Embedded Architectural Simulation within Behavioral Synthesis Environment

A. Jemai*, P. Kission** and A.A. Jerraya**

*RAM/ENSI Laboratory
16 rue 8010, Quartier Montplaisir
1002 Tunis Belvédère
Tunisia

**TIMA Laboratory
46 avenue Félix Viallet
38031 Grenoble Cédex
France

Abstract - This paper introduces one way to integrate an interactive simulator within a behavioral synthesis tool, thereby allowing concurrent synthesis and simulation. Such a simulator performs dynamic analysis and execution time evaluation.

This paper also discusses an implementation of this concept resulting in a simulator, called AMIS. This tool assists the designer for understanding the results of behavioral synthesis and for architecture exploration.

I Introduction

Starting from a behavioral description, the behavioral synthesis allows to produce an architecture made of a controller and a data path. The latter is generally given as an RTL description which is 5 to 10 times larger than the initial behavioral specification [1].

The first experiments using behavioral synthesis have shown that designing with behavioral synthesis is an iterative process. Starting from an initial specification, the user proceeds in several iterations. At each behavioral synthesis iteration, the hints extracted from earlier synthesis sessions are used in order to produce an architectural solution.

Such a scheme allows to produce very efficient results when using behavioral synthesis. Several works in the litterature report that behavioral synthesis may produce efficient design that can be compared well to manual design [2,3].

However this iterative scheme issues several challenges:

-1- The designer needs to understand the results of behavioral synthesis in order to be able to analyze it. This means that the behavioral synthesis tool should provide facilities that show correspondence between the architecture and the initial behavioral description.

-2- The analysis of the architecture may provide information on the use of the resources and on the execution time. Some of these informations need just static analysis, the other informations may need dynamic analysis. For example, in the case of a behavioral description that includes a data dependent loop, the execution time cannot be computed by a static analysis. Much work has been treating static analysis [3] but no work at our knowledge has treated dynamic analysis at the behavioral level within the synthesis process.

-3- Debugging the results of behavioral synthesis is very difficult and fastidious when performed on the resulting RTL description. This step is however needed because behavioral synthesis may produce unexpected architectures when the input description does not comply with the restrictions and writing style imposed by the behavioral synthesis tool.

As long as the three above-mentioned problems are not solved, behavioral synthesis will remain restricted to specialists who know very well the behavioral synthesis tool they are using.

I.1 Previous Work

Several tools published tackle one or two of the above mentioned problems. Some of them tried to solve the first problem, which is to link the behavioral description and the resulting architecture. In this field AWB [4] is a precurser, it provides an original model which allows the user to analyze and understand the decision of the behavioral compiler and thereby to explain the resulting solution.

The dynamic analysis of the behavioral description, pointed out as the second problem, can be done through a simulation of the corresponding RTL description operated by the behavioral compiler using standard simulators. Tools such as CATHEDRAL [3] and PHIDEO [5] include facilities for static analysis and are restricted to regular algorithms without data dependent computation. MIES [6] is an interesting approach to handle dynamic analysis. It is based on a micro-architectural model similar to those produced by behavioral synthesis. Unfortunately, it is not connected to most popular behavioral compilers.

Finally only few tools tackled the problem of debugging. Again, the architect workbench (AWB) is a precurser in this field [7]. ISE [8] provides an interactive environment allowing an easy interaction with the user for understanding the architecture and debugging the design.

As mentioned above, some works have tried to solve one of the three problems but none of the existing tools to our knowledge allows to solve the three above-mentioned problems.

I.2 Contribution

This paper introduces one way to solve the three problems: the integration of an interactive simulator within a behavioral synthesis tool. The simulator and the behavioral synthesis are based on the same model. This model allows to link the behavioral description and the architecture produced by synthesis. The simulation can be performed step by step. At each step the user can analyze the resources of the architecture and the correspondence with the behavioral description, thereby making specification debug easier. Of course, the simulation provides a dynamic analysis for data dependent computation. The simulator implemented is called AMIS and is integrated within AMICAL, an interactive behavioral synthesis tool based on VHDL.

ASP-DAC '97
0-89791-851-7$5.00 © 1997 IEEE

II Combining Behavioral Synthesis with Architectural Simulation

The key idea for combining behavioral synthesis and architectural simulation is to use the synthesis representation and internal data structures for simulation.

II.1 Behavioral Synthesis

Behavioral synthesis is generally organized in two major steps. The first step performs scheduling and allocation in order to produce the data path and to fix the controller. The second one performs the generation of the controller and produces the RTL description. Most of the design decisions are made during the first step. At this stage, we have enough information about the architecture to perform architectural simulation.

The scheduling and allocation steps fix the execution order of the operations of the behavioral description. All the complex operations are decomposed into basic transfers (and operator activations). The paths used to transfer data in the data path are fixed. These may be made through multiplexers, switches and buses. The intermediate model produced by this step is called an abstract architecture. This model can be used for architectural simulation. Figure 1 shows a design flow combining architectural simulation and behavioral synthesis.

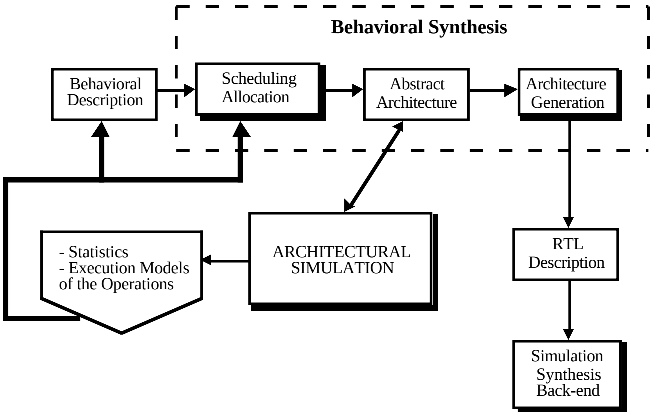

Fig. 1: Mixing Behavioral Synthesis and Architectural Simulation

The final step of the behavioral synthesis produces an RTL description that can feed simulation and synthesis tools acting at the logic level. However such a simulation would also need a long time (hours for description) for the compilation, elaboration and simulation.

II.2 Architectural Simulation

Architectural simulation is cycle based. It makes use of the abstract architecture to simulate the execution of the design at the architectural level. This simulation is performed at the clock cycle level. At each cycle both data path and controller execute one step. A controller step selects a transition and computes the next state. A data path step executes the transition selected by the controller. A transition is made of a set of elementary operations that have to be executed simultaneously in the same cycle. An operation may be a data transfer between resources or a functional unit selection. The execution of a transfer may need the activation of several components of the data path (switches, multiplexers, registers, ...).

More details on the simulation scheme will be given later (section IV). The simulator produces several results that may be used by the designer to understand and refine his solution during the iterative design process (figure 1).

The simulator produces:

- All the details about the execution of the operations of the behavioral description
- The detailed order of the execution of the operations by means of a trace of the controller execution
- The intermediate values of all the resources of the data path for each cycle
- The value of the outputs of the design at each cycle
- The operations executed by the functional units during each cycle.

In addition to this cycle based information, the simulator computes and provides dynamic statistics on the resource use. The cycle based informations are generally used to understand and debug the design. The statistics are used to analyze the architecture and to react for the modification of the synthesis script or the behavioral description (illustrated by bold arrows in figure 1). Section V will illustrate the use of architectural simulation in order to understand the design and to refine the architectures using statistics.

II.3 Combining Architectural Synthesis and Simulation

Starting with a pure behavioral or functional description, behavioral or architectural synthesis generates a cycle based description. Therefore the RTL description obtained after synthesis has an execution time different from that of the initial one. As a matter of fact, one of the major problems met when debugging RTL description produced by HLS is the definition of a new testbench or re-using the initial testbench used to behavioral description.

The validation of RTL descriptions with respect to the behavioral ones becomes a tough task when the latters include multi-cycle operations executed on complex functional units. The use of a complex functional unit that may execute several operations or that may run concurrently with respect to the rest of the design requires a model which combines its functionality with its execution scheme.

The interactive architectural simulation allows the designer to follow the data transfers and to understand the operation schedule. The combination of architectural synthesis and simulation allows the validation of the architecture with respect to synchronization with the external world, in taking into account local execution delay.

III Design Models Used by AMIS

This section introduces the different models used by AMIS. As stated above AMIS shares the concepts and the intermediate formats used by the behavioral synthesis tool AMICAL. More details about AMICAL can be found in [9], only the models needed to understand AMIS will be detailed.

III.1 Architectural Model

AMICAL is based on a flexible target architecture model composed of a top controller, a set of functional units and a communication network. These last two constitute the data path.

The architecture may include several functional units that may run in parallel. The functional units interact through the communication network which is composed of buses, multiplexers, switches and registers. With this scheme, large memorization blocks and I/O units are handled as functional units and managed by the user in the behavioral description. This architecture is general enough to represent a large class of designs. The target may be a simple ASIC or a complex application specific architecture where the top controller acts as a main processor and the functional units as co-processors.

III.2 Input Description

AMICAL starts with two kinds of inputs, a behavioral description and a functional unit library. The behavioral description is given in VHDL. Like most behavioral synthesis tools, the compilation unit of AMICAL is a single process.

The other part of the input description is a library of functional units. AMICAL does not assume a predefined library of operators. The user is requested to provide the set of functional units (or operators) needed to execute the operation of the behavioral description. An operation may be a standard VHDL operator (e.g. + ,-, etc.), a procedure call or a function call.

Each functional unit of the library is described according to several views. These different views are given for the adder-substractor (AS) in figure 2.

- Conceptual view: From the conceptual point of view, the functional unit is an object that can execute one or several operations which may share some data.
- Behavioral view: At the behavioral level, the functional unit is described through the operations it can execute. These may correspond to standard operations, procedures or functions. With respect to the conceptual view, the behavioral view defines additional informations about the data necessary for each operation as well as the ones generated by each of them.
- Implementation view: This model gives all the implementation details of the functional unit. It may be an RTL or a gate level description. This view includes all the data and control ports of the functional unit.
- Synthesis view: The high-level synthesis view of the functional unit links the behavioral and implementation views. It includes the interface of the functional unit, its operation-call parameters, the operation set executed by the functional unit as well as the parameter passing protocol for each operation.
- Simulation view: The simulation view of a functional unit is an advanced form of the behavioral view. It defines the functionality or algorithm corresponding to each functional unit. While the behavioral view defines only the different operations used in the behavioral description, the simulation view links the synthesis and behavioral views by giving an abstract description of the implementation view. In other words, it defines the behavior of the operations executed by each functional unit. Therefore the simulation view emulates the functional unit. However, during simulation it shall be combined to the synthesis view so that the timing characteristics are also taken into account.

The simulation view, illustrating the adder-substractor (AS) in figure 2, is given as an independent C program that executes the operations of the functional unit. This will be executed during the simulation process.

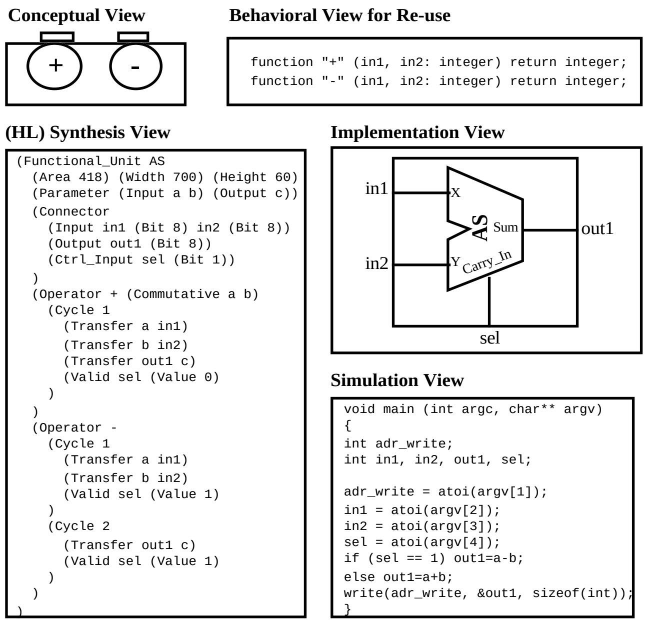

Fig. 2: View of the AS Functional Unit

III.3 Abstract Architecture

The abstract architecture is the model used during simulation. It is composed of a scheduled behavioral description and a data path. This model is the intermediate model used by AMICAL and obtained after the scheduling, the allocation and the data path generation step.

At this level an operation may hide a complex behavior and may therefore require several basic cycles (or clock cycles to execute). Each operation is decomposed into a set of elementary transfers by the synthesis process. An elementary transfer is composed of a source and a sink that may be a register, a port or a connector (input or output) of a functional module. During data path generation, a connection path (set of multiplexers, buses, switches) is associated to each elementary transfer. Of course, when several transfers have to be executed in parallel, separate connection paths should be allocated.

The FSM can be seen as the result of a two-level scheduling of the behavioral description. The first one fixes the parallelism of operation and produces an FSM where each transition is made of a macro-cycle. The second level decomposes each macro-cycle into a set of basic transitions that have to be executed in sequence. Each basic transition is a micro-cycle whose execution will take a single clock cycle.

Complex transfers may be decomposed into several basic transfers. For example, a transfer including an operation may be decomposed into several register-FU transfers (in order to feed the FU with input and to recover the outputs) and a control transfer that selects the operation that has to be executed by the functional unit.

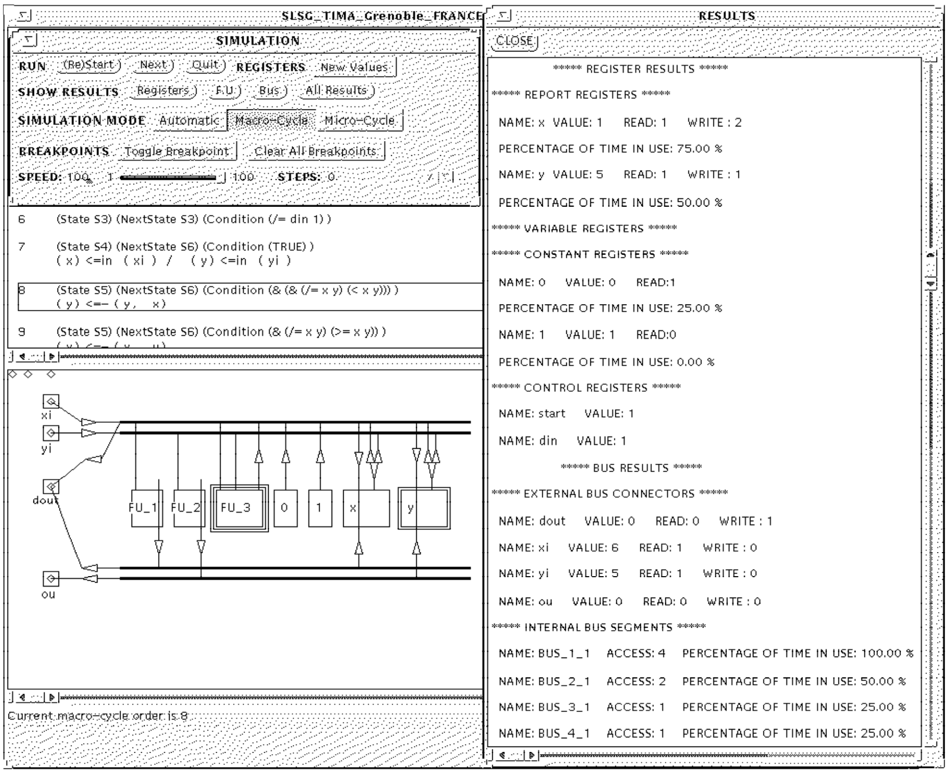

Fig. 3: Combined Simulation/Synthesis Session

IV AMIS: THE AMICAL ARCHITECTURAL SIMULATOR

AMIS is an architectural simulator embedded in AMICAL. Figure 3 shows a screen dump of a combined synthesis and simulation session.

The top left part of the screen shows the AMIS interaction window. The right window gives the corresponding architectural simulation report, while the windows underneath show the standard AMICAL results (controller and data path). The simulation and the synthesis tools are fully integrated. AMIS is invoked through a simple command from the AMICAL menu.

The rest of this section details the simulator organization and the simulation modes. The next section illustrates the use of AMIS within an interactive synthesis process.

IV.1 Simulation Scheme

As stated above, AMIS is embedded in the synthesis environment AMICAL. The simulation is made using three cooperating processors:

- A simulation engine in charge of executing the statements of the behavioral description in the right order.
- A functional unit emulator in charge of executing the simulation view of the functional units.
- An environment emulator in charge of providing the stimulus.

Figure 4 shows the AMIS organization. The simulation engine communicates with the functional unit emulator and with the environment emulator through UNIX-IPC [10].

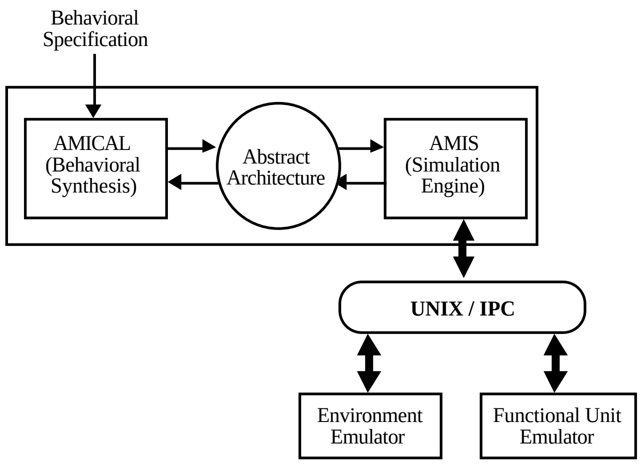

Fig. 4: AMIS Organization

IV.2 The Simulation Engine

The simulation engine performs the interpretation of the abstract architecture. The simulator is cycle based. The simulation unit is the micro-cycle. The simulation algorithm is given in figure 5.

The main steps performed by the engine are the sequencing of the macro-cycle (step 2 of the algorithm) and the transfer execution (step 3).

The present version of the simulator restricts the behavior to three kinds of data transfers and one type of control transfer:

```
1- Initialization
    Current -State:= initial state
        Set All variables to initial values
        Set All connections to undefined values

2- Select a transition (macro-cycle)
    a- Evaluate all the conditions of the outgoing
        transitions from current-state
    b- If there is one and only one true condition
        Then current-macro-cycle:= the transition
                                     with true condition
            Next-state:= successor state of the transition
        Else SPECIFICATION-ERROR

3- Execute a Macro-cycle
    For each micro-cycle in the macro-cycle
    Execute simultaneously all data and control transfers
                              (functional unit selections)

4- Current-state := next state
    Go to 2
```

Fig. 5: Simulation Algorithm

- Pure register transfer: data transfers between two registers of the data path without invocation of functional unit or I/O ports.
- Register-functional unit transfer: this kind of transfer allows to exchange data with functional units. The data is transferred from/to the emulator of functional units through system calls (UNIX-IPC) (see next section) [10].
- I/O Transfers: this kind of transfers is needed to exchange data with the environment of the simulation e.g. reading stimuli and writing results.
- Control transfer: this kind of transfer corresponds to commands from the controller to the data path. At the architectural level, only the control of functional units is concerned by this kind of transfer. It also needs communication with the functional unit emulator.

IV.3 The Functional Unit Emulator

The data path may include complex functional units executing multi-cycle operations and that may have internal states. This implies that the functional modules may have their own sequencing and that we may have several of these functional units running in parallel [11KiDJ94]. In order to cope with this concurrency AMIS makes use of a functional unit emulator. The emulator is in charge of executing the simulation view corresponding to the functional units. As explained above (Section III.2), each functional unit has a simulation view that may be given as a C program. This is made using separate UNIX processes and parameter exchanges (inputs and results) is made through a UNIX/IPC protocol [10].

IV.4 The Environment Emulator

This sub-system is in charge of providing stimulis (test vector for the simulation) and recovering the output of the circuit simulated. It acts as a testbench in a standard simulation environment. In the present version, the environment emulator is also an independent C-program executed as a separate program. The communication with the simulation engine is also made through a UNIX/IPC protocol.

V USING AMIS FOR THE ARCHITECTURE EXPLORATION

This section illustrates the use of AMIS for understanding, debugging and improving the results of behavioral synthesis. We will use the GCD example to illustrate this process. The next section will report on the design of larger examples.

V.1 Using AMIS to Understand the Results of Behavioral Synthesis

The main transformations performed by behavioral synthesis are scheduling, allocation and data path generation. In order to understand the resulting architecture the designer needs to relate it to the initial behavioral description. Without specific aid tools, the designer would have to decode the produced architecture in order to find the correspondence with the behavioral description. This is a fastidious task which can be made very easily with a tool like AMIS.

During simulation, at each step, AMIS shows the resources of the data path used for the execution of the current transition. For example, in figure 4, AMICAL framed macro-cycle 8 which is the current macro-cycle used. In the same time it highlighted the resources used to execute the operation of this macro-cycle in the data path. When parallel operations or parallel transfers are executed during a macro-cycle or a micro-cycle, the corresponding resources are highlighted using different colors. The designer can easily find which path is used to execute which transfer and which functional unit is used to execute which operation.

V.2 Using AMIS to Debug the Behavioral Specification

The architectural simulation allows to detect several kinds of specification errors that cannot be detected using behavioral simulation. Typical errors are those related to communication protocols with the external world and with functional units. In fact several operations of the behavioral description take zero-delay during behavioral simulation. After scheduling and micro-scheduling, the execution details of these operations are fixed and may introduce extra execution cycles. This may induce a change between the results of behavioral and RTL simulations. These changes may provoke errors, but they can be easily detected by architectural simulation.

During the simulation with AMIS, the users have access to all the intermediate values of the different resources. By this way, the execution can be followed micro-cycle by micro-cycle in order to detect the origin of changes in the behavior. In figure 4, the system shows the value of different registers and buses.

V.3 Using AMIS to Improve the Results of Behavioral Synthesis

AMIS provides a great deal of help in making the iterative synthesis process (introduced in section 1) easier.

Besides all the facilities provided to help the designer understand the architecture, AMIS provides statistics on the use of the resources of the architecture. These statistics are computed dynamically during simulation. Of course, the quality of these data depends on the quality of the test vectors used for architectural simulation. For example in the case of the GCD, table 1 gives the statistics corresponding to a typical testbench.

The table gives for each resource the percentage of cycles it is used. We can easily see from this table that the two I/O units and the buses 3 and 4 are used only during few cycles. In fact the solution includes 2 I/O units because the scheduling step produced a solution where several macro-cycles make use of 2 parallel I/O operations (transitions 7, 10 and 11) in figure 4.

Resource	AS (FU)	I/O (FU)	I/O 2 (FU-1)	Bus 1	Bus 2	Bus 3	Bus 4	X	Y
Frequency of Usage	95,93%	2,64%	2,64%	100%	50,61%	2,64%	2,64%	74,8%	73,17%

Table 1: Frequency of Use of the Resources

In this case, we can iterate in the design process by changing the synthesis script in order to restrict the number of I/O operations. This induces a serialization of the execution of the I/O operation in cycles 7, 10 and 11.

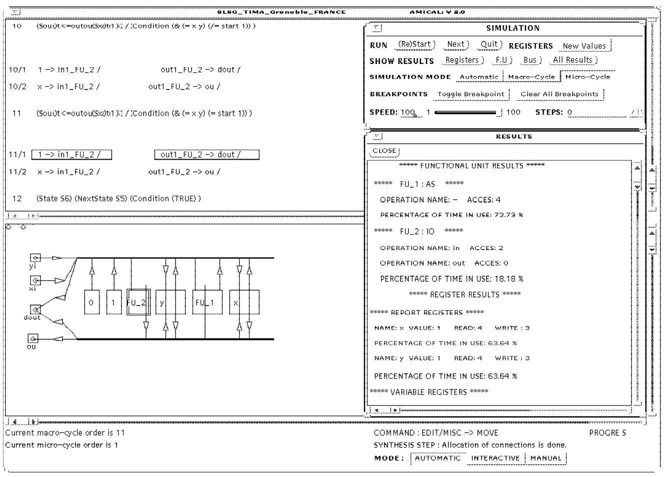

Fig. 7: New Solution after Resynthesis

Figure 7 shows a new screen where the scheduled behavioral description (top window) has no parallel I/O. In this case, transition 10 is scheduled into 2 mico cycles in order to allow the serial execution of the two operations. The bottom window shows the corresponding data path after a new synthesis session. The data path is simpler since it includes only one I/O unit and 2 buses.

In this case we obtain a smaller solution with only a small speed overhead (less then 3%) for our testbench.

CONCLUSIONS

This paper presented a solution for integrating an interactive simulator within a behavioral synthesis tool so as to help the user in the analysis of the resources of the architecture and the correspondence with the behavioral description. This scheme allows the designer to understand the architectural solution. It enables also to make specification debug easier. The simulation provides a dynamic analysis for data dependent computation. The simulator AMIS has been integrated within AMICAL, an interactive behavioral synthesis tool based on VHDL.

The architectural simulation allows to detect several kinds of specification errors that cannot be detected using behavioral simulation. Typical errors are those related to communication protocols with the external world and with functional units. In fact several operations of the behavioral description take zero-delay during behavioral simulation. After scheduling, the execution details of these operations are fixed. This may induce a change between the results of behavioral and RTL simulations. These changes may provoke errors, but they can be easily detected by architectural simulation.

REFERENCES

[1] T. Ly, D. Knapp, R. Miller, D. MacMillen "Scheduling using Behavioral Templates", Synopsys Inc., DAC, 1995.

[2] P.G. Paulin, J. Fréhel, M. Harrand, E. Berrebi, C. Liem, F. Naçabal, J.-C. Herluison, "High-Level Synthesis and Codesign Methods: An Application to a Videophone Codec", EuroDAC/EuroVHDL, 1995.

[3] S. Note, W. Geurts, F. Catthoor, H. De Man, "Cathedral-III: Architecture-Driven High-level Synthesis for High Throughput DSP Applications", 28th ACM/IEEE Design Automation Conference, 1991.

[4] D.E. Thomas, E.M. Dirkes, R.A. Walker, J.V. Rajan, J.A. Nestor, R.L. Blackburn, "The System Architect's Workbench", 25th ACM/IEEE Design Automation Conference, 1988.

[5] P.E.R. Lippens, J.L. van Meerbergen, A. van der Werf, W.F.J. Verhaegh et al, "PHIDEO, A Silicon Compiler for High Speed Algorithms", European Conference on Design Automation, 1991.

[6] J. Nestor, B. Soudan, Z. Mayet, “MIES: A Micro-Architecture Design Tool”, 22nd International Workshop on Microprogramming and Micro-Architecture, 1989.

[7] D.E. Thomas, E.D. Lagnese, R.A. Walker, J.A. Nestor, J.V. Rajan, B.L. Blackburn, "Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench", Kluwer Academic Publishers, 1990.

[8] V. Chaiyakul, D.D. Gajski, "Assignment Decision Diagram for High-Level Synthesis", Technical Report #92-103, Department of Information and Computer Science, University of California, Irvine, December 1992.

[9] A. A. Jerraya, H. Ding, P. Kission, M. Rahmouni, “Behavioral Synthesis and Component RE-USE with VHDL”, Kluwer Academic Publishers, Boston/London/Doredrecht, 1996.

[10] S.L. Coumeri, D.E. Thomas, "A Simulation Environment for Hardware-Software Codesign", International Conference on Computer Design, 1995.

[11] P. Kission, H. Ding, A.A. Jerraya, "Structured Design Methodology for High Level Design", 31st ACM/IEEE Design Automation Conference, 1994.