**SystemC\textsubscript{tlm}^\text{FL}: the Successor of SystemC\textsuperscript{FL}**

K.L. Man and M. Mercaldi

*Abstract*—In this paper, we introduce SystemC\textsubscript{tlm}^\text{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\textsuperscript{FL} language. The SystemC\textsubscript{tlm}^\text{FL} language extends SystemC\textsuperscript{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\textsubscript{tlm}^\text{FL} by means of several examples (including a TLM example).

**Index Terms**—SystemC, formal semantics, SystemC\textsuperscript{FL}, SystemC\textsubscript{tlm}^\text{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), Formal 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\textsuperscript{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\textsuperscript{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\textsuperscript{FL} is defined by means of deduction rules in a Structure Operational Semantics (SOS) [35] style that associates a time transition system (TTS) with a SystemC\textsuperscript{FL} process. A set of properties is presented for a notion of bisimilarity.

More precisely, SystemC\textsuperscript{FL} is aimed at giving formal specifications of SystemC designs and to perform formal analysis of SystemC processes. Furthermore, SystemC\textsuperscript{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\textsuperscript{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\textsuperscript{FL} can be purportedly used for formal verification of SystemC designs. For instance, safety properties of concurrent systems specified in SystemC\textsuperscript{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\textsuperscript{FL} can be fed into the SMV Model Checker [29] to verify them. Moreover, a formal translation was defined in [15] from SystemC\textsuperscript{FL} to a variant (with very general settings) of timed automata [2]. The practical benefit of the formal translation from a SystemC\textsuperscript{FL} specification (describing a real-time system) to a timed automaton is to enable verification of timing properties of the SystemC\textsuperscript{FL} specification using existing verification tools for timed automata, such as UPPAAL [14].

During the last few years, we have seen that SystemC\textsuperscript{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\textsuperscript{FL} specifications. To verify some desired properties of SystemC\textsuperscript{FL} specifications using existing formal verification tools (see also [15], [22], [24]), similarly, manual translations have been needed for turning SystemC\textsuperscript{FL} specifications into corresponding terms of the input language of the selected formal verification tool. Since manual transformation and translations between SystemC codes, SystemC\textsuperscript{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.
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 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 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 SystemCFL 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 SystemCFL have been being developed. These tools enable automatic translations from SystemC codes to SystemCFL specifications and from SystemCFL specifications to various formalisms that are the input languages of some existing formal verification tools. Using SystemCFL tools in combination with some formal verification tools yields automatic verifications of SystemC designs via SystemCFL 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 SystemCFL 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 SystemCFL, it is impossible to build a translator in such a way that it can be used to translate complex SystemC designs, because SystemCFL is not expressive enough to formally represent the current version of SystemC (2.2). For instance, SystemCFL (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 SystemCFL to cope with the features such as SystemC TLM and process instantiation; it turned out to be very difficult to show that SystemCFL with new operators could be an operational conservative extension [43] of SystemCFL as defined in [16], [20]. Furthermore, we had several ideas to improve SystemCFL in such a way that the semantics of SystemCFL would be more intuitive, simpler and elegant.

Hence, we made a decision to redesign SystemCFL. Recently, the successor of SystemCFL, called SystemCFL (SCFL2 in ASCII format), has been developed. The aim of SystemCFL 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 SystemCFL. The SystemCFL language extends SystemCFL 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 SystemCFL formalism that is relevant for the use in this paper. The motivations and outlines of the development of SystemCFL are given in Section III and Section IV presents the SystemCFL language including the syntax and the formal semantics. By means of several examples, the practical use of SystemCFL is illustrated in Section V and Section VI discusses the related work of SystemCFL. Finally, concluding remarks are made in Section VII and the direction of future work is pointed out in the same section.

### II. FORMAL LANGUAGE SYSTEMCFL

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 SystemCFL (that is relevant for the use in this paper). Also note that SystemCFL is strongly influenced by ACP and ATP. Hence, fundamental mechanisms used in SystemCFL to model processes are similar to those process algebras.

#### A. SystemCFL Data Types

In order to define the semantics of SystemCFL processes, we need to make some assumptions about the data types. Let \( \mathbf{Var} \) denote the set of all variables \( (x_{0}, \ldots, x_{n}) \), and \( \mathbf{Value} \) denote the set of all possible values \( (v_{0}, \ldots, v_{m}) \) that contains at least \( B \) (Booleans) and \( R \) (reals). A valuation is a partial function from variables to values (e.g. \( x_{0} \mapsto 7 \)). The set of all valuations is denoted by \( \Sigma \). The set \( Ch \) of all channels and the set \( S \) of all sensitivity lists with clocks maybe used in SystemCFL 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\textsuperscript{FL} Language

\( P \) denotes the set of process terms in SystemC\textsuperscript{FL} and \( p \in P \) are the core elements of SystemC\textsuperscript{FL}. The formal language SystemC\textsuperscript{FL} is defined according to the following grammar for process terms \( p \in P \):

\[
\begin{align*}
p & ::= \delta \mid \text{skip} \mid x := e \mid o \mid p \mid b \mid p \\
& \mid p \cdot p \mid \Theta p \mid p \cdot \gamma p \mid b \cdot p \mid k \cdot p \\
& \mid p \cdot k \mid p \cdot v \cdot p \mid \tilde{\chi}_0(p) \mid t_i(p) \mid \pi(p) \\
& \quad a(p) \mid Y \cdot A \cdot p
\end{align*}
\]

Below is a brief introduction of the syntax of SystemC\textsuperscript{FL}:

\( \delta \) The deadlock \( \delta \) is introduced as a constant, which represents no behavior.

\( \text{skip} \) The skip process term \( \text{skip} \) performs the internal action \( \tau \), which is not externally visible.

\( x := e \) The assignment process term \( x := e \) assigns the value of expression \( e \) to variable \( x \) (modeling a SystemC assignment statement).

\( o \) The delay process term \( o \) is able to first delay the value of numerical expression \( e \), and then terminates by means of an internal action \( \tau \).

\( b \cdot p \) The unbounded delay process term \( b \cdot p \) (modeling a SystemC wait statement) may delay for a long time that is unbounded or perform the internal action \( \tau \).

\( \Theta p \) The conditional composition \( p \cdot \gamma p \) 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.

\( k \cdot p \) The watch process term \( k \cdot p \) is used to model a SystemC construct of even control.

\( v \cdot p \) The sequential composition \( v \cdot p \) models the process term that behaves as \( v \), and upon termination of \( p \), continues to behave as \( q \).

\( \Theta \) The alternative composition \( \Theta p \) models the process term \( p \) \& \( q \) or \( p \) \& \( q \), and \( p \) \& \( q \) is atomic.

\( \gamma \) The timeout process term \( \gamma \cdot p \) defines a non-deterministic choice between \( p \) and \( q \).

\( \gamma \cdot k \) The parallel composition \( \gamma \cdot k \cdot q \) (modeling a SystemC time out construct) behaves as \( p \) if \( p \) performs a time transition before a time \( d \in R_{\geq 0} \). Otherwise, it behaves as \( q \).

\( \gamma \cdot q \) The watchdog process term \( \gamma \cdot q \) behaves as \( p \) during a period of time less than \( d \), at time \( d \), \( q \) takes over the execution from \( p \) in \( p \cdot q \); if \( p \) performs an internal cancel \( \gamma \) action, then the delay is cancelled, and the subsequent behavior is that of \( p \) after \( \gamma \) is executed.

\( \gamma \cdot k \) The repetition process term \( \gamma \cdot k \) (modeling a SystemC loop construct) executes \( p \) zero or more times.

\( \gamma \cdot k \cdot q \) The parallel composition \( k \cdot q \), the left-parallel composition \( \gamma \cdot k \cdot q \) and the communication composition \( p \cdot q \) 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 \( \gamma \in \lambda \times \lambda \rightarrow \lambda \) (the set \( \lambda \), is defined in Subsection II-C). For example, if the actions \( a \) and \( b \) synchronize, the resulting action \( c \) such that \( \gamma(a, b) = c \).

\( H \) The encapsulation of actions is allowed using \( \tilde{\chi}_0(p) \), where \( H \) represents the set of all actions to be blocked in \( p \).

\( t_i(p) \) The abstraction \( t_i(p) \) behaves as the process term \( p \), except that all action names in \( I \) are renamed to the internal action \( \tau \).

\( \pi(p) \) 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.

\( \Theta \) The grouping of actions in \( p \) and executing them in one single step can be done by using \( a(p) \).

\( \gamma \) The signal emission operator \( Y \cdot A \cdot p \) requires that the predicate \( Y \) always holds; if it is the case, \( Y \cdot A \cdot p \) behaves like \( p \), otherwise, it is a \( \delta \); this operator is needed for defining the translation from SystemC\textsuperscript{FL} to the SMV language [29] (see also [22]).

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

1) Outlines: It is worth to show some outlines concerning the syntax of SystemC\textsuperscript{FL}:

\( \delta \) 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\textsuperscript{FL}, whose actions are interleaved between process terms (e.g. \( x := 1 \cdot k \& y := 5 \)) in a parallel context. In addition, it is possible to describe synchronous systems, using the parallel composition operator in SystemC\textsuperscript{FL} with the application of the grouping operator on it (e.g. \( a(x := 1 \cdot k \& y := 5) \)), 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 \( a(x := 1 \cdot k \& y := 5) \), it can be rewritten to \( a(x := 1 \cdot y := 5 \& y := 5 \cdot x := 1) \) and then to \( a(x := 1 \cdot y := 5 \& y := 5 \cdot x := 1) \) using SystemC\textsuperscript{FL} axioms/properties [25]. Process term \( a(x := 1 \cdot y := 5 \& y := 5 \cdot x := 1) \) models a non-deterministic choice between \( a(x := 1 \cdot y := 5 \& y := 5 \cdot x := 1) \) and \( a(x := 5 \cdot y := 1) \). Let us first discuss the process term \( a(y := 5 \cdot x := 1) \). The application of the grouping operator to the process term \( x := 1 \cdot y := 5 \) 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 \( a(y := 5 \cdot x := 1) \). Putting all together, from an observer’s point of view, the process term \( a(x := 1 \cdot y := 5 \& y := 5 \cdot x := 1) \) only performs a single transition in which the assignments \( x := 1 \) and \( y := 5 \) are executed in parallel and simultaneously.

\( \gamma \) 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 modelling 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 \( k, k \cdot v, \tilde{\chi}_0, t_i \), and \( \pi \) (from ACP) were introduced in SystemC\textsuperscript{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\(_{FL}^{\text{TL}}\) (see [25] for details).

### III. FROM SYSTEMC\(_{FL}^{\text{TL}}\) TO SYSTEMC\(_{TLM}^{\text{TL}}\)

As we already mentioned in Section I, it turned out to be impossible to use SystemC\(_{FL}^{\text{TL}}\) to formally represent the current version of SystemC. The main clauses are the following:

- **Expressivity and TLM.** Clearly, SystemC\(_{FL}^{\text{TL}}\) (developed in 2004) is rather old. It is not expressive enough to formally represent SystemC (today) and SystemC\(_{FL}^{\text{TL}}\) 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\(_{FL}^{\text{TL}}\) 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\(_{FL}^{\text{TL}}\)) is not intuitive for designers and engineers in the electronic design community. Also, designers and engineers are uncomfortable with mathematical notations used in SystemC\(_{FL}^{\text{TL}}\).

- **Unintuitive semantics.** Needless to say that the formal semantics of SystemC\(_{FL}^{\text{TL}}\) is also not intuitive for designers and engineers in the electronic design community.

### IV. FORMAL LANGUAGE SYSTEMC\(_{TLM}^{\text{TL}}\)

Based on the concept of SystemC\(_{FL}^{\text{TL}}\) towards a slightly richer language, the successor of SystemC\(_{FL}^{\text{TL}}\), the formal language SystemC\(_{TLM}^{\text{TL}}\) 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\(_{TLM}^{\text{TL}}\) date types, syntax and semantics can already be found at [44], please refer to [44] for details.

### V. EXAMPLES IN SYSTEMC\(_{TLM}^{\text{TL}}\)

In this section, we aim to illustrate that SystemC\(_{TLM}^{\text{TL}}\) 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\(_{TLM}^{\text{TL}}\), 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:

\[
h \cdot \text{clk}^{-} \land \text{clk}^{+} \ast Q := d, \sigma_{O_{E}} \text{ for some } \sigma \text{ and } E \text{ such that } \\
\sigma = \{\text{clk} \rightarrow \text{false}, \ d \rightarrow \text{true}, \ Q \rightarrow \text{true}, \ \text{time} \rightarrow 0\} \text{ and }
\]

**Sample**

\[
\begin{array}{c}
\text{din} \\
\text{out} \\
\text{c1}
\end{array}
\]

**Mult**

\[
\begin{array}{c}
\text{din} \\
\text{out} \\
\text{c1}
\end{array}
\]

**Coeff**

\[
\begin{array}{c}
\text{din} \\
\text{out} \\
\text{c1}
\end{array}
\]

**B. Filter Design Example**

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

In SystemC\(_{TLM}^{\text{TL}}\), Sample\((\text{d}_{\text{out}}, \ \text{d}_{\text{in}}) = p_{r}, \text{Mult}(q^{0}, q^{0}, b^{0}) = p_{m}\) and Coef\((\text{out}) = p_{r}\) are the process term definitions for modules Sample, Mult and Coeff respectively, where \(p_{r}, p_{m}\) and \(p_{r}\) are process terms that describe some behavior of such modules Sample, Mult and Coeff. Notice that the variables with a prime (e.g. \(d_{\text{out}}^{0}\) 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, \(d_{\text{in}}\) of \(s_{1}\) (which is the instantiation of Sample) is connected to \(q\) of \(m_{1}\) (which is the instantiation of Mult). The SystemC\(_{TLM}^{TLM}\) filter process is given below:

\[
h \cdot Y \cdot (s_{1}(\text{d}_{\text{out}}, \ \text{d}_{\text{in}}) \ k m_{1}(q, a, b) \ k c_{1}(\text{out}), \) \sigma_{O_{E}} \text{ for some } \\
\ Y, \sigma \text{ and } E \text{ such that } Y \cdot t \cdot s_{1} \cdot \text{Sample} \land m_{1} = \\
\text{Mult} \land c_{1} = \text{Coeff} \land \text{d}_{\text{in}} = q \land \text{d}_{\text{out}} = a \land \text{out} = b, \\
\sigma = \{\text{d}_{\text{out}} = \text{d}_{\text{in}} = q = a = b = \text{out} \rightarrow \text{false}, \ \text{time} \rightarrow 0\} \text{ and } \\
E = (\ldots, \{\text{Sample}(\text{d}_{\text{out}}, \ \text{d}_{\text{in}}) = p_{r}, \text{Mult}(q^{0}, a^{0}, b^{0}) = p_{m}, \text{Coeff}(\text{out}) = p_{r}\}).
\]

For simplicity, the behavior of some variables (e.g. \(d_{\text{in}}\) and \(q\)) defined by means of the process terms (e.g. \(p_{r}, p_{m}\) and \(p_{r}\)) 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. \(s_{1}(\text{d}_{\text{out}}, \ \text{d}_{\text{in}}) \ k m_{1}(q, a, b) \ k c_{1}(\text{out})\)). 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. \(d_{\text{out}} = a\)) 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\(_{TLM}^{TLM}\) 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:

\[ \text{Status} \equiv \text{busy} := \text{false} \quad \text{busy} := \text{true}. \]

As mentioned already in Definition 5, actions are considered as parameters of SystemC\textsuperscript{FL} that can be freely instantiated. For this process term, we also write busy\textsubscript{m} and free\textsubscript{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\textsubscript{m} or action free\textsubscript{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:

\[ \text{ReadWrite} \equiv \text{data} := \text{true} \Theta \text{data} := \text{false}. \]

For this process term ReadWrite, we write actions read\textsubscript{m} and write\textsubscript{m} as the actions associating with the assignment process terms data := true and data := false respectively. When read\textsubscript{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\textsubscript{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:

\[ \text{Buffer} \equiv (\text{status} \cdot \tau (\circ \text{buffer} (\text{ReadWrite} \text{status}))), \]

where \( I = \{ \text{writeok}_m, \text{readok}_m \}, H = \{ \text{write}_m, \text{busym}, \text{free}_m \}, \gamma(\text{write}_m, \text{free}_m) = \text{writeok}_m, \text{and} \gamma(\text{read}_m, \text{busym}) = \text{readok}_m. \)

Clearly, process terms ReadWrite and Status execute concurrently with synchronization of actions between write\textsubscript{m}, read\textsubscript{m}, busym and free\textsubscript{m} over the channel \( m \). Intuitively, write\textsubscript{m} is synchronized with free\textsubscript{m} and leads to an action writeok\textsubscript{m} (let us say). Also, read\textsubscript{m} is synchronized with busym and leads to an action readok\textsubscript{m} (let us say). The execution of writeok\textsubscript{m} refers to the case that the buffer is free and then is written through the channels \( m \); and the execution of readok\textsubscript{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 encapsulator 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 0 and \( \text{tBuser, sOFe} \) for some \( \sigma \) and \( E \) such that \( \sigma = \{ \text{data} = \tau, \text{busym} = \text{false}, \text{time} = \tau, 0 \} \) and \( E = (\sigma, m) \).

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\textsuperscript{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) 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\textsuperscript{FL} specifications are completely executable (as in many process algebraic specifications). More precisely, the behavior of a specification described in SystemC\textsuperscript{FL} can be illustrated by means of transition traces according to SystemC\textsuperscript{FL} deduction rules together with the TTS associating to SystemC\textsuperscript{FL}. Similarly, formal analysis of the SystemC\textsuperscript{FL} specifications can be performed using SystemC\textsuperscript{FL} deduction rules together with the TTS associating to SystemC\textsuperscript{FL}.

However, in our view, SystemC\textsuperscript{FL} has generally some disadvantages. First, the formal language SystemC\textsuperscript{FL} is small
and has no well-defined semantics for TLM. Second, the syntax and semantics of SystemC\textsuperscript{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\textsuperscript{FL}.

Recently, as pointed out incorrectly by [42], that SystemC\textsuperscript{FL} claims its similarity with SystemC, does not have a non-preemptive scheduler 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 response to these, strictly speaking, SystemC\textsuperscript{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\textsuperscript{FL} operators (see [16], [20] for details); and clearly the watch process term in SystemC\textsuperscript{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\textsuperscript{FL} to model various systems.

Al-though, there are no deduction rules in SystemC\textsuperscript{FL}, explicitly defined for synchronous and asynchronous composition as defined in [40], the semantics of them can be captured in SystemC\textsuperscript{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\textsuperscript{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\textsuperscript{TM} SystemVerilog [13] designs. Clearly, SystemVerilog and SystemC are similar and the research work in PAFSV was highly inspired by the theoretical aspects of SystemC\textsuperscript{FL}. Hence, a formal comparison between them is indispensable as a future work.

VII. CONCLUDING REMARKS AND FUTURE WORK

Although the introduction of SystemC\textsuperscript{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\textsuperscript{FL} could not be practically used to formally repre-sent the current version of SystemC (2.2).

Indeed, the current semantics of SystemC\textsuperscript{FL} is rather old. In this paper, we have motivated and presented the newly developed language of SystemC\textsuperscript{FL}, the successor of SystemC\textsuperscript{FL}. The SystemC\textsuperscript{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\textsuperscript{FL} are much simpler, intuitive and elegant (than in SystemC\textsuperscript{FL}). We have illustrated the practical use of SystemC\textsuperscript{FL} through some examples of SystemC designs including a SystemC TLM design.

As future work, we plan to apply SystemC\textsuperscript{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\textsuperscript{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 Schifflers, Pieter Cuipers, Michel Reniers, Kees Middelburg, Uzma Khadin 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\textsuperscript{FL}, and Flaviano Garberoglio and Angelo Trischitta for their comments and support for the current research work on SystemC\textsuperscript{FL}.

REFERENCES

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