Architectural Simulation in the Context of Behavioral Synthesis

A. Jemai: INSAT, Tunis, Tunisia,
P. Kission, ANACAD, Grenoble France
A.A. Jerraya, TIMA Laboratory, Grenoble, France

Abstract

This paper deals with integrating an interactive simulator within a behavioral synthesis tool, thereby allowing concurrent synthesis and simulation. The resulting environment provides a cycle based simulation of a behavioral module under synthesis. The simulator and the behavioral synthesis are based on a single model that allows to link the behavioral description and the architecture produced by synthesis. The basic simulation-synthesis model is extended in order to allow for concurrent architectural simulation of several modules under synthesis.

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. It may also be used to debug the behavioral specification.

I. Introduction

Starting from behavioral description, behavioral synthesis allows to produce an architecture made of a controller and a data path. The later is generally given as an RTL description which is 5 to 10 times larger than the initial behavioral description [PFHB95]. A typical HLS flow is depicted in figure 1.

The first experiences using behavioral synthesis [Berr96] have shown that designing with behavioral synthesis is an iterative process where starting from an initial specification, the user proceeds in several iterations. At each iteration behavioral synthesis is used in order to produce an architectural solution. If the obtained solution is not "good enough", another behavioral synthesis iteration may be needed. In order to obtain a better solution the designer can refine the initial behavioral description or the synthesis script according to the hints extracted from earlier synthesis sessions.

On the other hand, experiences with complex system design have shown the need to allow for modular approaches. A complex system is seldom described as a single module. Even if behavioral synthesis is still restricted to a single process, the overall design methodology need to handle multi-module architecture.

It is then clear that combining modular design with iterative behavioral synthesis is needed for the design of complex systems [PFHB95]. However this combination issues several challenges:

-1- Analyzing the interaction between several modules under synthesis may be a nightmare for designers. Behavioral synthesis may introduces extra cycles that affects the inter-modules communication timing. When several modules needs to be synthesized, specific debug tools are required in order to check inter-module interaction after synthesis.

-2- Debugging the results of behavioral synthesis is fastidious and inefficient when performed on the resulting RTL description. First RTL simulation may take hours for the compilation, elaboration and simulation. Second, the RTL model may be very difficult to read because organization of the RTL model is generally very different from the initial behavioral description.. Some existing behavioral synthesis tools reduce the time needed for this step by producing an elaborated RTL description [Syno95].

-3- The analysis of the architecture may provide statistic information on the use of the resources and on the execution time. Some of these information need just static analysis. However in the case of a behavioral description that includes a data dependent loop, the execution time cannot be computed by a static analysis. In this case a simulation is needed to find the typical execution time according to the test vectors used.

As long as the three above-mentioned problems are not solved, behavioral synthesis will remain isolated and difficult to include in an overall design methodology.

1. Previous Work

Several tools published tackle one or two of the above mentioned problems. Multi-module debug and validation is handled quite well using simulation and co-simulation[PFHB95]. However none of the existing tools, to our knowledge, allows for concurrent debug of

several modules under synthesis.

Only few tools tackled the problem of debugging. Some of them tried to link the behavioral description and the resulting architecture. In this field AWB [TDWR88, TLWN90] is a precursor, it provides an original model allowing such links. The model allows the user to analyze these links and to understand the decision of the behavioral compiler and thereby to explain the resulting solution. ISE [ChaG92, Gajs96] provides also 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.

The dynamic analysis of the behavioral description, pointed out as the third problem, can be done through a simulation of the corresponding RTL description operated by the behavioral compiler using standard simulators. However this implies that the designers have to complete the behavioral synthesis before starting the analysis. Some behavioral synthesis tools include statistic analysis. CATHEDRAL [NGCD91] and PHIDEO [LMWV91] for example include such facilities. However, these tools are restricted to regular algorithms without data dependent computation. MIES [NeSM89] is an interesting approach to architecture simulation, 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.

2. Contribution

In a previous paper [JKJ97] we introduced an architectural simulator called AMIS which is integrated within AMICAL, an interactive behavioral synthesis This paper presents an extension of AMIS to handle distributed architectural simulation within behavioral synthesis. The goal is to allow for concurrent debug and performance analysis of multi-modules descriptions.

II. Synthesis and Simulation

The key idea for combining behavioral synthesis and architectural simulation is to use the representation and internal data structures of behavioral synthesis for simulation. . Figure 1 shows a design flow combining architectural simulation and behavioral synthesis.

1. Behavioral Synthesis

Behavioral synthesis allows to produce an RTL description starting from a behavioral specification. The RTL model is composed of a data path and a controller. 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.

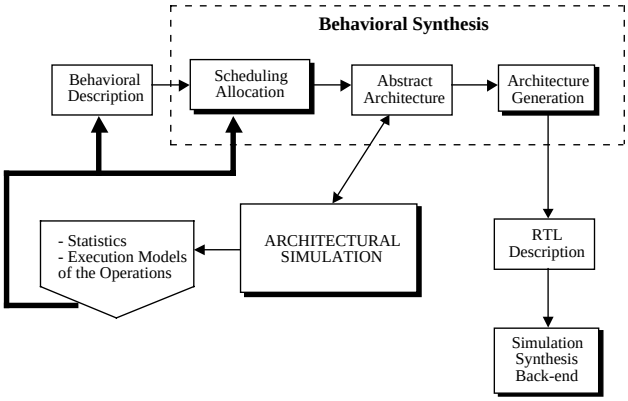

Fig. 1: Mixing Synthesis and Simulation

After scheduling and allocation we know how the behavioral description will be realized by the architecture. All the complex operations are decomposed into basic transfers (and operator activation). The paths used to transfer data in the data path are also 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

In this paper we will use AMICAL for behavioral synthesis. AMICAL is an interactive synthesis tool that starts from behavioral VHDL models and produces architectures also described in VHDL. The rest of this section introduces the internal models used by AMICAL. For more details about this system see [JDKR96].

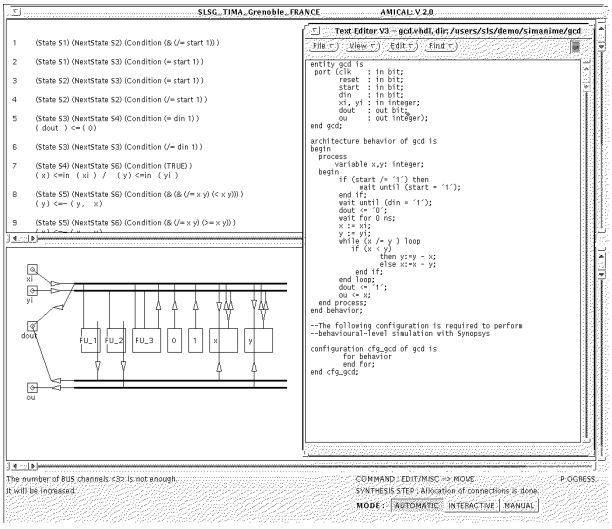

Fig. 2: Abstract Architecture of GCD

Figure 2 shows a screen dump of AMICAL during behavioral synthesis. This screen shows the intermediate models used by AMICAL. The example under synthesis is the GCD[Benc89]. The top left part of the window shows a portion of the scheduled behavioral description. This is organized as an FSM and printed as a transition table. Each transition is defined by two lines. The first gives the transition number, the present state, the condition and the next state. The second line shows the operations that have to be executed when the transition is selected.

The bottom window shows the corresponding data path. In this case the synthesis produces a bus based solution. The synthesis was made with an option that introduces I/O units for communication with the external world using a specific protocol.

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.

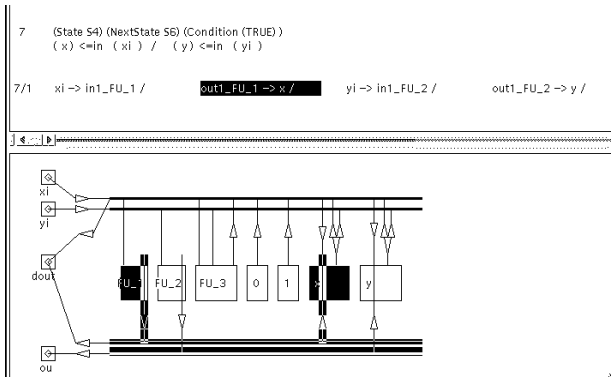

Fig. 3: Decomposition of Operations into Transfers and Transfer Execution Paths

Figure 3 shows these two decomposition levels using transition 7 from figure 2. This transition uses I/O operations that have to be executed in parallel. These operations are bound to two I/O units that need two transfers to execute each I/O operation. The top part of figure 3 shows the transition and the decomposition of the operations into transfers. The resulting transfers are organized into micro-cycles. Each micro-cycle is composed of a set of operations that have to be executed in parallel. Then each transition is decomposed into a set of micro-cycles according to the operations of the transition. In the following we will call the transition: macro-cycle. In fact 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.

Figure 3 also shows the correspondence between an elementary transfer and the connection path used to execute the transfer. Both the transfer in the top window and the connection path in the bottom one are highlighted. Besides data transfers, a macro-cycle may include several control transfers; these correspond to the activation of the data path components, e.g. functional unit selection and to I/O operations of the controller.

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 to recover the outputs) and a control transfer that selects the operation that has to be executed by the functional unit.

2. Architectural Simulation

Architectural simulation makes use of the abstract architecture (internal model of behavioral synthesis) 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, ...).

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 information related to cycle based execution of the behavioral model (usage of buses and operators, intermediate values of wires, I/O and registers).

In addition to this cycle based information, the simulator computes and provides dynamic statistics on the resource use (e.g. frequency of use of the buses). The cycle based information 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). More information about architectural simulation may be found in [JKJ97].

3. Combining Architectural Synthesis and Simulation

The key idea of this work is to use the internal model used during synthesis in order to perform an architectural simulation. In fact the abstract architecture used by behavioral synthesis provides all the information needed to perform a cycle based simulation. This way the simulation may be used to evaluate and validate the architecture before the generation of the RTL description. The main benefit of this scheme is to reduce the amount of RTL simulation. In fact architectural simulation may be used to validate the RTL descriptions with respect to the behavioral ones. Such a validation is needed because behavioral synthesis may introduce extra cycles that modify the behavior of the system under synthesis. The interactive architectural simulation allows the designer to follow the data transfers and to understand the operation schedule.

III. AMIS: The AMICAL Architectural Simulator

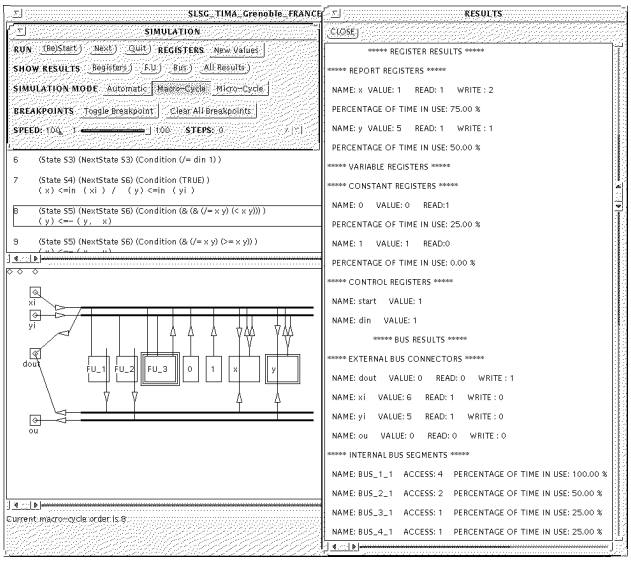

Fig. 4: Combined Simulation/Synthesis Session

AMIS is an architectural simulator embedded in AMICAL. The rest of this section outlines the simulator organization and the simulation modes. More details on AMIS can be found in [JKJ97].

Figure 4 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. The simulation and the synthesis tools are fully integrated. AMIS is invoked through a simple command from the AMICAL menu.

The simulation is made of 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. This engine is needed in case the data path includes functional units able to execute multi-cycle operations.
- An environment emulator in charge of providing the stimulus.

Figure 5 shows the AMIS organization. The simulation engine communicates with the functional unit emulator and with the environment emulator through UNIX-IPC [CouT95].

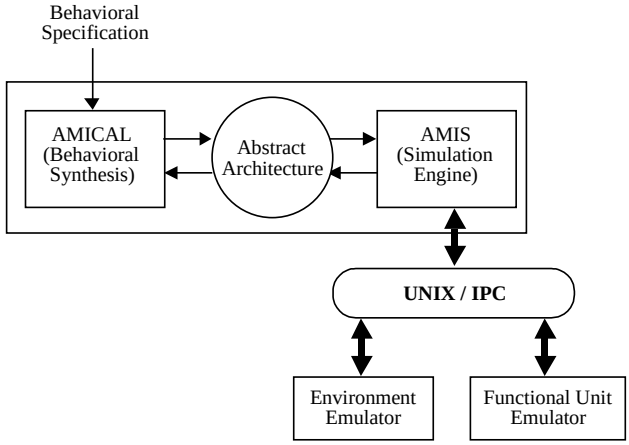

Fig. 5: AMIS Organization

The environment emulator is in charge of providing stimuli (test vector for the simulation) and recovering the output of the circuit simulated. It acts as a test program in standard simulation environment. IV.5 Interaction with AMIS

The functionality and user interface of AMIS are tailored to work in cooperation with a high level synthesis. The simulation can be performed in automatic mode or in a step by step mode. The granularity of a step is fixed by the user, it may be a macro-cycle, a micro-cycle or a simple transfer. At each step the operation executed and the resources used are highlighted in the synthesis screen. During a simulation session the user has a continuous access to the values of the register and the buses. In addition the simulation provides statistics on the use of the different resources.

IV. Using AMIS for single module design

This section illustrates the use of AMIS for debugging and dynamic analysis of the results of behavioral synthesis. We will use the GCD example introduced earlier to illustrate this process. The next section will report on the simulation of multi-module design.

1. Using AMIS to debug the Results of Behavioral Synthesis

In order to debug the architecture produced by behavioral synthesis 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.

The use of AMIS combined with AMICAL makes easier this correspondence. In fact the simulation may be executed in a step by step mode allowing to follow the execution details of the behavioral description. A step may be a basic transfer, a clock cycle or a macro-cycle. For each step executed, the system provides the corresponding VHDL line in the input description. It also uses the capabilities of AMICAL (figure 3) in order to show the resources of the data path used for the execution of the current step. This way the user can easily make the correspondence between the initial behavioral description and the scheduled behavioral description. It is easy to find which path is used to execute which transfer and which functional unit is used to execute which operation.

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 the scheduling may introduce extra steps that 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.

2. Using AMIS for data analysis of data dependant computation.

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 test program..

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

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

V. Multi-module Simulation

This sections explains the extension of AMIS to handle a multi-module systems. In this case a design is a heterogeneous system composed of programmable processors executing software and dedicated hardware processors communicating through a network. Such a system may be implemented as a single chip, a board or a geographically distributed system. In the rest of this paper we assume that the design starts with a heterogeneous model where the software modules are described C and the hardware modules are described in behavioral VHDL. We also assume that each software module will be executed on a SPARC processor. A cycle true model of the SPARC, called SPIM, is used for the simulation of software modules.

The simulation scheme is an extension of the one given in figure 5. A simulation configuration is made of several AMIS instances executing the different hardware modules and several software modules and several SPIM instances executing the different software modules. The different modules interact using UNIX/IPC communication.

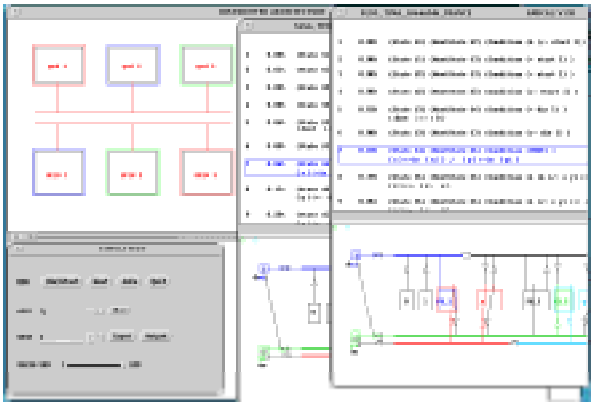

Fig.6 : Multi-component simulation

Fig. 6 shows an example of a multi-module simulation screen. The system under simulation is made of three hardware processors and three software processors. The hardware modules perform GCD operations. The right part of the screen shows the global configuration and the simulation control board. The left part of figure 6 shows two instances of AMIS.

The control board allow to select the module that needs to be shown during the simulation process. During simulation, SPIM provides a detailed report on the software execution (number of cycles, operations count …).

The multi-module simulation allows to debug both the modules and their interaction. It also allow to perform dynamic analysis of the system under test.

VI. Evaluation and Future Work

Although defined for AMICAL, most of the concepts underlying AMIS may be applied to other HLS tools. In fact, most of existing behavioral synthesis tools makes use of an internal representation of the architecture which can be used for simulation.

First experiments with AMIS tools have shown that such a tool brings a great deal of added value that makes easier the iterative design process required for behavioral synthesis. The paper discussed AMIS through a small example. However for large examples the need for such a tool is even bigger since the architecture is much more complex and harder to understand, to debug and to evaluate.

The main restriction of the present version is the non availability of a link to VHDL simulators. Such a link would allow to co-simulate behavioral modules with existing components described at the RTL and gate level. Such an extension requires the development of co-simulation techniques allowing to execute AMIS and SPIM concurrently with a VHDL simulator that emulates the hardware blocks which are not under synthesis. The communication between AMIS and the VHDL simulator may also use a protocol based on UNIX IPC as reported in [CouT95,PFHB95].

VII. Conclusions

This paper discussed the integration of an architectural simulator, called AMIS, within a behavioral synthesis tool, called AMICAL. This scheme allows the user to analyze the results of behavioral synthesis and to link this results to the initial behavioral model. The use of AMIS make the debug of the initial specification and the resulting architecture easier.

The simulation environment was extended to handle heterogeneous system composed of programmable processors executing software and dedicated hardware processors communicating through a network.

References

[Benc89] Benchmarks for the Fourth International Workshop on High Level Synthesis, 1989.

[Berr96] E. Berrebi al. “intensive use of Behavioral synthesis for the design of XXXXX”, 33rd Design Automation Conference, 1996

[ChaG92] 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.

[CouT95] S.L. Coumeri, D.E. Thomas, "A Simulation Environment for Hardware-Software Codesign", International Conference on Computer Design, 1995.

[Gajs96] D. Gajski, " Interactive behavioral synthesis", Invited paper, SASIMI 1996.

[JDKR96] A.A. Jerraya, H. Ding, P. Kission, M. Rahmouni, "Behavioral Synthesis and Component Re-use with VHDL", Kluwer Academic Publishers, Boston / London / Doredrecht, 1996.

[JKJ97] A. Jemai, P. Kission, A. Jerraya, “Combining Behavioral synthesis and Architectural simulation”, ASPDAC 1997.

[LMWV91] 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.

[NeSM89] J. Nestor, B. Soudan, Z. Mayet, “MIES: A Micro-Architecture Design Tool”, 22nd International Workshop on Microprogramming and Micro-Architecture, 1989.

[NGCD91] 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.

[PFHB95] 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.

[Syno95] Synopsys Inc., "Behavioral Compiler Methodology", Version 3.3.a, 1995.

[TDWR88] 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.

[TLWN90] 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.