SystemC^FL : a formalism for hardware/software co-design

DOI:
10.1109/ECCTD.2005.1522943

Document status and date:
Published: 01/01/2005

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

$SystemC^{\mathbb{FL}}$: A Formalism for Hardware/Software Co-design

K.L. Man*

Abstract **—** $SystemC^{\mathbb{FL}}$ **is a formal language for hardware/software co-design. Principally,** $SystemC^{\mathbb{FL}}$ **is the formalization of SystemC based on classical process algebra ACP. The language is aimed to give formal specification of SystemC designs and perform formal analysis of SystemC processes. This paper, designed for the first-time user of** $SystemC^{\mathbb{FL}}$**, guides the reader through modeling, analyzing and verifying designs using** $SystemC^{\mathbb{FL}}$**. This paper illustrates the use of** $SystemC^{\mathbb{FL}}$ **with two case studies taken from literature.**

1 Introduction

SystemC [1] is a modeling and simulation language (without formal semantics defined) based on C++ for hardware/software co-design. 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.

$SystemC^{\mathbb{FL}}$ [2] is a reasonable subset of SystemC that has a rigid formal basis (i.e. formal semantics). In principle, $SystemC^{\mathbb{FL}}$ is the formalization of SystemC based on classical process algebra (the Algebra of Communicating Processes) ACP [7]. The intended use of $SystemC^{\mathbb{FL}}$ is for giving formal specification of SystemC designs and performing formal analysis of SystemC processes. It was shown in [3] that $SystemC^{\mathbb{FL}}$ can be reasonably efficiently used to model software, hardware and concurrency.

A key feature of $SystemC^{\mathbb{FL}}$ is to have a single-formalism that is used to describe the various aspects of the system under consideration. Analysis/formal verification takes place by extracting simpler designs from $SystemC^{\mathbb{FL}}$ designs that are tailored to some specific properties of interest. More precisely, various desired properties of systems/designs modeled in $SystemC^{\mathbb{FL}}$ can be verified with existing formal verification tools by translating them formally into different formalisms that are the input languages of the existing formal verification tools. Hence, $SystemC^{\mathbb{FL}}$ can be purportedly used for formal verification. For instance, safety properties of concurrent systems modeled in $SystemC^{\mathbb{FL}}$ can be verified by translating those systems to PROMELA [12] that is the input language of the SPIN Model Checker [12]. Similarly, [5] reported that some desired properties of finite state systems described in $SystemC^{\mathbb{FL}}$ can be fed into the SMV Model Checker [14] to verify them.

Also, a formal translation was defined in [4] from $SystemC^{\mathbb{FL}}$ to a variant (with very general settings) of timed automata. The practical benefit of the formal translation from a $SystemC^{\mathbb{FL}}$ design (describing real-time systems) to a timed automaton is to enable verification of properties of the $SystemC^{\mathbb{FL}}$ design using existing verification tools for timed automata, such as Uppaal [13].

1.1 Related Work

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 [10] and [11] respectively.

It is general believed that the structured operational semantics (SOS) [8] is more intuitive [9], 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 SOS style.

1.2 Organization

The remainder of this paper is organized as follows. The next section introduces the formal language $SystemC^{\mathbb{FL}}$. Section 3 and 4 present some practical applications of $SystemC^{\mathbb{FL}}$ for modeling and formal verification. Section 5 contains our conclusions.

2 $SystemC^{\mathbb{FL}}$ 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].

2.1 $SystemC^{\mathbb{FL}}$ 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 $(v_0, \ldots, v_m)$ 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

*Formal Methods Group, Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands, e-mail: kman@win.tue.nl, tel.: +31 (0)40 2474139, fax: +31 (0)40 2475361.

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].

2.2 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, and it is defined according to the following grammar:

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

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

The operators are listed in descending order of their binding strength as follows : $\{\circlearrowleft, \bullet, \bigtriangleup, \blacklozenge, *\}, \{\blacktriangleleft\blacktriangleright, \Theta, \parallel, \parallel_{\ell}, \sim\}, \{\partial, \tau, \pi, \eth\}$. 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 a brief introduction of the formal language $SystemC^{\mathbb{FL}}$. The formal semantics of $SystemC^{\mathbb{FL}}$ is given in subsection 2.3. Due to limitation of pages, deduction rules[1] for operational semantics of $SystemC^{\mathbb{FL}}$ are not given in this paper. For those interested in more details, please read [2] and [6].

A constant called *deadlock* δ is introduced, which represents no behavior. The *skip* process term performs the internal action τ which is not externally visible. 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} 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 a long time that is unbounded or perform the internal action τ.

The *conditional composition* $p \blacktriangleleft b \blacktriangleright q$ operates as a SystemC "then_if_else" statement, where b denotes a boolean expression and $p, q \in P$. The *watching* process term $b \circlearrowleft p$ is used to model a SystemC "watching" statement. 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 non-deterministic choice between process terms p and q. The *timeout* process term $p \bigtriangleup_d q$ (modeling a SystemC "time out" construct) behaves as p if p performs a time transition before a time $d \in \mathbb{R}_{>0}$; otherwise, it behaves as q. The *watchdog* process term $p\blacklozenge^d q$ behaves as p during a period of time less than d, at time d, q takes over the execution from p in $p\blacklozenge^d q$; if p performs an internal *cancel* χ action, then the delay is canceled, and the subsequent behavior is that of p after χ is executed. The *repetition* process term $*p$ (modeling a SystemC "loop" construct) executes p zero or more times.

The *parallel composition* $p \parallel q$, the *left-parallel composition* $p \parallel_{\ell} q$ and the *communication* composition $p \sim q$ are used to express *parallelism* (actions are executed in an interleaving manner, with the possibility of communication of actions). 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. 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 and executing them in one step can be done by using $\eth(p)$.

Informal semantics of SystemC states that SystemC incorporates both point-to-point communication and multi-party communication mechanisms for the interaction amongst processes. However, there are no (implicit) statements in SystemC for modeling these communication mechanisms. In order to capture these facts in $SystemC^{\mathbb{FL}}$, operators $\parallel, \parallel_{\ell}, \sim, \partial_H, \tau_I$, and π are introduced to give formal specification for point-to-point communication and multi-party communication mechanisms.

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

Definition 1 *A $SystemC^{\mathbb{FL}}$ process is a quintuple $\langle P, \Sigma, \Sigma, S, Ch\rangle$. 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 sensitivity list with clocks; and m is a channel.*

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

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:*

[1]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*.

- *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 valuation of the process after the action a is executed,*

- *a termination transition $\langle p, \sigma', \sigma, s, m\rangle \xrightarrow{a} \langle \surd, \sigma, \sigma'', s, m\rangle$ is that the process executes the action a followed by termination, where $\surd$ is used to indicate a successful termination, and $\surd$ 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 $\langle p', \sigma, \sigma'', s, m\rangle$.*

Two valuations (e.g., previous accompanying valuation and current valuation) are defined (as arguments) in the quintuple, so that the change of valuation of variables in the sensitivity list of the quintuple can be observed. This is needed for defining the deduction rules of some $SystemC^{\mathbb{FL}}$ operators (e.g. the watching $\circlearrowleft$).

3 Modeling with $SystemC^{\mathbb{FL}}$

In this section, we apply $SystemC^{\mathbb{FL}}$ to give the formal specification of a case study taken from literature.

3.1 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<int> 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 $clock \in s$.

4 Purportedly Used for Formal Verification

In the section, we present the application of SPIN to verify the mutual exclusion algorithm of Dekker modeled in $SystemC^{\mathbb{FL}}$, by translating the$SystemC^{\mathbb{FL}}$ design to PROMELA.

4.1 Dekker's Mutual Exclusion

The mutual exclusion algorithm of Dekker is used by two processes (A and B) which communicate through shared variables. It is intended to prevent the processes being simultaneously entered in their critical section. Since Dekker's Mutual Exclusion is a well-known case study of the concurrency theory, we do not further give the description and explanation of this algorithm. We only give the formal specification of it in $SystemC^{\mathbb{FL}}$. In order to increase the readability, we introduce syntactic sugars for various process terms. The syntactic sugars for the process terms A and B of the mutual exclusion algorithm of Dekker are as follows:

$$A \equiv (x := 1 \bullet t = B_t) \bullet (mu := mu + 1 \bullet mu := mu - 1 \bullet x := 0) \blacktriangleleft (y = 0) \wedge (tu = A_t) \blacktriangleright \delta$$
$$B \equiv (y := 1 \bullet t = A_t) \bullet (mu := mu + 1 \bullet mu := mu - 1 \bullet y := 0) \blacktriangleleft (y = 0) \wedge (tu = B_t) \blacktriangleright \delta$$

Since the process terms A and B execute concurrently, the parallel composition is used to model the complete system. The complete system with initial value for variables is modeled as follows:

$\langle A \parallel B, \sigma', \sigma, \emptyset, \emptyset\rangle$ for some σ', where $\sigma = \{x \mapsto \perp, y \mapsto \perp, t \mapsto \perp, A_t \mapsto 0, B_t \mapsto 1\}$, $\emptyset$ and $\perp$ denote empty element and the undefinedness respectively.

Note that the variable mu is introduced as a flag. In this case study, if $mu = 2$, which indicates that process terms A and B enter simultaneously in their critical section.

Presenting the formal translation scheme from $SystemC^{\mathbb{FL}}$ to PROMELA, and the syntax and semantics of PROMELA are far beyond the scope of this paper, therefore we do not show them. Nevertheless, the translation of the above-given formal specification of Dekker's Mutual Exclusion algorithm from $SystemC^{\mathbb{FL}}$ to PROMELA model is straightforward and it is shown as follows:

```
/*declaration*/
bit x, y;
byte t, mu;
proctype A() {
  x = 1;
  t = Bt;
 ((y == 0)||(t == At))->
/* critical section */
  mu ++;
  mu --;
  x = 0;
}
proctype B() {
  y = 1
  t = At;
 ((x == 0)||(t == Bt))->
/* critical section */
  mu ++;
  mu --;
  y = 0;
}

proctype monitor() {
  assert (mu ! = 2);
}
init {
  run A(); run B(); run monitor()
}
```

Notice that the process *monitor* is introduced and needed in the PROMELA model to check whether mutual exclusion is valid. If the condition of the assert statement (i.e., $mu! = 2$) does not hold, SPIN will produce an error report. The process *init* is used to start processes.

5 Conclusions

In this paper, the main aspects of the current status of the formal language $SystemC^{\mathbb{FL}}$ are presented. Then, the use of the $SystemC^{\mathbb{FL}}$ through some case studies taken from literature is illustrated. Also, some practical applications of $SystemC^{\mathbb{FL}}$ are shown that can be purportedly used for formal verification.

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 - IEEE/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," in *INSPEC IEE Proceedings of the 9th Baltic Electronics Conference - IEEE/BEC 2004, Tallinn, Estonia*, pp. 155-158, October, 2004.

[5] K.L. Man, " Verifying $SystemC^{\mathbb{FL}}$ Designs Using the SMV Model Checker," in *Proceedings of the 8th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems - IEEE/DDECS 2005, Sopron, Hungary*, pp. 244-247, April, 2005.

[6] K.L. Man, "Formal Communication Semantics of $SystemC^{\mathbb{FL}}$," to appear in *IEEE Proceedings of the 8th Euromicro Conference on Digital System Design - IEEE/DSD 2005, Porto, Portugal*, September, 2005.

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

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

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

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

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

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

[13] 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.

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