International Journal of Computer Theory and Engineering, Vol. 1, No. 4, October2009
1793-8201

# SystemC$_{tlm}^{FL}$: the Successor of SystemC$^{FL}$

K.L. Man and M. Mercaldi

***Abstract*—In this paper, we introduce *SystemC*$_{tlm}^{FL}$, an algebraic theory based on classical process algebras "Algebra of Communicating Processes (ACP)" and "A Timed Process Algebra for Specifying Real-Time Systems (ATP)" that can be used to specify and analyze the behavior of SystemC designs. This language is the successor of the *SystemC* $^{FL}$ language. The *SystemC*$_{tlm}^{FL}$ language extends *SystemC*$^{FL}$ with the possibility to define process term instantiations and for the use of SystemC positional connections/named connection and Transaction Level Modeling (TLM). We illustrate the practical use of *SystemC*$_{tlm}^{FL}$ by means of several examples (including a TLM example).**

***Index Terms*—SystemC, formal semantics, *SystemC*$^{FL}$, *SystemC*$_{tlm}^{FL}$, process algebras, formal specification and analysis, transaction level modeling**

## I. INTRODUCTION

SystemC [12] is a modeling language consisting of C++ class libraries and a simulation kernel for *Hardware Description Language* (HDL) designs, encompassing system-level behavioral descriptions down to *Register Transfer Level* (RTL) representations. Nowadays, SystemC is becoming the de-facto-standard for system level modeling and design in industry. SystemC can also address the need for directly expressing heterogeneous and hierarchical behaviors for modeling specific embedded systems [36]; and to test such embedded systems [8]. Despite its successes, SystemC has no formal semantics. Although some attempts to apply formal methods to verify SystemC designs have been made, it still does miss the possibility of formal reasoning of designs described in SystemC. The goal of developing a SystemC formal semantics is to provide a complete and unambiguous specification of the language. It also contributes significantly to information sharing, to description portability, and to integration of various applications such as simulation, synthesis, and formal verification.

In a attempt (from the free time of a Ph.D. student with some knowledge in *Electronic Design Automation* (EDA), *For-mal Methods* and *Process Algebras* [4], [3]) to give a formal semantics of a reasonable subset of SystemC based on process algebras that could be used for the formal specification and analysis of SystemC designs, the formal language *SystemC*$^{FL}$ (SCFL in ASCII format) [7], [21], [19], [23], [17] was first defined in [16] (2004); and subsequently extended with some features in [20] (2005).

*SystemC*$^{FL}$ maybe regarded as the formalization of a reasonable subset of SystemC based on the classical process algebras *Algebra of Communicating Processes* (ACP) [4] and *A Timed Process Algebra for Specifying Real-Time Systems* (ATP) [32]. The semantics of *SystemC*$^{FL}$ is defined by means of deduction rules in a *Structured Operational Semantics* (SOS) [35] style that associates a time transition system (TTS) with a *SystemC*$^{FL}$ process. A set of properties is presented for a notion of bisimilarity.

More precisely, *SystemC*$^{FL}$ is aimed at giving formal specifications of SystemC designs and to perform formal analysis of SystemC processes. Furthermore, *SystemC*$^{FL}$ is a single formalism that can be used for specifying concurrent systems, finite state systems and real-time systems (as in SystemC). Desired properties of these systems specified in *SystemC*$^{FL}$ can be verified with existing formal verification tools by formally translating them into different formats that are the input languages of such tools. Hence, *SystemC*$^{FL}$ can be purportedly used for formal verification of SystemC designs. For instance, safety properties of concurrent systems specified in *SystemC*$^{FL}$ can be verified (see [24]) by translating those systems to PROMELA [11], which is the input language of the SPIN Model Checker [11].

Similarly, [22] reported that some desired properties of finite state systems specified in *SystemC*$^{FL}$ can be fed into the SMV Model Checker [29] to verify them. Moreover, a formal translation was defined in [15] from *SystemC*$^{FL}$ to a variant (with very general settings) of timed automata [2]. The practical benefit of the formal translation from a *SystemC*$^{FL}$ specification (describing a real-time system) to a timed automaton is to enable verification of timing properties of the *SystemC*$^{FL}$ specification using existing verification tools for timed automata, such as UPPAAL [14].

During the last few years, we have seen that *SystemC*$^{FL}$ has been successfully used to give formal specifications of SystemC designs (see also [16], [18], [20]). However, for formal analysis purposes, users have been required to manually transform their SystemC codes into corresponding *SystemC*$^{FL}$ specifications. To verify some desired properties of *SystemC*$^{FL}$ specifications using existing formal verification tools (see also [15], [22], [24]), similarly, manual translations have been needed for turning *SystemC*$^{FL}$ specifications into corresponding terms of the input language of the selected formal verification tool. Since manual transformation and translations between SystemC codes, *SystemC*$^{FL}$ specifications and various formalisms are quite laborious and therefore error-prone, these translations have to be automated.

Nowadays, *Transaction Level Modeling* (TLM) is indispensable to solve a variety of practical problems (e.g. pro-viding an early platform for software development and sys-tem 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 [10] for excellent surveys on the topic of TLM.

Ka Lok Man is with Solari, Hong Kong,
M. Mercaldi is with M.O.S.T., Turin, Italy

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 de-facto 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 complex-ity 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 [28], we have focused our attention on SystemC as a language for TLM, and selected *SystemC*$^{FL}$ as the language to formally represent SystemC designs.

In the frame of a tight collaboration between re-searchers/engineers from industrial entities and research institutes, several tools for *SystemC*$^{FL}$ have been being developed. These tools enable automatic translations from SystemC codes to *SystemC*$^{FL}$ specifications and from *SystemC*$^{FL}$ specifications to various formalisms that are the input languages of some existing formal verification tools. Using *SystemC*$^{FL}$ tools in combination with some formal verification tools yields automatic verifications of SystemC designs via *SystemC*$^{FL}$ 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*$^{FL}$ specifications that can be further mapped to the input languages of several formal verification tools (e.g. SPIN and NuSMV [33]). Recently, such an automatic translation tool SC2SCFL has been developed in the Java language (JDK 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 [27], [30]) to the corresponding specifications in SystemC, it is not applicable in practice to deal with the translation of industrial SystemC designs. Our experience with SC2SCFL tells us that, based on the current semantics of *SystemC*$^{FL}$, it is impossible to build a translator in such a way that it can be used to translate complex SystemC designs, because *SystemC*$^{FL}$ is not expressive enough to formally represent the current version of SystemC (2.2). For instance, *SystemC*$^{FL}$ (developed in 2004) has no well-defined semantics for TLM and cannot deal with SystemC process instantiations and positional connections modeling features.

After having several attempts, by means of defining new semantics and new operators, to extend *SystemC*$^{FL}$ to cope with the features such as SystemC TLM and process instantiation; it turned out to be very difficult to show that *SystemC*$^{FL}$ with new operators could be an operational conservative extension [43] of *SystemC*$^{FL}$ as defined in [16], [20]. Furthermore, we had several ideas to improve *SystemC*$^{FL}$ in such a way that the semantics of *SystemC*$^{FL}$ would be more intuitive, simpler and elegant.

Hence, we made a decision to redesign *SystemC*$^{FL}$. Recently, the successor of *SystemC*$^{FL}$, called *SystemC*$_{tlm}^{FL}$ (SCFL2 in ASCII format), has been developed. The aim of *SystemC*$_{tlm}^{FL}$ is to serve as formalism to formally rep-resent SystemC (current version) including SystemC TLM features. In this paper, we sketch the newly developed formal language *SystemC*$_{tlm}^{FL}$. The *SystemC*$_{tlm}^{FL}$ language extends *SystemC*$^{FL}$ with the possibility to define process term instantiations and for the use of SystemC positional connections/named connection and SystemC TLM.

### A. Structure

The reminder of the paper is organized as follows. In Section II, we give a brief overview of *SystemC*$^{FL}$ formalism that is relevant for the use in this paper. The motivations and outlines of the development of *SystemC*$_{tlm}^{FL}$ are given in Section III and Section IV presents the *SystemC*$_{tlm}^{FL}$ language including the syntax and the formal semantics. By means of several examples, the practical use of *SystemC*$_{tlm}^{FL}$ is illustrated in Section V and Section VI discusses the related work of *SystemC*$^{FL}$. Finally, concluding remarks are made in Section VII and the direction of future work is pointed out in the same section.

## II. Formal Language SystemC$^{FL}$

For the reason of space limitation, an overview of SystemC is not given in this paper. Some familiarity with SystemC is required. The desirable background can, for example, be found in [12]. In this section, we give a short overview of the formal language *SystemC*$^{FL}$ (that is relevant for the use in this paper). Also note that *SystemC*$^{FL}$ is strongly influenced by ACP and ATP. Hence, fundamental mechanisms used in *SystemC*$^{FL}$ to model processes are similar to those process algebras.

### A. SystemC$^{FL}$ Data Types

In order to define the semantics of *SystemC*$^{FL}$ processes, we need to make some assumptions about the data types. Let *Var* denote the set of all variables ($x_0$, . . . , $x_n$), and *Value* denote the set of all possible values ($v_0$, . . . ,$v_m$) that contains at least B (Booleans) and R (reals). A valuation is a partial function from variables to values (e.g. $x_0$ $7\rightarrow v_0$). The set of all valuations is denoted by Σ. The set *Ch* of all channels and the set *S* of all sensitivity lists with clocks maybe used in *SystemC*$^{FL}$ processes that are assumed.

Notice that the above proposed data types are the fundamental ones. Several extensions of data types (e.g. "sc bit" and "sc logic") were already introduced in [18].

*B. Syntax of the SystemC^FL^ Language*

P denotes the set of process terms in *SystemC*^FL^ and $p \in P$ are the core elements of *SystemC*^FL^. The formal language *SystemC*^FL^ is defined according to the following grammar for process terms $p \in P$ :

$p$ ::= $\delta$ | skip | $x := e$ | $e_{\mathrm{n}}$ | o | $p$ J $b$ I $p$
$b$ ª $p$ | $p \bullet p$ | $p\Theta p$ | $p\ 4_d\ p$ | $p^{\cdot\cdot d}p$ | $*p$
$p$ k $p$ | $\underline{p}$ k $p$ | $p$ v $p$ | $\partial_H(p)$ | $\tau_I(p)$ | $\pi(p)$
a($p$) | Y Â $p$

Below is a brief introduction of the syntax of *SystemC*^FL^:

- The deadlock $\delta$ is introduced as a constant, which represents no behavior.
- The skip process term skip performs the internal action $\tau$, which is not externally visible.
- The assignment process term $x := e$, which assigns the value of expression $e$ to variable $x$ (modeling a SystemC assignment statement).
- The delay process term $e_{\mathrm{n}}$ is able to first delay the value of numerical expression $e_{\mathrm{n}}$ and then terminates by means of an internal action $\tau$.
- The unbounded delay process term o (modeling a SystemC wait statement) may delay for a long time that is unbounded or perform the internal action $\tau$.
- The conditional composition $p$ J $b$ I $q$ 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 $b$ ª $p$ 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 non-deterministic choice between $p$ and $q$.
- The timeout process term $p\ 4_d\ q$ (modeling a SystemC time out construct) behaves as $p$ if $p$ performs a time transition before a time $d \in \mathrm{R}_{>0}$. Otherwise, it behaves as $q$.
- The watchdog process term $p^{\cdot\cdot d}q$ behaves as $p$ during a period of time less than $d$, at time $d$, $q$ takes over the execution from $p$ in $p^{\cdot\cdot d}q$; if $p$ performs an internal cancel $\chi$ action, then the delay is cancelled, and the subsequent behavior is that of $p$ after $\chi$ is executed.
- The repetition process term $*p$ (modeling a SystemC loop construct) executes $p$ zero or more times.
- The parallel composition $p$ k $q$, the left-parallel composition $\underline{p}$ k $q$ and the communication composition $p$ v $q$ are used to express parallelism in which actions are executed in an interleaving manner with the possibility of synchronization of actions. The synchronization of actions take place using a (partial, commutative and associative) synchronization function $\gamma \in A_\tau \times A_\tau\ 7{\rightarrow} A_\tau$ (the set $A_\tau$ is defined in Subsection II-C). For example, if the actions $a$ and $b$ synchronize, 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 $\tau$.
- • The maximal progress $\pi(p)$ assigns action transitions a higher priority over time transitions; this operator is needed to establish a desired communication behavior, 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 a($p$).
- The signal emission operator Y Â $p$ requires that the predicate Y always holds; if it is the case, Y Â $p$ behaves like $p$, otherwise, it is a $\delta$; this operator is needed for defining the translation from *SystemC*^FL^ to the SMV language [29] (see also [22]).

It is worth mentioning that the syntax in ASCII format of a subset of *SystemC*^FL^ was defined in [28] to ease the development of *SystemC*^FL^ toolset.

*1) Outlines:* It is worth to show some outlines concerning the syntax of *SystemC*^FL^:

- Synchronous and asynchronous systems. It is not hard to see that the behavior of asynchronous systems can be easily modeled using the parallel composition operator in *SystemC*^FL^, whose actions are interleaved between process terms (e.g. $x := 1$ k $y := 5$) in a parallel context. In addition, it is possible to describe synchronous systems, using the parallel composition operator in *SystemC*^FL^ with the application of the grouping operator on it (e.g. a($x := 1$ k $y := 5$)), where the assignments are taken into account in parallel and simultaneously. We illustrate it by means of a simple example. Let us consider the process term a($x := 1$ k $y := 5$), it can be rewritten to a($x := 1 \bullet y := 5\ \Theta\ y := 5 \bullet x := 1$) and then to a($x := 1 \bullet y := 5$) Θ a($y := 5 \bullet x := 1$) using *SystemC*^FL^ axioms/properties [25]. Process term a($x := 1 \bullet y := 5$) Θ a($y := 5 \bullet x := 1$) models a non-deterministic choice between a($x := 1 \bullet y := 5$) and a($y := 5 \bullet x := 1$). Let us first discuss the process term a($y := 5 \bullet x := 1$). The application of the grouping operator to the process term $x := 1 \bullet y := 5$ makes the assignment $x := 1$ followed by $y := 5$ become atomic (i.e. in one transition). A similar reasoning can be made to describe the behavior of the process term a($y := 5 \bullet x := 1$). Putting all together, from an observer's point of view, the process term a($x := 1 \bullet y := 5\ \Theta\ y := 5 \bullet x := 1$) only performs a single transition in which the assignments $x := 1$ and $y := 5$ are executed in parallel and simultaneously.
- TLM and communication mechanism in SystemC. Informal semantics of SystemC in [12] states that SystemC incorporates both point-to-point communication and multi-party communication mechanisms for the interaction between concurrent processes. However, there are no (specific) statements in SystemC for modeling these communication mechanisms. Loosely speaking, components in TLM are modeled as modules/processes in a parallel context. They are communicated in the form of transactions through an abstract channel. In order to capture the communication behaviors as indicated above between concurrent processes, operators k, k , v, $\partial_H$ , $\tau_I$ , and $\pi$ (from ACP) were introduced in *SystemC*^FL^. Over the years, the communication mechanism in ACP has been widely used for modeling such communication behaviors. In addition, the idea of using a

synchronization/communication function $\gamma$ (ACP) to define the synchronization/communication behaviors among parallel processes is particularly well-suited for defining TLM semantics possibly in *SystemC*$^{FL}$ (see [25] for details).

## III. From SystemC$^{FL}$ to SystemC$_{TLM}^{FL}$

As we already mentioned in Section I, it turned out to be impossible to use *SystemC*$^{FL}$ to formally represent the cur-rent version of SystemC. The main clauses are the following:

- Expressivity and TLM. Clearly, *SystemC*$^{FL}$ (developed in 2004) is rather old. It is not expressive enough to formally represent SystemC (today) and *SystemC*$^{FL}$ has no well-defined semantics for TLM.
- Lack of SystemC features. There are also quite a lot of SystemC constructs and features that were not formalized in *SystemC*$^{FL}$ 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*$^{FL}$) is not intuitive for designers and engineers in the electronic design community. Also, designers and engineers are uncomfortable with mathematical notations used in *SystemC*$^{FL}$.
- Unintuitive semantics. Needless to say that the formal semantics of *SystemC*$^{FL}$ 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*$^{FL}$ process is highly unintuitive for the users. According to the deduction rules of some *SystemC*$^{FL}$ operators (e.g. the watch operator $^{a}$), this is needed and can be used to observe the change of the valuation of variables in the sensitivity list.

## IV. Formal Language SystemC$_{TLM}^{FL}$

Based on the concept of *SystemC*$^{FL}$ towards a slightly richer language, the successor of *SystemC*$^{FL}$, the formal language *SystemC*$_{tlm}^{FL}$ 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. A detailed account of *SystemC*$_{tlm}^{FL}$ date types, syntax and semantics can already be found at [44], please refer to [44] for details.

## V. Examples in SystemC$_{TLM}^{FL}$

In this section, we aim to illustrate that *SystemC*$_{tlm}^{FL}$ can be used for SystemC positional connections/named connection modeling and SystemC TLM.

### A. Synchronous D Flip Flop Example

Using the syntax and semantics of *SystemC*$_{tlm}^{FL}$, we can obtain a much more simpler, intuitive and elegant specification of the synchronous D flip flop (as shown in Subsection II-E) as follows:

$h\neg clk^{-} \wedge clk^{+\,a}\, Q := d, \sigma O_E$ for some $\sigma$ and $E$ such that
$\sigma = \{clk\ 7\!\to\!\text{false},\ d\ 7\!\to\!\text{true},\ Q\ 7\!\to\!\text{true},\ time\ 7\!\to\!0\}$
and

$E = (\{clk\},\ \ ,\ \ ).$

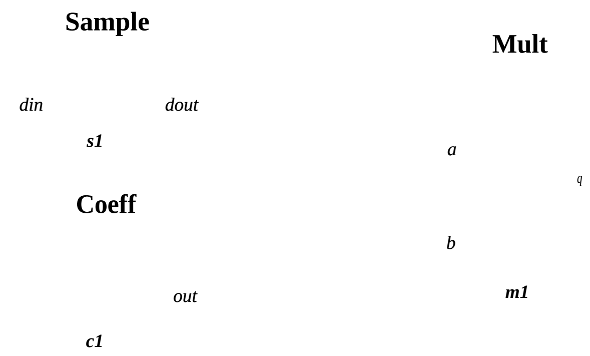


Fig. 2. A filter design.

### B. Filter Design Example

Figure 2 depicts a simple filter design. This example consists of three modules Sample, Mult and Coeff.

In *SystemC*$_{tlm}^{FL}$, $Sample(d^0_{out}, d^0_{in}) = p_s$, $Mult(q^0, a^0, b^0) = p_m$ and $Coe\ (out^0) = p_c$ are the process term definitions for modules Sample, Mult and Coeff respectively, where $p_s$, $p_m$ and $p_c$ are process terms that describe some behavior of such modules Sample, Mult and Coeff. Notice that the variables with a prime (e.g. $d^0_{out}$ and $q^0$) are the formal parameter variables in the process term definitions.

In the example, we use named connection for the component instantiations, for example, $d_{in}$ of $s_1$ (which is the instantiation of Sample) is connected to $q$ of $m_1$ (which is the instantiation of Mult). The *SystemC*$_{tlm}^{FL}$ filter process is given below:

$h$Y Â $(s_1(d_{out}, d_{in})\ k\ m_1(q, a, b)\ k\ c_1(out)), \sigma O_E$ for some Y, $\sigma$ and $E$ such that Y t $s_1 = Sample \wedge m_1 = Mult \wedge c_1 = Coe \quad \wedge d_{in} = q \wedge d_{out} = a \wedge out = b,$
$\sigma = \{d_{out} = d_{in} = q = a = b = out\ 7\!\to \perp, time\ 7\!\to\!0\}$ and
$E = (\ \ ,\ \ , \{Sample(d^0_{out}, d^0_{in}) = p_s, Mult(q^0, a^0, b^0) = p_m, Coe\ \ (out^0) = p_c\}).$

For simplicity, the behavior of some variables (e.g. $d_{in}$ and $q$) defined by means of the process terms (e.g. $p_s$,$p_m$ and $p_c$) in the process term definitions for modules (e.g. Sample and Mult) is not given. Clearly, all process term instantiations execute in a parallel context (i.e. $s_1(d_{out}, d_{in})\ k\ m_1(q, a, b)\ k\ c_1(out)$). The signal emission operator with the predicate Y applied on such a parallel context is used to enforce/ensure the correct named connection for the component instantiations (e.g. $d_{out} = a$) and process term instantiations (e.g. $s_1$ is the instantiation of Sample) always hold during the evolution (in terms of transitions) of the *SystemC*$_{tlm}^{FL}$ filter process. It is not hard to see that using the idea of positional connection to describe the filter design can also work well.

### C. TLM Buffer Example

This subsection presents an example which implements a 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:

$$Status \equiv busy := \text{false} \;\; \text{J} \;\; busy \;\; \text{I} \;\; busy := \text{true}.$$

As mentioned already in Definition 5, actions are considered as parameters of $SystemC_{\text{tlm}}^{\text{FL}}$ 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* := false and *busy* := true respectively. These actions are used for the synchronization 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* := true and *data* := 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 *Bu er* as follows:

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

Clearly, process terms *ReadWrite* and *Status* exe-cute concurrently with synchronization of actions between $write_m$, $read_m$, $busy_m$ and $free_m$ over the channel *m*. Intuitively, $write_m$ is synchronized with $free_m$ and leads to an action $writeok_m$ (let us say). Also, $read_m$ is synchronized 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 3 shows the interaction of synchronization 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 synchronization, while the abstraction operator makes synchronization actions over the channels *m* invisible. In order to make the specification of the process term *Buffer* more interesting, process terms o and are used to introduce some arbitrary delay and repetition to such a process term.

Finally, the $SystemC_{\text{tlm}}^{\text{FL}}$ process BUFFER is given below:

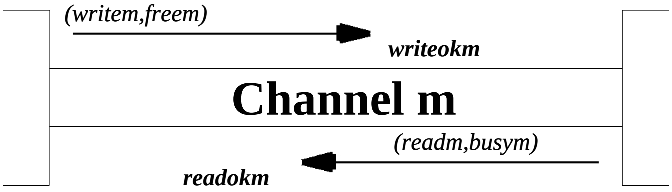


Fig. 3. Interaction of synchronization actions over the channel *m*.

BUFFER t $hBu\,er, \sigma O_E$ for some $\sigma$ and $E$ such that
$\sigma = \{data \; 7\!\!\to \; \perp, busy \; 7\!\!\to \text{false}, \text{ time } 7\!\!\to 0\}$ and
E = ( , {m}, ).

## VI. Related Work

Over the last ten years or so, research works in formal semantics in electronic design community that have targeted to obtain some applicable opportunity mainly focused on Verilog, VHDL and SystemC. Quite often, their definitions were based on Abstract State Machine (ASM) specifications, Denotational Semantics and rewrite rules [31], [37], [39], [38], [5], [6]; for instance (as related work for the research in SystemC semantics), [31], [37] addressed respectively the simulation semantics of SystemC in the form of distributed ASM specifications and in the denotational semantics for various subsets of SystemC. Recently, some research works on the SystemC TLM semantics have also been done by means of deduction rules [40] 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 behavior of processes [1]. Since processes are the basic units of execution within Verilog, VHDL and SystemC that are used to simulate the behavior 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 $SystemC^{\text{FL}}$ 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. [31], [37]) that can only be considered as theoretical frameworks, except a few trails (e.g. [9]), because they are not directly executable.

In contrast to such formal approaches and others [31], [37], [39], [38], [5], [6], [40], [42], $SystemC^{\text{FL}}$ specifications are completely executable (as in many process algebraic specifications). More precisely, the behavior of a specification described in $SystemC^{\text{FL}}$ can be illustrated by means of transition traces according to $SystemC^{\text{FL}}$ deduction rules together with the TTS associating to $SystemC^{\text{FL}}$. Similarly, formal analysis of the $SystemC^{\text{FL}}$ specifications can be performed using $SystemC^{\text{FL}}$ deduction rules together with the TTS associating to $SystemC^{\text{FL}}$.

However, in our view, $SystemC^{\text{FL}}$ has generally some disadvantages. First, the formal language $SystemC^{\text{FL}}$ is small and has no well-defined semantics for TLM. Second, the syntax and semantics of *SystemC*$^{FL}$ are probably not intuitive for one not having a strong background in Computer Science; and designers of system-level design are often uncomfortable with mathematical notations used in *SystemC*$^{FL}$.

Recently, as pointed out incorrectly by [42], that *SystemC*$^{FL}$ claims its similarity with SystemC, does not have a non-preemptive scheduler[1] and does not seem to manage a notion of "event" (which is the basic synchronization primitive on top of which everything else is built in SystemC), etc. In respond to these, strictly speaking, *SystemC*$^{FL}$ is the formalization 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 (as given in [42]) is ensured by the (termination, action and time) transition rules defined for various *SystemC*$^{FL}$ operators (see [16], [20] for details); and clearly the watch process term in *SystemC*$^{FL}$ is used to model the construct of a "event control" in SystemC (see the example given in Subsection II-E for details).

In SystemC, statements, macros, classes and other core language elements are predefined. Users/modelers can use such language elements in SystemC to make models, which repre-sent, for instance, state machines and asynchronous systems. With the same idea as in SystemC, users/modelers can use process terms in *SystemC*$^{FL}$ to model various systems. Al-though, there are no deduction rules in *SystemC*$^{FL}$ explicitly defined for synchronous and asynchronous composition as defined in [40], the semantics of them can be captured in *SystemC*$^{FL}$ in a combination of deduction rules of the parallel composition and the grouping operator (see also the example given in Subsection II-B1 for details). 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*$^{FL}$ yet. However, a well-defined semantics of SystemC delta cycle, for example, can be found at [34].

Based on the similar motivations and needs, **PAFSV** [41], [26] (a similar timed process algebra) was recently introduced for the formal specification and analysis of IEEE 1800™ SystemVerilog [13] designs. Clearly, SystemVerilog and SystemC are similar and the research work in **PAFSV** was highly inspired by the theoretical aspects of *SystemC*$^{FL}$. Hence, a formal comparison between them is indispensable (as a future work).

## VII. Concluding Remarks and Future Work

Although the introduction of *SystemC*$^{FL}$ (since three years ago) initiated an attempt to extend the knowledge and experience collected in the field of process algebras to system-level modeling and design, it turned out unfortunately that *SystemC*$^{FL}$ could not be practically used to formally repre-sent the current version of SystemC (2.2).

Indeed, the current semantics of *SystemC*$^{FL}$ is rather old. In this paper, we have motivated and presented the newly developed language of *SystemC*$_{tlm}^{FL}$, the successor of *SystemC*$^{FL}$. The *SystemC*$_{tlm}^{FL}$ language is extended with process term instantiations, SystemC positional con-nections/named connection modeling features as well as the semantics for SystemC TLM. In addition, the syntax and semantics of *SystemC*$_{tlm}^{FL}$ are much simpler, intuitive and elegant (than in *SystemC*$^{FL}$). We have illustrated the practical use of *SystemC*$_{tlm}^{FL}$ through some examples of SystemC designs including a SystemC TLM design.

As future work, we plan to apply *SystemC*$_{tlm}^{FL}$ 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*$_{tlm}^{FL}$), which is the next release of SC2SCFL.

## Acknowledgment

K.L. Man would like to thank Jos Baeten, Bert van Beek, Mohammad Mousavi, Koos Rooda, Ramon Schiffelers, Pieter Cuijpers, Michel Reniers, Kees Middelburg, Uzma Khadim and Muck van Weerdenburg for many stimulating and helpful discussions focusing on process algebras for distinct systems in the past few years.

The authors would like to thank Andrea Fedeli, Michel Schellenkens and Menouer Boubekeur for their cooperation of the previous research work on *SystemC*$^{FL}$; and Flaviano Garberoglio and Angelo Trischitta for their comments and support for the current research work on *SystemC*$_{tlm}^{FL}$.

## References

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

[2] R. Alur and D.L. Dill. A theory of timed automata. Theoretical Computer Science, 126:183–236, 1994.

[3] J. C. M. Baeten and C. A. Middelburg. Process Algebra with Timing. EACTS Monographs in Theoretical Computer Science. Springer-Verlag, 2002.

[4] 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.

[5] 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.

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

[7] SystemCFL. http://digilander.libero.it/systemcfl/.

[8] A. Fin, F. Fummi, and M. Signoretto. SystemC: A homogenous environment to test embeddded systems. In the IEEE Codesign Conference, Copenhagen, Denmark, 2001.

[9] 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 Engineer-ing, USA, 2004.

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

[11] Gerard J. Holzmann. The SPIN Model Checker: Primer and Reference Manual. Addison Wesley Professional, Boston, 2003.

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

[13] IEEE. IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language (IEEE STD 1800TM-2005). IEEE, 2005.

[14] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. Journal on Software Tools for Technology, 1(1–2):134–152, 1997.

[15] K. L. Man. Analyzing SystemC$^{FL}$ designs using timed automata. In the 9th IEEE Baltic Electronics Conference BEC, Tallinn, Estonia, 2004.

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

[17] K. L. Man. Extensions of SystemC$^{FL}$ for mixed-signal system and formal verification. In the PROGRESS 2004 Embedded Systems Symposium PROGRESS04, Nieuwegein, The Netherlands, 2004.

[18] K. L. Man. Modeling with the formal language of SystemC: Case

studies. In the 11th IEEE International Conference Mixed Design of Integrated Circuits and Systems MIXDES, Szczecin, Poland, 2004.
[19] K. L. Man. SystemC$^{FL}$: A formalism for hardware/software co-design. In the 17th European Conference on Circuits Theory and Design ECCTD05, Cork, Ireland, 2005. IEEE.
[20] K. L. Man. Formal communication semantics of SystemC$^{FL}$. In the 8th Euromicro Conference on Digital System Design DSD, Porto, Portugal, 2005. IEEE.
[21] K. L. Man. An overview of SystemC$^{FL}$. In the Ph.D. Research in Microelectronics and Electronics Conference PRIME05, Lausanne, Switzerland, 2005. IEEE.
[22] K. L. Man. Verifying SystemC$^{FL}$ designs using the SMV model checker. In the 8th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems DDECS, Sopron, Hungary, 2005.
[23] K. L. Man. SystemC$^{FL}$: Formal specification and analysis of hardware/software co-designs. The World Scientific and Engineering Academy and Society Transactions on Circuits and Systems, 3(5):361–368, 2006.
[24] K. L. Man. Formal verification of SystemC$^{FL}$ specifications using SPIN. In the 5th WSEAS International Conference on Microelectronics, Nanoelectronics and Optoelectronics MINO, Prague, Czech Republic, 2006. WSEAS.
[25] K. L. Man. Operational semantics of SystemC$^{FL}$ and SystemC$_{tlm}{}^{FL}$. Draft paper, 2007.
[26] K. L. Man, M. Boubekeur, and M. P. Schellekens. Process algebraic approach to SystemVerilog. In the 20th IEEE Canadian Conference on Electrical and Computer Engineering, Columbia, Canada, 2007. IEEE.
[27] K. L. Man, A. Fedeli, M. Mercaldi, M. Boubekeur, and M. P. Schellekens. SC2SCFL: Automated SystemC to SystemC$^{FL}$ trans-lation. In the 7th International Symposium on Systems, Architectures, Modeling and Simulation, Lecture Notes in Computer Science 4599, pages 34–45. Springer-Verlag, 2007.
[28] K. L. Man, A. Fedeli, M. Mercaldi, and M. P. Schellekens. SystemC$^{FL}$: 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.
[29] Ken L. McMillan. Symbolic Model Checking. Kluwer Academic Publisher, 1993.
[30] M. Mercaldi, A. Fedeli, and K.L. Man. SC2SCFL: An overview. In the 4th IEEE International SoC Conference, Seoul, South Korea, 2007.
[31] W. Mueller, J. Ruf, D. Hofmann, J. Gerlach, T. Kropf, and W. Rosen-stiehl. The simulation semantics of SystemC. In the Proceedings of Design, Automation, and Test in Europe, 2001.
[32] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. Information and Computation, 114:131–178, 1994.
[33] NuSMV. NuSMV Model Checker User Manual, 2006. http://nusmv.irst.itc.it/.
[34] Xiaoqing Peng, Huibiao Zhu, Jifeng He, and Naiyong Jin. An opera-tional semantics of an event-driven system-level simulator. In the Pro-ceedings of 30th Annual IEEE/NASA Software Engineering Workshop, 2006.
[35] G. D. Plotkin. A structural approach to operational semantics. Technical Report DIAMI FN-19, Computer Science Department, Aarhus University, 1981.
[36] Ivan Radojevic, Zoran A. Salcic, and Partha S. Roop. Modeling embedded systems: From SystemC and Esterel to DFCharts. IEEE Design & Test of Computers, 23(5):348–358, 2006.
[37] Ashraf Salem. Formal semantics of synchronous SystemC. In the Proceedings of Design, Automation, and Test in Europe, 2003.
[38] 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.
[39] 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.
[40] R. K. Shyamasundar, F. Doucet, R. Gupta, and I. H. Kruger. Com-positional 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.
[41] PAFSV. http://digilander.libero.it/systemcfl/pafsv.
[42] 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.
[43] C. Verhoef. A general conservative extension theorem in process algebra. In the Proceedings of PROCOMET, 1994.
[44] K.L. Man, M. Mercaldi, F. Garberoglio, A. Trischitta, H.Y. Lai, .M. Ho, SystemC$^{FL}{}_{tlm}$: Motivation and Development. in the Proceedings of the IAENG International MultiConference of Engineers and Computer Scientists 2008 (IMECS 2008), Hong Kong, 2008.

**Ka Lok Man** He holds a Dr. Eng. degree in Electronic Engineering from Politecnico di Torino, Italy, and a PhD degree in Computer Science from Technische Universiteit Eindhoven, The Netherlands.Currently, he is a senior researcher at the Centre for Efficiency-Oriented Languages (CEOL), Department of Computer Science, University College Cork, Ireland and a research and engineering consultant for Solari - Hong Kong. His research interests include logic synthesis, formal verification and low power design methodologies for integrated circuits and systems, formalization of SystemC and SystemVerilog designs including TLM, formal methods, process algebras, formal analysis of real-time and hybrid systems, communication and wireless sensor networks, reversible computing, and software development. In the above-mentioned topics, he has authored or co-authored about 80 refereed publications including books, edited books, journal articles, book chapters and conference proceedings. He has been a committee member, reviewer, session chair and special session/workshop organizer of different IEEE, IASTED and IAENG conferences.He is also the editor-in-chief of the International Journal of Design, Analysis and Tools for Integrated Circuits and Systems (IJDATICS), the editor of the Journal of Computers (JCP), the associate editor of the Journal of Engineering, Computing and Architecture (JECA), the associate editor of the Journal of Computer Science, Informatics and Electrical Engineering (JCSIEE), the editor of the Journal of Computer Science and Information Technology (JCSIT), the editor of the International Journal of Advancements in Computing Technology (IJACT), the editor of the Magazine of Wireless and Cellular Networks (MWCN), the editor of the International Journal of Digital Content Technology and its Applications (JDCTA) and the editor of the Journal of Convergence Information Technology (JCIT).

**Michele Mercaldi** He received his Dr. Eng, Degree in Software Engineering from the Politecnico di Torino (Italy) in 1998. From 1997 to to 1999 he worked at the Politecnico di Torino as software and database developer. From 2000 to 2007 he worked in MOST (Turin, Italy) which is a highly skilled firm for documental activation using Oracle and Postgresq running on Linux. Currently he is employed by the Critical Path (Turin, Italy) which is one of the world leading providers of internet messaging and directory software.