A Customizable Monitoring Infrastructure for Hardware/Software Embedded Systems

Martial Chabot, Laurence Pierre

A Customizable Monitoring Infrastructure for Hardware/Software Embedded Systems

Martial Chabot, Laurence Pierre

TIMA Lab. (CNRS-INPG-UJF)
46 Av. Félix Viallet - 38031 Grenoble - France

Abstract. The design of today's embedded systems requires a complex verification process. In particular, due to the strong hardware/software interdependence, debugging the embedded software is a demanding task. We have previously reported our results about the development of a framework that enables the Assertion-Based Verification (ABV) of temporal requirements for high-level reference models of such systems. In this paper, we describe conceptual and practical improvements of this monitoring infrastructure to give the user the possibility to customize and to optimize the verification process. Experimental results on industrial case studies illustrate the benefits of the approach.

1 Introduction

With the increasing complexity of systems on chips (SoC) and time-to-market pressure, rising the abstraction level to *ESL* (Electronic System Level) modeling is becoming indispensable. *Platform-based design* emerges as a new paradigm for the design and analysis of embedded systems [1]. In that context, SystemC TLM (*Transaction Level Modeling*) [2] is gaining acceptance, in particular because the simulation of TLM models is several orders of magnitude faster, thus considerably improving productivity in SoC design [3]. SystemC [4] is in fact a C++ library; a SystemC model is made of modules that have I/O ports and can run processes. They are interconnected using *channels*. A SoC model includes modules that are models of microprocessors (ISS, Instruction Set Simulator) which execute embedded software, busses, memories, hardware coprocessors like DSP (Digital Signal Processor), DMAs (Direct Memory Access, components that perform memory transfers),... In a transaction-level model (TLM), the details of communication among components are separated from the details of computation [5]. Transaction requests take place by calling interface functions of the channel models.

The challenge addressed here is Assertion-Based Verification (ABV) for complex SoCs modeled at the System level in SystemC TLM. Requirements to be verified at this level of abstraction express *temporal constraints* on the interactions (between hardware and software components) in the SoC. They can be formalized with a language such as the FL (Foundation Language) class of the IEEE standard PSL [6], which essentially represent linear temporal logic (LTL). In [7] we have presented a methodology that enables the runtime verification of

PSL assertions for TLM specifications: *assertion checkers* (monitors) are automatically generated from PSL properties, and the design is instrumented with these monitors using an ad hoc observation mechanism. Then, the SystemC simulation is run as usual, and the assertion checkers dynamically report property violations, if any. In that context, simulation traces must be sampled according to *communication actions* involved in the assertions, hence the observation mechanism detects the communication functions calls and triggers the monitors accordingly. A prototype tool called ISIS has been developed; experimental results on industrial case studies have been reported in [8], [9].

However, this original version of the tool offers few flexibility to the user. It is only possible to select, at the SystemC compile time, the assertions/monitors to be attached to the design. We present here conceptual extensions of the underlying observation model, and their implementation, that enable the designers - in particular the software developers - to customize and optimize the verification process and to get concise and easily analyzable verification results.

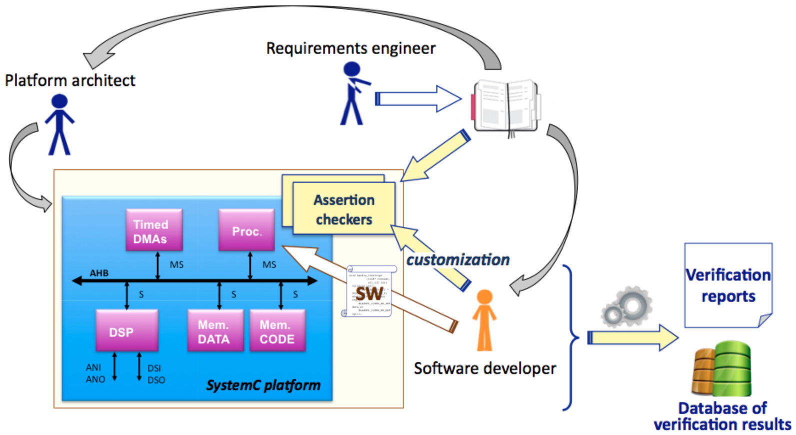

Fig. 1. Customizable Assertion-Based Verification infrastructure

2 A Customizable ABV Infrastructure

While acknowledging the tangible benefits of this verification environment, industrial partners noticed some matters of improvement. The ABV infrastructure pictured by Figure 1 targets the following characteristics:

- the SystemC platform is used by the software designer to develop the application, but he usually does not wish to recompile the platform (or even has it only under executable form). Hence selecting the assertions to be attached to the design should be feasible *at runtime*, not at the SystemC compile time,
- more essentially, it should be possible to configure the instrumented simulation to take into account relationships between assertions to dynamically disable/enable monitors, thus *clarifying and optimizing the verification*,
- it should also be possible to store concise information about assertions activations, satisfactions, violations, and to *easily analyze verification results.*

2.1 Configuration without Platform Recompilation

The assertion checkers generated from the PSL expression of the requirements, and the observation mechanism, are SystemC modules that instrument the SystemC platform. ISIS takes as input the platform source files and an XML configuration file that specifies the selected assertions, and produces this instrumented design. With the original version of the tool, the user must execute again this procedure, and recompile the resulting platform, every time he wants to choose other requirements to be checked. The tool has been improved to generate an instrumented source code that import all the assertion checkers but enables their *optional runtime instantiation*, according to choices given in a configuration file.

2.2 Monitors Enabling/Disabling

The observation mechanism is based on a model, inspired from the observer pattern, that allows to observe the transactional actions in the system and to trigger the monitors when needed [7]. Each monitor is enclosed in a wrapper that *observes* the channels and/or signals involved in the property associated with this monitor; the channels/signals are *observable* objects (or *Subjects*).

Originally, the assertion checkers remain active all along the simulation. However, requirements are usually *correlated* (not by purely *logical* relations, that could be identified by subsumption detection [10], but by *conceptual* relations, determined by the designer). The tool has been enhanced to *enable/disable monitors* according to these relations. The model has been extended with a *Verification Manager* component, as described by the class diagram of Figure 2.

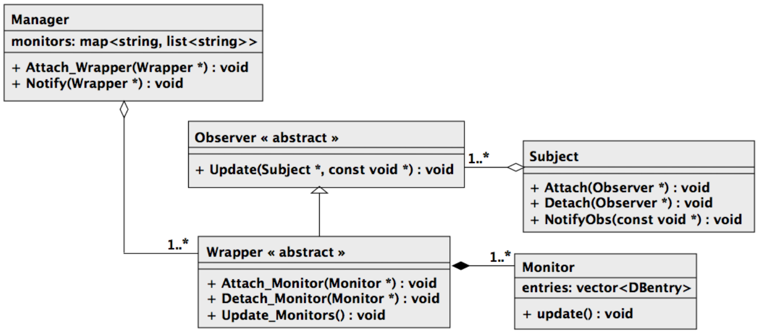
Manager
monitors: map<string, list<string>>
+ Attach_Wrapper(Wrapper *) : void
+ Notify(Wrapper *) : void
Observer « abstract »
+ Update(Subject *, const void *) : void
1..*
Subject
+ Attach(Observer *) : void
+ Detach(Observer *) : void
+ NotifyObs(const void *) : void
1..*
Wrapper « abstract »
+ Attach_Monitor(Monitor *) : void
+ Detach_Monitor(Monitor *) : void
+ Update_Monitors() : void
1..*
Monitor
entries: vector<DBentry>
+ update() : void

Fig. 2. Extended model - Verification manager

A Manager can include a collection of wrappers, and takes care of the associated monitors. By means of a configuration file, the user specifies relations of the form $A_i \mathcal{R} A_j$ which express that, if assertion A_i experiences violations, then checking assertion A_j becomes worthless. The Manager can be configured such that, when it detects that the monitor of A_i reports violations, either it only disables A_i immediately (Level 1), or it also disables the monitor of A_j (Level 2). As illustrated in section 3, this has two main advantages: simulation traces *only include the most relevant information* provided by the checkers, thus simplifying the interpretation of the verification results; the CPU time overhead induced by the checkers *may be minimized*, thus optimizing the verification process.

2.3 Database with Verification Results

To ease the analysis of the verification results, in addition to the textual verification reports, verification results are now stored in a database. To achieve this, the *Monitor* class has been extended with a member which is a vector of database entries (see Fig. 2). During a simulation, the monitor stores here the information about every assertion activation: start time, end time, status (pass or fail). This information is ultimately committed to the database. A post-processing tool extracts a concise and easily analyzable tabular representation of the results. Other post-processing tools e.g., for statistical analysis, are under development.

3 Experimental Results

3.1 Space High Resolution Image Processing

This case study (from Astrium) is an image processing platform that performs spectral compression, see Figure 3. It mainly includes two processors, *leon_a* and *leon_b*, two DMA components, memories, a FFT coprocessor, and an IO module.

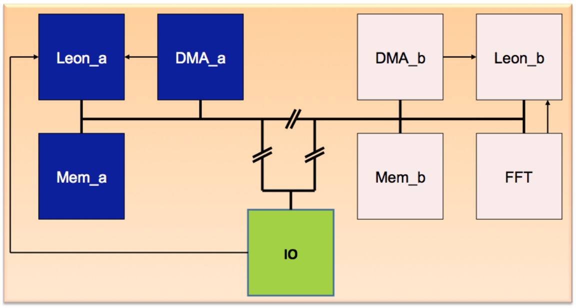

Fig. 3. Image processing case study

The *leon_a* processor configures the *DMA_a* component to get the image from the *IO* module to *Mem_a* where raw data are sub-sampled. Packets are then transferred to *Mem_b* and a 2D-FFT is applied to obtain the corresponding spectrum. The resulting packets are then compressed by the *leon_b* processor which configures *DMA_b* to send the output packets back to the *IO* module. Process synchronization is ensured by software. A set of requirements has been defined for this platform, some of them are presented below:

- A_1: *DMA_a must not be reconfigured before the end of a transfer.*
- A_2: *The FFT must not be reconfigured before the end of a computation.*
- A_3 (no loss of input data): *Every input data packet must be transferred by DMA_a to Mem_a before the IO module generates a new interrupt.*
- A_4 (software not too slow): *Every data packet stored in Mem_a must be processed by leon_a before being overwritten* (note: *Mem_a* is 32 MB large)

- A_5 (no loss of processed data): *Each incoming data packet* (read by DMA_a from the IO module) *must have a corresponding output packet* (written by DMA_b to the IO module)*, before 3 new incoming data packets are processed.*

There are conceptual dependencies between these requirements. For example, if A_3 is violated (meaning that input data are lost), the results of A_5 become irrelevant because paying attention to the loss of output data is useless, the bug revealed by A_3 must first be fixed. Property A_2 however might still be enabled, to continue checking the interactions between *leon_b* and its FFT coprocessor. Table 1 summarizes results for processing 10000 images, with configurations that can induce data loss: CPU times, total number of checkers activations (notifications on function calls), and of checkers actual triggerings. Column 1 is for the raw simulations, column 2 gives the full monitoring results (Level 0), column 3 corresponds to Level 1 of the Manager, columns 4 and 5 correspond to Level 2, with the management of one or several assertion correlations. Assertion A_4 is the most time-consuming one (it may require several thousands of concurrent monitor instances), configurations in which it may be disabled are optimum.

3.2 Avionics Flight Control Remote Module

This case study (from Airbus) is an avionics module dedicated to processing data from sensors and controlling actuators according to predefined flight laws. Certification issues and specific safety requirements must be taken into account. Properties are associated with the joint behaviour of the software and the DSP coprocessor (see Figure 4). This component stores and uses data that represent digital filter coefficients and analog input (ANI) calibration coefficients. Data integrity must be checked, to protect the device against SEU (Single Event Upset, change of state caused by energetic particles). Among the requirements:

- A'_1 : *Checksum computation must be performed every CHECKSUM_PERIOD ms* (the corresponding result is stored in the register STATUS of the DSP).
- A'_2 (ensures detection of potential checksum errors): *The software must read the content of STATUS every CHECK_PERIOD ms.*
- A'_3 : *When a checksum error is detected (wrong value in STATUS), the DSP function must be deactivated (within LIMIT ms).*

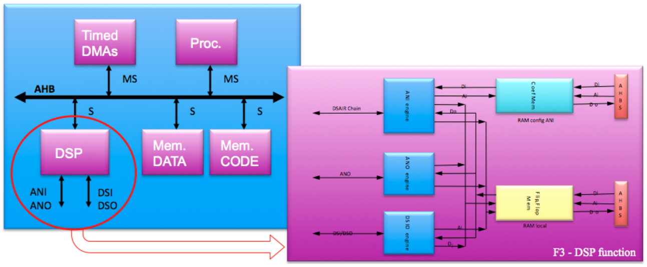

Fig. 4. Flight control case study

Those properties are clearly intercorrelated. Table 1 summarizes some results, for the processing of 25000 analog data coming from the ANI port, with configurations that can induce missed deadlines. Here full monitoring is not really time-consuming (about 2% CPU time overhead), therefore the benefit rather comes from the fact that the number of assertions triggerings significantly decreases, hence making the results more conveniently analyzable because avoiding irrelevant trigerrings means that only the most useful information appears in the simulation trace.

	Raw simulation	Full monitoring	Violation detect. (Level 1)	Level 2, 1 correlation	Several correlations
Image proc.	71.10 s	211.2 s 1311 10^6 activ. 2.13 10^6 trigerrings	219 s 1310 10^6 activ. 2.11 10^6 trig.	$A_3 \mathcal{R} A_5$ 213.84 s 1310 10^6 activ. 2.1 10^6 trig.	$\forall j, A_3 \mathcal{R} A_j$ 72.68 s 461340 activ. 632 trig.
Flight control	52.41 s	53.5 s 1.2 10^6 activ. 67 trigerrings	52.84 s 860000 activ. 58 trigerrings	$A'_1 \mathcal{R} A'_2$ 52.57 s 500000 activ. 8 trigerrings	$\forall j, A'_1 \mathcal{R} A'_j$ 52.52 s 135000 activ. 7 trigerrings

Table 1. Comparative results for various manager configurations

4 Conclusion

The results show that this version of the ABV tool can significantly improve the efficiency of the instrumented simulations (e.g., fifth column of the first row), and can produce more easily analyzable results, which is crucial to facilitate debug. On-going work includes the identification of correlations between monitoring results and specific patterns in the simulation traces.

References

1. Carloni, L., De Bernardinis, F., Pinello, C., Sangiovanni-Vincentelli, A., Sgroi, M.: Platform-Based Design for Embedded Systems. (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.7365)
2. Grötker, T., Liao, S., Martin, G., Swan, S.: System Design with SystemC. Kluwer Academic Pub. (2002)
3. Klingauf, W., Burton, M., Günzel, R., Golze, U.: Why We Need Standards for Transaction-Level Modeling. SOC Central (2007)
4. : IEEE Std 1666-2005, IEEE Standard SystemC Language Ref. Manual. (2005)
5. Cai, L., Gajski, D.: Transaction Level Modeling: An Overview. In: Proc. International Conference CODES+ISSS'03. (2003)
6. : IEEE Std 1850-2005, Standard for Property Specification Language (PSL). (2005)
7. Pierre, L., Ferro, L.: A Tractable and Fast Method for Monitoring SystemC TLM Specifications. IEEE Transactions on Computers **57** (2008)
8. Pierre, L., Ferro, L., Bel Hadj Amor, Z., Lachaize, J., Lefftz, V.: Runtime Verification of Typical Requirements for a Space Critical SoC Platform. In: Proc. Int. Workshop on Formal Methods for Industrial Critical Systems (FMICS). (2011)
9. Pierre, L., Ferro, L., Bel Hadj Amor, Z., Bourgon, P., Quévremont, J.: Integrating PSL Properties into SystemC Transactional Modeling - Application to the Verification of a Modem SoC. In: Proc. IEEE SIES'2012. (2012)
10. Biere, A.: Resolve and expand. In: Proc. of SAT'04, Springer (2004)