Theme 3 / Design and verification of Systems on a chip architectures
VDS Group: Verification

Themes

Test and verification methods for mixed or digital IP blocks
IP blocks synthesis from logic-temporal specifications
Test and verification methods for hardware/software on-chip systems
Verification methods for Networks on Chip
Formal methods for robustness analysis

Expertise

Scientific
Simulation and description semantics, systems modeling, requirements formalization, temporal logics, automatic proofs

Fields of expertise
Correctness verification at various levels of the design flow, robustness analysis and formal methods

Know-how
Formal specification of functional and robustness requirements, formal verification (mechanized proofs), assertion-based design, symbolic simulation

Industrial transfer
Transfer of HORUS technology in EDA tools for mixed systems by Dolphin Integration

Research keywords
Specification and verification of complex systems, EDA (Electronic Design Automation) assertion-based verification, design flow, correct-by-construction design

Contact

Laurence PIERRE
UJF Professor – TIMA
(+33) 476 57 49 92
laurence.pierre@imag.fr
Verification of Hardware/Software Systems on Chip

Key-words: Verification of SoC, Assertion-Based Verification (ABV), SystemC TLM

Members: L.Pierre, Z.Bel Hadj Amor, M.Chabot, D.Borrione

Debugging today’s hardware/software embedded systems is a complex process. We have previously described our tool, ISIS [1], that enables the runtime Assertion-Based Verification (ABV) of temporal requirements for high-level (SystemC TLM) models of such systems, by the automatic instrumentation of SystemC virtual platforms with assertion checkers. This verification infrastructure has been enhanced with two major features. The first one offers the possibility to customize and to optimize the verification process, and to get more easily analyzable results. The flexibility of the original version of ISIS was limited, it was only possible to select, at the SystemC compile time, the assertion checkers to be attached to the design. It now enables to customize at runtime and to optimize the process, and also to get concise information about checker's activations, thus improving the debugging task [2][3].

The second additional key feature is a companion tool that performs a semantic refinement of the temporal assertions, such that analogous requirements can be verified before and after ESL-to-RTL synthesis [4].

1. ABV toolchain

These improvements result in a toolchain that seamlessly embeds runtime verification in the design flow, as pictured by Fig.1.

While the initial system requirements can be used by the platform architect to propose a SystemC virtual platform as the outcome of architecture exploration and HW/SW partitioning, their PSL formalization is transformed by ISIS into assertion checkers that instrument this platform. At that point of the design flow, the “processor” representation in the SystemC model is often a simple traffic generator used to perform preliminary evaluations of the transactional behaviour of the platform. The checkers can be used to report first conclusions about the integration of the components (i.e., the functional correctness of the interactions between them), and to confirm the architecture exploration choices.

Once the hardware platform is considered adequate, a more realistic processor model (ISS) can be integrated, and software development can start. The software developer can use the monitors to check the functional correctness of the hardware/software interactions. To that goal, he can customize the checkers activity at runtime. It is also possible to configure the simulation to take into account relationships between assertions to dynamically disable/enable monitors, thus clarifying and optimizing the verification. Verification results are provided both as textual outputs and as concise information stored in a database to be analyzed by post-processing tools.

The next step of the design flow is to provide the RTL (Register Transfer Level) implementation of this abstract model, by means of HLS (High Level Synthesis) tools and/or by manual refinements. An assertion refinement process conveys assertions through this ESL-to-RTL refinement procedure. It gives designers the possibility to reverify, after synthesis, the exact semantical counterpart of the initial System-level requirements.

Let us discuss these improvements in more details.

2. Improvements of the verification process

In addition to enabling the optional runtime instantiation of the assertion checkers (w.r.t choices given in a configuration file - this file is generated with default values that enable all the assertions and that can be toggled to “disable”), ISIS now also allows to take into account the fact that requirements are usually correlated (not by purely logical relations, but by conceptual relations determined by the designer). For example, if a property that states that input data are not lost in a image processing platform is violated, checking that these data are not lost within the platform becomes useless.

To that goal, a Verification Manager component has been introduced in the observation model. By means of a configuration file, the user specifies relations of the form \( A \models p \rightarrow q \) which express that, if assertion \( A \) experiences violations, then checking assertion \( q \) becomes worthless. The Manager can
be configured such that, when it detects that the monitor of Ai reports violations, either it only disables Ai immediately, or it also disables the monitor of Aj. 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 and improving debugging; the CPU time overhead induced by the checkers may be minimized.

To ease the analysis of the verification results, in addition to textual reports, verification results are now stored in a database. To achieve this, the monitors have been extended with a member which is a vector of database entries. 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 result.

Fig. 2: Verification post-processing

Fig.2 shows how the post-processing tool displays simulation times and the results of the evaluations.

3. Assertions from TLM to RTL

For a comprehensive and seamless verification flow, analogous requirements should be verifiable before and after ESL-to-RTL hardware refinement. This requires the transformation of ESL assertions into their counterparts at the RT level. We have defined a methodology and implemented a tool that performs the automatic extraction of signal level assertions from system level transactional requirements. Beyond simple issues such as data concretization, the most salient concern is related to the modification of temporal granularity due to the introduction of actual communication channels in place of abstract components. A set of PSL transformation rules has been defined. These rules address cases where an atomic communication action is transformed into a given sequence of actions, and cases where a concrete communication action is expected after a certain number of clock ticks. A transformation rule $L \rightarrow C R$ maps a TLM temporal expression $L$ into the corresponding RTL expression $R$, according to given time constraints $C$. The role of a constraint is to specify delays introduced by the actual communication protocol (for example, for a single write operation for an AHB bus, the constraint related to the "write" TLM function specifies that there is a one-cycle delay between sending the control and address and sending data).

Given a TLM assertion, the tool recognizes where some of the transformation rules can be applied (using unification) and applies them, see Fig.3. The implementation is such that minimal user guidance is required: the designer has to disclose the actual protocols and timing that are introduced in the TLM-to-RTL design flow, but we provide an easy-to-use procedure to guide this process.

To the best of our knowledge, this tool is the first implementation of a refinement process that takes into account the conversion of temporal granularity to automatically generate RTL temporal assertions from ESL requirements.

References
Automatic Compilation of Properties into Synthesizable Designs

Key-words: Correct by construction, Assertion-Based Design (ABD), PSL

Members: N.Javaheri, K.Morin-Allory, D.Borrione

Declarative specifications are now widely adopted in the context of verification: declarative properties about the behavior of a design (Assertions) or its environment (Assumptions) are checked using dynamic or static verification tools. Once refined down to the register transfer level (RTL), a complete set of assertions unambiguously characterizes how a module reacts to signals sent to it, logically and temporally. Many tools are now available to compile assertions into monitors, i.e. verification IP's that check the design correctness, either by simulation or emulation.

Our current project considers the direct production of compliant control and communication modules from a set of assertions. A property is seen as the specification of the module to be designed, and we directly produce the synthesizable RTL design from its assertions. For each property, we obtain a compliant RTL component called reactant: its inputs and outputs are operands of the property, it reacts to the input values and produces output values so that the property holds.

Previously published works are based on automata and game theory. In contrast, our method is modular: it is based on the interconnection of primitive library modules for the logical and temporal operators of the property, according to its syntactic structure, a technology that we initially introduced to compile assertions into monitors (Horus). Fig.1 illustrates the construction.

Property P2 is always \(\{ \text{BtoS\_ACK\_0 and StoB\_REQ\_0} \} \rightarrow \text{next} \{ \text{BtoS\_ACK\_0} \} ;\)

Fig. 1: Reactant architecture for property P2

In general, the specification has many properties, and a same variable may appear in several distinct properties. The originality of our approach is to avoid combining all the properties into one big automaton. Our method constructs the dependence between the design variables, and identifies which properties monitor a variable, and which properties generate its value. If a variable is an output for several reactants, these are combined with a solver to produce the final design.

SynthHorus-2 is a new software tool that implements these principles [1]. It takes as input the entity (interface) declaration of the specified module and a set of properties written in the simple subset of PSL, and produces a RTL design in the synthesizable subset of VHDL. SynthHorus2 compiles several dozens properties in seconds, and produces a reasonably sized RTL circuit model. Experiments and performances were reported in 2013. In 2014, new optimizations have been performed, leading to significant improvements in the size of the combinatorial part of the reactants. Original benchmarks have been specified and processed: a high-level data link control (HDLC) module, a SDRAM controller, and a CRC; the results obtained show the versatility of the method.

For all the temporal operators, we have defined a Dependence relation \(\Delta\) between its operands \(\exp\) and \(\cond\), in accordance with the trace semantics of PSL. It is based on writing it under one of these two generalized expressions:

\[
\forall i \in [k_1, k_2], [\exp \Delta \cond]_i^k
\]

\[
\exists i \in [k_1, k_2], [\exp \Delta \cond]_i^k
\]

where \(k_i\) is the first time point of the trace when some formula \(F\) has been true exactly \(i\) times (\(F\) depends on the temporal operator). The hardware semantics of the \(\Delta\) relation, of a counting function, of the \(\forall\) and \(\exists\) quantifiers were defined as logic circuits. From there, the construction of the reactants is obtained by mere interconnection of the primitive modules for the operators generalized expression. The construction was proved correct by inductive reasoning [2].

Fig. 2: Hardware semantics for Next_e [kmin, kmax]

What remains to be done is the synthesis of properties written with SEREs. A partial solution is under construction.

References

[1] Morin-Allory K., Javaheri F.N., Borrione D., "Design Understanding with Fast Prototyping from Assertions", Workshop on Design Automation for Understanding Hardware Designs (Friday Workshop DATE’14), Dresden (Germany), 2014