High Level Synthesis of Protocols Described by a Formal Description Technique

L. Pirmez (1) (2)
A. Pedroza (1) (3)
A. Mesquita (1)

(1) Coppe/UFRJ - Program of Electric Engineering
(2) NCE/UFRJ - Electronic Computer Centre
(3) EE/UFRJ - Department of Electronics

NCE - 04/97

High Level Synthesis of Protocols Described by a Formal Description Technique

L. Pirmez (1) (2), A. Pedroza (1) (3), A. Mesquita (1)

(1) Coppe/UFRJ - Program of Electric Engineering
(2) NCE/UFRJ- Electronic Computer Centre
(3) EE/UFRJ - Department of Electronics
Tel: +55 21 598-3159 -- Fax: +55 21 598-3156
P.O. Box 2324
20001-970, Rio de Janeiro, RJ, Brazil
e-mail: luci@.nce.ufrj.br, aloysio, mesquita@coe.ufrj.br

Abstract

A methodology that efficiently translates Estelle formal specifications into a VHDL description, suitable for High Level Synthesis of communication protocols is proposed. The effect of the protocol description style in VHDL on the result of the HLS scheduling step is discussed by report to the Dynamic Loop Scheduling algorithm. An example using a test protocol is given.

Keywords

Protocol, high-level synthesis, VLSI, VHDL, Estelle.

1 INTRODUCTION

The increasing performance of the computer networks relies basically on the increasing speed, reliability, decreasing cost of the VLSI circuits and the development of optical fibers as a transmission media that enabled the emergence of new applications in multimedia.

Despite the progress on the VLSI circuits design methodologies and fabrication technologies, the main limiting factor in the performance of a computer network still is the amount of processing needed to run protocols in workstations and network servers. These technological issues require modifications in the protocol design and implementation. High performance in communication subsystems can be obtained by software optimization, by the use of parallel structures in the construction of protocols and by hardware implementation of these protocols in a VLSI circuit.

In this last case, the protocol designer should be concerned by the time required to design a VLSI circuit as compared with its expected commercial lifetime.

Integrated circuits are currently designed in a semi-automatic way by using a set of design tools generally integrated into a design environment. The design process generally begins with a high level system specification where this term refers to a system level behavioral hardware description. But, in the case of the communication protocols, the design process involves system specifications written in a level of abstraction even higher than that commonly used for digital systems, as is the case of the Formal Description Techniques (FDTs) such as the ISO Estelle language (Budkowski, 1987). As a consequence, the high speed protocols designer will face the problem of interacting with two design environments: one for the protocol specification, verification and testing and the other for the synthesis, simulation and physical implementation of a circuit. In general, these two environments are based on different description languages requiring an interface to translate one language into the other. Several efforts have been made to translate a system level specification into a Hardware Description Language (HDL) such as the work of Kloos (Kloos, 1993) on the translation of Lotos into VHDL and those of Pirmez (Pirmez, 1995), and Wytrebowicz (Wytrebowicz, 1995), on the Estelle - VHDL mapping.

The High_Level Synthesis of Protocol controllers requires the use of specific HLS tools called Control Flow Dominated Behavioral Compilers (Walter, 1991). These tools are able to handle large control structures that include sophisticated handshaking and non-structured sequences. However, most of these compilers have a drawback: the compilation results depend on the quality of the input description because a specification in VHDL (Ashenden, 1990) may include explicit synchronization points (e.g. a wait in a VHDL description) which restricts the scope of the transformation and optimization usually performed in behavioral descriptions (Bhasker, 1990). This paper presents a methodology to efficiently translate Estelle specifications into VHDL descriptions to be used by synthesis tools. The effect of the protocol description style in VHDL (Pirmez, 1996) on the result of the HLS scheduling step is discussed by report to a typical scheduling algorithm, the Dynamic Loop Scheduling (DLS) (Rahmouni, 1994), implemented in the HLS tool AMICAL, a VHDL behavioral compiler for control flow dominated machines (Park, 1993). In order to test the proposed methodology, a high speed protocol based on the ISO reference protocol ABRACADABRA called ABRACADABRA_HS was devised. Section 2 describes a methodology to translate Estelle specifications into VHDL descriptions, suitable for synthesis tools. The different styles that can be used to describe a protocol in VHDL and theirs results of HLS are discussed in Section 3. The last section deals with the results obtained so far.

2 TRANSLATING A ESTELLE** INTO A VHDL

The main steps involved in the proposed methodology of HLS of communication protocols are shown on Figure 1.

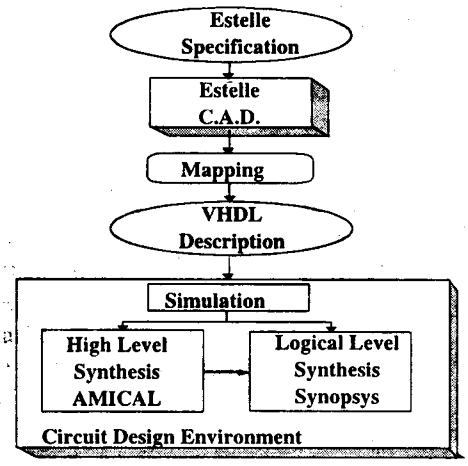
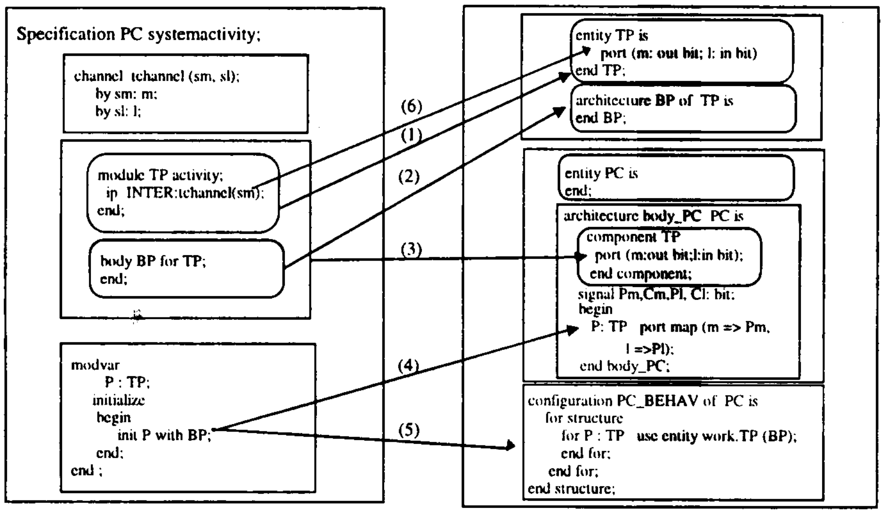

Estelle Specification
Estelle C.A.D.
Mapping
VHDL Description
Simulation
High Level Synthesis AMICAL
Logical Level Synthesis Synopsys
Circuit Design Environment

Figure 1 Protocol design environment

The starting point is a correctly compiled and simulated Estelle specification, obtained in a suitable protocol design environment, that is mapped into a VHDL description which is used by the circuit design environment. The following discussion concerns the translation of this specification into a VHDL description. Particularly, we are interested on criteria to guide the protocol's designer among the many different possible solutions, leading to an HLS-efficient Estelle-VHDL translation.

2.1 The Translation Strategy

An Estelle specification is composed of a set of **modules** hierarchically organized and a VHDL description is composed of a set of entities described separately. Thus, each Estelle module maps naturally into a VHDL entity, as shown in Figure 2-(1).



```
Specification PC systemactivity;
    channel tchannel (sm, sl);
        by sm: m;
        by sl: l;

    module TP activity;
     ip INTER:tchannel(sm);
    end;

    body BP for TP;
    end;

    modvar
        P : TP;
      initialize
       begin
          init P with BP;
       end;
    end ;
```

```
entity TP is
   port (m: out bit; l: in bit)
end TP;

architecture BP of TP is
end BP;

entity PC is
end;

architecture body_PC PC is
  component TP
   port (m:out bit;l:in bit);
  end component;
  signal Pm,Cm,Pl, Cl: bit;
  begin
    P: TP   port map (m => Pm,
          l =>Pl);
  end body_PC;

configuration PC_BEHAV of PC is
   for structure
      for P : TP   use entity work.TP (BP);
      end for;
   end for;
end structure;
```

Figure 2 Mapping Estelle into VHDL

The hierarchy among Estelle modules is maintained through a VHDL component declaration statement, as shown in Figure 2.(3).
Each VHDL component statement declares an entity and its description is placed in design libraries. The VHDL component instantiating construct is used for creating VHDL entities instances (Estelle *init function*) whose ports are either connected to actual signals or to the entity ports (Estelle *connect and attach functions*), as shown in Figure 2.(4). The binding of a VHDL entity to its instance (component instantiation) is achieved through a configuration declaration, as shown in Figure 2.(5).

Each Estelle module is composed of two parts: the header (*module*) and one or more bodies. This corresponds to the VHDL entity-architecture pair, with each Estelle header (*module*) corresponding to a VHDL entity and each Estelle body corresponding to a VHDL architecture, as shown in Figure 2.(1) and 2.(2). The main Estelle module (specification) does not has an external view and, consequently, its corresponding interface is empty.

Since there are no VHDL equivalent mechanisms to the Estelle synchronous/ asynchronous types of parallelism among module instances, an efficient Estelle-VHDL translation will require the restrictions included in Estelle* by Courtiat (Courtiat, 1988). This allows to represent in asynchronous form the parallelism among activity type module instances within a system activity type specification. As a consequence, only the semantic of an Estelle activity module is implemented in VHDL by creating an entity called HIERARCHY_QUEUE_ENTITY. This entity will forward an interaction to its queueing entity.

The Estelle language has a built-in FIFO queue mechanism that has no equivalent in VHDL. The solution adopted and implemented to overcome this limitation was to create an entity in VHDL, called QUEUE_ENTITY, to reproduce only the queueing management mechanism of an Estelle module. Since there is one FIFO queue for each interaction in a Estelle specification, it will be necessary to allocate VHDL signals to provide the communication among entities. Furthermore, a port called *next* is added to the entity that implements a module and to the queue management entity of this module. This port enables the entity that implements a module to search for the next message into its corresponding QUEUE_ENTITY.

In the header of an Estelle module, each interaction contained in the set of interactions of each interaction point of an Estelle module will correspond a port (signal) of a VHDL entity, as shown in Figure 2.(6). The signal type will be *input (output)* if the interaction point *role* is *reception (transmission).*

An Estelle body is composed of three parts: declaration, initialization and transition part. The declarative part contains Pascal-like statements and the declarations of the Estelle objects. The Pascal-like statements in Estelle may be converted into similar VHDL declarations with some restrictions: variables with tails variables are not allowed in VHDL; and the variables in the declarative part of a Estelle body should be declared in the declarative part of a VHDL process instruction.

By report to the Estelle objects declarations, the following procedure applies:

- A Estelle module translates into a VHDL entity;
- To specify the behavior of each Estelle module, defined in terms of a state machine, a signal will be created to store the current state of the corresponding VHDL state machine;
- The interactions associated to each internal interaction point of each Estelle module types will correspond to the internal signals of a VHDL entity.

The execution of the initialization part of a module is performed sequentialy and only when the given module instance is initialized. The variables initialization process, *"var:= x"*, in both languages is similar. In Estelle, the state initialization instruction "to state", the module instance creation *"init"* and the creation of bindings between interaction points *"attach"* and *"connect"* are directly converted into a signal attribution *"state <= value"* plus a component instantiation and signal attribution instruction, respectively.

The behavior of a module is described by the initialization and the transition parts of the body. The execution of both parts is sequential. The translation of Estelle clauses into VHDL structures is trivial and the set of structures obtained in VHDL is included in a process statement. The Estelle clauses *"from state"*, *"when interaction-point interaction"*, *"provided expression"*, *"to state"* and *"output* *interaction-point interaction(msg)"* are directly converted into *"when state => ... of case structure"*, *"wait until interaction_condition"*, *"if expression then ... "*, *"state : = state value"* and *"interaction <= msg"* in VHDL. The Estelle instructions *"procedure name (formal parameters queue)"* and *"function name (formal parameters queue): return "* are directly converted into the equivalent VHDL declarations.

Only the Estelle delay clauses *"delay(T)"* or *"delay (T,T)"* will be translated in VHDL. An entity called CLOCK_ENTITY_TYPE is created in VHDL as a time counter with the following ports: *Count, Period and Finished.* The time counting starts when the CLOCK_ENTITY_TYPE entity receives a *Count* signal and the QUEUE_ENTITY of a given entity returns a *no message* signal when this last entity asks for a next message. If the counting value is bigger than the value buffered in *Period,* the CLOCK_ENTITY_TYPE entity sends a time-out signal, *Finished,* to the entity that executes the state machine. When the entity that executes the state machine receives the time-out signal, the time-out treatment instructions are executed.

The mapping strategy may be divided in three steps. Initially, all module types in a Estelle specification and their hierarchic relation are identified. Then, each module type is implemented as an independent VHDL entity. The modules hierarchy is established by means of a VHDL component construct. Finally, queue management entities are created, one for each entity and one for each group of child entities.

To structure each VHDL entity the following procedure is proposed:

To a FSM described in the body of an Estelle module, there will correspond a state machine described in VHDL. However, VHDL does not have a build-in concept of state, it will be necessary to build-up the VHDL state machine. To this end a VHDL entity is created with an associated architecture containing a behavioral description of the FSM, obtained directly from the Estelle specification, as follows:

1. The set of different states of an Estelle module will form a VHDL case structure.
2. For each state, transitions associated with this state are grouped by interaction. The *"wait until condition_interaction"* instruction will stop the execution when the clause *condition_interaction* is evaluated to true. If there is more than one *condition_interaction* to be evaluated in the same state, an *"if"* VHDL construct will be placed just after the *''wait until condition_interaction'*'statement.
3. To avoid code duplication when specifying exceptions, these signals (*reset, error situations*) should be created and added to the sensitivity list of the *"wait"* statement. Thus, the exceptions signals are checked immediately after this *"wait"* statement.
4. Only one state at a time is active in this state machine. Furthermore, due to the filtering performed by the QUEUE_ENTITY entity, only one interaction is activated at a time. The resulting state machine should be placed inside a VHDL

process, enabling the repeated execution of the state machine. This state machine will remain in stand-by until a signal change occurs.

2.2 The AMICAL Environment

The AMICAL environment used for the HLS was based on a structured design methodology (Park, 1993) to allow the use of the hierarchy concepts in the synthesis process (Kission, 1995).

The synthesis process starts with two types of information: a behavioral specification given in VHDL and an external library of functional units. In this discription, each entity is composed of only one process. This process may include complex sub-systems by using procedure and function calls implementing in this way the concept of hierarchy among modules. However, for each procedure or function used, the library should contain at least one functional unit able to execute the corresponding operation.

From the point of view of the HLS, a structured design is composed of a system and a set of components, described at the behavioral level. At the behavioral level, a component is described by a VHDL entity that, in the AMICAL environment, is invoked through either procedure or function calls. A component may correspond to a design produced by external tools or to a sub-system originated on a previous design. The abstract concept of behavioral component was used to allow re-use.

As each sub-system must be translated into an AMICAL compatible VHDL input description, the following restrictions apply:

- The VHDL ''*entity*'is restricted to the declaration of I/O signals.
- AMICAL does not accept the following types: *physical, floating point, enumeration, arrays and records.*
- Operators are permitted under the assumption that there are functions(functional units) able to execute them.
- VHDL description using only ''*wait until...*'statements.
- The "*wait for time*" statement is in fact ignored by AMICAL. To consider time delays a possible solution consists in adding a clock signal, to start the timer and wait for the time-out signal.
- Each architecture must contain only one concurrent statement and this statement must be a ''*process ...*'statement.
- It is not possible to use a signal assignment with time expression. Consequently, one should activate first the timer, use a "*wait*" statement associated to a time-out signal and then use signal assignment.

Further restrictions apply to the use of the following Estelle signal types:
•*real, record, pointer, aliases* types and *recursion* are not allowed;
• only one dimensional arrays are allowed;

The above rules enable the generation of feasible Estelle-VHDL mappings, which means that to each Estelle** specification there exists a semantically equivalent description in VHDL* that is accepted by the AMICAL HLS tool. This do not implies a one-to-one correspondence between the subsets of each language. In fact, there are many possible VHDL translations for the same Estelle specification. The choice of a suitable VHDL description style for protocols is an important issue when the HLS is sought, since this affects directly the structure of the synthesized hardware (Bhasker, 1990) as will be seen in the following discussion.

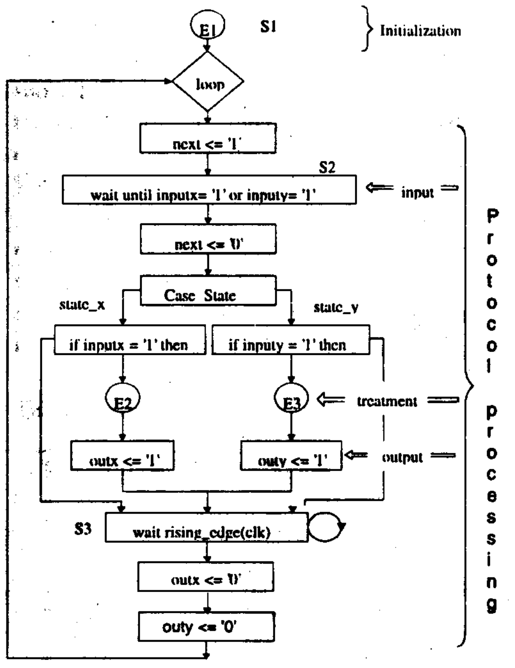
3 DESCRIPTION STYLES FOR PROTOCOLS IN VHDL

A protocol may generally be described by many different VHDL behavioral description styles. These styles provide semantically equivalent descriptions that differ only in the quality of the HLS result. The quality of this HLS output structural description is measured by the number of states and paths generated by the scheduling process performed by the HLS tool. It will be shown that the quality of the HLS output description will depend on the type of statements used and on the order in which they appear in the input description.

To illustrate this fact, consider a protocol modelled by a state machine that performs handshaking operations with another state machine as shown on Figure 3. This generic protocol has two phases: the initialization phase and the protocol processing phase. The last one consists of a loop that waits for an input signal, process the input data and outputs the corresponding result, through a signal. In the processing phase one may further distinguish three steps: data input, treatment and output, assuming no data dependencies in the model.

This model will be used to illustrate the several description styles and the consequences on the HLS results. The main tasks involved in HLS are scheduling and allocation. The analysis will be restricted to the scheduling step which is responsible by the complexity of the controller.

The First Description style for implementing a state machine associates a VHDL *"wait until condition"* statement to each branch of a Estelle *"case state"* statement. The second description style places the *"wait until input_condition"* statement outside the *"case"* statement. The Third employs an *"if ... then ... elsif ... end if"* structure instead of a *"case state"* statement inside a *"process"* declaration. This construct uses the state as the outer variable. A *"wait until input_condition"* statement is placed outside the nested if clauses. The Fourth also employs an *"if ... then ... elsif ... end if"* structure but with the input as the outer signal. The Fifth alternative for implementing a state machine is a variation of the second one; the different output control signals are settled to zero immediately after the *"case state"* statement. Figure 3 shows the VHDL description and the corresponding CFG resulting from the Fifth style. Table 2 shows the state table resulting from the scheduling for this description using the Dynamic Loop Scheduling algorithm.



```
1  process
2    begin
3      -- E1
4    MEF: loop
5       proximo <= '1';
6       wait until inputx=1 or inputy=1;
7       proximo <= '0';
8       case state is
9         when statex =>
10          if inputx = '1' then
11            -- E2
12            outx<= '1';
13          endif;
14        when state y =>
15          if inputy = '1' then
16            -- E3
17            outy<= '1';
18          end if;
19      end case;
20      wait until rising_edge(clk);
21      outx <= '0';
22      outy <= '0';
23    end loop;
24 end process;
```

Figure 3 Control flow graph and VHDL process for the Fifth Description Style

Table 2 Transition Table of the Fifth Description

Transition	State	Operation	Next	Condition
1	S1	3,5	S1	---
2	S2	7,11,12	S2	state = statex and inputx =1
3	S2	7	S2	state = statex and inputy =1
4	S2	7,16,17	S2	(state /= statex and inputy =1
5	S2	7	S2	state /= statex and inputx =1
6	S2	---	S2	(inputx /=1) or (inputy /= 1)
7	S3	21,22,5	S2	---

3.1 Example of Application

The five description styles discussed previously were used, as an example, to describe the Signaling state machine of the ABRACADABRA_HS protocol defined on (Pirmez, 1995). The resulting descriptions were synthesized by

AMICAL using the DLS algorithm. The DLS algorithm is optimized (Rahmouni, 1994) for the treatment of control-flow dominated descriptions. It is based on the same principles of path scheduling (Camposano, 1991), but significantly reduces the number of the generated paths and hence the computation cost. The scheduling results obtained by the DLS algorithm, for each description style, are displayed in Table 3.

The following information is displayed in the table: the number of paths, the number of states generated by each description, the number of operations, the number of waits, the number of if's, the number of cases, the number of VHDL lines of code in the input description and an estimate of the area of the controller, in number of transistors.

Table 3 DLS Results

Description	Path	States	Operations	Wait	Ifs	Case	Lines	Area
First	614	88	400	54	34	1	653	206.183
Second	138	70	406	45	42	1	622	46.473
Third	136	69	413	44	51	0	625	45.904
Fourth	137	69	451	44	52	1	616	46.237
Fifth	104	36	325	11	42	1	557	35.641

It can be observed from Table 3 that globally the Fifth Description presented the best results. It significantly reduced the number of *"wait until input_condition"* and the number of *"wait until rising_edge (clk)"* statements. As a consequence, the number of paths generated is reduced and, hence, the computational cost. The DLS algorithm generates a new path each time a *"wait until ..."* statement is found thus, the number of wait's is the main cause of paths generation in the algorithm. For this reason the First Description style presents the worst results. The other description styles significantly reduce the number of *"wait until input_condition"* reducing considerably the generation of paths. The Second and Third Description styles produce similar results since the *"if ... then ...elsif ... endif"* and *"case ..."* statements have, in fact, similar structures. For this particular example the results show a similar behavior of the Third and Forth styles. But in general one can expect better results from the Third Style when the protocol processing is more dependent on the number of states than on the number of inputs, reducing, accordingly, the number of generated paths and operations. On the opposite case, one should expect a better performance for the Forth style.

As a consequence of the previous discussion, to produce efficient protocol descriptions for HLS, the following general procedure should be adopted :

- The FSM protocol model corresponds to a VHDL process. The process is composed of two loops: an external loop that models the restart of the process and an internal loop that models the FSM of the protocol.

- The *"wait until ..."* statement is combined with the inner loop where the control signals are checked.
- In order to avoid duplications in the code when specifying exceptions, the exceptions signals (reset, error situations) should be created and added to the sensitivity list of the wait statement. Thus, the exceptions signals are immediately checked after this wait statement.
- A state machine can be built using a "case ..." or "if ... then ... elsif ..." statement. In both solutions, the decision to test initially the state or the input will depend on the processing. That is, if it is more dependent on the state than on the input or vice-versa.
- After a *"case..."* or after an outer *"if ... then ... elsif ... end if"* statement, the output control signals should be set to zero.

4 FINAL CONSIDERATIONS

A Formal Description technique as Estelle is a natural choice to the protocol designer when specifying and analyzing communication protocols. On the other hand, VHDL is the best choice for dealing with run time problems and interfacing with synthesis tools. This work presented a translating strategy for HLS using the VHDL based AMICAL environment. Initially, the Estelle constructs were analyzed by report to their equivalence to VHDL. A simplified version of Estelle, the Estelle**, was proposed to simplify the translation to VHDL.

Five alternative VHDL description styles for FSM's that can be used to describe a protocol and the constraints that determine the most suitable style to synthesis tools are discussed.

Finally, the proposed methodology was applied to an example, the high performance ABRACADABRA-HS protocol. The conclusions drawn from the ABRACADABRA_HS protocol example can be extended to any system modelled by a FSM.

5 REFERENCES

Ashenden P. J., (1990) The VHDL Cookbook. Dept. Computer Science of University of Adelaide.

Bhasker J., Lee H. C. (1990) An optimizer for Hardware Synthesis. IEEE Design and test of Computer. special issue on High-Level Synthesis, pp 20-36.

Budkowski S. and Dembinski P. (1987) An Introduction to Estelle : A Specification Language for Distributed Systems. Computer Network and ISDN System 14(1), pp. 3-23, North-Holland.

Camposano R., Saunders L. F., Tabet R. M. (1991) VHDL as Input for High_Level

Synthesis. IEEE Design & Test of Computers, pp 43-49.

Courtiat J. P. (1988) Estelle* : A powerful Dialect of Estelle for OSI Protocol description. In: Proceeding of the VIII International Symposium on Protocol Specification, Testing and Verification, pp. 171-186,France, North-Holland.

Kission P., Ding H., Jerraya A. (1995) VHDL Based Design Methodology for Hierarchy and Component Re-Use at Behavioral Level. EURODAC'95, Brighton, UK, September.

Kloos C. D. et all, (1993) VHDL generation from a timed extension of the formal description technique LOTOS within the FORMAT project. Microprocessing and Microprogramming 38, pp. 589-596, North-Holland.

Park, O'Brien K., Jerraya A. A. (1993) Tutorial - AMICAL :Interative Architectural Synthesis Based on VHDL. In Synthesis for Control Dominated Circuits, pp. 219-234,North-Holland.

Pirmez L., Pedroza A., Mesquita C. (1995) A Methodology to the Implementation of Distributed System in Hardware from a Formal Description', In: Proceedings of the IFIP WG. 6.1 Fifteenth Int. Symposium on PSTV, pp. 419-434,Warsow, Poland, june, Chapman&Hall.

Pirmez L . et al (1996) Analysis of Different Protocol Description Styles in VHDL to High-Level Synthesis. In: Proceedings of the EURODAC'96, pp. 490-495, September,Geneva,Switzerland.

Rahmouni M., O'Brien K.and.Jerraya A. A. (1994) A Loop-based Scheduling Algorithm For Hardware Description Languages. Parallel Processing Letters, World Scientific Publishers, Vol. 4(3), pp 351-364.

Walter R.A., Camposano R (1991) A Survey of High-Level Synthesis. Systems. Kluwer Academic Publishers.

Wytrebowicz J. (1995) Hardware Specification Generated from Estelle. In: Proceedings of the IFIP WG. 6.1 Fifteenth Int. Symposium on PSTV, pp. 435-450,Warsow, Poland, june, Chapman&Hall.

6 BIOGRAPHY

A. Mesquita received the E.E. degree from PUC/MG, the M.Sc. degree from PUC/RJ and the Docteur d'Etat degree from Université Paul Sabatier of Toulouse, France. Current research interests: Circuits and Systems Theory, VLSI circuit design and Signal processing. Associate Professor at COPPE/UFRJ.

A. Pedroza received the E.E. degree from UFRJ, the M.Sc. degree from COPPE/UFRJ and the Doctorat degree from Université Paul Sabatier/LAAS. Current research interests: Formal Specification, Verification and Implementation of Protocol. Associate Professor at UFRJ.

L. Pirmez received the E.E. degree from UFRJ, the M.Sc. degree from COPPE/UFRJ and the D. Sc. thesis at COPPE/UFRJ. Current research interests: Formal Specification, Verification and Implementation of Protocol.