. Clearly, $SystemC^{\mathbb{FL}}$ (developed in 2004) is rather old. It is not expressive enough to formally represent SystemC (today) and $SystemC^{\mathbb{FL}}$ has no well-defined semantics for TLM.
- Lack of SystemC features. There is also quite a lot of SystemC constructs and features that were not formalised in $SystemC^{\mathbb{FL}}$ yet. As examples, instantiation of SystemC modules, positional connections in SystemC and some C++ constructs.
- Unintuitive syntax. Generally speaking, the common syntax used in process algebraic theories (including $SystemC^{\mathbb{FL}}$) is not intuitive for designers and engineers in the electronic design community. Also, designers and engineers are uncomfortable with mathematical notations used in $SystemC^{\mathbb{FL}}$.
- Unintuitive semantics. Needless to say that the formal semantics of $SystemC^{\mathbb{FL}}$ is also not intuitive for designers and engineers in the electronic design community. In our experience, for example, the use/definition of two valuations (e.g. previous accompanying valuation and current valuation) in the quintuple of a $SystemC^{\mathbb{FL}}$ process is highly unintuitive for the users. According to the deduction rules of some $SystemC^{\mathbb{FL}}$ operators (e.g. the watch operator $\circlearrowleft$), this is needed and can be used to observe the change of the valuation of variables in the sensitivity list.

3 Formal Language $SystemC^{\mathbb{FL}}_{\text{tlm}}$

Based on the concept of $SystemC^{\mathbb{FL}}$ towards a slightly richer language, the successor of $SystemC^{\mathbb{FL}}$, the formal language $SystemC^{\mathbb{FL}}_{\text{tlm}}$ has been recently developed, which can be used to formally represent (most of the features of) the current version of SystemC (2.2) including SystemC TLM features.

3.1 $SystemC^{\mathbb{FL}}_{\text{tlm}}$ Data Types

In order to define the semantics of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ processes, we need to make some assumptions about the data types:

1. Let Var denote the set of all variables $(x_0, \ldots, x_n, \texttt{time})$. Besides the variables $x_0, \ldots, x_n$, the existence of the predefined reserved global variable `time` which denotes the current time, the value of which is initially zero, that is assumed. This variable cannot be declared. Note that the reason for introducing the predefined variable `time` representing time in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ is to allow a better capture of the notion of time in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ and a more straightforward translation from $SystemC^{\mathbb{FL}}_{\text{tlm}}$ to other formalisms with timing (e.g. timed automata).
2. Also, we make use of the sets of variables $\text{Var}^- = \{x^- \mid x \in \text{Var}\}$ and $\text{Var}^+ = \{x^+ \mid x \in \text{Var}\}$, modelling the current value and future value (after a transition) of a variable, respectively. The introduction of these two sets of variables aims to ease modelling. Similarly, e^- and e^+ are used to represent the current value and future value of e respectively, where e is an expression over variables from Var.
3. Let Value denote the set of all possible values $(v_0, \ldots, v_m, \bot)$ that contains at least all Integers, all

Reals and Booleans, where $\perp$ denotes the "*undefinedness*". As in $SystemC^{\mathbb{FL}}$, the set Value can be easily extended with some other data types (e.g. "sc_bit"). We usually use true and false to represent the predicates "true" and "false" respectively.

4. We then define a *valuation* as a partial function from variables to values. Syntactically, a valuation is denoted by a set of pairs $\{x_0 \mapsto v_0, \ldots, x_n, \mapsto v_n, \mathtt{time} \mapsto t\}$, where $x_i \in \text{Var}$ represents a variable and $v_i \in \text{Value}$ represents the associating value to x_i; and $t \in \mathbb{R}_{\geq 0}$. Further to this, the set of all valuations is denoted by Σ.
5. P^{ω} is the set of all process term instantiations of the form $P^{\wp}(x_1, \ldots, x_n)$, where $P^{\wp}$ denotes a process term label and $x_1, \ldots, x_n \in \text{Var}$.
6. An environment E is a tuple of (s, m, i) such that $s \in S$, $m \in Ch$ and $i \in P^{\omega}$, where S and Ch denote the sets of all sensitivity lists with clocks and all channels may be used in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ processes, respectively, that are assumed. Also, we choose to use $\aleph$ to denote the set of all environments.

Note that the type "array" in SystemC has not been formalised in $SystemC^{\mathbb{FL}}_{\text{tlm}}$, because the behaviour of elements in an array in SystemC can be modelled in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ by introducing fresh variables. As an example, for an array $A[0..10]$ in SystemC, we can introduce fresh variables $A_0, \ldots, A_{10}$ in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ to associate correspondingly $A0$ with A_0, $A1$ with A_1 and so on.

3.2 Syntax of the $SystemC^{\mathbb{FL}}_{\text{tlm}}$ Language

The formal language $SystemC^{\mathbb{FL}}_{\text{tlm}}$ is defined according to the following grammar for process terms $p \in P$:

$$\begin{aligned} p = \;& \delta \mid \text{skip} \mid x = e \mid \Delta e_{\text{n}} \mid \ggg \mid p \blacktriangleleft b \blacktriangleright p \\ & \mid b \circlearrowleft p \mid p \bullet p \mid p \Theta p \mid p \triangle_d p \mid p \blacklozenge^d p \\ & \mid *p \mid p \parallel p \mid p \between p \mid p \rightsquigarrow p \mid \partial_H(p) \mid \tau_I(p) \\ & \mid \pi(p) \mid \ni(p) \mid \Upsilon \Rsh p \mid P^{\wp}(x_1, \ldots, x_n) \end{aligned}$$

Here, x and $x_1, \ldots, x_n$ are variables taken from Var and $d \in \mathbb{R}_{\geq 0}$. b and e denote a boolean expression and an expression over variables from Var respectively; and Υ denotes a predicate over variables from Var and/or P^{ω}; and e_{n} denotes an numerical expression. Moreover, H and I are sets of actions. In $SystemC^{\mathbb{FL}}_{\text{tlm}}$, we allow the use of common arithmetic operators (e.g. $+$, $-$), relational operators (e.g. $=$, $\geq$) and logical operators (e.g. $\wedge$, $\vee$) as in mathematics to construct expressions over variables from Var. The operators are listed in descending order of their binding strength as follows : $\{\_ \circlearrowleft \_, \_ \bullet \_, \_ \triangle_{\_} \_, \_ \blacklozenge^{\_} \_, \_ \Rsh \_\}$, $\{\_ \blacktriangleleft \_ \blacktriangleright \_, \_ \Theta \_, \_ \parallel \_, \_ \between \_, \_ \rightsquigarrow \_\}$. The operators inside the braces have equal binding strength. In addition, operators of equal binding strength associate to the left, and parentheses may be used to group expressions. For example, $p;q;r$ means $p;(q;r)$, where $p, q, r \in P$. Note that more intuitive and user-friendly syntax of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ will be defined in ASCII format for the $SystemC^{\mathbb{FL}}_{\text{tlm}}$ tool development.

Process terms δ and skip are mainly introduced for calculation and axiomatisation purposes. Below is a brief introduction of the syntax of $SystemC^{\mathbb{FL}}$:

- The *deadlock* δ is introduced as a constant, which represents no behaviour.
- The *skip* process term skip performs the internal action τ, which is not externally visible.
- The *assignment* process term $x = e$, which assigns the value of expression e to variable x (modelling a SystemC `assignment` statement).
- The *delay* process term Δe_{n} is able to first delay the value of numerical expression e_{n} and then terminates by means of an internal action τ.
- The *unbounded delay* process term $\ggg$ (modelling a SystemC `wait` statement) may delay for a long time that is unbounded or perform the internal action τ.
- The *conditional composition* $p \blacktriangleleft b \blacktriangleright q$ operates as a SystemC `if_then_else` statement, where b denotes a boolean expression and $p, q \in P$. If b holds, p executes. Otherwise, q executes.
- The *watch* process term $b \circlearrowleft p$ is used to model a SystemC construct of `even control`.
- The *sequential composition* $p \bullet q$ models the process term that behaves as p, and upon termination of p, continues to behave as q.
- The *alternative composition* $p \Theta q$ models a nondeterministic choice between p and q.
- The *timeout* process term $p \triangle_d q$ (modelling a SystemC `time out` construct) behaves as p if p performs a time transition before a time $d \in \mathbb{R}_{>0}$. Otherwise, it behaves as q.
- The *watchdog* process term $p \blacklozenge^d q$ behaves as p during a period of time less than d, at time d, q takes over the execution from p in $p \blacklozenge^d q$; if p performs an internal *cancel* χ action, then the delay is cancelled, and the subsequent behaviour is that of p after χ is executed.
- The *repetition* process term $*p$ (modelling a SystemC `loop` construct) executes p zero or more times.
- The *parallel composition* $p \parallel q$, the *left-parallel composition* $p \between q$ and the *communication* composition $p \rightsquigarrow q$ are used to express *parallelism* in which actions are executed in an interleaving manner with the possibility of synchronisation of actions. The synchronisation of actions take place using a (partial, commutative and associative) synchronisation function $\gamma \in A_{\text{tlm}} \times A_{\text{tlm}} \mapsto A_{\text{tlm}}$ (the set A_{tlm} is defined in Subsection 3.3). For example, if the actions a and b synchronise, the resulting action is c such that $\gamma(a, b) = c$.
- The *encapsulation* of actions is allowed using $\partial_H(p)$, where H represents the set of all actions to be blocked in p.
- The *abstraction* $\tau_I(p)$ behaves as the process term p, except that all action names in I are renamed to the internal action τ.
- The *maximal progress* $\pi(p)$ assigns action transitions a higher priority over time transitions; this operator is needed to establish a desired communication behaviour, that is, both the sender and the receiver must be able to perform time transitions, but if two of these can communicate (i.e. performing action transitions), they should not perform time transitions.
- The *grouping* of actions in p and executing them in one single step can be done by using $\ni(p)$.

- The *signal emission operator* $\Upsilon \Rsh p$ requires that the predicate Υ always holds; if it is the case, $\Upsilon \Rsh p$ behaves like p, otherwise, it is a δ; this operator is needed for defining the translation from $SystemC^{\mathbb{FL}}$ to the SMV language [15] (see also [11]).
- The *process term instantiation* $P^{\wp}(x_1, \ldots, x_n)$ is defined in $SystemC^{\mathbb{FL}}_{\text{tlm}}$, which is used to refer to a process term declared by the process term definition of the form $P^{\wp}(x'_1, \ldots, x'_n) = p$, where $x_1, \ldots, x_n$ are (actual parameter) variables and $x'_1, \ldots, x'_n$ are (formal parameter) variables. This form of definition declares that the behaviour of the process reference $P^{\wp}(x'_1, \ldots, x'_n)$ is given by $px'_1 \ldots, x'_n/x_1, \ldots, x_n$, which means that all free occurrences of variables $x_1, \ldots, x_n$ are replaced by $x'_1, \ldots, x'_n$ in p.

Here, we highlight some features in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ that lead $SystemC^{\mathbb{FL}}_{\text{tlm}}$ to outperform $SystemC^{\mathbb{FL}}$:

1. As said already, SystemC TLM semantics is covered in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ (via its syntax and deduction rules).
2. In $SystemC^{\mathbb{FL}}_{\text{tlm}}$, no explicit operator has been defined for positional connections/named connection in SystemC, because SystemC positional connections/named connection can be easily modelled using process term instantiations in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ together with the signal emission operator.
3. It is not hard to see that the specifications described in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ are much more intuitive and elegant than the specifications given in $SystemC^{\mathbb{FL}}$.

3.3 Semantics of the $SystemC^{\mathbb{FL}}_{\text{tlm}}$ Language

Definition 1 *A $SystemC^{\mathbb{FL}}_{\text{tlm}}$ process is a tri-tuple $\langle P, \Sigma, \aleph \rangle$. We use the convention $\langle p, \sigma, E \rangle$ to write a $SystemC^{\mathbb{FL}}_{\text{tlm}}$ process, where $p \in P$ is a process term, $\sigma \in \Sigma$ is a valuation and $E \in \aleph$. We also assume that the variables occurring in p must be defined and this also means that such variables are from the domain of σ and/or in E.*

Definition 2 *The set of actions A_{tlm} contains at least $aa(x, v)$, χ and τ, where $aa(x, v)$ is the assignment action (i.e. the value of v is assigned to the variable x), χ is the internal cancel action and τ is the internal action. The set A_{tlm} is considered as a parameter of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ that can be freely instantiated.*

Definition 3 *We give a formal semantics for $SystemC^{\mathbb{FL}}_{\text{tlm}}$ processes in terms of a* Timed Transition System *(TTS), and define the following transition relations on processes of $SystemC^{\mathbb{FL}}$:*

- $\_ \xrightarrow{\_} \_ \subseteq (P \times \Sigma \times \aleph) \times A_{\text{tlm}} \times (P \times \Sigma \times \aleph)$*, denotes action transition;*
- $\_ \xrightarrow{\_} \langle \checkmark, \_, \_ \rangle \subseteq (P \times \Sigma \times \aleph) \times A_{\text{tlm}} \times (\Sigma \times \aleph)$*, denotes termination, where $\checkmark$ is used to indicate a successful termination, and $\checkmark$ is not a process term;*
- $\_ \xmapsto{\_} \_ \subseteq (P \times \Sigma \times \aleph) \times \mathbb{R}_{>0} \times (P \times \Sigma \times \aleph)$*, denotes time transition (so-called delay).*

The three kinds of transition relations can be explained as follows:

- Firstly, an action transition $\langle p, \sigma, E \rangle \xrightarrow{a} \langle p', \sigma', E \rangle$ is that the process $\langle p, \sigma, E \rangle$ executes the action $a \in A_{\text{tlm}}$ starting with the current valuation σ (at the moment of the transition taking place) and by this execution p evolves into p'. Notice that σ' represents the accompanying valuation of the process after the action a is executed.
- Secondly, a termination $\langle p, \sigma, E \rangle \xrightarrow{a} \langle \checkmark, \sigma', E \rangle$ is that the process executes the action a followed by termination.
- Thirdly, a time transition $\langle p, \sigma, E \rangle \xmapsto{d} \langle p', \sigma', E \rangle$ is that the process $\langle p, \sigma, E \rangle$ may idle during a time d and then behaves like $\langle p', \sigma', E \rangle$.

For brevity, in what follows, we abbreviate $\langle p, \sigma, E \rangle$ as $\langle p, \sigma \rangle_E$

3.4 Deduction Rules and Improvement of the Semantics

The above transition relations are also defined through deduction rules (SOS style). We refer to [12] for a detailed account of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ deduction rules, the congruence result and the set of axioms/properties of $SystemC^{\mathbb{FL}}_{\text{tlm}}$. There is a technicality issue about the definition of a $SystemC^{\mathbb{FL}}_{\text{tlm}}$ process and the way to define deduction rules for $SystemC^{\mathbb{FL}}_{\text{tlm}}$ semantics, which make the specifications described in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ are much more intuitive and elegant (than the $SystemC^{\mathbb{FL}}$ specifications in general).

A $SystemC^{\mathbb{FL}}$ process (e.g. $\langle p, \sigma', \sigma, s, m \rangle$) consists of two valuations (the previous valuation σ' and the current valuation σ). Based on the variation/information of the valuations (σ' and σ), some deduction rules of $SystemC^{\mathbb{FL}}$ operators were defined in such a way that the behaviour of such a process could be predicted/defined after a possible transition (e.g. $\langle p, \sigma', \sigma, s, m \rangle \xrightarrow{a} \langle p', \sigma, \sigma'', s, m \rangle$). The disadvantage of this approach to define deduction rules is that one additional valuation is always needed to keep as a part of the $SystemC^{\mathbb{FL}}$ process definition. Also, this approach was only needed to define semantics for very few operators in $SystemC^{\mathbb{FL}}$ and made some $SystemC^{\mathbb{FL}}$ deduction rules unintuitive.

To improve the readability of and to simplify the deductions in $SystemC^{\mathbb{FL}}_{\text{tlm}}$, only one valuation is defined in a $SystemC^{\mathbb{FL}}_{\text{tlm}}$ process. For a given current valuation σ and a given future valuation σ' (after a transition), deduction rules in $SystemC^{\mathbb{FL}}_{\text{tlm}}$ have been defined in such a way that, starting from the current state/valuation (i.e. σ), a more specific transition (e.g. $\langle p, \sigma \rangle_E \xrightarrow{a_{specific}} \langle p', \sigma' \rangle_E$) is precisely defined to reach the future state/valuation (i.e. σ'). In other words, such a transition is highly restricted by the current state/valuation and future state/valuation. The advantage of this approach to define deduction rules is that the defined deduction rules become simpler and more intuitive. It also helps to reduce nondeterminism regarding the behaviour of the $SystemC^{\mathbb{FL}}_{\text{tlm}}$ process.

4 TLM Buffer Example in $SystemC^{\mathbb{FL}}_{\text{tlm}}$

In this section, we aim to illustrate that $SystemC^{\mathbb{FL}}_{\text{tlm}}$ can be used for SystemC TLM by means of an example of TLM: *one*

slot buffer. In the example, a process term *ReadWrite* issues randomly and continuously write and read actions to an one slot buffer and a process term *Status* describes the availability of the buffer if it is ready for reading from the channel m (i.e. when the flag variable *busy* evaluates to true) or if it is free for writing to the channel m (i.e. when the flag variable *busy* evaluates to false). The process term *Status* is defined as follows:

$$Status \equiv busy = \text{false} \blacktriangleleft busy \blacktriangleright busy = \text{true}.$$

As mentioned already in Definition 2, actions are considered as parameters of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ that can be freely instantiated. For this process term, we also write $busy_m$ and $free_m$ as the actions associating with the assignment process terms $busy = \text{false}$ and $busy = \text{true}$ respectively. These actions are used for the synchronisation with other actions (see the process term *Buffer* below for details) for writing to the buffer through the channel m when the buffer is free or reading from the buffer through the channel m when the buffer is occupied. As shown in the process term *Status*, depending on the status of the flag variable *busy*, a choice is made between performing action $busy_m$ or action $free_m$. In either case, the value of the flag variable *busy* will be converted (from true to false or vice versa) after performing such actions. The process term *ReadWrite* is defined below:

$$ReadWrite \equiv data = \text{true} \ \Theta \ data = \text{false}.$$

For this process term *ReadWrite*, we write actions $read_m$ and $write_m$ as the actions associating with the assignment process terms $data = \text{true}$ and $data = \text{false}$ respectively. When $read_m$ executes, the reading action is performed through the channel m and leads to free the buffer by means of assigning a predicate true to the variable *data* (to denote that the buffer is not busy). Similarly, when $write_m$ executes, the writing action is performed through the channel m and leads to occupy the buffer by means of assigning a predicate false to the variable *data* (to denote that the buffer is busy). The complete system is described by the process term *Buffer* as follows:

$$Buffer \equiv *(\ggg \bullet \tau_I(\partial_H(ReadWrite \parallel Status))), \text{ where } I = \{writeok_m, readok_m\}, H = \{write_m, read_m, busy_m, free_m\}, \gamma(write_m, free_m) = writeok_m, \text{ and } \gamma(read_m, busy_m) = readok_m.$$

Clearly, process terms *ReadWrite* and *Status* execute concurrently with synchronisation of actions between $write_m$, $read_m$, $busy_m$ and $free_m$ over the channel m. Intuitively, $write_m$ is synchronised with $free_m$ and leads to an action $writeok_m$ (let us say). Also, $read_m$ is synchronised with $busy_m$ and leads to an action $readok_m$ (let us say). The execution of $writeok_m$ refers to the case that the buffer is free and then is written through the channels m; and the execution of $readok_m$ refers to the case that the buffer is occupied and then is read from the channels m.

Figure 1 shows the interaction of synchronisation actions over the channel m. It is not hard to see that the encapsulation operator is used to enforce actions over the channel m into synchronisation, while the abstraction operator makes synchronisation actions over the channels m invisible. In order to make the

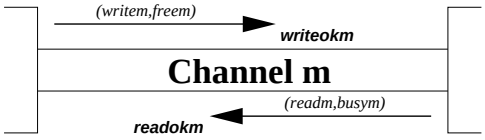

Figure 1: Interaction of synchronisation actions over the channel m.

specification of the process term *Buffer* more interesting, process terms $\ggg$ and $*$ are used to introduce some arbitrary delay and repetition to such a process term. Finally, the $SystemC^{\mathbb{FL}}_{\text{tlm}}$ process BUFFER is given below:

$$\text{BUFFER} \approx \langle Buffer, \sigma \rangle_E \text{ for some } \sigma \text{ and } E \text{ such that } \sigma = \{data \mapsto \bot, busy \mapsto \text{false}, \texttt{time} \mapsto 0\} \text{ and } E = (\emptyset, \{m\}, \emptyset).$$

5 Related Work

Recently, some research works on the SystemC TLM semantics have also been done by means of deduction rules [25] and via PROMELA (an asynchronous formalism). It is generally believed that a SOS provides more intuitive descriptions and that ASM specifications and denotational semantics appear to be less suited to describe the dynamic behaviour of processes [1]. Since processes are the basic units of execution within Verilog, VHDL and SystemC that are used to simulate the behaviour of a device or a system, process algebras with a SOS style semantics are more immediate choices for giving formal specifications of systems in electronic design community (these motivated us to develop $SystemC^{\mathbb{FL}}$ in a process algebraic way with SOS deduction rules).

In the recent years, various formal approaches (based on ASM specifications, deduction rules and denotational semantics) have already been studied and investigated for SystemC (e.g. [17, 22]) that can only be considered as theoretical frameworks, except a few trails (e.g. [6]), because they are not directly executable. In contrast to such formal approaches and others [17, 22, 24, 23, 3, 4, 25, 27], $SystemC^{\mathbb{FL}}$ specifications are completely executable (as in many process algebraic specifications). More precisely, the behaviour of a specification described in $SystemC^{\mathbb{FL}}$ can be illustrated by means of transition traces according to $SystemC^{\mathbb{FL}}$ deduction rules together with the TTS associating to $SystemC^{\mathbb{FL}}$. Similarly, formal analysis of the $SystemC^{\mathbb{FL}}$ specifications can be performed using $SystemC^{\mathbb{FL}}$ deduction rules together with the TTS associating to $SystemC^{\mathbb{FL}}$. Furthermore, we believe that, among other formal approaches on research of SystemC semantics, $SystemC^{\mathbb{FL}}$ is the only one that the correctness of the semantics of $SystemC^{\mathbb{FL}}$ was carefully validated (see the well-definedness of the semantics and congruence result presented in [9, 10] for details).

Recently, as pointed out incorrectly by [27], that $SystemC^{\mathbb{FL}}$ claims its similarity with SystemC, does not have a non-preemptive scheduler (as given in [27]) and does not seem to manage a notion of "event" (which is the basic synchronisation primitive on top of which everything else is built in Sys-

temC), etc. In respond to these, strictly speaking, $SystemC^{\mathbb{FL}}$ is the formalisation of a subset of SystemC based on the classical process algebras and it is not a claim of certain similarity with SystemC; the notion of a non-preemptive scheduler is ensured by the (termination, action and time) transition rules defined for various $SystemC^{\mathbb{FL}}$ operators (see [9, 10] for details); and clearly the watch process term in $SystemC^{\mathbb{FL}}$ is used to model the construct of a "event control" in SystemC.

In SystemC, statements, macros, classes and other core language elements are predefined. Users/modellers can use such language elements in SystemC to make models, which represent, for instance, state machines and asynchronous systems. With the same idea as in SystemC, users/modellers can use process terms in $SystemC^{\mathbb{FL}}$ to model various systems.

Although, there are no deduction rules in $SystemC^{\mathbb{FL}}$ explicitely defined for synchronous and asynchronous composition as defined in [25], the semantics of them can be captured in $SystemC^{\mathbb{FL}}$ in a combination of deduction rules of the parallel composition and the grouping operator. As shown previously, SystemC is currently aimed to use as a vehicle to perform formal analysis of SystemC processes and not for simulation. So, the semantics of delta cycle is not well-defined in $SystemC^{\mathbb{FL}}$ yet. However, a well-defined semantics of SystemC delta cycle, for example, can be found at [20].

6 Concluding Remarks and Future Work

This paper motivated and presented the newly developed language $SystemC^{\mathbb{FL}}_{\text{tlm}}$ (the successor of $SystemC^{\mathbb{FL}}$). The $SystemC^{\mathbb{FL}}_{\text{tlm}}$ language is extended with process term instantiations, SystemC positional connections/named connection modeling features, as well as the semantics for SystemC TLM. In addition, the syntax and semantics of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ are much simpler, intuitive and elegant (than in $SystemC^{\mathbb{FL}}$). We have illustrated the pratical use of $SystemC^{\mathbb{FL}}_{\text{tlm}}$ by means of a TLM example.

As future work, we plan to apply $SystemC^{\mathbb{FL}}_{\text{tlm}}$ for the formal specification and analysis of larger SystemC designs. Also, we focus on SystemC parsing for the making of the automatic translator SC2SCFL2 (from SystemC to $SystemC^{\mathbb{FL}}_{\text{tlm}}$), which is the next release of SC2SCFL.

References

[1] Luca Aceto, Willem Jan Fokkink, and Chris Verhoef. Structural operational semantics. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 3, pages 197–292. Elsevier, 2001.

[2] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, United Kingdom, 1990.

[3] J. Bowen. Animating the semantics of Verilog using Prolog. Technical Report UNU/IIST Report No. 176, International Institute for Software Technology, United Nations University, Macau, 1999.

[4] P.T. Breuer and C. Delgado Kloos, editors. *Formal Semantics for VHDL*. Kluwer Academic Publishers, 1995.

[5] $SystemC^{\mathbb{FL}}$. http://digilander.libero.it/systemcfl/.

[6] A. Gawanmeh, A. Habibi, and S. Tahar. An executable operational semantics for SystemC using abstract state machines. Technical report, Concordia University, Department of Electrical and Computer Engineering, USA, 2004.

[7] Frank Ghenassia, editor. *Transaction-Level Modeling*. Springer, 2005.

[8] IEEE. *IEEE Standard for SystemC Language Reference Manual (IEEE STD 1666TM-2005)*. IEEE, 2005.

[9] K. L. Man. $SystemC^{\mathbb{FL}}$: Formalization of SystemC. In *the 12th Mediterranean Electrotechnical Conference MELECON*, Dubrovnik, Croatia, 2004. IEEE.

[10] K. L. Man. Formal communication semantics of $SystemC^{\mathbb{FL}}$. In *the 8th Euromicro Conference on Digital System Design DSD*, Porto, Portugal, 2005. IEEE.

[11] K. L. Man. Verifying $SystemC^{\mathbb{FL}}$ designs using the SMV model checker. In *the 8th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems DDECS*, Sopron, Hungary, 2005.

[12] K. L. Man. Operational semantics of $SystemC^{\mathbb{FL}}$ and $SystemC^{\mathbb{FL}}_{\text{tlm}}$. Draft paper, 2007.

[13] K. L. Man, A. Fedeli, M. Mercaldi, M. Boubekeur, and M. P. Schellekens. SC2SCFL: Automated SystemC to $SystemC^{\mathbb{FL}}$ translation. In *the 7th International Symposium on Systems, Architectures, Modeling and Simulation*, Lecture Notes in Computer Science 4599, pages 34–45. Springer-Verlag, 2007.

[14] K. L. Man, A. Fedeli, M. Mercaldi, and M. P. Schellekens. $SystemC^{\mathbb{FL}}$: An infrastructure for a TLM formal verification proposal (with an overview on a tool set for practical formal verification of SystemC descriptions). In *the 4th East-West Design & Test Workshop EWDTS*, Sochi, Russia, 2006. IEEE.

[15] Ken L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.

[16] M. Mercaldi, A. Fedeli, and K.L. Man. SC2SCFL: An overview. In *the 4th IEEE International SoC Conference*, Seoul, South Korea, 2007.

[17] W. Mueller, J. Ruf, D. Hofmann, J. Gerlach, T. Kropf, and W. Rosenstiehl. The simulation semantics of SystemC. In *the Proceedings of DATE*, 2001.

[18] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.

[19] NuSMV. *NuSMV Model Checker User Manual*, 2006. http://nusmv.irst.itc.it/.

[20] Xiaoqing Peng, Huibiao Zhu, Jifeng He, and Naiyong Jin. An operational semantics of an event-driven system-level simulator. In *the Proceedings of 30th Annual IEEE/NASA Software Engineering Workshop*, 2006.

[21] G. D. Plotkin. A structural approach to operational semantics. Technical Report DIAMI FN-19, Computer Science Department, Aarhus University, 1981.

[22] Ashraf Salem. Formal semantics of synchronous SystemC. In *the Proceedings of DATE*, 2003.

[23] G. Schneider and X. Qiwen. Towards a formal semantics of Verilog using duration calculus. In *Formal Techniques for Real-Time and Fault Tolerant Systems (FTRTFT'98)*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[24] G. Schneider and X. Qiwen. Towards an operational semantics of Verilog. Technical Report UNU/IIST Report No. 147, International Institute for Software Technology, United Nations University, Macau, 1998.

[25] R. K. Shyamasundar, F. Doucet, R. Gupta, and I. H. Kruger. Compositional reactive semantics of SystemC and verification in rulebase. In *the Proceedings of the Workshop on Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, 2007.

[26] SystemC. *SystemC Users Guide and SystemC Language Reference Manual (Version 2.2)*. http://www.systemc.org.

[27] Claus Traulsen, Jerome Cornet, Matthieu Moy, and Florence Maraninchi. A SystemC/TLM semantics in Promela and its possible applications. In *the 14th Workshop on Model Checking Software SPIN*, 2007.

[28] C. Verhoef. A general conservative extension theorem in process algebra. In *the Proceedings of PROCOMET*, 1994.