On the Reuse of VHDL Modules into SystemC Designs

N. Agliada # A. Fin # F. Fummi # M. Martignano ‡ G. Pravadelli #
DST Informatica, Università di Verona, ITALY
{agliada, fin, fummi, pravadelli}@sci.univr.it
‡ Sitek S.p.A., Verona, ITALY
mm@sitek.com

Abstract

More and more the SystemC description language is used to model and simulate new hardware designs. The need of integrating new designs with already existing modules may lead to the generation of heterogeneous models, specifications, based both on SystemC and on VHDL (or Verilog). In this scenario cosimulation is necessary to verify the heterogeneous system description. This paper presents a method to remove the above heterogeneity based on automatic translation of VHDL descriptions into SystemC. Of course, the presented approach is viable only if the automatic translation respects the original behaviour, specified in VHDL. This way, the adoption of a homogenous system model allows the efficient execution of simulations sessions aiming to assess the feasibility and convenience of reusing already existing modules.

1 Introduction

The rapid development of new and powerful systems on silicon can only be based on the reuse of previously designed modules belonging to a proprietary library or obtained from third parties (IP-cores). In a core-based design framework, the configuration management task absorbs a considerable part of the entire design effort [1, 2]. Similar problems have to be solved at this moment when the SystemC [3] description language is being added to an existing design framework.

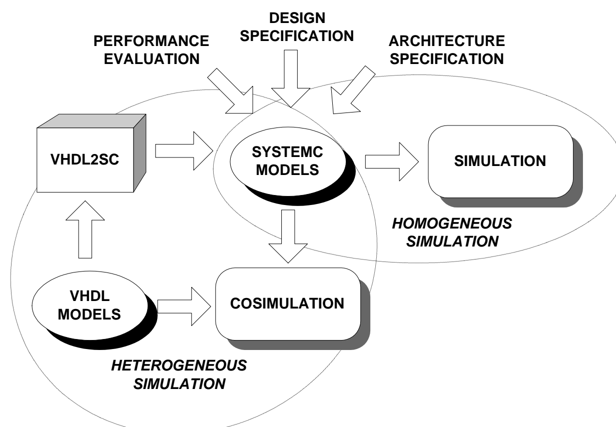

Figure 1: Design strategies to mix VHDL and SystemC.

The majority of projects are not completely new, but they are based on predesigned components that are usually available ad the RT-level as VHDL (or Verilog) descriptions. The potential integration of such components must be carefully investigated by extensively simulating their interactions with the rest of the design. These simulation sessions can usually give enough information on the expected behaviors and performance. In this context, the designer can tackle a new project by following two different strategies as described in Figure 1.

- **Heterogeneous** modeling. A VHDL component selected from the library can be integrated with the SystemC description by using only a cosimulator [4, 5, 6]. In this case, cosimulation implies an interface between the event-driven VHDL simulation and the cycle-based SystemC simulation.
- **Homogeneous** modeling. In this case the VHDL original descriptions are first automatically translated into SystemC. The invariance of the original behavior can be guaranteed whenever event-driven specifications can be described in cycle-based terms. This is possible when all signals are updated synchronously with one or more clock signals. In this context, the behavior of asynchronous signals becomes synchronous.

This paper investigates the potentialities of the second approach. The manual remodeling of VHDL components into SystemC modules is not feasible due to the long time required to perform the translation, inherently prone to errors. So, the paper presents a set of rules aiming at automatically translate VHDL descriptions into SystemC modules with equivalent behavior under the assumption of cycle-based simulation. This way it is possible to obtain homogeneous representations and perform efficient architectural analyses and investigations.

While a first set of rules for manual translation have been reported in [7], this paper presents an automatic system (a translator) implementing the complete translation from VHDL to SystemC.

This translation has been implemented on top of the Savant environment [8] as shown in Figure 2. The SystemC code generator is based on the intermediate format IIR that is built from VHDL code. All identified translation rules (Section 2) have been implemented as C++ methods manipulating the IIR description. Simulation measurements (Section 3) show which VHDL descriptions have more advantages in the SystemC conversion from the point of view of simulation performance.

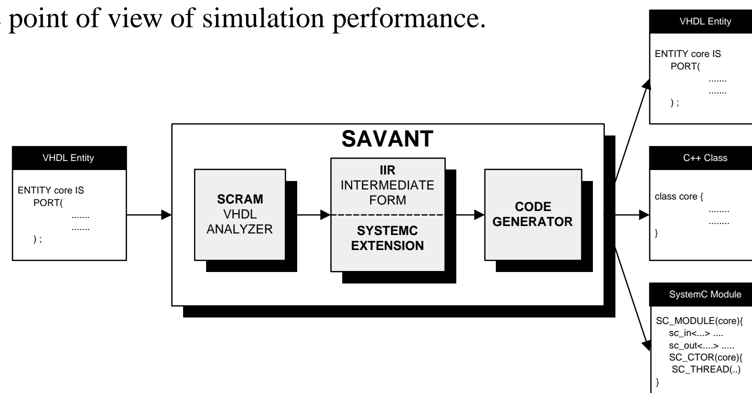

Figure 2: Program architecture based on the Savant environment.

2 VHDL to SystemC Conversion

Translation rules for all the considered VHDL constructs have been inserted in a database of rules and implemented in the `vhdl2sc` tool. Only partial examples are reported in the following for lack of space. For instance, Table 1 shows the rule to translate a fixed-dimension integer, considering: $sup \geq inf \geq 0$ and $\log_2(sup - inf + 1) \in \mathbf{N}$.

In order to respect the semantic of the original VHDL descriptions, some of the translation rules require a more detailed translation process than a pure syntax conversion. Figure 3 presents some interesting features related to the translation of VHDL models. The considered model describes a synchronous sequential machine. Due to the `wait` statement the process has been translated by `vhdl2sc` into a SystemC `SC_THREAD`. The SystemC construct `wait_until()`, required to translate the VHDL `wait until`, can be inserted only in a `SC_CTHREAD`. Therefore we use the `do-while` cycle to achieve the same behaviour. The VHDL variables have to be translated in C++ static variables in order to preserve their value for the next execution of the SystemC process.

Syntax	
VHDL	`integer range` inf `to` sup
SystemC	`sc_uint`$< \log_2(\sup - \inf + 1) >$
Example	
VHDL	`variable x : integer range 0 to 15;`
SystemC	`sc_uint<` 4 `> x;`

Table 1: Translation of integer range.

```
ENTITY mouse IS
 PORT(clk,xc,xd,rst:IN BIT;
      xp:OUT INTEGER );
END mouse;
ARCHITECTURE bhv OF mouse IS
 SIGNAL pbs_state,x:INTEGER;
BEGIN
 pbs_machine:PROCESS
  VARIABLE pbs_tok :INTEGER;
 BEGIN
  WAIT UNTIL clk'EVENT AND
             clk='1';
  pbs_tok:=0;
  IF (reset='0') THEN
      ...
  END IF;
 END PROCESS;
END bhv;
```

```
SC_MODULE(mouse) {
 sc_in<sc_bit> clk,xc,xd,rst;
 sc_out<int> xp;
 sc_signal<int> pbs_state,x;
 void pbs_machine();
 SC_CTOR(mouse){
  SC_THREAD(pbs_machine);
  sensitive << clk;
} };

void mouse::pbs_machine(){
 static int pbs_tok;
 do{ wait(); }
 while(!(clk.event()&&clk.read()==1));
 pbs_tok=0;
 if (reset==0){
     ...
} }
```

Figure 3: Template of a synchronous sequential machine.

Figure 4 shows the translation of an array variable. If there is no match between VHDL and SystemC data type, then the syntax can be different.

The VHDL `case` statement can be translated into the C++ `switch`, but the `break` instruction has to be inserted at the end of each branch. The `others` branch has to be declared as the C++ `default` branch.

A *ad-hoc* C++ function has to be defined to translate any not directly mappable VHDL arithmetic operator, e.g., VHDL `mod` operator. There are some SystemC restrictions about vectors `sc_bv` and `sc_lv`, for instance, no arithmetic operators are defined for them. This translation problem can be solved using the appropriate SystemC integer through cast operator (as shown in Figure 5) or support variables.

```
variable str :  string(1 to 20);            char str[20];
variable bv :  bit_vector(1 to 8);          sc_bv<8> bv;
```

Figure 4: Different syntax for array type definition.

```
VARIABLE xlv:STD_LOGIC_VECTOR(7 DOWNTO 0);          sc_lv<8> xlv;
VARIABLE ylv:STD_LOGIC_VECTOR(7 DOWNTO 0);          sc_lv<8> ylv;
VARIABLE zlv:STD_LOGIC_VECTOR(7 DOWNTO 0);          sc_lv<8> zlv;
xlv:="11001100";                                    xlv="11001100";
ylv:="11001100";                                    ylv="11001100";
zlv:=xlv + ylv);                                    zlv=(sc_int<8>)xlv + (sc_int<8>)ylv;
```

Figure 5: Use of cast operator for `sc_lv`.

Both VHDL functions and procedures have to be translated into C++ methods. In the case of procedures, method parameters have to be passed by reference in order to model side-effects of VHDL procedures into

the calling process/procedure.

A `namespace` has to be defined to translate each VHDL package. It can include: functions, procedures, variables and constants. This approach allows the declaration of different objects with the same name in separated `namespaces`, thus behaving as a package.

Currently, the translator is able to convert to SystemC the majority of VHDL constructs: entity and architecture definitions, signal and variable declarations, arithmetic and logic operators, assignments, functions and procedures, processes, control statements, types and subtypes, packages, event attribute, generic constants, ... The translation of the remaining VHDL constructs (resolved type, records, slice names, generates, configurations and attributes) will be included inthe next version of the translator.

3 Simulation Results

First of all, we compared performance of the simulation approach versus the cosimulation strategy. We selected an example composed of a CPU core (described in VHDL) and surrounding blocks (designed in SystemC and VHDL). The manual translation of the CPU in SystemC is a too complex task, it requires an estimated effort of 30 hours, while its automatic translation requires few milliseconds. The cosimulation of the heterogeneous model or the simulation of the automatically translated CPU description in SystemC represent viable solutions to produce a simulatable model of the entire system.

	Real Time	User Time	System Time
VHDL only	1.5	1.2	2.1
VHDL+SystemC	10.3	7.5	5.3
SystemC only	1.0	1.0	1.0

Table 2: Simulation versus cosimulation normalized times.

VHDL simulation has been performed by using a very efficient commercial VHDL simulator. The cosimulation approach has been implemented by using a socket-based interface between the executable SystemC description and the VHDL simulator. Simulation results are reported in Table 2 as normalized time with respect to the simulation of a homogeneous SystemC description of the entire system. The times indicated in Table 2 have been obtained with the Unix command `time`.

Simulation time required by the SystemC only description is sensibly lower (50%) than VHDL simulation. The cosimulation technique is the more complex task and it is sensibly slower than the SystemC-based simulation, since it requires to interact an event-driven simulator with a cycle-based one. In conclusion, the homogeneous SystemC-based simulation shows better performance with respect to the cosimulation approach, but also with respect to VHDL simulation. This last comparison has been further investigated by examining different sets of VHDL constructs.

The first analysis concerns the presence of some function calls and parallel processes. We selected three single-process VHDL descriptions (`ex1`, `ex2` and `ex3`) with a high number of function calls (respectively 97, 623 and 1283). SystemC descriptions have been automatically generated and simulation times have been measured for both VHDL and SystemC descriptions.

	ex1	ex2	ex3
VHDL	20%	57%	109%
SystemC	77%	177%	310%

Table 3: Percentage of delay introduced in the simulation by the function calls.

	ex1	ex2	ex3
SystemC vs. VHDL	29.0	12.0	4.0

Table 4: Ration between SystemC and VHDL simulation times with function calls.

Table 3 shows the delay introduced by the function calls in the simulation of VHDL and SystemC descriptions stimulated with 100,000 input vectors. Results show that function calls overhead in simulation time is higher in SystemC than in VHDL because of the minor complexity of SystemC processes regarding to VHDL ones. However, Table 4 shows that the simulation times of SystemC are clearly better than VHDL ones. Moreover, further experiments have shown that when the number of processes rises, the advantage of SystemC over VHDL increases as well. Note that, changing the C++ compilation optimizations and the way VHDL functions are translated in SystemC (e.g., static functions, in-class methods and inline methods) impacts on SystemC simulation efficiency up to 44%.

# Components	100	500	1000	5000	10000
VHDL	10.0	4.5	2.8	2.5	2.1
SystemC	1.0	1.0	1.0	1.0	1.0

Table 5: VHDL and SystemC simulation normalized times with respect to components instantiation.

The last set of experiments measures the impact of components instantiation in SystemC and VHDL. A structural VHDL description with a parametric number of components has been selected and translated in SystemC. Simulation results are shown in Table 5. It is evident that, the higher is the number of components, the lower is the advantage of SystemC over VHDL. Moreover this observation can be motivated by the minor complexity of SystemC processes regarding to VHDL ones.

In conclusion, the automatic translation of VHDL code into SystemC allows to simulate a homogeneous SystemC model of an entire system even if some VHDL modules are reused in the design. SystemC simulation outperforms cosimulation and it is also more efficient than VHDL simulation particularly when the number of component instantiations and function calls is limited.

References

[1] J. Notbauer, T. Albrecht, G. Niedrist and S. Rohringer. Verification and management of a multimillion-gate embedded core design. *Proc. ACM/IEEE DAC*, pages 425–428, 1999.

[2] S. Olcoz, A. Castellvi, M. Garcia and J.A. Gomez. Static Analysis Tools for Soft-Cores Reviews and Audits. *Proc. IEEE DATE*, pages 935–936, 1998.

[3] SystemC User's Guide. *Synopsys, CoWare, Frontier Design, version 1.1beta*, 2000.

[4] C.A. Valderrama, F. Nacabal, P. Paulin, A.A. Jerraya, M. Attia and O. Cayrol. Automatic Generation of Interfaces for Distributed C-VHDL Cosimulation of Embedded Systems: an Industrial Experience. *Proc. IEEE International Workshop on Rapid System Prototyping*, pages 72–77, 1996.

[5] J.P. Soininen, T. Huttunen, K. Tiensyrja and H. Heusala. Cosimulation of Real-Time Control Systems. *Proc. IEEE Euro-VHDL, Euro-DAC*, pages 170–175, 1995.

[6] W. Fornaciari, D. Sciuto and F. Salice. A two-level Cosimulation Environment. *IEEE Computer*, vol.30(6), pages 109–111, 1997.

[7] G. Bollano, P. Garino, M. Turolla, M. Valentini. SystemC's Impact on the Development of IP Libraries. *IP00 Europe Conference*, 2000.

[8] Savant Programmer's Manual. *Dept. of ECECS, University of Cincinnati*, 1999.