SystemC: A Homogenous Environment to Test Embedded Systems

Alessandro Fin [#] Franco Fummi [#] Maurizio Martignano [‡] Mirko Signoretto [#]

[#] Università di Verona
DST Informatica
Verona, ITALY

[‡] Sitek S.p.A.
S.Giovanni Lupatoto
Verona, ITALY

Abstract

The SystemC language is becoming a new standard in the EDA field and many designers are starting to use it to model complex systems. SystemC has been mainly adopted to define abstract models of hardware/software components, since they can be easily integrated for rapid prototyping. However, it can also be used to describe modules at a higher level of detail, e.g., RT-level hardware descriptions and assembly software modules. Thus, it would be possible to imagine a SystemC-based design flow, where the system description is translated from one abstraction level to the following one by always using SystemC representations. The adoption of a SystemC-based design flow would be particularly efficient for testing purpose as shown in this paper. In fact, it allows the definition of a homogeneous testing procedure, applicable to all design phases, based on the same error model and on the same test generation strategy. Moreover, test patterns are indifferently applied to hardware and software components, thus making the proposed testing methodology particularly suitable for embedded systems. Test patterns are generated on the SystemC description modeling the system at one abstraction level, then, they are used to validate the translation of the system to a lower abstraction level. New test patterns are then generated for the lower abstraction level to improve the quality of the test set and this process is iterated for each translation (synthesis) step.

Keywords: functional testing, C++ models, embedded systems verification.

1. INTRODUCTION

SystemC is a C++ class library and a design methodology [1] that is able to create efficient and accurate models of software algorithms, hardware architectures, system-level designs and interfaces; in other terms, all components of an embedded system. The potential success of this design methodology is due to the easy simulation, validation and alternatives exploration that can be performed on a homogeneous C++ description. Moreover, the design team can be provided with an executable specification of the entire system, which helps in the design and development of related components. This executable specification is a C++ program that, when executed, exhibits the same behavior of the described system. SystemC allows the increase of abstraction level of a project and the number of different architectural alternatives analyzed before the actual synthesis of the hardware part.

Testing SystemC descriptions is still an open issue, since the language is new and researches are looking for efficient error models and coverage metrics, which can be indifferently applied to hardware and software modules. The adoption of typical and well analyzed software coverage metrics [2] does not seem to be acceptable for verifying hardware design errors as pointed out in [3]. The opposite solution could be the adoption of error models and test generation techniques developed in the past years for hardware RT-level descriptions (e.g., [4, 5, 9]. Many error models and coverage metrics has been defined to take advantage from the analyzed abstraction-level [5, 6]. Some probalistic Test Pattern Generators, based on genetic algorithms, have been developed with the main goal of improving the performances of the deterministic TPGs [7, 8, 6]. They achieve good fault coverage when applied to RT-level hardware representations with the target of testing gate-level faults.

However, the proposed testing methodology tries to cover the whole design flow of an embedded system from the system-level description to its structural representation, by using the same error model and test generation technique. The aim of this testing technique is a simulation-based verification of the design transformations necessary to map a system description into hardware and software modules. The accurate description level and simulation performance reached by a SystemC description allows the definition of a high efficient testing procedure. Its description is the aim of this paper.

The rest of paper is organized as follows. Section 2 describes the use of SystemC for modeling different im-

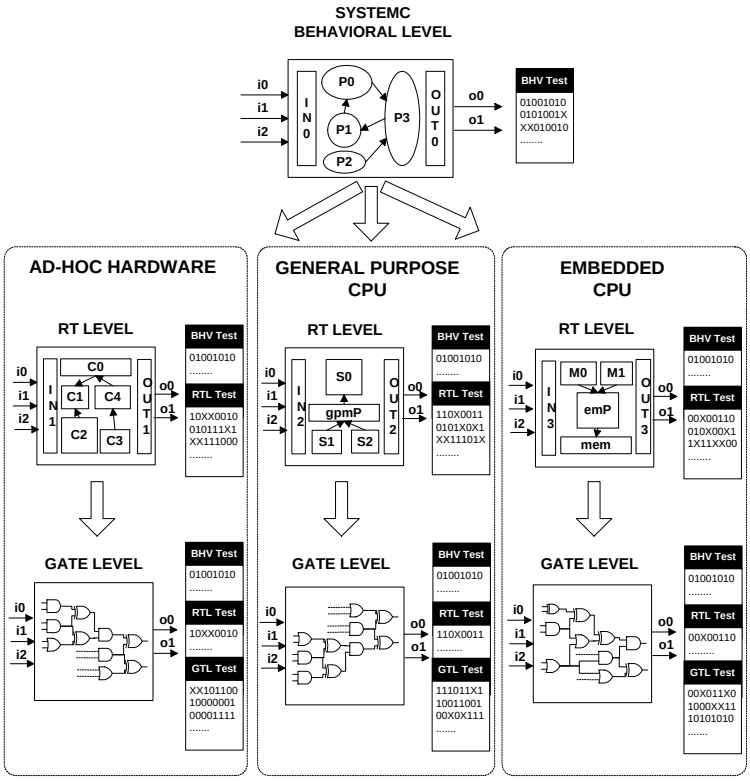

Figure 1: Design alternatives for the same system description.

plementation choices and the possibility of generating functional test patterns on this homogeneous description. The fault model and the fault injection technique are described in Section 3. The implementation choice, based on an embedded CPU, is analyzed in Section 4, where an entire design cycle is presented and incremental functional test patterns are generated. The last section is devoted to concluding remarks and future works.

2. DESIGN ALTERNATIVES

Let us now examine some different design alternatives for an embedded system modeled by using SystemC. Figure 1 shows three different alternatives originated from the same SystemC model.

We assume to represent the system as a set of interacting processes, which will be implemented by using ad-hoc hardware components, a software program running on a general purpose embedded CPU, a software program running on an ad-hoc developed CPU or a mix of these three implementation strategies.

The same initial description is used to generate test patterns accordingly to the *bit-coverage* error model [11] and the functional test generation approach [6] applied in this case to a SystemC representation. SystemC plays a different role for test pattern generation with respect to these three different scenarios.

- **Ad-hoc Hardware.** No software components are presented in this case, since the initial SystemC description is completely mapped onto hardware components. The design team uses SystemC to define an initial model of the whole system, but the following design phases are based on VHDL (or Verilog). The translation of a SystemC description into VHDL must be validated. This task is complex since it implies a translation between a cycle-based language (SystemC) and a event-driven language (VHDL, Verilog). To reduce the complexity of this task, the behavioral description should be transformed into a SystemC description at the RT level. After this step, the translation of RT-level SystemC code into RTL VHDL code is simpler, since it consists of a syntactic translation rather than a semantic translation [1]. Test patterns inherited from the system level can be used to check the new hardware description. The change of the design environment causes the adoption of a different testing technique and error model for the following design phases, increasing the global effort for the project. On the contrary by using a uniform SystemC description, a uniform test methodology can be applied.

- **General Purpose CPU.** The design goes on by selecting a general purpose CPU description, bought by an external vendor, and by adapting

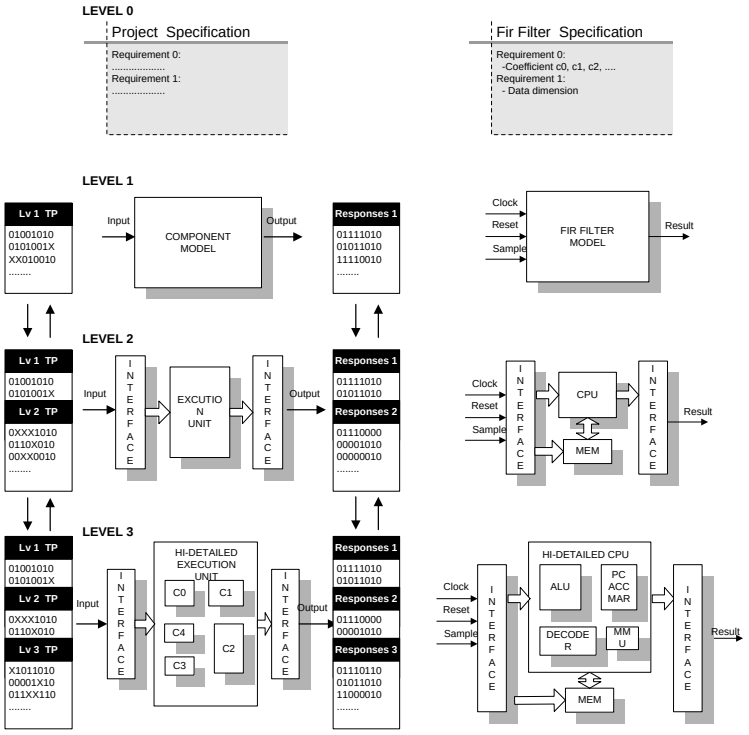

Figure 2: Application of the testing methodology to an example.

the SystemC system description to the lower design level. The model in SystemC can be used as executable specification of the software to be implemented on the target CPU. The SystemC interacting processes become software modules running on the CPU. The shorter is the abstraction gap between the SystemC models and the target CPU run time support, the easier is the translation process. The system architecture must be completed by a memory model, to store software, and some other hardware components working as interfaces. Both software and extra modules can be modeled using SystemC. The adoption of a homogeneous design environment, as SystemC, makes the testing procedure easier, since the same test techniques and methodologies previously used can be applied. The externally bought descriptions are considered tested, therefore the inherited test set is helpful to check the software routines and all added components. Moreover, designers could buy the SystemC description of each added hardware component and translate the remaining processes of the behavioral model as software routines. Adopting this approach the inherited test set would be used to test software and components interaction.

- **Ad-hoc Embedded CPU.** In this case, project constraints imply the design of an ad-hoc embedded CPU. Typical examples are Systems on Chip (SoC's). The design team has to define every module: CPU, memory and interfaces. The flexibility and description power of SystemC must be exploited to quickly find the best configuration (hardware and software). Designers can easily partition processes between hardware and software to observe the performance of these different configurations. Again, it is possible to maintain the same test technique and error model applied at the system level by using SystemC to describe the system architecture at the lower description level. Test sets derived from the first test phase are essential to check this critical step. The number of components to test is larger than those of the previous design alternative. Moreover, the complexity of components under test is high (i.e. embedded CPU). Thus, the development of test patterns on a more simple description, the system description, can simplify this task.

For each design alternative, test sets obtained from previous design phases are used to validate each new design step. Moreover, test sets are improved, at each design step, by specific-level test patterns.

3. TESTING TECHNIQUE

Every time a design description, at an identified abstraction level, is converted to the description at a lower level, the necessity of a validation phase is necessary. This task is usually performed by using a simulation-based approach whenever formal verification techniques cannot be applied. This test is performed by applying the

whole set of test patterns, identified at the higher abstraction level, on the new description at the lower level. Moreover, this set of test patterns is extended at each development phase with some new level-specific test vectors that are simulated on the previous-level representation to identify correct responses. By using a common description language covering all phases of a design hierarchy, this testing strategy could be repeated from the system level to the structural level. SystemC could be this common description language. Moreover, in the case tird-party blocks are used, an authomatic translation of VHDL or verilog descriptions into SystemC-based representations can be performed as described, for instance, in [10].

The proposed testing strategy is based on the *bit-coverage* error model [11], which shows a high correlation between errors modeled on descriptions of different abstraction levels. This error model has been used to correlate test patterns between the behavioral, RT and gate-level for hardware components only. In this paper, a similar approach is applied to embedded systems, thus testing both hardware and software parts. Each bit of each variable, condition and input/output port is set stuck-at zero or one to obtain erroneous SystemC descriptions that are compared to the error-free description in order to identify functional test patterns.

Two different programs have been adopted to test the embedded CPU. The FIR filter algorithm, available from the SystemC package, and a test oriented algorithm. The two algorithms have different targets. The first is application-oriented and it can be useful to test the realization of a specific algorithm on the developed embedded CPU. It can not reach a high global error coverage on the average, because it use a subset of the instruction set and the errors in the memory can not be detected if they are outside the data and code segments. However it is shorter and it requires less knowledge about the CPU. Moreover, it can be enough to design a embedded CPU for a specific target application. Otherwise if the target is to design an embedded CPU reusable for more programs and it is possible to spend more time on testing then the test-oriented algorithm has to be applied. It use all the instrucions available and it is allocated in differen positions in the memory of the embedded architecture to reach a higher global error coverage.

The test patterns are randomly generated. For both testing algorithms, the test pattern are the data manipulated by the algorithms. To reach a higher error overage, the random test pattern generator can be repleaced by a deterministic one.

4. AD-HOC EMBEDDED CPU

The proposed testing methodology is applied in this section to the design alternative based on an ad-hoc embedded CPU. We consider the problem of designing and verifying the FIR filter included in the SystemC documentation [1].

	#SystemC lines	#Errors
level 1	1529	606
level 2	2637	750
level 3	3043	890

Table 1: Description features.

Figure 2 shows the application of the proposed testing methodology to the FIR example. General transformations of SystemC models are reported on the leftmost side of the figure. The system specification (level 0) is implemented by using an embedded CPU. On the rightmost side, the exemplification on the FIR filter is shown. The specification is converted at first into a description (level 1) composed of interacting processes representing:

- the system interfaces;
- the system behavior.

The adopted error model is used in this level to generate a first set of functional test patterns to verify the coherence with the specification.

Then, an abstract model of the CPU is written in SystemC (level 2) in order to verify the correct execution of the assembly code derived from the software system behavior of level 1. This check is performed by applying the previous set of test patterns to the description of level 2, where a new set of errors is considered. Such errors are identified by using the same error model, but they are injected into the level 2 SystemC description. New test patterns are added to the initial set in order to cover such new errors. Correct output responses of such new test patterns are obtained by simulating them on the level 1 SystemC description.

In the analyzed case, the CPU is designed to be embedded into the chip, thus, the CPU is remodeled in SystemC at the RT level (level 3 in 2). Previously identified test patterns are used at this step to verify the functionality of the CPU with respect to the embedded program. Moreover, this set is extended by using the same error model applied to the SystemC description at level 2. Correct output responses of such new test patterns are obtained by simulating them on the level 2 SystemC description.

This analysis reaches the RT level, but it could be extended to the gate level by using the same testing strategy. In this case a useful tool would be an automatic translator of VHDL or verilog descriptions into SystemC-based representations as the one described in [10].

4.1 Application to the FIR benchmark

Partial SystemC representations covering the different design levels are reported in the following. The partial level 1 system architecture is reported in Figure 3, while Figure 4 and Figure 5 show, respectively, the system at level 2 and 3.

```
SC_MODULE(fir) {
   sc_in<bool>     reset;
   sc_in<bool>     input_valid;
   sc_in<int>      sample;
   sc_out<bool>    output_data_ready;
   sc_out<int>     result;
   sc_in_clk       CLK;
   ...
   SC_CTOR(fir) {
      SC_CTHREAD(entry, CLK.pos());
      watching(reset.delayed() == true);
      #include "fir_const.h"
   }
   ...
};
```

Figure 3: Partial SystemC level 1 description of the FIR filter.

The assembly language of the embedded CPU is summarized in Table 2. Each 16-bit instruction is composed of a 10-bit field for the address and a 6-bit field for the instruction code. The C++ program describing the FIR filter is compiled into 320 assembly instructions, which are modeled by using SystemC as shown in Figure 5.

Table 3 describes the main features of the adopted benchmark: SystemC code lines and injected errors at each design level. Summarized results on the application of the described testing methodology are reported in Table 4.1, in terms of modeled errors, test patterns identified, error coverage of the test set of the previous level and of the test set extended by modeling and covering new errors of the level.

Instruction	Explanation	Code
ADD	ACC + MEM[IND] $\rightarrow$ ACC	000001
SHIFT_R	SHIFT_RIGHT(ACC) $\rightarrow$ ACC	000010
LOAD	MEM[IND] $\rightarrow$ ACC	000100
STORE	ACC $\rightarrow$ MEM[IND]	001000
JMP	IND $\rightarrow$ PC	010000
JNZ	IF (ACC $\neq$ 0) IND $\rightarrow$ PC	010001
HALT	Halt the CPU	100000
NOP	No operation	others

Table 2: Assembly Language of the embedded CPU.

Functional test patterns have been easily generated and they allowed the correct translation of the system description from one design level to the following one.

5. CONCLUDING REMARKS

The paper analyzes three different design strategies to implement an embedded system starting from an initial SystemC description. In all cases, the use of SystemC as a homogeneous development environment produces some benefits with respect to testing purposes. The same error model can be applied to the system description at the different abstraction levels, thus allowing the application of the same functional test generation technique during the entire design flow. Moreover, same technique is adopted for testing both hardware and software components. This technique produces test patterns aiming at verifying the correct translation of the design from the system to the structural level.

```
SC_MODULE(cpu_mono) {
   sc_in<bool>             start;
   sc_in<bool>             clear;
   sc_in<sc_lv<16> >       data_in;
   sc_in<bool>             clk;

   sc_out<bool>            wr;
   sc_out<sc_lv<16> >      accout;
   sc_out<sc_lv<10> >      addr;
   ...
   SC_CTOR(cpu_mono) {
      SC_METHOD(evaluate);
      sensitive_pos << clk << start << clear;
   }

   void evaluate();
};

SC_MODULE(mem) {
   sc_in<sc_lv<10> >       addr;
   sc_in<bool>             wr;
   sc_in<sc_lv<16> >       accout;
   sc_out<sc_lv<16> >      data_in;

   sc_lv<16>               memory[1024];

   void mmu();

   SC_CTOR(mem) {
   for (int i=0;  i<1024; i++)
        memory[i] = "0000000000000000";
        ...
        SC_METHOD(mmu);
        sensitive << addr;
        ...
   }
};
```

Figure 4: Partial SystemC level 2 description of the FIR filter.

The proposed testing methodology has been applied to an example architecture, where an embedded CPU is designed and integrated with embedded software. The entire testing cycle is covered by constantly expanding and refining functional test patterns generated by using the same error model and test generation technique. Future work will improve the effectiveness of the adopted functional test generation technique in order to analyze more complex systems.

6. REFERENCES

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

[2] G.J. Myers. *The Art of Software Testing.* Wiley -

Description	Embedded Algorithm			Test Oriented Algorithm		
	#Test Patterns	%Prev. Err.Cov.	%Err.Cov.	#Test Patterns	%Prev. Err.Cov.	%Err.Cov.
level 1	50	-	98.5	-	-	-
level 2	177	44%	63%	93	-	90.3%
level 3	321	69%	71%	178	78.5%	88.9%

Table 3: Functional test generation for embedded and general porpose programs.

```
SC_MODULE(cpu_rtl) {
   sc_in<sc_logic>        start;
   sc_in<sc_logic>        clear;
   sc_in<sc_logic>        clk;
   sc_in<sc_lv<16> >      data_in;

   sc_out<bool>           wr;
   sc_out<sc_lv<16> >     accout;
   sc_out<sc_lv<10> >     addr;

   // Internal Signals
   ...

   //Modules pointers
   controller_ov_fsm_syn  *controller_instance;
   clock_gen_fsm_syn      *clockGenerator_instance;
   clock_gen_fsm_syn      *clockGenerator_1_instance;
   clock_gen_fsm_syn      *clockGenerator_2_instance;
   clock_gen_fsm_syn      *clockGenerator_3_instance;
   IR_stx_syn             *instruction_register_instance;
   PC_fsm_syn             *program_counter_instance;
   acc_fsm_syn            *accumulator_instance;
   decoder_sse_syn        *decoder_instance;
   mar_stx_syn            *memory_address_instance;
   out_mod1               *out_module1_instance;
   out_mod2               *out_module2_instance;
   out_mod3               *out_module3_instance;

   SC_CTOR(cpu_rtl_autl) {
   ...
   // Port binding between modules
   ...
   }
};
```

Figure 5: Partial SystemC level 3 description of the FIR filter.

Interscience, New York, 1979.

[3] M.B. Santos, F.M. Gonalves, I.M. Teixeira and J.P. Teixeira. RTL-based Functional Test Generation for High Defect Coverage in Digital SOCs. *Proc. IEEE European Test Workshop (ETW)*, pp.99–104, 2000.

[4] R. Vemuri and R Kalyanaraman. Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming. *IEEE Trans. on VLSI Systems*, 3(2):201–214, June 1995.

[5] F. Corno, M. Sonza Reorda, G. Squillero. High-Level Observability for Effective High-Level ATPG. *Proc. IEEE VLSI Test Symposium*, 2000.

[6] F. Ferrandi, A. Fin, F. Fummi and D.Sciuto, An Application of Genetic Algorithms and BDDs to Functional Testing. *Proc. IEEE International Conference on Computer Design (ICCD)*, pp.48–56, 2000.

[7] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, A genetic algorithm framework for test generation. *IEEE Trans. Computer-Aided Design*, vol. 16, no. 9, pp. 1034–1044, Sept. 1997.

[8] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, Application of genetically engineered finite-state-machine sequences to sequential circuit ATPG. *IEEE Trans. Computer-Aided Design*, vol. 17, no. 3, pp. 239–254, March 1998.

[9] F. Fallah, S. Devadas, and K. Keutzer. OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Verification. *Proc. IEEE DAC*, pp. 152–157, 1998.

[10] N. Agliada, A. Fin, F. Fummi, and M. Martignano. On the Reuse of VHDL Modules into SystemC Designs. submitted.

[11] F. Ferrandi, F. Fummi, L.Gerli, and D. Sciuto. Symbolic Functional Vector Generation for VHDL Specifications. *Proc. IEEE Design Automation and Test in Europe Conference (DATE)*, pp. 442–446, 1999.