26

A methodology for the implementation of protocols in hardware from a formal description

L. Pirmez (1) (2), A. Pedroza (1) (3), A. Mesquita (1)
(1) Coppe/UFRJ - Program of Electric Engineering
Tel: +55 21 2605010
Po. Box 68504 - 21495 Rio de Janeiro RJ Brazil
E-mail: luci@barra.nce.ufrj.br
(2) NCE/UFRJ - Electronic Computer Centre
(3) EE/UFRJ - Department of Electronics

Abstract

A methodology[1] that efficiently translates Estelle formal specifications into a Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) description, suitable for the high level syntesis of an integrated circuit, is proposed. It will be shown that, in order to efficiently map Estelle into VHDL, a number of constrains must be imposed on the set of possible constructs within each language. An example based on the specification of a high speed protocol is discussed.

Keywords

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

1 INTRODUCTION

The emergence of new technologies in VLSI, the high efficiency of nodes (commuters), the reduction of memory and processor costs and the development of optical fibbers allowed the development of networks of increasing speed and reliability. The bottleneck to increase the performance in computer networks is no more the transmission media but the amount of processing needed to run the protocols in the workstations and the network servers. In fact, the main source of errors in these systems is the loss of packets due to the lack of buffers.

[1] Research supported by CNPq / ProTem-CC,project TRAVEL.

P. Dembiński et al. (eds.), *Protocol Specification, Testing and Verification XV*

These technological issues and the emergence of multimedia applications require modifications in the protocol design. On the one hand, protocols must be light enough to allow high speed operation and, on the other her hand, some additional functions should be included to perform errors and losses recovery, traffic control and to support multimedia applications. These requirements led to the development of several high speed protocols such as DATAKIT by Fraser (1989), XTP by Strayer (1985) and the one proposed by Netravali (1990). Other adequate protocol design techniques for high performance communication systems are described by Doshi & Johri (1992) and Doeringer et al (1990).

High performance in communication subsystems can be obtained through software optimisation, by the use of parallel structures to built up the protocol and also by implementing these protocols in VLSI hardware. An integrated circuite that implements one or more protocol layers is called a protocol controller [Krishnakumar(1989)]. A classical example is the Protocol Engine which implements the high speed protocol XTP. Another example is described by Braun (1994) who showed the implementation of a transport protocol, the Patroclos, using VLSI components assigned only to perform the functions that usually are responsible for the bottlenecks such as those related to timing, memory management and to the transmission support.

The time necessary to design a circuit and its commercial lifetime are today of the same order of magnitude. In order to reduce the design time, the integrated circuits are currently designed in a (semi) automatic manner from high level system specifications. It should be pointed out that the term high level system specification refers to any hardware specification that is not an exact description of an implementation of the manufacture mask. This (semi) automatic way to design integrated circuits has led to the development of techniques that allow to obtain an integrated hardware from a high level specification (silicon compilation technique). Thus, to further reduce the hardware implementation time of computer network protocols, one should design integrated circuits in VLSI from system specifications written in a very high level of abstraction, using, for example Formal Description Techniques (FDTs) such as the ISO Estelle language [Budkowski (1987)].

In this paper, a methodology leading to a (semi) automatic procedure for the implementation of integrated circuits for high speed communication systems is proposed. This implementation is performed from Estelle specifications of efficient protocols using a silicon compilation technique [Gajsky (1992)].

In order to apply the proposed methodology, a high speed protocol based on the ISO reference protocol ABRACADABRA called ABRACADABRA_HS was devised. This protocol includes the parameters of most of the existing high performance protocols such as the XTP and the DATAKIT.

Section 2 presents the preliminary concepts concerning the formal specifications of protocols [Chanson (1993)], the main characteristics of the Estelle language[Budkowski (1987)], the definition of silicon compilation technique and the main characteristics of the VHDL [Ashenden (1990), Navabi (1992)]. Section 3 describes a methodology to convert Estelle specifications into VHDL description. Section 4 introduces our case study: the proposed methodology is applied to a high performance protocol, the ABRACADABRA_HS. The last sections deals with the results obtained so far, the perspectives of our project and final considerations.

2 FORMAL SPECIFICATION AND HIGH-LEVEL SYNTHESIS

Communication protocols are a set of rules and procedures that allow the interaction among communicating entities and are consequently crucial to the operation of computer networks and distributed systems. The increasing use of distributed systems and the vast acceptation of the reference model and protocols standardised by ISO motivated many researches to improve protocol engineering in the last decade. The importance of formal methods of Protocol Engineering was noticed since the beginning, particularly in the specification of Communication Protocols. The term Formal Description Technique (FDT) was created in the late 70's to refer to the techniques which have been used since then, to describe protocols and data communication services in a more accurate way. In parallel to the academic efforts in research of FDTs, the ISO and the CCITT contributed to the development of standardised FDTs in the late 80's. Estelle and LOTOS languages were defined by ISO, and the SDL language by CCITT.

This work presents a methodology that integrates an Estelle C.A.D. to a micro electronic C.A.D, allowing a high level synthesis out of a VHDL specification. To this propose it will ne necessary to examinate the caracteristics of the Estelle into VHDL mapping. The mapping of Estelle into VHDL is facilitated because the VHDL has similar structures to the Estelle.

2.1 The Estelle Language

The Estelle language is a Formal Description Technique developed by ISO for the specification of distributed systems, specifically the standards related to services and to communication protocols.

In Estelle, a *specification* is composed of a set of *modules*. A module can be refined in sub modules, defining an hierarchical structure among themselves. A module is composed of the header and the body. The interaction points, the exported variables and the class attribute are defined in the header. The module behaviour is defined in the body. The behaviour of an Estelle module is specified through an extended state machine. This state machine is a transition system presented in the form of Pascal-like instructions. The evolution of the module corresponds to the shooting of transitions in this state machine. The transitions are considered to be atomic. The declaration of the module defines a new type in the specification. Variables of type module specify the instances of the module.

Courtiat (1988) introduced a simpler and more efficient version of Estelle, the Estelle*. This version allows obtaining protocol specifications with a high level of abstraction, using a semantic of parallelism among module instances consistent with Petri networks. Estelle* is one of the starting points of our work, and some additional restrictions to Estelle to improve the conversion to VHDL are found in section 3.

2.2 The High-Level Synthesis

VLSI integrated circuits have been designed in a (semi) automatic manner from high-level system specifications to decrease the project time. This line leads to the development of the

silicon compilation technique, that is, the development of techniques that allow an integration hardware layout from a high level specification.

The design of integrated circuits can be classified by their domain and by their level. The many existing representations can be discussed in a clearer way through the Gajski-Kuher's Y-chart [Gajski (1992)], showed in figure 1. By convention, as we move farther from the Y centre, the level of description becomes more abstract.

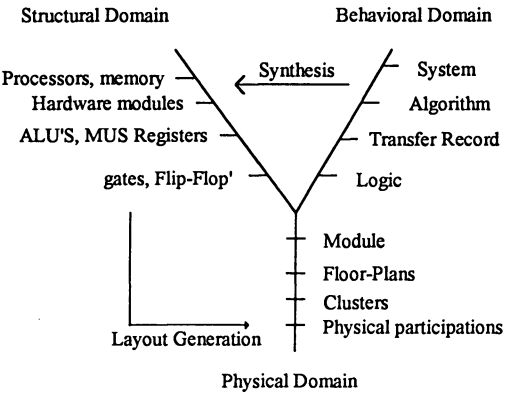

Figure 1 The Y-chart.

There are three domains of hardware specification related to the domain representation*: the behavioural, the structural and physical.* In the *behavioural domain* we are interested in what a design does and not in how it is built. In this domain, a design corresponds to one or more black boxes where only their interface with the external environment and a set of functions are considered. This functions describes the behaviour of each output in terms of the inputs over time. The structural domain is a bridge between the behaviour and physical domain. In this domain, a system is specified through their components and connections, such as the processors, the memories, the flip-flops and so on. The physical domain describe the allocation of the hardware module in the area designated to the circuit manufacture.

As for the level representation, there are various levels, such as *architectural level, logic level* and *circuit level.* A level is characterised by elements that it uses, as we can see in figure 1.

Silicon compilation technique of a hardware specification can be divided into two different stages: the synthesis and the layout generation. The synthesis is a translation from a *behavioural* specification into a *structural* specification. The synthesis process is not the only one because there are various hardware structures which can correspond with the same behavioural specification. This work will use AMICAL [25] to do the synthesis. AMICAL is one of the environments of high-level synthesis that presents suitable characteristics to support the synthesis of specifications written in Estelle. AMICAL allows high-level synthesis from VHDL protocol specification. VHDL is a language that is becoming a standard to hardware specifications in the various domains of the Gajski-Kuher's Y-chart. Thus, the synthesis of a

specification in Estelle is obtained after the conversion of an Estelle specification into VHDL behavioural descriptions. The layout generation is responsible for the translation of the hardware structure in to physical information equivalent to the allocation of hardware structure over the chip area.

2.3 The VHDL Language

VHDL [Ashenden (1990), Navabi (1992)] is a standard language developed by the U.S. Department of Defence (DoD) to describe digital electronic systems. VHDL supports the design, description and simulation of hardware structures in different abstract levels, from the architectural level to the logic level. This language was projected to be independent of any specific technology, design environment, or design method, and thus, it should be possible to integrate it into any combination of environment, technology, and methodology.

VHDL is a multilevel language that allows structural and behavioural descriptions. The structural description allows to describe the structure of a design, their decomposition in sub-designs and the interconnection among themselves. The behavioural description allows the specification of the function of a design using familiar programming language structures.

A digital system is usually designed as a hierarchical set of modules. Each module has a set of ports that constitute its interface with the external world. In VHDL, an entity represents a module. The entity may be used as a component in a design, or it may be the top level module of the design that is, the module in which the hierarchical level is the highest. An entity, as showed in figure 2, is composed of two segments, an interface (*entity*) and a set of one or more bodies (*architectures*). The interface defines the entity's externally observable characteristics: the input/output ports. The body defines the implementation of an entity that is not visible from the entity's environment.

```
entity example is
        port (ConReq, ConRsp : in bit; ConInd ,ConCnf : out bit);
end example;
architecture example_body of exemple is
        component conection                                    -- declaration
                port (CR: in bit; CI : out bit);
        end component;
        other declarations
begin
        CON : conection port map (CR => ConReq, CI => ConInd); -- Instance
        other concurrent instructions;
end example_body;
```

Figure 2 Example of interface segment and of body segment of an entity.

The input/output port declarations, whose class is *signal*, allow an entity exchange information with other entities in the interface segment of the entity structure. In the body segment of the entity structure is met the implementation of an entity. The body segment is divided into two parts: declarative part and instruction part where the concurrent statements are. Signals and components can be declared in a *body*, thus allowing the building of a structural description in terms of component instances, as illustrated in figure 2. Signals are

used to connect *sub modules* in a design. The *component* declarations allow an entity to use other entities described separately and placed in design libraries. In order to do this, the body must declare a component and then it must be instanced within its instruction part. Later, the configuration declaration can be used to match the component with its specific library entity.

2.4 Restrictions to the VHDL language to obtain efficient VLSI circuits

The conceiving process of a design has a main role in the high-level synthesis, since that the way as a design is modelled and described has a direct impact in its final implementation, both in terms of performance and cost. Thus, an efficient high-level synthesis will depend on a good match between the language's model and the target hardware architectural model. However, if the language was developed to attend a large spectrum of designs and applications, the language semantic model may be quite different from the target architecture generated by the high-level synthesis tools.

This discrepancy between language semantic models and target architectural models is especially true with VHDL. VHDL was designed to attend a broad range of design descriptions. Consequently, this language has various language constructs that allow the description of the same behaviour in many different ways. After a synthesis process, some descriptions generate hardware structures that are not needed. These inefficient structures must be removed by the adopted high-level synthesis tool to generate a reasonable design. Unfortunately, there are few tools that facilitate the task of modelling and describing of designs in high-level synthesis; the major part of this work is in the designer's mind.

One of the goals of this work is to propose guidelines to a good match between the language's semantic model used in a communication protocol and the target hardware architectural model generated after the synthesis which implements the considered protocol. A FSMD (Finite State Machine with Datapath) is a design model to a communication protocol. FSMD corresponds to the RT (Register Transfer) design. Hence, the behavioural description of an RT design is naturally expressed on a state-by-state basis wherein each state specifies the conditions to be tested in the control unit, the operation to be performed in the Datapath, and the next state for the design. Unfortunately, VHDL language does not have the concept of states built into the language. Consequently, we can model the state machine behaviour in several ways and some of then will result in inefficient synthesis.

3 METHODOLOGY TO SYNTHESIS OF DISTRIBUTED SYSTEM FROM A FORMAL SPECIFICATION

This section proposes a methodology which is able to make an efficient and easy conversion from an Estelle specification protocol into VHDL descriptions. We can generate an integrated circuit from those VHDL descriptions and through other synthesis tools. The design time can be reduced with this methodology. In addition, the synthesis should be a transparent process for the Estelle users.

The methodology to obtain VLSI circuits from an Estelle specification is shown in figure 3. The methodology start from a suitable environment to the development of protocols in Estelle, the Estelle C.A.D. After the generation of an Estelle specification correctly simulated, this

specification is converted into VHDL. This conversion is made by the CONVERTER. Afterwards, the output of the CONVERTER will be used as an input to a micro electronic C.A.D. The micro electronic C.A.D. is a suitable environment to the design of integrated circuit using high-level synthesis. It should be noted that the two environments, Estelle C.A.D. and micro electronic C.A.D., are independent. The CONVERTER must be a bridge between these environments.

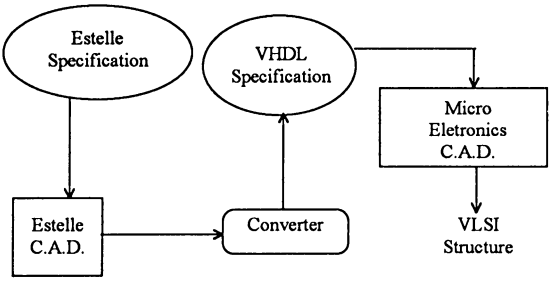

Figure 3 Methodology to obtain VLSI circuits from an Estelle specification.

3.1 Description of Estelle aspects concerning to the generic conversion

The CONVERTER should connect the Estelle C.A.D. to the micro electronic C.A.D.. The input to the CONVERTER is an Estelle specification and the output is the corresponding VHDL description. It is the CONVERTER responsibility to find out VHDL constructs that matches Estelle constructs. But this is not always possible, and we must then specify the functioning of this inexistent constructs which usually generates overspecified texts. We can point out some aspects that were not found in VHDL language but that exists in Estelle, such as queue mechanism, exported variables, the existence of different types of system. In this section a possible solution of a conversion from Estelle into VHDL is presented.

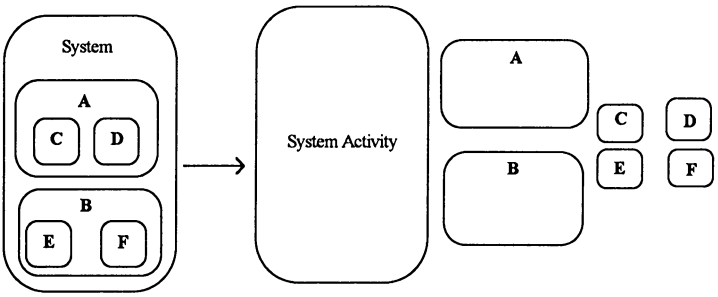

Figure 4 Convertion of Estelle modules into VHDL entities

Each module concerning the Estelle specification will correspond to a VHDL entity, see figure 4. The roles of the communication channels (interactions) of each type of module will correspond to the ports of the interface of a VHDL entity. The ports in VHDL will be input type (in), output type (out) or input/output type (inout). The declarations of the data types and objects types in VHDL are converted into similar Estelle declarations (Pascal-like declarations).

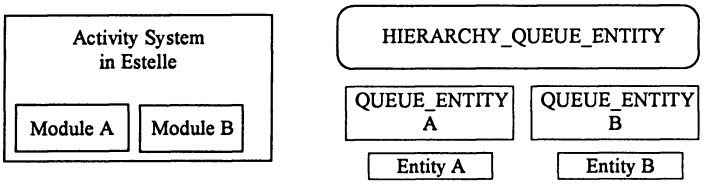

Figure 5 Entity simulating the queues system.

The FIFO queue mechanism included in Estelle language cannot be converted directly into VHDL. The queue mechanism can be implemented by an entity, we call QUEUE_ENTITY, responsible only for queue management of an Estelle module (see figure 5). Furthermore, we will create another entity, called HIERARCHY_QUEUE_ENTITY, responsible for the management of queues of module instances with the same hierarchy. For each communication channel concerning an Estelle module, the entity responsible for the queue implementation of this module must manage two FIFO queues. The first queue keeps the types of received/transmitted interactions and the other one keeps the contents.

The exported variables are another aspect met in Estelle that does not exist in VHDL. If there is a exported declaration in a module, two ports, one input and another output, must be created in the VHDL entities that will relate this module to this father module. In VHDL, the entities are independent so that the communication among entities is allowed only through input/output ports. Besides that, another entity, which will be only responsible for assuring the mutual exclusion, is created. In this entity, pairs of ports (input, output) are also created, one for each entity that has access to this declarations. All the writing (reading) operations made in the exported variables will be replaced by a corresponding *output* (*when)* clause.

Estelle has two types of parallelism among module instances (activity type and process type). There is no equivalent construct in VHDL. The module instances of process type perform in a synchronous parallelism while the modules instances of activity type perform in a non-deterministic way. The conversion of a process type module or of an activity type module from Estelle to VHDL cannot be obtained in a direct manner. The semantic of an activity type module in Estelle is implemented in VHDL by a HIERARCHY_QUEUE_ENTITY entity. This entity must select a module instance for execution. A process type module in Estelle can't be converted directly into a VHDL entity whose body has the *process* instruction because in VHDL the parallelism is asynchronous. The semantic of a process type module in Estelle is implemented in VHDL by a HIERARCHY_QUEUE_ENTITY entity. This entity must select a group of instances with no parent/child conflict which are ready to be shot for execution. Besides, the HIERARCHY_QUEUE_ENTITY entity must make sure that the next selection will take place only after all the selected instances execution are finished.

The ***delay*** clause in Estelle is more generic that the similar VHDL clause. A solution to the conversion is to restrict the use of this clause in Estelle. In Estelle, the delay clause will be used only in the form "**delay** (E1)" or "**delay** (E1,E1)".

The utilisation of the ***priority*** clause in Estelle is more generic than the similar VHDL clause. The conversion of the *priority* clause will not be implemented because the obtained hardware would not have a simple related logic circuit.

The Estelle clauses "**from** *state*", "**when** *Interaction-point.interaction*", "**provided** *expression*", "**to** *state*" and "**output** *Interaction-point.interaction(msg)*", in VHDL, are directly converted into "**when** *state* => ... *of case structure*", "**wait on** interaction **until** *condition*", "**if** *expression* **then** ... ", state : = state valour and *interaction* <= *msg.*

The Estelle instructions "**procedure** *name(formal parameters queue)*" and "**function** *name(formal parameters queue)* : return type" are directly converted into the equivalent VHDL declarations. It should be reminded that VHDL does not allow recursion. No solution was given to this problem in this fist version of our work.

The Estelle extensions, which are responsible for dynamic configuration, will not be converted into VHDL. This is not an important restriction if we consider that it is not economical to define a dynamic architecture in hardware.

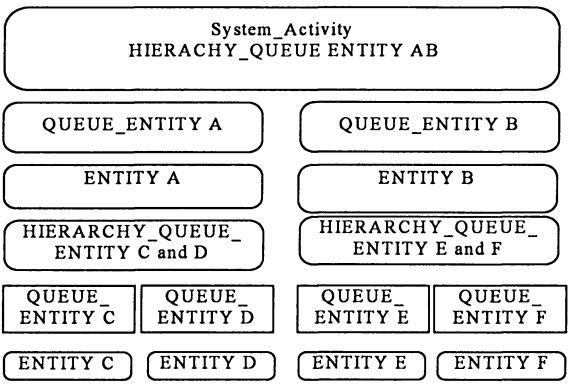

Figure 6 Estelle module convertion into VHDL entity

Estelle modules are specified in a nested form and VHDL entities are specified separately. Its hierarchy is obtained through a VHDL *component declaration* construct. Then, each VHDL *component declaration* construct declare an entity and its description is placed in design libraries. The VHDL *component instantiation* construct is responsible for the creation of a VHDL entity instance (Estelle "init" function) and for the connections of the component ports with actual signals or entity ports (Estelle "connect" and "attach" functions). The binding of an entity to this entity declaration (component) is achieved through a *configuration declaration.* Figure 6 represents an activity system formed by two activity module instances (A and B). Each instance is formed by another activity module instance (C, D, E and F). The Estelle main module (*specification*) is converted into a VHDL entity.

The proposed methodology for synthesis will be applied to the Estelle activity systems. This methodology is divided into four stages as shown in figure 6. At first, all module types in a Estelle specification and their hierarchic relation will be identified. Then, each module type is implemented as a VHDL entity. Each module implementation is made independently of each other and their hierarchy is established through the VHDL *component* construct. Next, queue management entities are created, one for each created entity and one for each group of child entities. A topology does not apply to systems that have only process type modules or in systems that have process type modules and activity type modules.

Following, a methodology to structure each Estelle module is proposed. Each module is implemented independently from the others. The proposed methodology is formed by a sequence of steps. At first, states and transitions belonging to the body of each module type are settled. States of each module type will correspond to the states of the related state machine written in VHDL. This state machine implements the body of an entity VHDL equivalent to this module in Estelle. Transitions are equivalent to the shoot of an action. This state machine will execute these actions of according to the received interactions, that is, the received signals. To each state, group their related transitions. Finally, the interactions associated which each state and transitions are obtained. These interactions will coincide with the signals which are responsible for the evolution of the state machine built in VHDL.

The body of an entity will be built in accordance with a behavioural description. This description will be built directly from the Estelle state machine description. A variable (STATE) is defined with the current state of the VHDL state machine. Now we have two options: either the transitions are joined from states of state machines or transitions are joined from received interactions by the module. In this paper the first option will be examined:

1. The set of different states of a module will form a VHDL case structure.
2. For each state, transitions associated with this state are grouped by interaction. The "**wait until** *condition_interactions*" instruction will suspend or return the execution until the clause *condition_declaration* evaluates to *true*. If more than one *condition_interaction* can be evaluated to *true* in the same state, an "**if**" VHDL construct will be placed just after the "**wait until** *condition_interactions*".

```
case STATE of
  when state-1 =>                                  -- procedure of state 1;
       wait until condition_interaction_1, condition_interaction_2 ;
       if condition_interaction_1 then ...         -- transitions;
       elsif ...
       else ...                                    -- spontaneous transitions;
       end if;
  ...
  when others                                      -- other procedures;
end case;
```

3. Spontaneous transitions are performed when no interaction is received in a given state. These transitions are considered in the "**else**" clause of the "**if**" construct obtained in the previous item. If the "**wait until**" construct has a "**for x**" clause then, after an interval **x** of time with no interaction received, a spontaneous transition will occur. However, the clause "**for** x ns" is used only for simulation propose. Thus, during the high-level synthesis phase,

we must implement a busy wait procedure that will verify if the condition *condition_interation* has a *true* value.

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 the VHDL *process* construct, thus allowing that the state machine to be performed repeatedly. This state machine will remain suspended while no interaction (signal changing) happens. If more than a process is active at a time, only one process will be selected by the HIERARCHY_QUEUE_ENTITY.

3.2 Estelle language restrictions : Estelle **

The conversion of a generic specification written in full Estelle into VHDL description can result in an inefficient VLSI implementation. Some mechanisms found in Estelle generate overspecified text such as implicit synchronization. As a result, to get an efficient hardware implementation the Estelle language must be restricted.

```
Specification PROD_CONS systemactivity;
  default individual queue ; timescale seconds;    (* ... *)
  const Ndates = 5; type TMSG = array [0 ..Ndates] of char;
  channel tchannel (sendmsg, sendlib);
     by sendmsg: msg (Message : TMSG); by sendlib: lib;

  module TPROD activity; ip INTER : tchannel(sendmsg); end;
  body BPROD for TPROD;
    state P0,P1; var var_msg : TMSG;
    procedure CreateMsg (var msg: TMSG); begin (*... *) end;
    initialize to P0 begin (*...*) end;
      trans
        from P0 to P1 when INTER. lib begin end;
        from P1 to P0 begin CreateMsg (varmsg); output INTER. msg(varmsg); end;
  end (* BPROD*);

  module TCONS activity; ip INTER : tchannel (sendlib); end;
  body BCONS for TCONS;
    state S0, S1;
    initialize to S0 begin end;
      trans
        from S0 to S1 begin output INTER. lib; end;
        from S1 to S0
          when INTER.msg
          begin (*consumer*) end;
end (* BCONS *);

modvar
  P : TPROD; C : TCONS;
  initialize
    begin init P with BPROD; init C with BCONS; connect P.INTER to C.INTER; end;
end (* PROD_CONS *);
```

Figure 7 Estelle specification of the Producer_Consumer Protocol.

```
package PROD_CONS_PACK is
   constant time_scale : time := 1 sec; constant Ndates : integer := 5;
   type TMSG is array (0 to Ndates) of character;
end PROD_CONS_PACK;

use work.PROD_CONS_PACK.all;
entity TPROD is port (msg: out TMSG; lib: in bit) end TPROD;
architecture BPROD of TPROD is
  signal varmsg : TMSG;  type TSTATE is (P0,P1);
begin
    process
      variable state : TSTATE :=P0;
      procedure CreateMsg (var msg : out MSG_TYPE); begin end;
      begin
        case state is
          when P0 => wait until lib = '1'; State := P1;
           when P1 => CreateMsg (varmsg); msg<=varmsg; State := P0;
        end case;
    end process;
end BPROD;

use work.PROD_CONS_PACK.all;
entity TCONS is port (msg : in TMSG; lib: out bit); end TCONS;
architecture BCONS of TCONS is
    type TSTATE is (S0,S1);
begin
    process
      variable state : TSTATE :=S0;
      begin
        case state is
          when S0 => lib <= '1'; state := S1;
           when S1=> wait until msg;
                     -- consumer
                     state:=S0;
        end case;
    end process;
end BCONS;
use work.PROD_CONS_PACK.all;
entity PROD_CONS is end PROD_CONS;
architecture structure of PROD_CONS is
    component TPROD  port (msg : out TMSG; lib : in bit);  end component;
    component TCONS  port (msg : in TMSG; lib : out bit);  end component;
    signal Pmsg, Cmsg : TMSG;  signal Plib,Clib : bit;
    begin
       P: TPROD   port map (msg => Pmsg,lib =>Plib);
       C: TCONS   port map (msg => Cmsg, lib =>Clib);
       Cmsg <= Pmsg;  Plib <= Clib;
    end structure;
configuration PROD_CONS_BEHAVIOUR of PROD_CONS is
   for structure
     for P : TPROD use entity work.TPROD (BPROD); end for;
     for C : TCONS use entity work.TCONS (BCONS); end for;
   end for;
end structure;
```

Figure 8 VHDL specification of the PRODUCER_CONSUMER protocol.

The definition of a set of restrictions Estelle in order to obtain an efficient conversion into VHDL resulted in a language version called Estelle. This version includes some propositions made by Courtiat in Estelle* and others that improve the conversion to VHDL**. The restrictions imposed to Estelle are:

- In static configurations among module instances, considere these conditions: there is not a priority clause associated to any transition defined by the ***when*** clause and the only active module instances are the leaves of the module instances tree. This conditions impose some restrictions to the utilisation of some Estelle constructs. These requirements allow to represent in an asynchronous form the parallelism among module instances. The relationship of module instance is not of the ascendant/descendent kind.
- The other communication mechanism used is the rendez-vouz [Courtiat(1988)].
- Restrict the use in Estelle of the ***delay*** clause to the forms ***delay(E1)*** and ***delay***(E1,E1).

An Estelle** and the resulting VHDL specification for the PRODUCER-CONSUMER protocol are presented in figures 7 and 8.

These restrictions will result in a more efficient conversion to VHDL and thus to simpler VLSI circuits. The conversion of an Estelle** specification into a VHDL specification is done using the same general rules already presented for the conversion of an Estelle specification into VHDL with some restrictions: the exported variables, the process type modules and the Estelle clauses "wait interaction_point.interaction" and "output interaction_point.interaction (msg)" will not be implemented.

In order to implement in VHDL the rendezvous mechanism between two VHDL entities a simple protocol is needed. The code for the synchronisation is then given in figure 9.

Estelle:	Activity Module 1 ("transmitter")	Activity Module 2 ("Receiver")
	IP ! ConReq	IP ? ConReq
VHDL:	Entity 1 ("transmitter")	Entity 2 ("Receiver")

	Tx_ready <= 1;
	wait until rx_ready = "1";
	data <= 7;
	tx_ready <= 0;
	
	wait until rx_ready = "0";
		wait until tx_ready = "1";
		rx_ready <= "1";
		
		wait until tx_ready = "0";
		aux <= data;
		rx_ready <= "0";

Figure 9 Communication mechanism (Rendezvous) in ESTELLE and in VHDL

4 - A HIGH PERFORMANCE PROTOCOL : ABRACADABRA-HS

The ABRACADABRA-HS protocol is based on the ISO's ABRACADABRA reference protocol and was created to demonstrate the proposed methodology. This protocol is a more complex example than the producer-consumer protocol. ABRACADABRA_HS protocol has different state machines running parallel and thus has many interesting problems that should be treated by a high-level synthesis environment. The translation shown in figure 10 is a base for our future work.

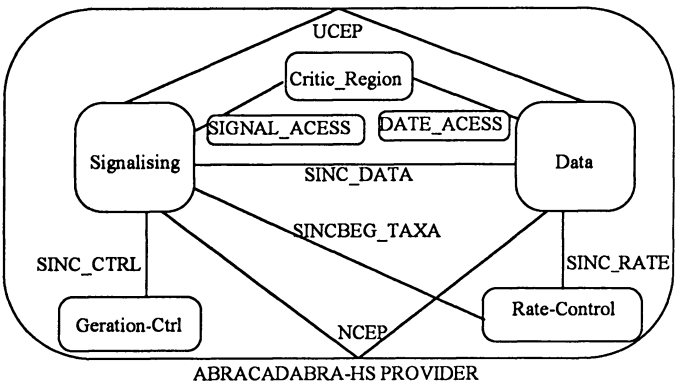

Figure 10 Architecture of ABRACADABRA_HS provider

```
...
body BABRA_HS_ENTITY for TABRA_HS_ENTITY;
channel SINC_CTRLTIMER (mod_ctrl,mod_timer); by mod_ctrl: ConReq;DisReq; by mod_timer: CtrlReq;
...
module TSIGNAL_ENTITY activity; ip SINC_CTRL : SINC_CTRLTIMER(mod_ctrl); (*...*) end;
body BSIGNAL_ENTITY for TSIGNAL_ENTITY; (* ... *) end;
...
module TCTRL_ENTITY activity; ip SINC_CTRL : SINC_CTRLTIMER(mod_timer); end;
...
modvar
  SIGNAL: TSIGNAL_ENTITY; CTRL: TCTRL_ENTITY;
    ...
  initialize
   begin
    init SIGNAL with BSIGNAL_ENTITY; init CTRL with BCTRL_ENTITY;
    connect SIGNAL.SINC_CTRL to CTRL.SINC_CTRL;
    ...
   end;
  trans
    ...
end; (*ABRA_HS_ENTITY_BODY *)
```

Figure 11 Estelle specification of part of the ABRACADABRA_HS protocol

The ABRACADABRA_HS protocol is a connection oriented protocol that joins characteristics of a number of high performance protocols as XTP, DATAKIT.
The ABRACADABRA-HS specification in Estelle is structured into three levels: the user level; the media level; and the ABRACADABRA-HS protocol. The ABRACADABRA-HS protocol, as illustrated in figure 10, is structured in five modules called Signalising, Data, Generation-ctr, Critic_Region and Rate-Control. The *Signalising module* is responsible for the message signalising management, that includes the message used during the connection and the disconnection association phases and also the control messages. The *Data module* is responsible for the data message management. The *Generation-Ctrl* module, when active, is

responsible for the periodic generation of a signal to the Signalising module. The Critic-Region module is responsible for certify the mutua exclusion in the access of the global variable. The *Rate_Ctrl module*, when active, is responsible for the periodic generation of a signal to the Data module. In figure 11 and 12 are showed an Estelle specification and a VHDL specification of part of the ABRACADABRA-HS protocol applied conversion methodology.

```
...
architecture ABRA_HS_ENTITY_BODY of ABRA_HS_ENTITY_TYPE is
  component TCTRL_ENTITY    port (ConReq, DisReq: in bit;CtrlReq: out bit);       end component;
  component SIGNAL_ENTITY   port( CConReq, CDisReq: out bit;CCtrlReq: in bit);  end component;
  ...
  signal SC_ConReq, SC_DisReq, SC_CtrlReq: bit;
  begin
   CTRL: TCTRL_ENTITY
     port map(ConReq => SC_ConReq, DisReq => SC_DisReq,CtrlReq =>SC_CtrlReq);
   SIGNAL: SIGNAL_ENTITY
     port map(CConReq => SC_ConReq,CDisReq => SC_DisReq,CCtrlReq =>SC_CtrlReq);
   ...
end ABRA_HS_ENTITY_BODY;
```

Figure 12 VHDL specification of part ABRACADABRA_HS protocol

5 FINAL CONSIDERATIONS

This paper presented a methodology to obtain a semiautomatic implementation of integrated circuits from an Estelle specification. Initially, the Estelle constructs were analysed and a simplified version of Estelle was proposed, the Estelle**. This simplified version allows efficient Estelle specifications and easy translation to VHDL. Then, the rules for a good match between the language's semantic model to be used in a communication protocol and the architecture created after the synthesis was established together with the rules to obtain a VHDL system from its Estelle specification. Finally, the proposed methodology was applied to an example, the high performance ABRACADABRA-HS protocol.

The proposed methodology integrates an Estelle C.A.D. with a micro electronic C.A.D. The Estelle C.A.D. developed by COPPE/UFRJ has a set of tools to perform the verification and to generate test sequences. The micro electronic C.A.D. (AMICAL plus Synopsis) is a high level synthesis environment that presents suitable characteristics to the Estelle specification conversion. The implementation in VLSI hardware of the high performance protocol proposed, the ABRACADABRA-HS, will be obtained by using the tools made available by the National Polytechnique Institute of Grenoble in France from a technological co-operation agreement.

VLSI implementations of high speed protocols will certainly have a better performance than its software implementation counterparts. The obtained architectures will be evaluated. Future work includes: to provide a solution to the implementation of a system described by a set of parallel state machines in the AMICAL environment, to provide the high-level synthesis of the ABRACADABRA_HS protocol using the AMICAL environment and to evaluate the ABRACADABRA_HS protocol architecture obtained.

6 REFERENCES

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

Braun T. , Shiller J. et Zitterbart M. (1994) Implementation of Transport Protocols using Parallelism and VLSI Components. SBT/IEEE ITS 94, Rio de Janeiro, Brasil.

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

Chanson et al, (1993) On tools supporting the use of formal description techniques in protocol development. Computer Networks and ISDN Systems, **25**, 723-39.

Courtiat, J. P. Estelle* (1988) A powerful Dialect of Estelle for OSI Protocol description, International Workshop on Protocol Specification verification and testing, France.

Doeringer, W. et al (1990) A Survey of Light-Weight Transport Protocols for High-Speed Networks. IEEE transactions on Communications, vol. 388, NO. **11**, November, 2025-38.

Doshi, B. and Johri, P.(1992) Communications Protocols for High Speed Packet Networks. Computer Networks and ISDN Systems, **24**, 243-73.

Fraser, A. Marshall, W. (1989) Data Transport in a Byte Stream Network. IEEE Journal on Selected Areas in Communications, Vol. 7, NO.7, September.

ISO/TC97/SC21/WG16-1, (1987) Estelle : A formal Description Technique Based on a Extended State transition Model, DP9074.

Gajski, D. D. et al, (1992) High - Level Synthesis : Introduction to Chip and System Design. Kluwer Academic Publishers.

Kloos, C. D. et al, (1993) VHDL generation from a timed extension of the formal description technique LOTOS within the FORMAT project. Microprocessing and Microprogramming, **38**, 589-96, North-Holland.

Krishnakumar, A. S. and Sabnani K.K. (1989) VLSI Implementation of Communication Protocols - A survey. IEEE Journal on Select Areas in Communications,Vol. 7,7,1082-90.

Navabi, Z. (1992) A high-level language for Design and Modeling of Hardware. J. System Software,**18**, 5-18.

Netravali, A. et al (1990) Design and Implementation of a High-Speed transport Protocol. IEEE transaction on Communications, Vol. 38, No. **11**, November, 2010-24.

Park, O'Brien,K., Jerraya A. A. (1992) Tutorial - AMICAL : Interactive Architectural Synthesis Based on VHDL. Internal Report, IMAG/TIM3, March.

Strayer W. et al, (1985) XTP: The Xpress Transfer Protocol, Addison-Wesley Publishing Company.

Vissers C. et al (1983) Formal Description Techniques.Proceeding of IEEE,Vol. 71,**12**,1356-64.

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 prepares a D. Sc. thesis at COPPE/UFRJ. Current research interests: Formal Specification, Verification and Implementation of Protocol.