Property Verification in the Design of Telecom Applications*

M. Bombana
DRSI
Italtel
Milano, ITALY 20019
Tel: +39-2-4388-7431
Fax: +39-2-4388-8593
e-mail: bombana@italtel.it

P. Cavalloro
DRSI
Italtel
Milano, ITALY 20019
Tel: +39-2-4388-7431
Fax: +39-2-4388-8593
e-mail: cavallor@italtel.it

F. Ferrandi
Dip. di Elettronica e Informazione
Politecnico di Milano
Milano, ITALY 20133
Tel: +39-2-2399-3636
Fax: +39-2-2399-3411
e-mail: ferrandi@elet.polimi.it

Abstract— The industrial interest in the application of formal methods in the design of complex ASICs is noteworthy to improve the efficiency of the design process (reduced time-to-market) and to increase the quality of the final products (increased competitive profile). In this paper we focus our attention on design capture and functional verification, two critical phases in the current design methodologies. A modular toolset built around a model checker is described. A telecom co-processor is presented, and general properties derived. A user-oriented taxonomy of properties is introduced to support the design practice. Guidelines for the application of this technique are inferred from the example and generalized.

I. Introduction

In the telecom market the design time of a new component becomes more and more comparable with the mean expected life on the market of its current implementation. Functional and real-time constraints, changing rapidly, make it soon obsolete, i.e. requiring for example a different implementation with a new technology, or overcome by competition. Moreover complexity is expected to grow at a rate of 50% per year, and this must not impact negatively on the product quality. For these reasons new design methodologies attract a great interest from manufactures, as the only available mean to support efficiently such market requirements. Specifically design improvements can be obtained by applying exact and rigorous design methods. As a consequence, the industrial interest in the application of formal methods applied to the design of complex ASICs is noteworthy. Design methodologies have been proposed ([1], [4], [5], [6], [7], [11],[12] [13]) and commercial tools are now available ([2], [8]) to partially support the design practice with the power of formal reasoning at different levels of the design flow. Anyway the application of formal methods is not trivial due to the innovative content of the theoretical approach and to the fact that some of the supporting tools are not mature to cope with the industrial needs in terms of device complexity, computational efficiency or user-friendliness. Moreover guidelines in terms of good practise for their application in-field are completely missing, so diminishing considerably their impact on potential users and design flows.

To overcome these problems it is necessary to clearly identify the role that these techniques can play in the standard design flows and to highlight the advantages that they can offer in comparison with standard approaches in terms of both improved efficiency of the design process (reduced time-to-market) and of the increased quality of the final products (increased competitive profile) [3].

Manufactures' design flows are complex multi-vendor environments, organized in a top-down sequence of tools encapsulated into ESDA (Electronic System Design Automation) frameworks, which provide homogeneous environments and user-interfaces. From an operational point of view design flows include three different abstraction levels: a functional phase, a logical phase and a technological phase. In the first phase design capture is implemented, the system specification is generated, and system level verification is applied to the generated specification. Verification is accomplished through simulation, even if this technique is non-exhaustive. Hardware description languages, and specifically VHDL and Verilog, have reached the stage of de-facto standards in most design sites. They are supported by almost all ESDA tools and well known by designers. Anyway the first stages of design are the most critical vs the introduction of errors and inconsistencies, due to the complexity of designs. VHDL code in particular needs to be verified in terms of design requirements at the functional level. The second phase is called logical design because logic synthesis tools are used to produce low-level netlists starting from RTL structural VHDL architectures. The third phase, called technological design, takes care of generating the layout and deals with the physical implementation.

Recent surveys of the time spent in the various phases of the design flow point to the fact that the functional phase becomes more and more predominant in the global design

*This research was partially supported by ESPRIT projects FORMAT n. 6128 and REQUEST n. 20616

ASP-DAC '97
0-89791-851-7$5.00

time. Moreover errors made at this level are transmitted and amplified in the following phases. In this paper we show how property checking can play a predominant role in assessing the functional correctness of control-oriented VHDL models vs the initial abstract requirements (design capture). The missing guidelines for good practice for the application of this technique are provided on the base of the experience obtained from the analysis of the control part of a telecom co-processor. In this way property checking can be considered a useful complement to simulation.

In section II. we describe the applied toolset and its corresponding design flow, underlying mainly two aspects:

- the modularity of the toolset and its user-friendliness when it is used on a stand-alone basis;
- feasibility of integration into an industrial design flow and ESDA framework.

In section III. we briefly introduce the telecom application to which the design flow has been applied.e this goal, modularity and flexibility are required to guarantee an easy encapsulation in ESDA environments. Section IV. provides a taxonomy of functional properties based on design experience. The following section describes the obtained results and generalizes them into guidelines for future application of this approach, from the designers' point of view. Finally, Section VI. is devoted to conclusions and directions for future work.

II. Design flow for properties checking in VHDL

Industrial design flows are basically multi-vendor environments, where different needs are satisfied by the most appropriate tools. In order to achieve this goal, modularity and flexibility are required to guarantee an easy encapsulation in ESDA environments. The toolset we considered [10] for property verification in VHDL models satisfies these requirements.

The core of the proposed design set is a powerful model checker [14]. This tool uses BDD based model checking techniques ([15]) and verifies that an implementation, in the form of a transition system (model), satisfies a specification given in terms of a formula in temporal logic (property) ([16], [14]). The model checker can also prove (by a process of weak simulation) that one design entity is refined by another. A tautology checker, included in the same package, is able to determine the universal correctness of a Temporal Logic Formula. An advanced feature of this package, seldom found in similar tools, is the sophisticated error path that is produced when the property is not satisfied in the chosen model. This feature is fundamental, as it enables an easy identification of the error and the subsequent correction of the VHDL model. The error path consists of a simulation sequence, leading from the activation of the process till the event that contradicts the required property. A graphical interface supports the user through a visualization of the sequence of patterns, providing both a simulation counter-example and a graphical representation of the found inconsistency.

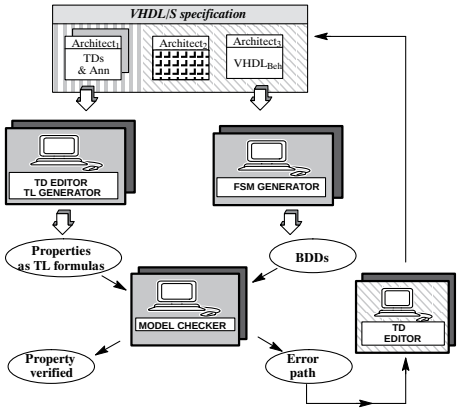

Fig. 1. Verification Flow for design properties through model-checking.

Properties cannot be directly specified in temporal logic, because designers have no background in this type of formalism. To overcome this difficulty, a graphical language has been developed, provided of sound semantics, able to capture the meaning of temporal properties as timing diagrams [4] through a graphical editor. The same editor visualizes the error path generated by the model checker, so providing a unified interface to the user, from specification to debugging.

A synthesisable VHDL subset is used as entry language for the description of behavioral models. This description of the behavioral code is processed by a commercial compiler [9] which provides the syntactic checks, and stores the information into an object-oriented data base. A finite state machine representation for the control modules is generated from this internal format and formalized as BDDs, while the timing diagrams are translated into Temporal Logic Formulas.

The various modules nicely build up an integrated design flow (see figure 1). Moreover the most complex operations and internal formalisms used in formal design are hidden from the average designer, who can use known specification styles, i.e. VHDL and timing diagrams. The entire path is characterized by a high degree of user interaction: both the compiler into BDDs and the Model Checker require user defined parameters, giving the designer a full control of the design process. The flow is meant to be applied several times, in a design loop fashion. In fact, corrections are introduced following the indications of the error path when inconsistencies are discovered in the provided specifications.

A design flow including this verification step is shown in figure 2 in the box labeled 'verification'. Clearly it

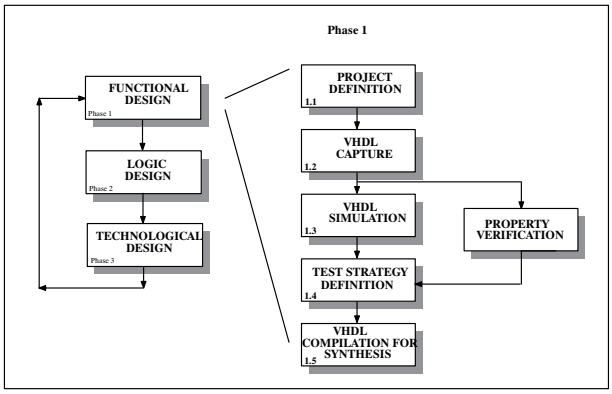

Fig. 2. Design flow including formal verification.

TABLE I
CHARACTERISTICS OF THE CONSIDERED MODULES.

Module	S	T	IV	LV	IB	SB	BDD	VHDL
`main module`	226	319	21	89	39	255	745	332
`control block #1`	155	214	15	60	15	81	155	307
`control block #2`	73	94	9	52	10	69	75	183
`computation block`	63	75	19	64	46	182	762	131

S = # States
T = # Transitions
IV = # Input variables
LV = # Local variables
IB = # Inputs bits
SB = # State bits
BDD = # BDD nodes
VHDL = # lines of VHDL code without standard packages

complements the simulation task and operates in parallel with it.

III. Device specification

In the telecom domain, the percentage of devices implemented as Application Specific Integrated Circuits (ASICs) is expected to grow considerably. Their complexity and the strict design constraints they have to satisfy, put under a hard test actual and future ESDA environments. The device we chose is designed to monitor the run-time behavior (incoming rate) of the Asynchronous Transfer Mode (ATM) connections, acting as a filter when they do not comply to specifications. The module is part of a large board composed of several ASICs, memories, busses and controlled by a micro-processor. The unit operates as an arithmetic co-processor devoted to data packages analysis and translation. The co-processor applies a computationally simple algorithm, including multiplications, enhanced to avoid data overflow. The computation of the algorithm is parameterized according to different classes of users. Computation parameters are stored in a register file upon the initialization phase. In the selected implementation each class of users is characterized by a specific set of the parameters.

The design capture of the co-processor is handled in VHDL applying a 'divide and conquer' strategy to decrease the behavioral complexity of the system. The result of the partitioning is the identification of a data path and a control module, with the addition of supporting logic and memories to comply with the required functional specification. When the control module is still too complex, it is further partitioned into interacting finite state machines.

The result of a possible hierarchical partitioning specification identifies two FSMs: one to control specifically the data path, the second to control I/O operations and interaction with the main memory. Communication protocols are defined to model the communication with the environment, and among communicating modules. Such protocols provide a clear path to identify in part the required properties (in terms of communication) that the VHDL models must satisfy. In figure 3 the protocols between the control module and the surrounding blocks are sorted out. Two different levels of complexity are identified: the first regards the verification of each protocol by itself (i.e., considered as a binary interaction among two modules, or one module and the environment). A second level of complexity involves functional sub-parts of the VHDL model where different protocols are subject to a strict sequencing structure. Both cases can be verified formally.

Statistics related to the size and complexity of the considered modules are reported in table I.

Each block is defined as an abstract VHDL entity, to which different architectures are associated. In addition to the traditional structural and behavioral descriptions, new architectures include timing diagrams (TDs). To allow an automatic verification of the properties of a module both specifications and implementations are VHDL architectures associated with the same VHDL entity. Usually not all the component referenced in the structural description must have timing diagram specifications. This means that the verification can deal with incomplete specifications. The designer incrementally describes all the modules of the hierarchy of the device and correspondingly verify step by step the device design. The applied methodology considers the possibility of using logic synthesis tools after the functional verification phase (logic phase of figure 2). In this case the initial VHDL belongs to the subset accepted by the logic synthesis tool used in the manufacturer's design flow (e.g. Synopsys and Mentor).

When the modules are still rather complex the partitioning strategy can be applied again, producing further structural decompositions into simpler elements at a later stage. In this way it is natural to include also design constraints and implementation details, like word length, data handling in records, memory elements initialization,

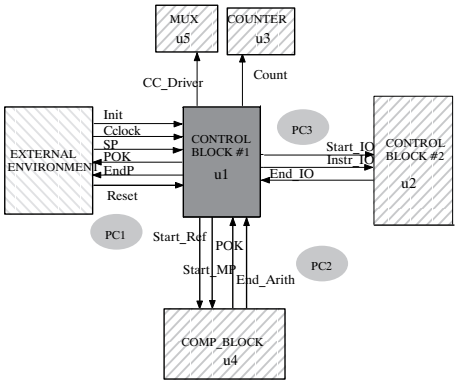

Fig. 3. Structural simplification of the protocol involving Control Block #1.

available RAM components, testability issues and so on. The process of partitioning ('design-capture' phase) ends when all the blocks of all the levels are defined in forms suitable for logic synthesis or specified as netlists of components belonging to a reference CAD library.

IV. Design Properties

Properties consist of assumptions and commitments. Assumptions are meant to model a 'good behavior' from the environment. For instance in figure3, they concern the activation of the algorithm, which is constrained by two events (external signals' edges): the `Reset` signal must go down (after having gone high, meaning that the reset phase is terminated) and the `CClock` signal must go high (meaning that a ATM cell has arrived and the computation can start). Other assumptions are necessary from the point of view of the logic representation, but are usually 'assumed' so by designers. So the system automatically takes into account the latter, without explicit mentioning. On the contrary commitments are user defined and represent the properties to be checked. To make the task of property selection and specification easier to the average designer, we introduced a taxonomy of hardware properties, which is more user-oriented than the usual division in liveness or safety properties. This taxonomy mainly applies to control-dominated applications, and includes six main classes:

1. required sequences of events or states;
2. prohibited conditions involving multiple signals;
3. requirements of the behavior on reset;
4. application-specific requirements on behavior;
5. actions performed in a state of the FSM;
6. stability in basic states.

Each class generally includes both liveness and safety properties. This taxonomy helps the user in the definition of the properties, because they belong to the usual background of a hardware designer and provides a better framework to support the specification of the selected properties. Some examples of properties are given in the following (names are self explaining).

ReadtoWriteseq (Group 1). This property expresses the requirement for correct READ/WRITE sequencing. The output instruction signal `Instr_IO` contains the sequence of values READ and WRITE, and the transition to the WRITE value is triggered by the falling edge of the incoming enabling signal (control signal indicating the end of computation). This property is based on two assumptions (in addition to the standard ones): that a cell arrives and the algorithm is activated.

NeverEnd_ArithandInstrIOeqwrite (Group 2). This property expresses the requirement that a WRITE value on the output instruction signal `Instr_IO` must never occur when the incoming enabling signal is high. This condition is expressed using a probe in which the boolean expression `not(En = '1' and Instr_IO = WRITE)` is always set to true.

GoingtoR1onReset1 (Group 3). This property expresses the fact that from any state, the rising edge of the reset signal will bring the system to state R1 . Also in the VHDL model the asynchronous reset signal is treated differently from the others.

Cell_accepted (Group 4). This property expresses the requirement that the signal specifying that a cell arrived and was accepted goes high after some other signals went high.

EventsfromI0 (Group 5). This property expresses the requirements on the behavior specified in the corresponding FSM when the system is in a state and evolves to the next state. Also in this case, these properties are completely dependent on the definition of the FSM, so on the application.

GoingtoRR1onInitdown (Group 6). This property expresses the requirement for the system to go back to a fundamental state RR1 when an input signal has a transition. The property is similar to the one for the reset, but in this case no asynchronous behavior is present. In the test-bench this case is verified each time the initialization signal is activated by the microprocessor to store the reference value in registers and memory.

A large set of properties were described as timing diagrams, automatically translated into Temporal Logic Formulas and tested on SparcStation 10 with 32Mb of RAM ([14]).

Execution times are not too large and in average comparable with simulation times for the same models. The advantage of the approach lies in the automatic definition of the debugging sequence. Anyway designers have to select properties with some care. In fact execution times strongly depend on the type of properties. Then, for the same property, the execution time depends on the size of the model, i.e., the number and range of IO ports and the number of the internal states of the finite state machine.

The model of the previous example is manageable because all the IO ports are control signal, i.e., bits.

V. From the user's point of view

A. Good practice in properties definition

A large percentage of properties tested on the device resulted false. Even if it is expected that industrial designs contain errors in the specification and in the implementation the number of failures is higher than expected. In fact, a closer analysis of these cases has allowed to correct errors in the properties definition. Let us consider some examples.

The property *NeverEnd_ArithandInstrIOeqwrite* has been proved `False`. A counter-example has been generated by the system in which the input enabling signal switches to '1' and `Instr_IO` switches to WRITE. The sequence involves 142 steps from the initial state to the state that contradicts the commitment. This result depends on the fact that the designer is testing a relationship involving an input signal and an output signal. This is not correct because, while the output signals are constrained by the behavior under test, the input signals are allowed (by definition) to cover exhaustively the range of values allowed by their type definition. So the property is not `False`, but only badly formulated.

Also the property *ReadtoWriteSeqSafe* has been proved `False`. The execution time is high because the timing diagram is rather complex. A counter-example has been generated by the system in which the signal assessing the end of computation switches to '0' while the instruction signal `Instr_IO` did not switch to WRITE. In fact this behavior is forbidden in the operative phase, but it is allowed in one case of the initialization phase. So, in this case the property has been stated without correctly restricting the model to which it applies (too general assumptions). In fact the initialization phase should have been excluded by the model under test for this property checking. The counter-example is very complex and involves 106 steps from the initial state to the state that contradicts the commitment.

In general, from the analysis of the properties considered during the verification of our application the following rules can be assessed:

- Better state a relationship between output signals only, or directly constrain (with assumptions) the values of the input signals.
- Split a complex property into several simpler ones, otherwise, execution time grows exponentially.
- Avoid relationships holding only in some states of the FSM, or restrict the model with further assumptions.

Finally some other properties, well defined in terms of application domain in the model, proved false. In these cases the model was not compatible with the assessed property. This does not mean that the VHDL code representing the model must be changed. On the contrary, it shows clearly a contradiction in terms of requirements for the device to be designed. In this way a better understanding of the functionality required by the application was highlighted. In some cases it resulted that the property, even if stated correctly from the syntactical point of view, was not correct 'semantically' for the functionality to be designed. In this case the property was dropped and another one specified in its place. In other cases the model was found defective, or wrong, and then modified accordingly. The following rule can be generalized: when a property is false, first verify the application domain, to make sure it is correct syntactically, second reexamine the specification, to check why a contradiction exists, third modify the property or the model according to the results obtained by the analysis.

B. Simulation vs properties verification

Some of the property verifications were compared with the results obtained by the simulation of the model (Mentor Graphics QuicksimII). Observing the simulation results the following conclusions can be derived:

- Simulation clearly identifies local relationships between the various signals.
- Properties express relationships which are proved correct over the whole behavior of the model.
- Simulation results are strongly dependent on the correct sequence of the input signals given by the designer, which is not exhaustive by construction. As a consequence it is not possible to guarantee that an unexpected but possible sequence of events may generate a misbehavior.
- On the contrary, when a property has been proved `True` by the model checking technique, it is so for all the possible sequences of input events.

Another advantage over simulation, which is allowed by the described tool-set, is the *Error path* generated when a property is not verified. In fact, by following the indications of the error path, the designer can identify and correct the inconsistencies among high-level descriptions and specifications. Simulation is still a necessary step in the design verification process, because the technique we described is not covering all the possible functionalities of the model. The final goal of the joint application of simulation and properties verification is to better understand the specification and get an enhanced assessment on the global results of functional verification.

C. Impact of properties verification on design practice

Local variables and states are almost always present in the behavioral code representing the model to be veri-

fied. Properties are very often dependent on these internal signals (local variables) and states. In order to verify properties addressing this kind of signals, they must be explicitly referenced in the VHDL entity of the model. Basically they are declared as output ports, in order to be observable (probes). This implies a modification of the VHDL model to accommodate this feature.

As one can see, the changes to the code are not dramatic. Anyway it is evident that inserting them prior to the verification phase, and deleting them afterwards implies a manipulation of the specification which is not guaranteed to be error free.

The execution times, quite acceptable for the control part of the device, increase considerably when the data path is involved, or when the model of top entity (including control and data path prior to partitioning) is verified. This observation proves that the model checking technique well suited to control-dominated applications, has still some limitations when the data path is involved.

For example, the description of the package for our application presents declarations of local variables with a rather great range of data. This feature is often shared by applications in the telecom domain. In fact, all the functions performed in the data path (arithmetic operations, including multiplication) apply to data with a large range. The top level architecture presents many local variables with wide range of data. In this situation the generation of the BDD representation is unmanageable, because of the sizing problem. So, the module cannot be verified, unless the range of the data is artificially reduced. With this limitation, many properties of the data path can be verified. From the users' point of view, this step is critical: first because it implies a manual modification of the specification, so increasing the possibility of introducing errors; second, the specification with decreased values of the range is almost often not equivalent to the initial one. So, one looses the soundness claimed by the proof. In the telecom applications paths are always present, so the best technique is to apply partitioning and verify the control part alone, where most of the behavior complexity of the application is involved. When data path also must be verified, ad hoc techniques must be defined on the base of heuristics and design practice.

VI. Conclusions

We have presented in this paper some guidelines for good practice in the application of property checking at the behavioral level for complex VHDL specifications. This research was motivated by the need to enhance the quality assessment of functional verification required by manufacturers. The results are positive, considering the possibility to include the tool presented in the paper into proprietary design flows and ESDA frameworks. Even if very promising, the technique must be improved to cope with the growing complexity of designs. Partitioning into data path and control modules is not enough: new users driven strategies must be thought and formalized to reduce the complexity and allow the performing of the computations in a reasonable time. These issues will be the core of the next step of this research.

References

[1] L. Claesen, M. Genoe, E. Verlind, F. Proesmans, H. De Man, "SFG-Tracing: a methodology of design for Verifiability", *Proc. Adv. Res. Workshop on Correct Hardware Design Methodologies*, Turin, 1991.

[2] AHL: Lambda Reference Manual Version 4.1, London (1992)

[3] G. Gorla, M. Bombana, "A different approach to ipr sharing", *Proc. of 1996 OMI Conference*, Berlin Sept., 1996.

[4] R. Schlör, W. Damm, "Specification and verification of system-level hardware designs using timing diagrams", *EDAC '93: IEEE European Conference on Design Automation*, 1993.

[5] S. Olcoz, J. M. Colom, "Toward a Formal semantics of IEEE Std. VHDL 1076", *EURO-DAC '93: European Design Automation Conference*, Hamburg 1993.

[6] C. Bolchini, M. Bombana, P. Cavalloro, C. Costi, F. Fummi, G. Zaza, "A design methodology for the correct specification of VLSI systems", *Euromicro '93*, Barcelona, 1993.

[7] T. Robles Valladares, A. Marén López, C. Delgado Kloos, T. de Miguel Moro, G. Rabay Filho, "Automatic Hardware Implementation of Formal Specifications", *III Jornadas de Concurrencia*, Gandéa, 1993.

[8] CLSI Solutions, VFormal, 1993.

[9] *LEDA - VHDL System*, Meylan 1993.

[10] J. Bormann, J. Lohse, M. Payer, G. Venzl, "Model Checking in Industrial Hardware Design", *DAC '95: ACM/IEEE Design Automation Conference*, San Francisco 1995.

[11] W. Grass, M. Mutz, W.D. Tiedemann, "High Level Synthesis Based on Formal Methods", *Euromicro '94*, 1994.

[12] K. L. McMillan, "Fitting Formal methods into the Design Cycle", *DAC '94: ACM/IEEE Design Automation Conference*, San Diego 1994.

[13] R. B. Hughes, "Design-flow graph partitioning for formal hardware/software codesign", *chapter in HW/SW Codesign*, IEEE Press 1994.

[14] To appear in *The FORMAT Approach to correct Hardware Design*, C.D. Kloos and W. Damm (Editors), Ed. Springer-Verlang, in press.

[15] K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[16] E. M. Clarke, E.A. Emerson, "Characterizing properties of parallel programs as fixpoints", *Seventh International Colloquium on Automata, Languages, and Programming*, vol. 85 of LNCS, 1981.