Extensions of SystemC^FL for mixed-signal systems and formal verification

Extensions of $SystemC^{\mathbb{FL}}$ for Mixed-Signal Systems and Formal Verification

K.L. Man
Formal Methods Group, Department of Mathematics and Computer Science
Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands
Phone: +31 (0)40 2472993, Fax: +31 (0)40 2475361
E-mail: kman@win.tue.nl

Abstract—**The formal language $SystemC^{\mathbb{FL}}$ is the formalization of SystemC. The language semantics of $SystemC^{\mathbb{FL}}$ was formally defined in a standard structured operational semantics (SOS) style. In this paper, we first provide an overview of the current status of the formal language $SystemC^{\mathbb{FL}}$ and show some practical applications of $SystemC^{\mathbb{FL}}$. Then, we give an outline for the latest developments of $SystemC^{\mathbb{FL}}$. These developments include extensions of $SystemC^{\mathbb{FL}}$ for modeling mixed-signal systems and formal verification.**

Keywords— **SystemC, $SystemC^{\mathbb{FL}}$, Hardware/software, Process Algebra, Structured Operational Semantics (SOS), System Level Design Modeling, Mixed-signal Systems, Formal Verification**

I. Introduction

SystemC [1] is a modeling and simulation language (without formal semantics defined) based on C++ for hardware and system level design modeling. Recently, SystemC has received an extreme increase in industrial acceptance for system specification and simulation.

The goal of developing a formal semantics is to provide a complete and unambiguous specification of the language. It also contributes significantly to the sharing, portability and integration of various applications in simulation, synthesis and formal verification.

Due to the above-mentioned motivations, we developed the formal language $SystemC^{\mathbb{FL}}$ [2] (a portable subset) of SystemC. Since processes are the basic units of execution within SystemC that are used to simulate the behavior of a device or a system, *Process Algebra* [8] was chosen as the mathematical framework for $SystemC^{\mathbb{FL}}$. Process algebra was used, because it provides an elegant notation for transition, and allows for axiomatic reasoning. The main goal of $SystemC^{\mathbb{FL}}$ is to provide the formal reasoning of SystemC designs and the formal analysis about the behavior of SystemC processes.

Based on the informal semantics presented in [1], the language semantics of $SystemC^{\mathbb{FL}}$ was formally defined in a standard structured operational semantics (SOS) style [9]. Furthermore, a strong state based bisimulation on $SystemC^{\mathbb{FL}}$ processes was defined in [2] and shown to be a congruence. Moreover, a set of useful axioms was also introduced in [2]. Recently in [6], we extended the formal language $SystemC^{\mathbb{FL}}$ to deal with concurrency and interaction. The newly developed communication semantics of $SystemC^{\mathbb{FL}}$ was also formally defined in SOS style. The proposed semantics can incorporate both point-to-point communication and multi-party communication.

A. Related Works

The simulation semantics (including watching statement, signal assignment, and wait statement) of SystemC in the form of distributed *Abstract State Machine* (ASM) specifications and the *Denotational Semantics* for a synchronous subset of SystemC were studied by [11] and [12] respectively.

It is general believed that the SOS style semantics is more intuitive [10], and the methods of ASM specifications and denotational semantics appear to be difficult to apply to describe the dynamic behavior of processes. Therefore, the language semantics of $SystemC^{\mathbb{FL}}$ was formally defined in a standard structured operational semantics (SOS) style.

It should be noted that the fundamental mechanisms used to model processes in $SystemC^{\mathbb{FL}}$ were inspired by similar constructs in the formal specification *Chi* language [7] for hybrid systems and the Algebra of Communicating Processes (*ACP*) [8].

B. Organization

The remainder of this paper is organized as follows. The next section introduces the formal language of $SystemC^{\mathbb{FL}}$. Section III and IV present some practical applications of $SystemC^{\mathbb{FL}}$. Various possible extensions for $SystemC^{\mathbb{FL}}$ are shown in Section V. Section VI contains our conclusions.

II. $SystemC^{\mathbb{FL}}$ LANGUAGE

In this section, we introduce the formal language $SystemC^{\mathbb{FL}}$. For the syntax and the formal semantics of $SystemC^{\mathbb{FL}}$, we also refer to [2] and [6].

A. $SystemC^{\mathbb{FL}}$ Data Types

In order to define the semantics of $SystemC^{\mathbb{FL}}$ processes, we need to make some assumptions about the data types. Let Var denote the set of all variables $(x_0, \ldots, x_n)$, and $Value$ denote the set all possible values $(v_0, \ldots, v_n)$ that contains at least $\mathbb{B}$ (booleans) and $\mathbb{R}$ (reals). A valuation is a partial function from variables to values (e.g. $x_0 \mapsto v_0$). 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 may be used in $SystemC^{\mathbb{FL}}$ 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 [3].

B. Syntax of the $SystemC^{\mathbb{FL}}$ Language

A process term P in $SystemC^{\mathbb{FL}}$ is built from atomic process terms AP. $SystemC^{\mathbb{FL}}$ consists of various operators that operate on process terms. The formal language $SystemC^{\mathbb{FL}}$ is defined according to the following grammar:

$$AP ::= \quad \delta \mid \text{skip} \mid \text{x} := \text{e} \mid \Delta \text{e}_\text{n} \mid \ggg$$

$$\begin{aligned} P ::= \quad & AP \mid P \blacktriangleleft b \blacktriangleright P \mid b \circlearrowleft P \mid P \triangle_d P \mid P \bullet P \mid \\ & P \Theta P \mid P \blacklozenge^d P \mid *P \mid P \parallel P \mid P \parallel_\ell P \mid \\ & P \sim P \mid \partial_H(P) \mid \tau_I(P) \end{aligned}$$

The operators are listed in descending order of their binding strength as follows : $\{\circlearrowleft, \bullet, \triangle, \blacklozenge, *\}, \{\blacktriangleleft\blacktriangleright, \Theta, \parallel, \parallel_\ell, \sim\}, \{\partial, \tau\}$. 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. Below is an introduction of the formal language $SystemC^{\mathbb{FL}}$.

A constant called *deadlock* δ is introduced, which represents no behavior. The *skip* process term performs the internal action τ. The *assignment* process term $x := e$, which assigns the value of expression e to x (modeling a SystemC "assignment" statement). The *delay* process term Δe_n, which is able to delay the value of numerical expression e_n. The *unbounded delay* process term $\ggg$ (modeling a SystemC "wait" statement) may delay for arbitrary long duration of time or perform the internal action τ.

Complex process terms are constructed using several operators. The *conditional composition* operator $p \blacktriangleleft b \blacktriangleright q$ operates as a SystemC "then_if_else" statement, where b denotes a boolean expression and $p, q \in P$. The *watching* operator $b \circlearrowleft p$ is used to model a SystemC "watching" statement. The *timeout* process term $p \triangle_d q$ (modeling a SystemC "time out" construct) behaves as p if p performs a time transition before a duration of time $d \in \mathbb{R} : d > 0$. Otherwise, it behaves as q. The *sequential composition* $p \bullet q$ models the process term that behaves as p, and upon termination of p, continues to behave as process term q. The *alternative composition* $p \Theta q$ models a nondeterministic choice between process terms p and q. The *watchdog* process term $p \blacklozenge^d q$ behaves as p during a duration 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 canceled, and the subsequent behavior is that of p after χ is executed. A *repetition* $*p$ (modeling a SystemC "loop" construct) executes p zero or more times. The *parallel composition* $\parallel$, the *left-parallel composition* $\parallel_\ell$ and the *communication* composition $\sim$ are used to express *parallelism*. 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 actions names in I are renamed to the internal action τ. Notice that we always assume that the execution of action transitions has priority over time transitions (i.e. the *maximal progress* operator is not defined).

C. Semantics of the $SystemC^{\mathbb{FL}}$ Language

Definition 1: A $SystemC^{\mathbb{FL}}$ process is a quintuple $\langle P, \Sigma, \Sigma, S, Ch \rangle$[1]. We use the convention $\langle p, \sigma', \sigma, s, m \rangle$ to write a $SystemC^{\mathbb{FL}}$ process, where p is a process term; σ, σ' are valuations; s is a set of sensitivity lists with clocks; m is a channel.

Definition 2: The set of all actions A_τ is defined as follows: $A_\tau = \{aa(x, v), s(m), r(m), com(m), \chi, \tau\}$, where $aa(x, v)$ is the assignment action (i.e. the value of v is assigned to x), $s(m)$ is the parameterized send action, $r(m)$ is the parameterized receive action, $com(m)$ is the parameterized communication action between $s(m)$ and $r(m)$, χ is the internal cancel action and τ is the internal action.

Definition 3: A formal semantics for $SystemC^{\mathbb{FL}}$ processes is given in terms of a *Labelled Transition System* (LTS). We define the following transition relations on $SystemC^{\mathbb{FL}}$ processes:

- an *action transition* $\langle p, \sigma', \sigma, s, m \rangle \xrightarrow{a} \langle p', \sigma, \sigma'', s, m \rangle$ is that the process $\langle p, \sigma', \sigma, s, m \rangle$ executes the action $a \in A_\tau$ starting with the current valuation σ (at the moment of the transition taking place) and by this execution p evolves into p', where σ' represents the previous accompanying valuation of the process, and σ'' represents the accompanying

[1] The definition of $SystemC^{\mathbb{FL}}$ process used here is an enriched dialect of the definition of $SystemC^{\mathbb{FL}}$ process presented in [2]. The component Ch is added into the tuple.

valuation of the process after the action a is executed,

- a *termination transition* $\langle p, \sigma', \sigma, s, m\rangle \xrightarrow{a} \langle \checkmark, \sigma, \sigma'', s, m\rangle$ is that the process executes the action a followed by termination, where $\checkmark$ is used to indicate a successful termination, and $\checkmark$ is not a process term,
- a *time transition* (so-called delay) $\langle p, \sigma', \sigma, s, m\rangle \overset{d}{\rightsquigarrow} \langle p', \sigma, \sigma'', s, m\rangle$ is that the process $\langle p, \sigma', \sigma, s, m\rangle$ may idle for a duration of time d and then behaves like $\langle p', \sigma, \sigma'', s, m\rangle$.

D. Deduction Rules

The above transition relations are defined through deduction rules (SOS style). These rules (of the form $\frac{premises}{conclusions}$) have two parts: on the top of the bar we put *premises* of the rule, and below it the *conclusions.* If the premises hold, then we infer that the conclusions hold as well. Giving the deduction rules for all atomic process terms and other operators of $SystemC^{\mathbb{FL}}$ is far beyond the scope of this paper, we refer those rules to [2] and [6]

III. Modeling with $SystemC^{\mathbb{FL}}$

The formal language $SystemC^{\mathbb{FL}}$ can be reasonably efficiently used to model software, hardware and concurrency [3]. In this section, we apply $SystemC^{\mathbb{FL}}$ to model two nontrivial case studies. All two case studies are taken from [1], rather than devised by us.

Synchronous D Flip Flop

D flip flops are one of the most basic building blocks of RTL designs. Below is a SystemC implementation that implements a synchronous D flip flop.

```
// dff.h
# include ''systemc.h''
SC_MODULE(dff) {
  sc_in<bool> din;
  sc_in<bool> clock;
  sc_out<int> dout;

  void doit() {
    dout = din
  };

  SC_CTOR(dff) {
     SC_METHOD(doit);
     sensitive_pos << clock;
  }
};
```

A formal $SystemC^{\mathbb{FL}}$ specification of the above synchronous D flip flop is given as follows:

$\langle Cond_{clock}(\sigma', \sigma, s) \circlearrowleft (d_{out} := d_{in}), \sigma', \sigma, s, m\rangle$, for some σ', σ, s, m.

$Cond_{clock}$ is a function that returns true if a positive edge occurs on port clock. The formal $SystemC^{\mathbb{FL}}$ specification of the above synchronous D flip flop has a clock input ($clock$), a data input (d_{in}), and a data output (d_{out}). When a positive edge occurs on the clock input (which means the function $Cond_{clock}$ returns true), the input port data (d_{in}) is assigned to the output port d_{out}. Notice that $clock, d_{in}, d_{out} \in \mathrm{dom}(\sigma'), \mathrm{dom}(\sigma)$, and only $clock \in s$.

Remote Procedure Call (RPC) Protocol

With RPC, a process in a module can call a function in another module, which is called the slave process. This is very similar to RPC semantics in Unix. The two processes must be connected through specialized ports to a specialized communication link. Below shows a $SystemC^{\mathbb{FL}}$ implementation of RPC communication.

```
SC_MODULE(producer) {
  sc_outmaster<int> out1;
  sc_in<bool> start;

  void generate_data () {
     for (int i = 0; i < 10; i++) {
        out1 = i;
     }
  }

  SC_CTOR(producer) {
     SC_METHOD(generate_data);
     sensitive << start;
  }
};

SC_MODULE(consumer) {
  sc_inslave<int> in1;
  int sum;

  void accumulate() {
     sum += in1;
     cout <<''Sum =  ''<< sum << endl;
  }

  SC_CTOR(consumer) {
      SC_SLAVE(accumulate, in1);
      sum = 0;
      }
};
```

```
SC_MODULE(top) {
  producer *A1;
  consumer *B1;

    sc_link_mp<int> link;

    SC_CTOR(top) {
       A1 = new producer(''A1'');
       A1.out1(link1);
       B1 = new consumer(''B1'');
       B1.in1(link1);
    }
};
```

A formal $SystemC^{\mathbb{FL}}$ specification of the above RPC communication is given as follows:

$\langle(\tau_{\{c_m(d_a)\}}(\partial_{\{s_m(d_a),r_m(d_a)\}}((Cond(\sigma', \sigma, \{start\})$ $\circlearrowleft$ $producer) \parallel consumer)), \sigma', \sigma, s, m\rangle$ for some $\sigma', \sigma, s, m, d_a, s_m(d_a), r_m(d_a), c_m(d_a)$, where the process $producer \equiv i < 10 \circlearrowleft (out_1 := i; i := i + 1)$ and the process $consumer \equiv sum := sum + in_1$ respectively. Notice that $\{i \mapsto 0, sum \mapsto 0\} \in \sigma$, and $start \in s$.

We model the formal $SystemC^{\mathbb{FL}}$ specification of the above RPC communication slightly different from the SystemC implementation, because we would like to show how to model communication between processes through channel (rather than multi-point link) using $SystemC^{\mathbb{FL}}$. In the above formal $SystemC^{\mathbb{FL}}$ specification, the process $producer$ (sensitive to $start$) produces a set of numbers that each number invokes the process $consumer$, which accumulates the numbers. These two processes execute concurrently (modeled by the $\parallel$ operator) and communicate over channel m. We write $s_m(d_a), r_m(d_a)$ and $c_m(d_a)$ for the action of sending datum d_a through channel m, the action of receiving datum d_a through channel m, and the action of communicating datum d_a through channel m. Intuitively, the process $producer$ sends the value of out_1 through channel m, and the process $consumer$ receives the value of in_1 through channel m. The action $c_m(d_a)$ is the action that is left when $s_m(d_a)$ and $r_m(d_a)$ are performed synchronously (i.e. the process $producer$ and the process $consumer$ communicate over channel m and $out_1 = in_1$ necessarily). The encapsulation operator (∂) and the abstraction operator (τ) are needed to enforce the process $producer$ and the process $consumer$ to communicate, and to make the communication action $c_m(d_a)$ internal.

IV. Verification of $SystemC^{\mathbb{FL}}$ Designs

In this section, we briefly describe how $SystemC^{\mathbb{FL}}$ design properties (e.g. safety property) can be verified using various formal methods.

A. Analyzing $SystemC^{\mathbb{FL}}$ Designs Using Timed Automata

A formal translation was defined in [4] from $SystemC^{\mathbb{FL}}$ to a variant (with very general settings) of timed automata [13]. The practical benefit of this translation is to enable verification of properties of $SystemC^{\mathbb{FL}}$ designs using existing verification tools for timed automata, such as Uppaal [17].

However, specifications of timed automata are not always trivial and intuitive for users not having a computer science background. In addition, variants of timed automata are used for different verification tools for timed automata. Users are required to adapt manually the settings of the variant of timed automata proposed in [4] for various verification tools.

B. Formal Verification of $SystemC^{\mathbb{FL}}$ Designs Using the SPIN Model Checker

In [5], an approach was introduced to use the SPIN model checker ([14] and [15]) as a verification engine for $SystemC^{\mathbb{FL}}$ designs, by translating $SystemC^{\mathbb{FL}}$ designs to PROMELA [16] that is the input language of SPIN.

Among various formal verification tools, the SPIN model checker was chosen, because it is one of the most successful software tools that can be used for the formal verification of distributed software systems.

Furthermore, the input language of the SPIN model checker is PROMELA that is a popular language for building verification models. It is widely used in industrial and academic fields. Moreover, PROMELA is similar to the *language C*. This makes PROMELA easy to understand by verification engineers, researchers and even students.

V. Extensions for $SystemC^{\mathbb{FL}}$

This section describes our on-going research works to develop mixed-signal system extension and to get a formal verification framework using existing verification tools.

A. Mixed-Signal Systems

A number of research works (e.g. [20] and [21]) has already been done to develop SystemC extensions for modeling and simulating mixed-signal systems. To our best understanding, there is still no formal semantics defined for those extensions. We are now defining the formal semantics (also in SOS style) in $SystemC^{\mathbb{FL}}$ for modeling

mixed-signal systems. This semantics intends to support the development of system-level analog and mixed-signal specifications, and will be a conservative extension to the existing $SystemC^{\mathbb{FL}}$.

B. Formal Verification of $SystemC^{\mathbb{FL}}$ Designs Using the SMV/NuSMV Model Checker

Nowadays, formal verification of hardware plays an very important role in electronic industry. The formal verification approach proposed in subsection IV-B was not specifically used to verify hardware designs, because SPIN is a verification tool for software systems.

The SMV[18] and NuSMV[19] are well-known model checkers. They can be reliably used for the verification of industrial designs. Various successful applications of SMV and NuSMV to verify hardware designs can be easily found in the literature. We are currently defining the formal translation from $SystemC^{\mathbb{FL}}$ to the SMV language [18] that is the input language of SMV and NuSMV model checkers. This approach enables verification of properties of $SystemC^{\mathbb{FL}}$ designs using SMV and NuSMV model checkers.

VI. Conclusions

We gave the main aspects of the current status of the formal language $SystemC^{\mathbb{FL}}$, and showed some practical applications of $SystemC^{\mathbb{FL}}$. We also presented some possible extensions of $SystemC^{\mathbb{FL}}$. These extensions can be used for modeling mixed-signal systems and formal verification of hardware designs written in $SystemC^{\mathbb{FL}}$.

Acknowledgments

The author would like to thank D.A. van Beek, P.J.L. Cuijpers, M. Mousavi, M.A. Reniers, and R.R.H. Schiffelers for many stimulating and helpful discussions. The author is grateful to J.C.M. Baeten for his support and encouragement.

References

[1] "SystemC User's Guide and SystemC Language Reference Manual (version 2.0) ".

[2] K.L. Man. "$SystemC^{\mathbb{FL}}$: Formalization of SystemC," in *IEEE Proceedings of the 12th Mediterranean Electrotechnical Conference - MELECON 2004, Dubrovnik, Croatia*, Vol. 1, pp. 201-204, May, 2004.

[3] K.L. Man. "Modeling with the Formal Language of SystemC : Case Studies," in *Proceedings of the 11th International Conference Mixed Design of Integrated Circuits and Systems (IEEE) - MIXDES 2004, Szczecin, Poland*, pp. 407-411, June, 2004.

[4] K.L. Man. "Analyzing $SystemC^{\mathbb{FL}}$ Designs Using Timed Automata," to appear in *INSPEC/IEE Proceedings of the 9th Baltic Electronics Conference - BEC 2004, Tallinn, Estonia*, October, 2004.

[5] K.L. Man. " Formal Verification of $SystemC^{\mathbb{FL}}$ Designs Using the SPIN Model Checker," submitted paper in ASP-DAC, 2005.

[6] K.L. Man. "Formal Communication Semantics of $SystemC^{\mathbb{FL}}$," submitted paper in DATE, 2005.

[7] R.R.H. Schiffelers, D.A. van Beek, K.L. Man, M.A. Reniers and J. E. Rooda. "Formal Semantics of Hybrid Chi," in *Lecture Notes in Computer Science 2791*, pp. 151 - 165. Springer-Verlag Heidelberg, 2004.

[8] J.C.M. Baeten, W.P. Weijland. "Process Algebra". Number 18 in *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[9] Gordon D. Plotkin. "A Structural Approach to Operational Semantics". Report *DAIMI FN-19*, Computer Science Department, Aarhus University, 1981.

[10] Luca Aceto, Wan Fokkink, Chris Verhoef. "Structural Operational Semantics," in Bergstra et al. *BPS01*, pp. 197-292, 1999.

[11] W. Mueller, J.Ruf, D. Hofmann, J. Gerlach, T. Kropf, W.Rosenstiehl. "The Simulation Semantics of SystemC," in *Proceedings of DATE*, 2001.

[12] Ashraf Salem. "Formal Semantics of Synchronous SystemC," in *Proceedings of DATE*, 2003.

[13] R. Alur, D.L. Dill. "A Theory of Timed Automata," in *Theoretical Computer Science*. Vol. 126, No. 2, pp. 183-236, 1994.

[14] G. J. Holzmann, "The Model Checker SPIN," in *IEEE Transactions on Software Engineering*, Vol. 23, no. 5, pp. 279-295, May, 1987.

[15] G. J. Holzmann, *The SPIN Model Checker*, Addison-Wesley, 2003.

[16] G. J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall Software Series, Prentice Hall, 1991.

[17] Kim G. Larsen, Paul Pettersson, Wang Yi. "UPPAAL in a Nutshell, " in *Journal of Software Tools for Technology Transfer (STTT)*. Vol 1, No. 1-2, pp. 134–152, 1997.

[18] The SMV model checker is available at *http://www-2.cs.cmu.edu/ modelcheck/*.

[19] The NuSMV model checker is available at *http://nusmv.irst.itc.it/*.

[20] Karsten Einwich, Christoph Clauss, Gerhard Noessing, Peter Schwarz, Herbert Zojer. "SystemC Extensions for Mixed-Signal System Design," in *Proceedings of FDL*, 2001.

[21] Karsten Einwich, Peter Schwarz, Christoph Grimm, Klaus Waldschmidt. "Mixed-Signal Extensions for SystemC," in *Proceedings of FDL*, 2002.