Design–Flow and Synthesis for ASICs: a case study(*)

Massimo Bombana (*), Patrizia Cavalloro (*), Salvatore Conigliaro (*),
Roger B. Hughes (**), Gerry Musgrave (***), Giuseppe Zaza (*)
(*) ITALTEL SIT – 20019 Settimo Milanese (MI) –ITALY
(**) Abstract Hardware Limited – Uxbridge, Middlesex, UK
(***) Dept. Elec. Eng., Brunel University, Uxbridge, Middlesex, UK

Abstract – The growing complexity of devices to be designed and manufactured, and the need to reduce the time–to–market, stress the importance of sound design methodologies. In this framework formal synthesis has the advantage of increasing the quality both of the design process and of the realized devices. The problem of relating the different abstraction levels involved in the extended design process is solved through the use of logic synthesis tools. The evaluation of the design constraints, characterizing optimal implementations such as area and timing, provide the most pragmatic approach to identify efficient guidelines applicable in the abstract phases of the design flow. The resulting design methodology combining both formal and more traditional design tools has been tested on a complex device in the area of telecommunications.

I. INTRODUCTION

What are the problems vis a vis existing design flows and methods? Simulation cannot possibly provide exhaustive coverage, the test vectors used are always, of necessity, a sub–set of the exhaustive aims. There are problems in knowing if the appropriate test–sets really do cover the anticipated behavior and use of the system being designed. There remain problems such as bus contention, deriving scheduling information, the re–use of previous designs and redesign which make the engineer's task more difficult. Current approaches do not always follow the ideal of starting with a specification and there is certainly no mathematical rigour involved in the intermediary stages which lead to an implementation. The traditional design flow can be viewed by the following figure 1.

In order to validate the design process, simulation is used to show conformity of the underlying stages of the design with those higher up in the design hierarchy. The designer is restricted by the usual problems caused by ever more lengthy simulation runs which can only provide a validation, i.e. a sanity check on the design; such simulation does not and cannot provide verification.

32nd ACM/IEEE Design Automation Conference ®

What is needed is an approach which also provides the designer with the possibility to conduct an exploratory activity in the early design phases, quick and automatic generation of mathematically correct–by–construction qualitative design alternatives ([1],[2],[3]), the generation of explicit cost/benefit parameters to support the designer in the choice between alternatives and the presence of a commercial framework able to encapsulate the tools applied at different levels of abstraction and in different phases of the design process so as to support a unified design flow ([4],[5]). To provide a winning strategy, the next generation of design environments should guarantee satisfying these requirements.

In this paper we describe an innovative design methodology, viz. a formal approach, which is able to provide such capabilities (figure 2). Our goal is to derive a critical assessment on benefits and drawbacks provided by the use of such a design flow from the users' point of view.

The key aspects of this approach are outlined in the following sections, but in essence rely on a rigorous initial specification which allows the implementation to be explored

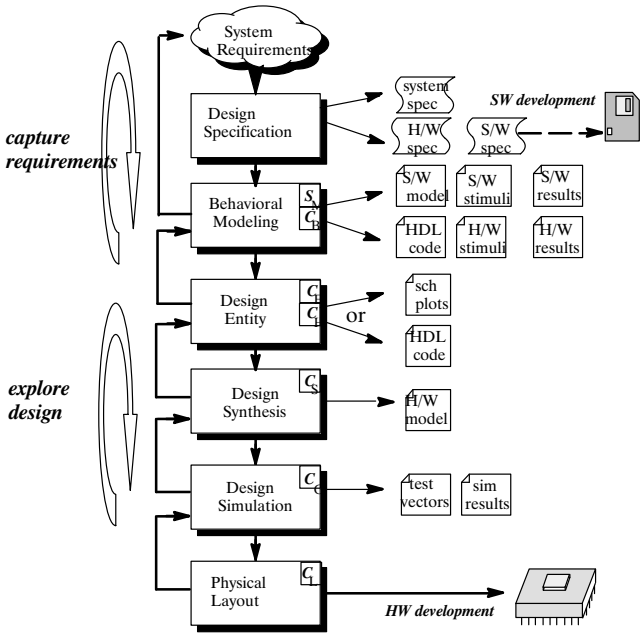

Figure 1. The conventional design flow.

(*) This research activity is part of PATRICIA (ESPRIT II n. 5020) and FORMAT (ESPRIT III n.6128).

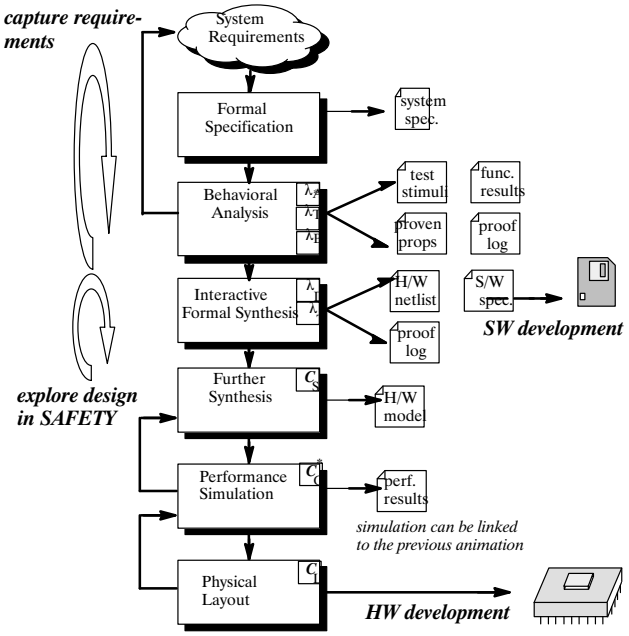

Figure 2. The Lambda design flow.

at any abstracted level to gain: insight into partitioning, design exploration etc. but at an early stage in the design flow, enabling the designer to make key decisions at the right time. In addition, the implementation alternatives are mathematically proven to satisfy the original specification as a result of the design process. These facts result in a significant reduction in the time–to–market parameter.

II. The Real Environment

Re–design steps, caused by errors introduced in the various phases of the design activity, are in many cases responsible for delays in the production cycle [6]. The introduction of sound design methodologies ([7],[8],[9],[10],[11]) can substantially improve the quality of the design process and the reliability of the manufactured devices. The synergies existing between the exploitation of formal reasoning in the abstract phases of design and of logic synthesis in the more implementation oriented steps of the design flow are worth exploring in view of the improvement of the quality of the design cycle.

Formal synthesis tools operate at very abstract levels, generating and validating a logic partitioning of the global specification of the device and producing a netlist involving functional macro–blocks. Many implementation aspects are on purpose left unspecified, because this allows a more efficient handling of the general characteristics of the behavioral description. The definition of lower level details and constraints is postponed to the subsequent phases of the design process. On the other side logic synthesis tools operate from descriptions represented using a specific subset of VHDL, the de–facto standard hardware description language ([12],[13]), in which all the implementation details have been correctly specified. The application of both techniques require the filling of the substantial gap existing between these different levels of abstraction.

In a top–down approach to design, the instantiation mechanism corresponds to design transformations from the more abstract design levels to the implementation oriented descriptions of the device. This process is supported by complex automatic or man–controlled transformational modules (figure 3) in the design environment. Such interfaces access a library of blocks, described in VHDL [14], apt for logic synthesis. The parametric components of this library are aligned with the cells formally specified and applied in the high–level synthesis process. In this way the initial high–level description is transformed into a structural/behavioral one of lower–level. The instantiation process involves various phases.

The first step takes care of the environment definition and the interface instantiation of the entities. Moreover the range and direction of the I/O ports of the components are declared. The second step identifies the standard library blocks in the netlist. For each of them the corresponding VHDL description is derived through the instantiation of the parametric elements. Finally user–defined modules, for which a behavioral description is not pre–defined, are identified. In this case the designer directly provides the VHDL description of the behavior.

The final VHDL representation is simulated to complement the animation of the initial specifications. This phase verifies that all the added implementation details have not changed the global behavior of the device. After a satisfying simulation phase, the VHDL structural/behavioral description is fed to the logic synthesis phase producing a low level netlist. A technology mapping and optimization phase for area or timing concludes this process producing an object oriented data base. From this point on the consolidated design flow is applied guiding the designer in the physical implementation of the device.

Following this path, the exploration of the design space, in order to identify the most efficient architectural implementation, is guaranteed. The integration of these tools into an industrial design environment (section III) supports the generation of low level implementations of the architectural alternatives. Comparisons based on implementation parameters (area dimension and timing efficiency) that constitute the final constraints of designs are produced.

The unified aspects of the design flow are managed through CAD frameworks ([15],[16]). These are environments explicitly developed to allow the integration of different–vendor tools into customized and specialized user environments.

III. Formal Synthesis: Application And Benefits

Formal methods have been widely applied in the *post–design* verification ([17],[18],[19],[20],[21]) of the correctness of the implementations versus the initial specification. The difference between this technique and the traditional simulation phase lies in the fact that in the latter a special set of test signals must be identified and the final result

is dependent on this choice (non–exhaustive validation), while in the former all the cases are covered (exhaustive validation). Formal validation can guarantee the logical equivalence of the two representations. Simulation analyzes details much closer to the physical functionality of the device.

A complementary approach consists in applying formally based synthesis tools (*transformational* approach). Due to the abstraction of the involved formalisms, the initial phases of the design are the most appropriate for the application of this methodology. The LAMDBA system [22] is a commercial tool that already satisfies most of the requirements of the users. Being an open system, it looks promising for adding the extra features required by the market in the future. The associated DIALOG interface, equipped with a window for schematic editing, is quite similar to what the user already applies for schematic entry. This feature increases users' acceptance, following the rule of incremental introduction of innovation into the designers' community. The interface supports the designer in the process of transforming step–wise the initial specification into a constrained implementation. The specification language (ML [22]) owns the necessary expressiveness to operate at the required abstraction levels. ML is a functional language provided of a specific semantic and an enriched syntax most apt to the application of tactics on which the tool is built. An ML animator is used to verify the correctness of the specification before starting the implementation phase.

The LAMBDA output interface is based on the use of standard description languages including EDIF and VHDL.

A. Formal specification and synthesis of a complex device

In LAMBDA/DIALOG a mixed design strategy is applied. Top–down and bottom–up approaches ([3],[23]) are applied,

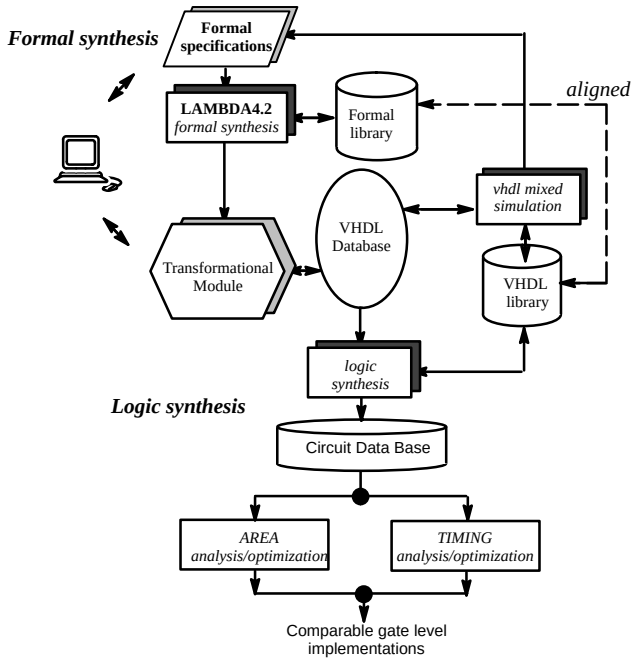

Figure 3. Tools and interfaces in the applied design flow.

at the designer's choice, and enhance usability. A formal synthesis process starts with the formalization of the specification of the device. This phase involves:

- the definition of the external interface for the global device;
- the partitioning of the global specification in an architectural description of high level modules;
- the definition of formal specifications for the high level modules.

A DSP–like complex ASIC, typical of the telecom application area, has been chosen to test the design flow going from high–level formal based synthesis down to the physical level. The device implements an arithmetic co–processor to control the incoming rate of Asynchronous Transfer Mode (ATM) connections. It applies a specific algorithm and is part of a board including other ASICs, memories, an internal bus and I/O interfaces. The device was completely specified and formally synthesized. Sub–modules of this ASIC are used in the paper as working test–benches to support with practical examples and figures the proposed approach.

B. The top level analysis of the device

The goal of the initial design phase is to define the global behavior of the component through the definition of its external interface and the identification of the relationship with the environment. In many cases the global behavior of the device is usually not consolidated at this stage, and various details are subject to changes to accommodate requirements from the board on which the device will be placed. The specification of other components of the board can also impact on the device under design. As a consequence, the specification must be abstract and flexible enough to cope with progressive refinements required by the environment. Moreover the specification has to be modified in order to accommodate a more efficient implementation in hardware.

When an algorithmic description of the behavior of the device is available, the definition of the interface between the external environment and the module involves the definition of control signals and data busses. A first logic partitioning of the module identifies a control/computational part and memories. The formalization of the overall control flow of the behavior of the component involves the identification of macro functionalities. The constraints relative to the parallel/sequential execution are expressed involving internal control signals and delay elements. Data flow is naturally described in ML, but control flow is not, due to the functional nature of the language. This limitation imposes the definition of a partitioning strategy in blocks and the description of the control flow in terms of communication/activation protocols. From this description the specification in ML can be efficiently derived. This solution constraints the theoretical potentiality of generating formal alternative architectures because the initial partitioning phase must always be pre–defined by the designer.

The macro functionalities belonging to this top–level netlist can be classified in three groups: memory elements and registers, control blocks, usually represented as Finite State

Machines (FSMs), and computational blocks. The specification strategy is different from case to case. Registers and FSMs are characterized by recursive functions representing the states and by a set of functions representing the combinational parts. Recursive functions update the values of the internal states and of a subset of the output signals from previous states and a subset of the input signals. On the contrary, a more traditional functional description is applied to describe the computational modules.

The ML code for the top level specification consists in a term, called 'abbreviation', representing the chosen partitioning, the external interface and the internal signals (30 lines of code). After partitioning, when the specification of the macro functionalities and associated blocks are considered, the specification amounts to 700 lines of code, including almost 150 ML functions, of which 25 are recursive. Four datatypes are defined and 4 different types of registers are introduced.

This specification differs considerably from those written for logic synthesis or simulation. In fact hardware description languages, such as VHDL, maintain the imperative approach, and their semantics ([24],[25]) are defined in terms of simulation, so stressing considerably the timing properties. In addition to this, hardware description languages utilize models (signals, timing properties, asynchronous processes, etc.) that have been introduced in order to express more naturally the behaviors of hardware components.

On the other side the advantages involved in the use of formal–based tools in the first phases of the design flow are quite evident. Approaches based on interfacing VHDL to ML have been considered to solve the problem of having simultaneously a visible user–friendly and hardware oriented language for designers and a good internal logic representation language for formal manipulation. In a recent approach [26] the full VHDL language has been considered with only a few restrictions. The result of the translation phase is not only an ML program but also a file that contains rules allowing the automatic simplification in LAMBDA of the generated description. An alternative approach consists in introducing specific graphical tools (i.e. timing diagrams editors, spec charts editors, etc.) that simplify the designer's specification effort and in linking them to a formal based environment which allows to check the correctness of their definition ([27],[28], [29]).

C. Alternative, formally derived implementations of the device

In the formal synthesis phase the designer proceeds to derive a hierarchical implementation of the device starting from the previously described specification. The top–down approach is mainly used during the first steps of the synthesis. Functions included in the specification are instantiated as Black Boxes, in order to identify and introduce different hierarchic levels. The bottom–up approach is useful to assemble and merge pre–existing modules in order to derive different partitioning solutions to improve area overhead and timing properties.

The results of two alternative formal synthesis processes applied to the ASIC described in section III,A have been compared. The two implementations are characterized by the fact that the components have been differently assembled into user–defined blocks. Both cases represent hierarchic descriptions of the device with four levels of depth.

One case resulted more compact in terms of the number of elementary cells included in the final netlist (236 elementary cells vs 330). This effect is due to a refinement in the specification style for control blocks and finite state machines. This result highlights the relevant role of the designer's choices in determining the global area overhead for the implementation of the device. Even if the number of cells is not directly related to the number of equivalent gates appearing in the final implementation (further optimization procedures can reduce dramatically this figure), it gives anyway an idea of the quality of the synthesized device. Alternative partitioning schema, obtained using the tactics for merging/assembling, can help in exploring the design space in order to reach different implementations.

IV. Design Constraints And Feedbacks To The Specification Phase

The activity of the lower level design phase starts from the RTL description provided by LAMBDA in terms of a set of VHDL entity declarations and structural architectures. After instantiating the necessary parameters, using an interactive interface prompting the designer for the missing information (i.e. range of busses, and so on), a gate level implementation of the device is generated through logic synthesis. At this stage figures of merit are available to quantify the compliance of the obtained implementation to the environmental constraints. In this way qualitative back–annotations are defined going from the lower levels towards the upper and more abstract levels of the design flow. The final goal is to derive an automatic implementation through synthesis that is comparable, in terms of physical constraints, with the one optimized 'by hand' by the designer (in the following referenced as Imp_0), characterized by an area of 8192 equivalent gates and a clock period of 51.44 nsec.

Only a qualitative comparison of these figures is possible because of the different implementation technologies (CMOS 1.2 μm and 0.8 μm "Sea of Gates") and the different number of accesses to RAM for a read/write operation on a 64 bits word (4 words of 16 bits are read/write one after the other).

A. Alternative implementations exploring the design environment

The first synthesis in LAMBDA (L_imp1) refers to an architecture of the device including instances of all the three different classes of modules considered in the previous section: control blocks (finite state machines), memory elements (registers and register files) and computational blocks operating in synchronous mode. The modules are here constrained by a start/end communication protocol for mutual activation/deactivation. The specifications of the FSM blocks follow a classical style (different actions are performed according to the value of an internal state and the input patterns). Their implementations are characterized by a

register for storing the internal states and combinational blocks to implement the evaluation part. The specification of the computational block follows a functional approach, implementing a hierarchy of functions at different levels of detail. The synthesis of this module follows the schema of instantiating different blocks for every different functional level and identifying basic blocks for each functional operator. Specific delay properties are associated to each operational block, representing the computational timing effort of the function. These delay properties are relevant because they are propagated upwards till the top–level, and generate the global timing property of the device.

	Registers	Counter	FSMs	Comput. module	ASIC tot.
L_imp1	5212	527	527	26560	33023
Imp1opt.	5052	513	418	26362	32641
L_imp2	4097	212	394	4712	9705
Imp2opt.	4095	210	282	4697	9571

Table1. Synthesis results in eq. gates for the main modules of the ASIC for alternative implementations.

Some optimization steps have been subsequently applied by the logic synthesis tool in order to minimize the area and the timing. Area is expressed by the number of resulting equivalent gates of the final implementation. The timing property is expressed by the clock phase generated, i.e. the maximum delay associated to all the combinational sub–nets included between sequential elements. The first row in Table 1 contains the number of equivalent gates resulting from the first mapping of the circuit L_imp1 in the chosen technology (H4CP3 MOTOROLA – CMOS 0.8 μm ”Sea of gates”) subdivided by modules. The second row contains the values of the same parameters after a first area optimization phase. The timing value corresponding to this implementation was equal to 102 nsec.

The first remark concerns the value of the resulting area (about 4 times larger than that of Imp_0). The area optimization process of the synthesis tool is not able to reduce significantly this parameter. The timing parameter (clock phase) is also not satisfactory, even if a significant improvement was obtained in the optimization on the critical block (computational module) from 102 to 67 nsec. On the contrary the two control blocks based on FSM representations are efficiently implemented, and the registers and memory blocks (including the internal counter) don’t represent a bottleneck neither from the area point of view (16.8% of the complete device) nor from the timing aspects. The computation block is the most critical. Its dimension (80% of the total area) is not justified by the complexity of the algorithm. The first sub–module, that computes an addition and a comparison, requires an area comparable with that of the RegFile. The second sub–module, where two subtractions, one multiplication and one complex shift operation are implemented, requires the 43% of the global are. The additional logic included at the top level of the module uses the rest of the total area (30%).

A closer analysis of the top level of the computational module shows the presence of delay components, introduced during the formal synthesis phase for synchronization purposes. The assumption is that they balance the delay property associated to every computational block of the netlist allowing data to reach synchronously the following block. These components are logic synthesized as arrays of latches (in case of delay equal to one clock step) and arrays of FIFOs (in case of delay greater than one clock step). The delay property associated to the operational blocks were based on the computational complexity of the functionality and not on realistic implementation aspects. As a consequence, the resulting memory elements introduced can effectively generate the observed area explosion (30% of total).

To remove this arbitrary overhead, synchronization components are to be minimized, i.e. the formal synthesis of computational modules must be performed reducing as much as possible the number of delay components in the netlist. To do so, no delay properties are associated to the computational blocks belonging to the lower levels. In this way a reduced number of synchronization components is added. This action is legal because it has no impact of the semantic of the specification. Further improvements can be obtained optimizing the VHDL functional library. The behavior of the library elements must be written avoiding as much as possible the use of statements that involve a complex hardware implementation after the logic synthesis phase. Applying these guidelines, L_imp2 was obtained (final two raws of Table 1). The area is considerably reduced. The timing value on the contrary increased up to 152 nsec.

B. Results and figures of merit

The resulting clock period characterizing L_imp1 is much higher than that of Imp_0, so undermining the obtained solution. Further guidelines were identified with the goal of decreasing this parameter, without increasing the area. The computational module resulted the most critical one in terms of physical constraints. So the design efforts were focused mainly on this module. The alternative implementations differ mainly in two aspects: different design strategies, including varying degrees of component re–use, and different granularity for computations. Every different implementation is represented by a point in a bi–dimensional space (figure 4), whose axes are the area dimension (number of equivalent gates) and the timing value (the clock phase in nanoseconds). Implementations related to the same synthesis strategy have been connected with dashed lines. This diagram shows clearly that there exists a circumscribed area including most of the obtained implementations (dashed area in figure 4). Most of theses cases are comparable with Imp_0 even if none of them is more efficient in both area and timing. Implementation L_imp7, for instance, has an area dimension 4.54% and a clock phase 16.12% greater than Imp_0. In most cases area and timing show an opposite trend: the ”best” area implementation (L_imp5) requires a timing value greater than other

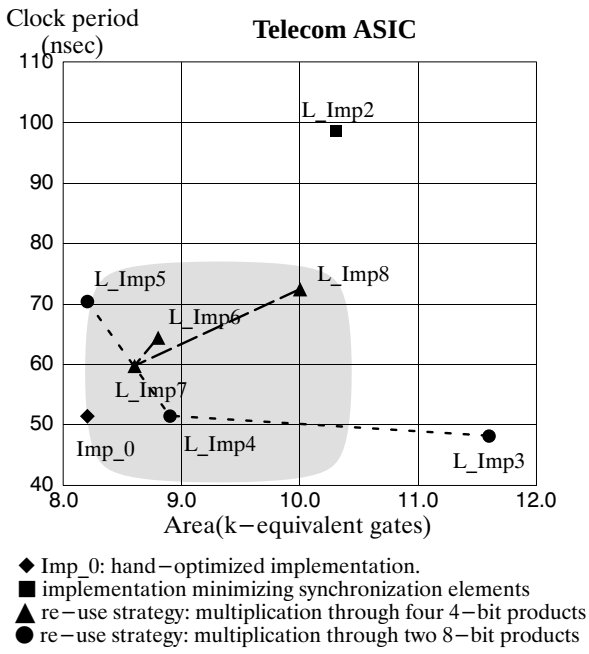

Figure 6. Comparison of the obtained implementations

implementations; and, on the other way, the best timing implementation (L_imp3) requires the largest area.

These results show that it is not possible to identify a unique synthesis strategy leading to an *absolute optimal* solution. The area and timing constraints are the milestones that allow the identification of an optimal circuit implementation in compliance with the system level design requirements. For this reason these two parameters must be kept under control even at the highest levels of specification and partitioning. The described pragmatical guidelines allow the designer to keep trace of low level constraints when operating at an abstract level so enhancing the final quality of the implementations, which is the ultimate goal of any industrial design flow.

V. Conclusions

In this paper we have described the prototype of an innovative design methodology with associated tools supporting both formal reasoning and standard design techniques in a unified design flow. This new approach can be applied to the design of complex ASICs, to increase the global quality of the produced devices coupled with a decreased time–to–market. From this research activity suggestions have been collected to make the tools more user–friendly and more accepted to the users' community.

Based on the design experience of a meaningful example, figures of merit have been provided in order to prove the advantages of quantitative evaluation means. They complement the qualitative, and somehow subjective, analysis which can be applied in the high level design phase. Design guidelines have been suggested, based on the outcome of the experimental results. Limitations of the approach are related to complexity. An engineering phase on the tools must precede the successful application of this methodology to different classes of designs of increasing complexity.

References

[1] M.Fourman, E. Mayger, 'Formally Based System Design – Interactive Hardware scheduling', G. Musgrave, U.Lauther (eds), VLSI '89, Elsevier (1989)

[2] S. Finn, M. Fourman, M. Francis, R. Harris, 'Formal System design – Interactive Synthesis based on Computer–Assisted Reasoning', Proc. IFIP WG 10.2, 10.5 Workshop, North Holland, (1990)

[3] R.B. Hughes, G. Musgrave, 'Design Flow Graph Partitioning', HOL '92, IMEC Leuven Belgium, (1992)

[4] M. Bombana, P. Cavalloro, G. Zaza, 'Specification and formal synthesis of digital circuits', Proc. IFIP TC10/WG10.2 Workshop, North Holland, (1992)

[5] 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)

[6] G. Bezzi, M. Bombana, P. Cavalloro, S. Conigliaro, G. Zaza, 'Quantitative evaluation of Formal–based Synthesis in ASIC Design', Proc. II Conference on TPCD'94, Bad Herrenalb (1994)

[7] 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)

[8] T. Robles Valladares, A. Marín López, C. Delgado Kloos, T. de Miguel Moro, G. Rabay Filho, 'Automatic Hardware Implementation of Formal Specifications', III Jornadas de Concurrencia, Gandía, (1993)

[9] W. Grass, M. Mutz, W. D. Tiedeman, 'High–level synthesis based on formal methods', EUROMICRO'94, Liverpoool (1994)

[10] T. Kropf, K. Schneider, R. Kumar, 'A formal framework for high level synthesis', Proc. II Conference on TPCD'94, Bad Herrenalb (1994)

[11] K. L. McMillan, 'Fitting Formal methods into the Design Cycle', DAC '94, San Diego (1994)

[12] D. L. Perry: VHDL, McGraw–Hill, Inc. (1991)

[13] LEDA: VHDL System, Meylan (1993)

[14] C. Costi, 'A VHDL subset definition for simulation and synthesis in the Italtel environment', ITALTEL Technical Report, (1992)

[15] M. Miserandino, 'ITL_TOOLKIT 1.0', Italtel Sit, Milano (1992)

[16] Mentor Graphics: Getting started with Falcon Framework, (1991)

[17] F. Anceau, 'Formal verification in industrial environment',Workshop on Formal Methods, L'Aquila, (1989)

[18] F. Anceau, 'Panel on formal methods in hardware design', 10th Int. Symp. on Computer Hardware Description Languages, Marseille, (1991)

[19] CLSI Solutions: VFormal, (1993)

[20] W. Damm, H. Hungar, P. Kelb, R. Schloer, 'Using graphical specification languages and simbolic modelchecking in the verification of a production cell', FZI'93, (1993)

[21] T. Filkorn P. Varkentin,'Internal representation of Transition Systems based on BDDs', V1.1 tech. rep. ZFE BT SE 11/F1, Siemens AG (1993)

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

[23] R. B. Hughes, G. Musgrave, 'Design–flow graph partitioning for formal hardware/software codesign', in Software/Hardware Codesign, ch. 10, Rozenblit and Buchenrieder (eds), IEEE Computer Society Press, (1994)

[24] S. Olcoz, J. M. Colom, 'Toward a Formal semantics of IEEE Std. VHDL 1076', EURO–DAC '93, Hamburg (1993)

[25] W. Damm, B. Josko, R. Schloer, 'A net–based semantics for VHDL', EURODAC'93 Hamburg (1993)

[26] G. Umbreit, 'Providing a VHDL interface for proof systems', EURODAC'92, Hamburg (1992)

[27] R. Schlör, W. Damm, 'Specification and verification of system–level hardware designs using timing diagrams', EDAC '93, (1993)

[28] W. D. Tiedeman, S. Lenk, C. Grobe, W. Grass, 'Introducing structure into behavioral descriptions obtained from timing diagram specifications', EUROMICRO'93, Barcelona (1993)

[29] J. Helbig, P. Kelb, 'An OBDD–representation of state charts', EDAC'94, Paris (1994)