Design for Hardware Implementation of Communication Protocols from a Formal Description Technique

R.N.B Lima [a], E. Carli [a], A.C.P. Pedroza [a,c], L. Pirmez [b] and A.C. de Mesquita [a]

[a] *Grupo de Teleinformática e Automação (GTA), COPPE/UFRJ - Program of Eletric Engineering, Tel: +55 21 2605010, P.O.Box 68504, 21945-970 - Rio de Janeiro - RJ - Brasil*

[b] *NCE/UFRJ - Electronic Computer Center*

[c] *EE/UFRJ - Department of Electronic*

Abstract

High performance in communication subsystems is obtained with software optimization of high speed protocols and with their implementation in hardware. This work follows a methodology that efficiently translates specifications described in a Formal Description Technique (Estelle) into VHDL, allowing hardware implementation and rapid prototyping of protocols. The implementation results of different description styles of protocols, the simulation results and constraints in the use of this translation strategy are discussed. The results include area analysis and clock frequency evaluation of synthesized hardware of the ABRACADABRA, AAL5 and IPv6 protocols.

Key words: Formal Description Technique, Hardware Implementation, High Level Synthesis.

1 Introduction

Advances in optical transmission technologies, the emergence of new technologies in VLSI, the high efficiency of nodes (commuters) and the reduction of memory and processor costs are leading to the development of faster and more reliable communication networks. In these networks, the performance

Preprint submitted to Elsevier Preprint *29 January 1999*

bottleneck becomes the amount of data processing needed to run protocols in workstations and network servers and not on the transmission capacity of the physical media.

Multimedia applications and services generate additional demand for data processing that require modifications in the design approach of protocols. In this case, the protocols need to be simple to allow high-speed operations and should be more efficient, as far as data processing is concerned, in order to support multimedia traffic. These performance requirements may not always be possible to obtain by traditional protocol design and software optimisation. This suggests alternative protocol design approaches that assume the hardware implementation of the protocol controllers.

In this paper, a design for hardware implementation approach to protocol design is proposed, based on the use of Estelle language [1,2] , VHDL (*Very High-speed Integrated Circuits Hardware Description Language*) [3,4] and High Level Synthesis (HLS) techniques for digital systems design automation [5].

The hardware implementation of protocols is not a trivial task. Integrated circuits are currently designed in a semi-automatic way 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 description. Generally, the design process of communication protocols involves system specifications written using a Formal Description Technique (FDT) [6–8] such as the ISO (*International Standard Organization*) Estelle language. These languages allow to generate specifications in a level of abstraction even higher than that commonly attained in the description of digital systems. As a result, the high speed protocol designers will face the problem of interacting with two design environments: one for protocol specification, verification and testing and the other for 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.

In the proposed methodology the design process assumes as input description a correctly compiled and simulated Estelle specification, obtained in a suitable protocol design environment, that is mapped into a VHDL behavioral description, similar to the protocol functional specification which is translated into a RTL (*Register Transfer Logic*) equivalent level through the Synopsys Behavioral Compiler tool [9]. The resulting lower-level design specification is then used as input of the logical synthesis tool Synopsys [10] and finally the Opus [11] tool generates the circuit layout. In this work three protocols - the ISO reference protocol ABRACADABRA, the IPv6 and AAL5 protocols [12–14] - are synthesized as a means to evaluate the complexity of the resulting hardware.

The paper is organized as follows. Section 2 presents the main characteristics of the Estelle and VHDL languages. The HLS concept is presented in section 3. In section 4, a specific and important step of our approach is outlined: a strategy to convert Estelle specifications into VHDL descriptions. Different description styles for protocols in VHDL efficient for HLS are presented in section 5. Section 6 contains the design for hardware implementation of protocols concepts and the main steps involved in the application of this methodology are discussed. In section 7, the obtained results are examined for the effectiveness of the hardware implementation proposed in terms of area the synthesized circuits area and clock frequency. Final considerations are drawn on section 8.

2 Estelle and VHDL Languages Characteristics

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 interest of protocols standardised by ISO encouraged researchers to improve protocol engineering in the past.

The importance of formal methods in protocol engineering was noticed since the beginning, particularly in the specification of communication protocols. The term FDT was created 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 of FDTs researches, the ISO and the ITU-T (*International Telecommunication Union*) contributed to the development of standardised FDTs. As examples, Estelle and LOTOS were defined by ISO, and the SDL language by ITU-T.

2.1 The Estelle Language

The Estelle language is a FDT developed by ISO. In Estelle, a *specification* is composed of a set of *modules*. A module can be refined in submodules, defining a hierarchical structure among themselves. A module is composed of a *header* and a *body*. The interaction points, the exported variables and the class attribute are defined in the header. The module behavior 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.

2.2 The VHDL Language

VHDL is a standard language developed by the U.S Department of Defense (DoD) to describe digital eletronic systems. The VHDL language is a multilevel language that allows structural and behavioural descriptions. The structural description allows to describe the structure of a design, its decomposition in subdesigns and the interconnection among them. The behavioural description allows the specification of the function of a design using familiar programming language structures.

The basic component of a VHDL description is the *entity/architecture* pair. A VHDL “entity” describes, through a set of input/output ports, the external interface of a given system. Ports are the signals that enable the exchange of information between VHDL entities. The “architecture” describes the entity’s function either in the behavioral or the structural domains. For an efficient information exchange between VHDL entities describing protocols it is necessary some kind of synchronization among them. Thus an entity will not be allowed to send data to another one which is enabled to receive the data. This synchronization is stablished by control signals included in both entities.

3 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 of research lead to the development of the silicon compilation technique, allowing an integration hardware layout from a high level specification.

A VLSI integrated circuit can be described by one of three representation domains: the *behavioral*, the *structural* and the *physical* domain. At the behavioral domain the functionality of the system is described without refering any structure. At this domain, a design corresponds to one or more black boxes where only their external interface and a set of functions are considered. The structural domain is a bridge between the behavioral and the physical domains. The structural domain specifies how components of a design are connected together to effect the prescribed behavior. At the physical domain what a design does is ignored. This domain specifies how to actually build a structure that has the required connectivity to implement the prescribed behavior.

Each domain can be hierarchically divided in several abstraction levels, causing the design of integrated circuits to be classified by their domain and by their level. These domain are discussed in a clearer way through the Gajski [5] Y-chart, shown in figure 1. Concentric circles around the center indicate the various levels of abstraction that are common in a integrated circuit design. By convention, as far it is from the center, more abstract becomes the description level.

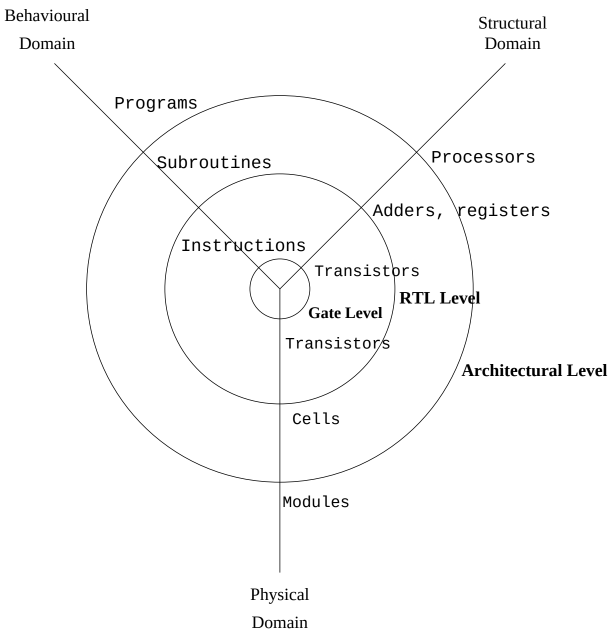

Fig. 1. The Y chart.

High Level Synthesis (HLS) corresponds to a behavioral-to-structural specification translation in a higher level than RTL. In this work will be employed the Behavioral Compiler tool to map a high-level circuit design specification to a register transfer-level (RTL) equivalent level and then to compile the lower-level design specification with the Synopsys RTL and gate-level logic tools. Synopsys Behavioral Compiler synthesizes hardware by:

- scheduling operations, I/O, and memory accesses to cycles;
- allocating hardware by assigning variables to registers and assigning operations to DesignWare modules;
- creating finite state machine (FSM) and memory-interface control logic;
- timing all components according to the HDL functional dependencies.

Instead of starting a design at RTL level, HLS begins at the algorithmic level. A behavioral VHDL specification contains operations, variables, and arrays similar or identical to the original software algorithm. The behavioral description, however, adds an I/O specification capability to the simple functional description.

The target architecture of HLS is a general computing model that contains data path, memory, and control elements. Designers no longer manually build the data path out of registers, higher order functional units (such as adders and multipliers), and multiplexers. Additionally, designers no longer need to build the controller at the RTL to sequence and control the actions of the data path and memory I/O.

Synopsys Behavioral Compiler builds this architecture by using automated methods of scheduling, hardware allocation, register sharing, memory and control inferencing. Designers do not need to specify the exact architecture of a design, and they can explore many implementations to find the optimal architecture.

The advantages of HLS use are:

- shorter specification time because the VHDL behavioral description is similar to the original algorithm and less complex than an equivalent RTL specification;
- the process is target architecture independent;
- faster simulation because the behavioral description is more abstract than with RTL;
- architectural exploration, with quick creation and evaluation of many implementations;
- higher quality of results.

4 The Estelle-to-VHDL Translation Strategy

This section outlines a Estelle-to-VHDL translation strategy proposed by Pirmez [15,16] which is able to make an efficient and easy conversion from a Estelle specification protocol into VHDL descriptions. This strategy is an important step of the proposed Design for Hardware approach and has the advantage of reducing the design time.

The translation strategy can be seen in figure 2. An Estelle specification is composed of a set of modules hierarchically organized while a VHDL description is composed of a set of entities described separately. Consequentilly, each Estelle module is mapped into a VHDL entity, as shown in mapping 1. The hierarchy among Estelle modules is mantained through a VHDL *component* declaration statement, as shown in mapping 4. Each VHDL *component* statement declares an entity and its description is placed in design libraries. The VHDL *component instantiation* construct is used to create VHDL entities instances (as Estelle *init* function) whose ports are either connected to actual signals or to the entity ports (as Estelle *connect* and *attach* functions), as shown in mapping 5. The binding of a VHDL entity to its instance is achieved by the *for all* declaration.

Each Estelle module is composed by two parts: the header (*module*) and by

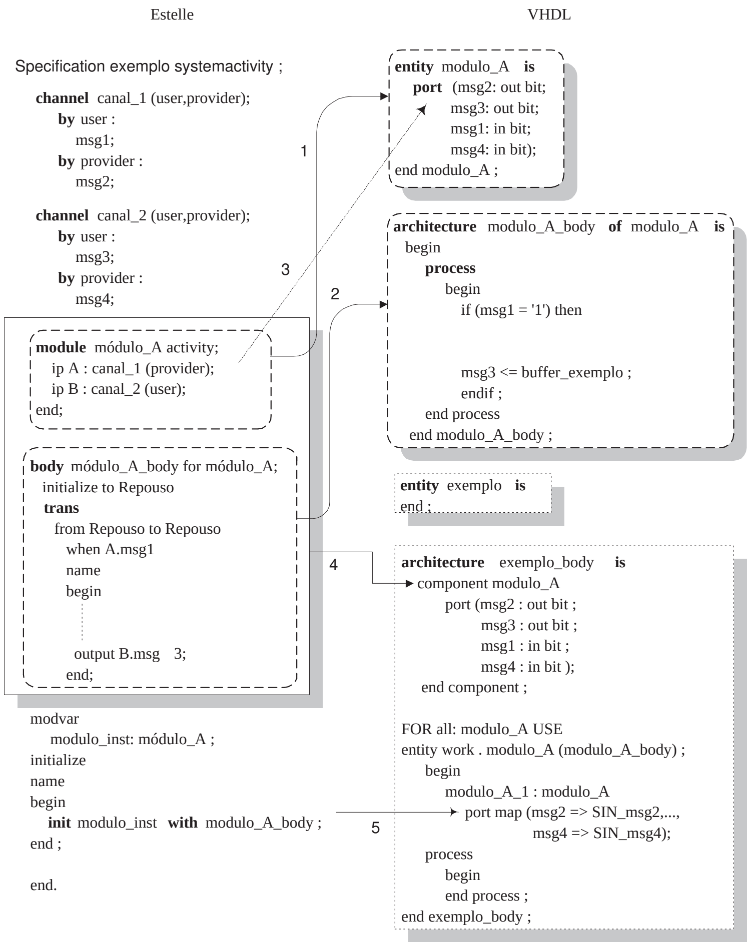

Fig. 2. The Estelle-to-VHDL translation strategy.

one or more bodies (*body*). The external view of a module is defined at the body and the internal behaviour of a module is described at the body. This corresponds to the VHDL entity-architecture pair, with each Estelle header corresponding to a VHDL entity and each Estelle body corresponding to a VHDL architecture, as shown in mappings 1 and 2, respectively. The main Estelle module (*Specification*) does not have an external view and, for this reason, its corresponding interface is empty.

Since there are no VHDL equivalent mechanisms to the Estelle synchronous and asynchronous types of parallelism among module instances, an efficient Estelle-VHDL translation will require the restrictions included in [17]. This allows to represent in asynchronous form the parallelism among activity type module instances within a system activity type specification. The Estelle language has a built-in FIFO queue mechanism that has no equivalent in VHDL. A solution adopted and implemented to overcome this limitation may be found in [18].

In the header of an Estelle module, each interaction contained in the set of interactions of each interaction point will correspond a port (*signal*) of a VHDL entity. The signal type will be *input* or *output* if the interaction point role is *reception* or *transmission*, as shown in mapping 3.

The behavior of an Estelle module is described by the initialization and transition parts of the body. The execution of both parts is sequential. The translation of Estelle clauses into VHDL structures is trivial as follows:

- an Estelle "**from** *state*" clause is directly converted into a VHDL "**when** *state => ... of case structure*" clause;
- an Estelle "**when** *interaction-point*" clause is directly converted into a VHDL "**wait until** *interaction_condition*" clause;
- an Estelle "**provided** *expression*" clause is directly converted into a VHDL "**if** *expression* **then...**" clause;
- an Estelle "**to** *state*" clause is directly converted into a VHDL "**state** *:= state value*" clause;
- an Estelle "**output** *interaction_point.interaction(msg)*" clause is directly converted into a VHDL "*interaction <= msg*" clause;
- the Estelle "**procedure** *name (formal parameters queue)*" and "**function** *name (formal parameters queue)*" instructions are directly converted into the equivalent VHDL declarations.

The body of an entity will be built in accordance with a behavioral 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. The transitions should be joined at the two options: from the states of state machine or from the received interactions by the module. The VHDL state machine will be built with "*case*" statement or with "*if..then..elsif..end if*" construct. The use of different types of structures will generate differents description styles in VHDL.

5 Description Styles for Protocols in VHDL

A protocol can be described by many different VHDL behavioral description styles [19,18]. The styles provide semantically equivalent descriptions that differ only in the quality of the resulting hardware. The quality of HLS output description is measured by the area analysiss and by the number of control states and control inputs/outputs generated by the scheduling process performed by the Synopsys Behavioral Compiler tool. It will be shown that the quality of the HLS output description will depend of the type of statements used and of the order in which they will appear in the input description.

To illustrate this fact consider the ISO's ABRACADABRA reference protocol modelled by a state machine that performs operations with another state machine. These operations are responsible for the opening/closing connection process. This protocol has two phases: the initialization phase and the 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.

This model will be used to illustrate the several description styles and the consequences on the HLS results. The mains tasks involved in HLS process are scheduling and hardware allocation. The analysis will be performed in the synthesized circuit for the Synopsys Behavioral Compiler tool and Design Compiler tool.

The First description style for implementing a state machine associates a VHDL "*wait until...*" statement to each branch of a "*case state*" statement and the different output control signals are reseted inside this "*case*" clause. Figure 3 shows the VHDL description for the first style.

```
...
process
begin
   -- initialization
   MEF: loop
     case state is
       when state_x =>
           next <= '1';
           wait until input_x = '1';
           next <= '0';
           -- processing
           out_x <= '1';
           wait until clk'event and clk='1';
           out_x <= '0';
       when state_y =>
            next <= '1';
            wait until input_y = '1';
            next <= '0';
            -- processing
            out_y <= '1';
            wait until clk'event and clk='1';
            out_y <= '0';
      end case;
   end loop;
end process;
```

Fig. 3. VHDL process for the first description style.

The Second description style for implementing a state machine is a variation of the first one. In this case, a "*wait until...*" statement is placed outside the "*case*" statement. Figure 4 shows the VHDL description for the second style.

The Third description style employs an "*if..then..elsif..end if*" construct instead of a "*case state*" statement into a *process* statement. This construct uses the *state* as the outer variable. A "*wait until...*" statement is placed outside the nested if clauses. The VHDL description for the third style is showed in figure 5.

The Fourth description style is a variation of the third one. It also employs an

```
...
process
begin
  -- initialization
  MEF: loop
    next <= '1';
    wait until input_x = '1' or input_y = '1';
    next <= '0';
   case state is
     when state_x =>
       if input_x = '1' then
         -- processing
         out_x <= '1';
         wait until clk'event and clk='1';
         out_x <= '0';
       end if;
     when state_y =>
       if input_y = '1' then
         -- processing
         out_y <= '1';
         wait until clk'event and clk='1';
         out_y <= '0';
       end if;
    end case;
  end loop;
end process;
```

Fig. 4. VHDL process for the second description style.

```
...
process
begin
  -- initialization
  MEF: loop
    next <= '1';
    wait until input_x = '1' or input_y = '1';
    next <= '0';
    if state = state_x then
       if input_x = '1' then
         -- processing
         out_x <= '1';
         wait until clk'event and clk='1';
         out_x <= '0';
       end if;
    elsif state = state_y then
       if input_y = '1' then
         -- processing
         out_y <= '1';
         wait until clk'event and clk='1';
         out_y <= '0';
       end if;
    end if;
  end loop;
end process;
```

Fig. 5. VHDL process for the third description style.

"*if..then..elsif..end if*" structure but with the *input* as the outer signal. The VHDL decsription for the fourth style is showed in figure 6.

The Fifth alternative to describe a state machine is a variation of the Second description style. In this style, the different output control signals are reseted immediatly after the "*case state*" statement. Figure 7 shows the VHDL description for the fifth style.

The Sixth style uses the "*case state*" statement for implementing the state machine in VHDL. This style employs the "*if..then..elsif..end if*" structure

```
...
process
begin
   -- initialization
   MEF: loop
      next <= '1';
      wait until input_x = '1' or input_y = '1';
      next <= '0';
      if input_x = '1' then
           if state = state_x then
               -- processing
               out_x <= '1';
               wait until clk'event and clk='1';
               out_x <= '0';
           end if;
      elsif input_y = '1' then
           if state = state_y then
                -- processing
                out_y <= '1';
                wait until clk'event and clk='1';
                out_y <= '0';
           end if;
      end if;
   end loop;
end process;
```

Fig. 6. VHDL process for the fourth description style.

```
...
process
begin
   -- initialization
   MEF: loop
      next <= '1';
      wait until input_x = '1' or input_y = '1';
      next <= '0';
     case state is
        when state_x =>
           if input_x = '1' then
               -- processing
               out_x <= '1';
           end if;
        when state_y =>
           if input_y = '1' then
                -- processing
                out_y <= '1';
           end if;
      end case;
      wait until clk'event and clk='1';
      out_x <= '0';
      out_y <= '0';
   end loop;
end process;
```

Fig. 7. VHDL process for the fifth description style.

instead "*wait until...*" statement. The VHDL description for the sixth style is showed in figure 8.

These different description styles are introduced in Synopsys Behavioral Compiler tool and these results are used by the Design Compiler tool. Different circuits are synthesized and the complexity of the resulting hardware will be analyzed in the section 7.

```
...
process
begin
   -- initialization
   MEF: loop
      next <= '1';
      if input_x = '1' or input_y = '1' then
         next <= '0';
        case state is
          when state_x =>
               if input_x = '1' then
                    -- processing
                    out_x <= '1';
               end if;
          when state_y =>
               if input_y = '1' then
                     -- processing
                     out_y <= '1';
               end if;
         end case;
      else
         out_x <= '0';
         out_y <= '0';
      end if;
   end loop;
end process;
```

Fig. 8. VHDL process for the sixth description style.

6 The Proposed Protocol Design Methodology

The steps involved in the proposed methodology of design for hardware implementation of protocols are summarized in the diagram of figure 9. This methodology starts from a suitable environment to the development of protocols in Estelle, the CAD Estelle. After the generation of an Estelle specification correctly simulated (1,2), this specification is converted into VHDL using the Estelle-to-VHDL Translation Strategy (3) viewed in section 4.

This VHDL description will be used as an input (4) to a micro electronic CAD, called IC-CAD. The IC-CAD is a suitable environment to the design of integrated circuit using HLS. It should be noted that the two environments, the CAD EStelle and the IC-CAD, are independent. The IC-CAD uses a set of standard tools for VLSI circuit design. This includes the Synopsys Behavioral Compiler tool, Synopsys logic synthesizer and Opus layout editor.

The Synopsys Behavioral Compiler is a tool that maps a high-level circuit design specification into a RTL or equivalent level and then to compile the lower-level design specification with the Synopsys RTL and gate-level logic tools. Synopsys can be used also to simulate VHDL behavioral and structural descriptions and to perform logical synthesis (5) of structural descriptions. Opus is applied to the layout generation (6) of the VLSI structures.

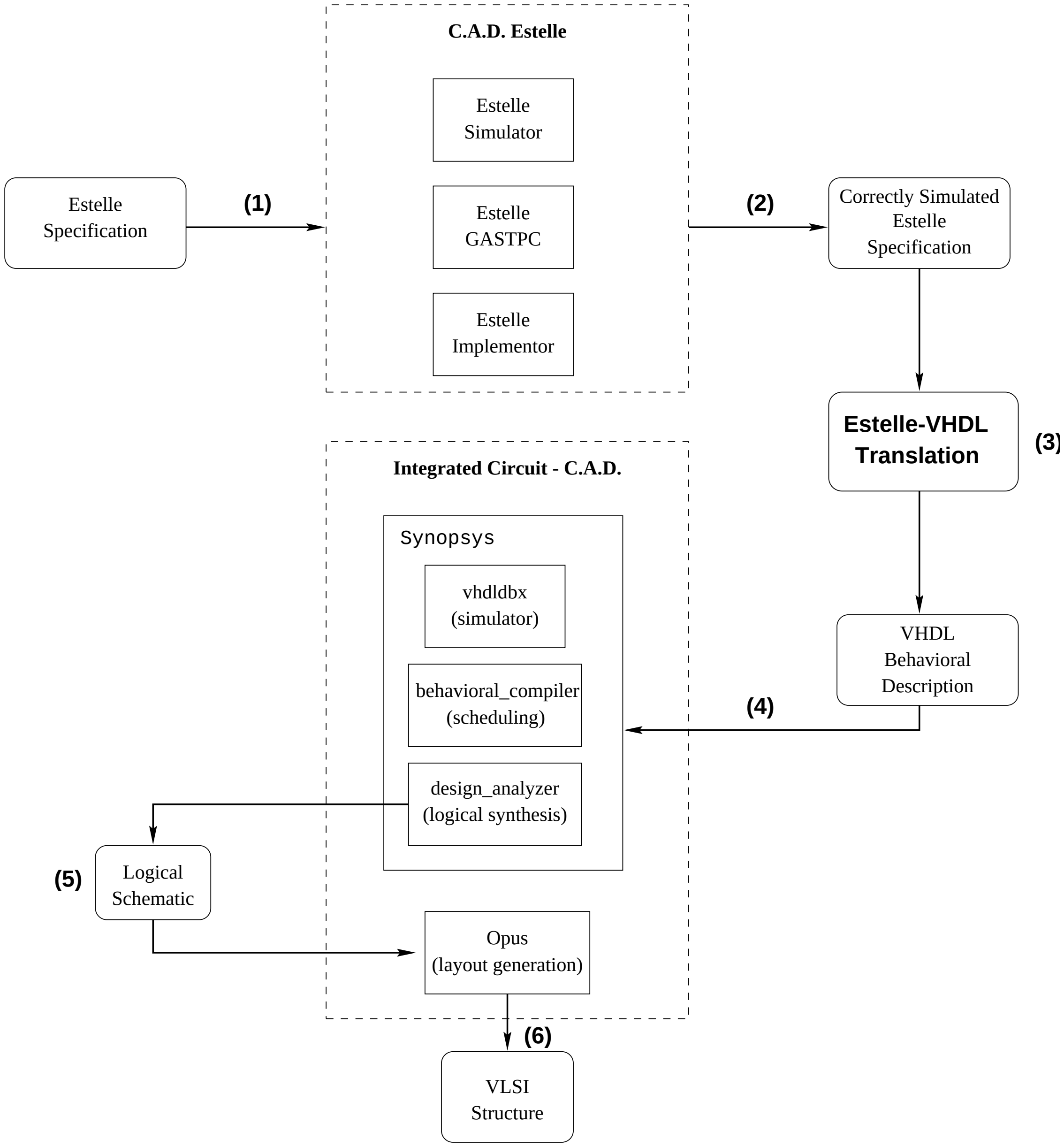

Fig. 9. The proposed design methodology.

7 Results

In this section the results obtained by the application of the proposed methodology for the synthesis of communication protocols based on the Estelle-VHDL translation strategy are discussed.

The topics of concern are:

- the simulation process of the Estelle and VHDL specifications of protocols;
- the effect of the different VHDL behavioral description styles in the synthe-

sized hardware;

- the precautions that one should have when using integer variables and I/O signals in VHDL descriptions;
- the data storage in protocols.

7.1 Simulation Results

Simulation is an important step when specifying and analyzing protocols. This process is used to verify the functionality of the specifications. The input data to the simulation process is a set of Test Vectors generated by a specific tool called Test Sequences Automatic Generator (GASTPC) [20]. This program generates a report with several sequences of send/receive service primitives according to the service offered by the considered protocol. These sequences are inserted in the Estelle Simulator [21] and show all the possible transitions that the state machine should make to accomplish in an effective way the proposed service.

Simulations are performed twice along the synthesis process. Initially at the Estelle specification level and then at the VHDL level, after the Estelle-VHDL translation step. The objective of these simulations is to verify the correct operation of the opening/closing connection process in the ABRACADABRA protocol and of the control signals used for the synchronization of the data transfer process in the AAL5 and IPv6 protocols. In the case under study the performed simulations showed that the Estelle specifications and VHDL descriptions are in agreement with their functional specifications. As an illustration, in the figure 10 it is shown part of the simulation performed in the VHDL description of the ABRACADABRA protocol, focusing the change of primitive used during the connection phase.

7.2 Different Description Styles

VHDL allows the description of a given behavior in many different ways. These descriptions are equivalent from the point of view of the simulation but can generate a wide variety of different hardware structures after synthesis. In order to improve the performance of the hardware generated by the synthesis process it is necessary to adopt a VHDL style of description that takes into account the structure of the scheduling algorithm used by a specific synthesis tool.

As an example, a comparison among the synthesis results generated by the Synopsys Behavioral Compiler tool for six different description styles is shown on table 1. This comparison is done in terms of the number of states corre-

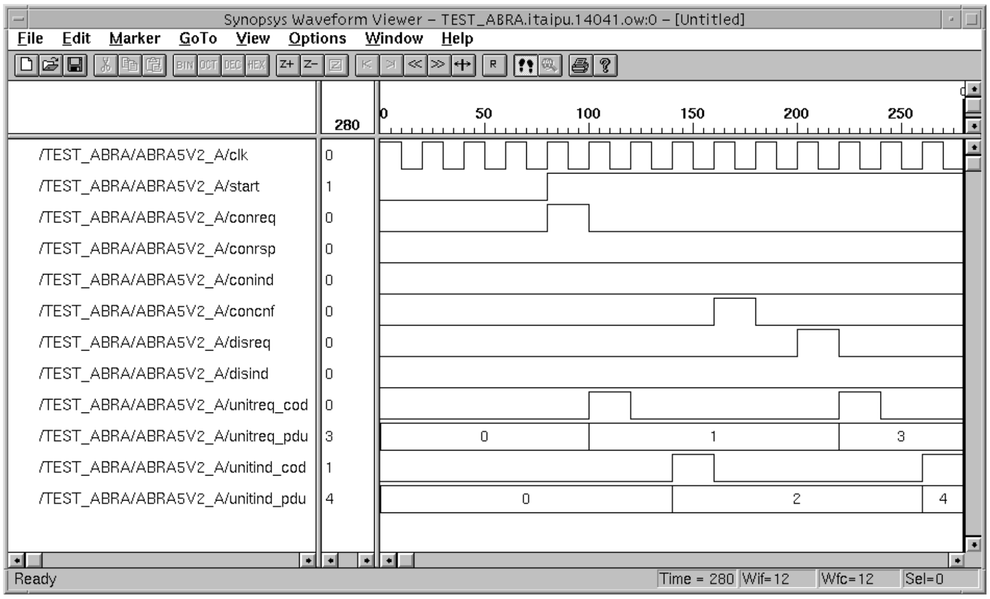

Fig. 10. Simulation of a part of ABRACADABRA protocol.

sponding to the state machines generated by the synthesis algorithm for each description style, circuit area and clock frequency. Note that, the complexity of the synthesized hardware is directly proportional to the number of states extracted by the synthesis tool.

Table 1
Comparison of different description styles.

Styles	**Control States**	**Control Inputs**	**Control Outputs**	**Area (mm^2)**	**Clock Frequency**
First	14	39	80	1,33	125 MHz
Second	9	39	73	1,10	149 MHz
Third	9	37	76	1,13	130 MHz
Fourth	9	39	118	1,37	130 MHz
Fifth	9	39	98	1,27	154 MHz
Thirdversion2	9	37	103	1,32	135 MHz
Sixth	4	31	34	0,66	154 MHz

From the discussion above some important conclusions are drawn:

(1) use the "*wait until...control signals*" statement outside the "*case state*" clause. The comparison among the first and second styles shows that the circuit area decreasing and clock frequency incresing;
(2) use the "*case*" statement instead "*if..then..elsif..end if*" structure. The comparison among the second and third styles and the comparison among

the fifth and thirdversion2 shows that the circuit area decreasing and clock frequency incresing;

(3) comparing the third and thirdversion2 styles and the second and fifth styles we can observe that reseting the different output control signals inside the "*case*" clause the circuit area decreasing. While reseting the different output control signals immediatly after the "*case*" the clock frequency increasing;

(4) employ the "*if..then..elsif..end if*" with the *state* variable as the outer signal instead the *input* variable. The comparison among the third and fourth styles shows that the circuit area decreasing and the output controls is smaller;

(5) use the "*if..control signals*" structure instead "*wait until...control signals*" constructs. The comparison among the fifth and sixth styles shows that the circuit area decreasing.

The figure 11 illustrate the logical schematic of the connection/disconnection process of the ABRACADABRA protocol described with the sixth description style. The synthesized circuit has a 0,66 mm^2 estimated area, obtained using the AMS, 0,8μm technology standard cells library.

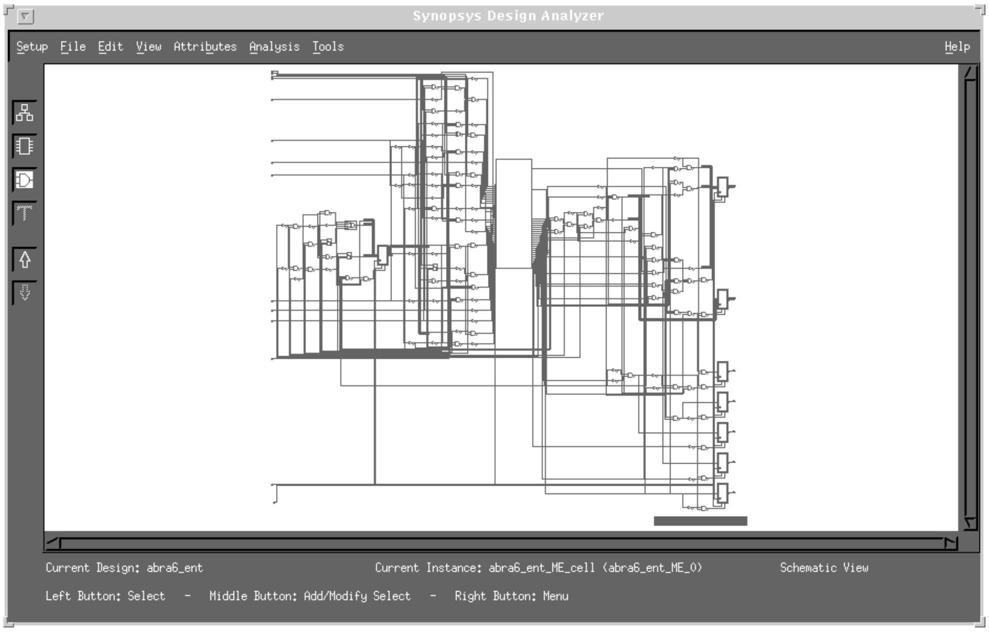

Fig. 11. Logical Schematic of part of ABRACADABRA protocol.

7.3 Integer Variables Constraints

When a software implementation of a protocol is though it is common practice to use integer variables and signals in the formal protocol specification. At the Estelle description level, this should cause no major problems. Nevertheless, when hardware description languages, like VHDL, are used it is necessary to limit the range of each integer variable or signal in order to reduce the number of bits of the registers associated to each variable by the synthesis tool.

For example, if no range is attributed to an integer variable the Synopsys Behavioral Compiler will allocate a default number of 32 bits to each register. To illustrate this fact, two VHDL descriptions of the connection/disconnection process of the ABRACADABRA protocol containing one integer variable were synthesized. In the first description the range of the integer variable was not limited. In the second description the range was set from 0 to 7. The table 2 shows a comparison of the synthesized circuit area and the number of bits allocated to the registers corresponding to each description.

Table 2
Comparison of synthesized circuits.

Description	**Synthesized Registers**	**Area (mm^2)**
Range	1 of 7 bits	1,27
No range	3 of 32 bits	2,37

Observe that in the description with no range limitation, three 32 bits register were associated to the integer variable, while in the other description only a 7 bits register was used. This represents a reduction in area of the synthesized circuit of approximately 50%.

7.4 Data Storage Constraints

The first step in circuit synthesis consists in partitioning the specification in two blocks: the logic structure and the the data storage structure. The data storage structure is generally implemented through an external memory. Thus, before performing the Estelle-VHDL translation all data storage structures corresponding to internal variables, as buffers, should be implement as a separate VHDL entity.

This fact is illustrated in table 3 by the implementation of the SAR module of the AAL5 protocol. This module is responsible for the segmentation of the received data. In this implementation, two buffers are allocated to the storage of characters, the first buffer stores 88 characters and the second one 48 characters. Both were specified by internal variables in the VHDL description.

It should be noted that the Synopsys Behavioral Compiler synthesizes these buffers as a group of registers internal to the circuit. A better solution would be to synthesize the control logic and connect it to an external memory.

Table 3
Comparison of synthesized circuits.

Protocol	**Synthesized Registers**	**Standarcell Memory**	**Total area (mm^2)**
SAR no memory	3 of 704 bits 1 of 384 bits	NO	77,41
SAR with memory	NO	1K x 8bits	9,79

8 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 the circuit designer when describing and simulating hardware and is the used by the majority of synthesis tools. A translation strategy from Estelle into VHDL allowing hardware implementation and rapid prototyping of protocols is proposed. The main motivation for using this methodology is to simplify the protocol designers task.

The advantages of this methodology are:

- the use of a FDT allows the specification and the verification of a given communication system;
- the Estelle-VHDL translation rules are well documentated and very simple;
- it enables the protocol designers to rely on the standard protocol specification techniques while the circuit implementation details are carried on by automatic synthesis procedures;
- the hardware implementation in an application specific circuit and the rapid prototyping of protocols are of concern when high speed protocols are considered.

A comparison among the different description styles has shown that the description style of the hardware in the behavioral level affects the quality of the synthesized circuit. This was evalueted by comparing the circuit area and the clock frequency of the circuit generated for the description style discussed. A number of constraints in the use of integer variables and signals and the problem of storage data implementation were examined. The logical schematic of part of ABRACADABRA protocol considering the most efficient description style was presented.

References

[1] Budkowski, S., Dembinski, P., "An Introduction to Estelle: A Specification Language for Distributed Systems", *Computer Network and ISDN System, North-Holland*, vol. 14, pp. 3–23, 1987.

[2] ISO/TC97/SC21/WG16-1, "Estelle: a Formal Description Technique based on a Extended State Transiction Model", *DP9074*, 1987.

[3] Ashenden, P. J., *The VHDL Cookbook.* Computer Science Dept. of University of Adelaide, 1990.

[4] IEEE Std 1076-1987, *IEEE Standard VHDL Language Reference Manual*, Mar. 1988.

[5] Gajski, Daniel D. et al., *High-Level Synthesis - Introduction to Chip and System Design.* University of California, Irvine: Kluver Academic Publishers, 1992.

[6] King, P., "Formalization of Protocol Engineering Concepts", *IEEE transactions on Computers*, vol. 40, no. 4, pp. 387–403, Apr 1991.

[7] Vissers, C. et al., "Formal Description Techniques", *Procceding of IEEE*, vol. 71, no. 12, pp. 1356–1364, Dec 1983.

[8] Bochmann, G., Sunshine, C., "Formal Methods in Communication Protocol Design", *IEEE transactions on Communications*, vol. 28, no. 4, pp. 624–631, Apr 1980.

[9] Synopsys, Inc., *Behavioral Compiler User Guide Online Documentation, v1998.02*, 1998.

[10] Synopsys, Inc., *Synopsys Online Documentation, v1998.02*, 1998.

[11] Cadence Design Systems, Inc., *Cadence Online Library*, 1994.

[12] Carli, E., Lima, R. N. B., Pedroza, A.C.P. et al., "Implementação em Hardware de Protocolos para Redes de Alta Velocidade a partir de uma Descrição Formal", in *II Simpósio Franco-Brasileiro de Sistemas Informáticos Distribuídos, SFBSID'97*, Fortaleza, CE, pp. 211–220, Nov 1997.

[13] Carli, E., "*Projeto de Protocolos visando a Implementacao em Hardware: aplicacao ao Protocolo IPv6 no Ambito da Abordagem IP over ATM*", Master's thesis, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

[14] Lima, R.N.B., "*Projeto de Protocolos visando a Implementacao em Hardware: Aplicacao ao Caso do Protocolo AAL5 para Redes ATM*", Master's thesis, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

[15] Pirmez, L., *Uma Metodologia para a Implementação de Sistemas Distribuídos em Hardware a partir de uma Descrição Formal.* PhD thesis, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1996.

[16] L. Pirmez, A. Pedroza and A. C. Mesquita, "A methodology to the implementation of distributed system in hardware from a formal description", in *Fifteenth International Symposium on Protocol Specification, Testing and Verification*, Varsow, Poland, IFIP WG 6.1, 1995.

[17] Courtiat, J. P., "Estelle *: A powerful Dialect of Estelle for OSI Protocol description", in *International Workshop on Protocol Specification, Verification and Testing*, Toulouse-Moissac, França, 1988.

[18] L. Pirmez, A. Pedroza and A. C. Mesquita, "High level synthesis of protocols described by a formal description technique", in *IX IFIP International Conference on VLSI'97*, Gramado, RS, Brazil, 1997.

[19] L. Pirmez, M. Rahmouni, P. Kission, A. Pedroza, A. C. Mesquita and A. A. Jerraya, "Analysis of different protocol description styles in vhdl to high-level synthesis", in *EURODAC'96*, Geneve, 1996.

[20] Soares da Silva, G., "*Geração Automática de Sequências de Testes para Protocolos de Comunicação*", Master's thesis, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1992.

[21] Resende Loureiro, F., "*Um Simulador para a Linguagem Estelle*", Master's thesis, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1995.