27

Hardware Specification Generated from Estelle

Jacek Wytrębowicz[*]
Institut National des Télécommunications[†]
9, rue Ch. Fourier, 91011 EVRY, France, jacek@hugo.int-evry.fr

Abstract

A specification written in the ISO standardized Estelle language can be translated into the standard hardware description language VHDL. In this paper we present such a translator and we discuss an example of its application. The Estelle formal description technique is used for specification and validation of communication protocols. VHDL is considered as an intermediate step, using of the existing simulation and synthesis tools.

Keywords

Estelle, VHDL, protocol prototyping

1. INTRODUCTION

The early protocol specification does not distinguish between hardware and software. The system level description methods are used to deal with a task and state nature of protocols. Subsequently the software-hardware partitioning is performed and the specification parts are addressed respectively to software and hardware designing environments. A specification given in a hardware description language can allow to start designing process taking advantage of existing CAD tools. There are many tools, which take on their input a hardware specification, and which are used in designing process. A register transfer level specification can be automatically synthesized. The methods and algorithms for synthesis of a hardware specification given on behavioral level, are under development, and some synthesis systems are offered for designers.

For different designing stages we use system level, software and hardware specification languages. Using the existing system level languages, as Estelle, SDL, CSP, we can relatively easily model hierarchy and inter-process communication, but we can not synthesize hardware

[*]This work was, in part, supported by CNET (93PE7404).

[†]On leave of absence from Warsaw University of Technology, ul.Nowowiejska 15/19, 00-665 Warszawa, Poland.

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

starting from them. Using the hardware description languages (HDLs), as VHDL, Verilog HDL, UDL/I we can immediately interfacing synthesis tools.

The HDLs can be used for hardware specification at a system level, but the languages has some disadvantages. They do not have task abstraction and they require detailed low level interface description in the data manipulation specification. The HDLs, contrary to system level languages, are inefficient for system level modelling - a corresponding HDL description is unseemly large. While software specification languages can not specify hardware and structure.

Several research works are trying to bridge the gap between system level specification and existing hardware CAD tools. One of the explored approaches is to translate a system level specification into hardware environment, e.g., translation from Lotos into VHDL (Delgado Kloos, 1993), from SDL into VHDL (Glunz, 1993), from StateCharts and from Petri nets into VHDL (Woo, 1992).

Second approach is to create an intermediate form such as BIF (Dutt, 1991) or SOLAR (Jerraya, 1994) that allows to build a common platform for different languages, e.g., SDL and VHDL (Jerraya, 1993), and to link the appropriate tools to them.

Another explored approach is to use a hardware language to system-level designing, e.g., (Bauer, 1993) (Benders, 1991). In this approach a subset of VHDL is used with build-in semantics for representing system level properties. These methods use a semaphore protocol to control access to shared resources. These resources can only be accessed via channels that use this protocol.

An approach, the most difficult to introduce and to be accepted, is to create a new specification language, e.g., SpecCharts (Vahid, 1991), TL (Benders, 1993).

Three formal specification languages are the international standards for communication protocols, namely Estelle, LOTOS, and SDL. There are some tools associated with these languages like, compilers, simulators, test generators, and universal programming languages generators. Up till now there are no methods and tools for hardware designing starting from this level of specification. The mentioned simulators allow for verifying the ideas and the algorithms, but do not allow for verifying a designed realization.

This paper introduces the e2v (Estelle to VHDL) translator. It is a continuation of the article (Wytrebowicz and Budkowski, 1994). It shortly discusses the implemented translation model from Estelle to VHDL. The main novelty of this paper consists in describing an application example (simple Estelle specification and the translation results). The example is a small finite state automaton that has been chosen to demonstrate the e2v tool, and to show the equivalence of the Estelle and VHDL simulation results. Introduction of the Estelle and VHDL languages can not be included in this paper due to its size limitation. Readers unfamiliar to these languages can easily find literature on these topics, e.g., (Budkowski, 1987) (Coelho, 1989).

2. ESTELLE TO VHDL TRANSLATION

This chapter describes the translation model, that is implemented in the e2v translator. It discusses the translation of the basic architectural elements of an Estelle specification, i.e., modules, channels, interaction points, and queues. After it deals with the main behavioral elements of an Estelle specification, i.e., module synchronization, time model and transition declarations. An exhaustive description of the translation model is given in (Wytrebowicz, 1994).

2.1 Architectural Part of Specification

The both Estelle and VHDL languages can specify a hierarchical structure of a given system. The structure is build from modules and entities respectively. En Estelle module is an extended finite state machine, which describes a functional element. A VHDL entity may represent any piece of hardware such as a functional block, a printed circuit board, an integrated circuit, and a macro cell, or even a whole system.

Figure 1 shows the obvious mappings between Estelle and VHDL structure elements. An Estelle module definition is composed of a module header declaration (external view), and of a module body declaration (internal view). The external view of an Estelle module (1)* can be translated to the external view of a VHDL entity, i.e., the entity declaration. The internal view of a module (2) can be translated to the internal view of an entity, i.e., architecture declaration. Several body declarations may be given for one header declaration. In this case, several architecture declarations are generated. The Estelle specification module does not have an external visibility, consequently the corresponding entity (3) declaration is empty. A component instance (4) represents a child module instance. A configuration statement (5) gives the binding between the component instance and an entity.

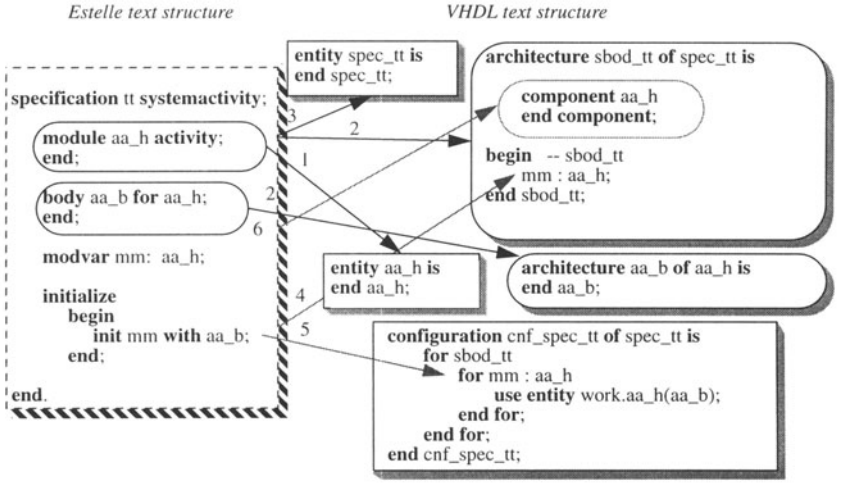


Figure 1 The structure correspondences between Estelle and VHDL.

Channels

An Estelle channel declaration specifies all possible messages, which can be sent in two directions through a connection link. An Estelle messages may be implemented as:

- signal transition (here, the signal means a value that can be read from a wire),
- signal state (e.g. *Reset*), which is active by realization dependent time period, and

*The numbers given in the parentheses are the arrows numbers on Figure 1.

- data packet, which involves a sending and a receiving circuit to synchronize the packet transmission.

The last interpretation is the most general, it can be applied to any messages, no mater of number of parameters and their types. The two first realizations can be applied only for messages, which can be represented by the *bit* data type. Moreover, the implementation time dependences have to guarantee, that no one message will be lost - to satisfy the Estelle unbounded queue model.

We represent a message in VHDL as a value carried by a signal. A pair of VHDL type declarations corresponds to an Estelle channel declaration. Each of these types enumerates values, which are allowed to be sent in one direction. Figure 2 shows a translation example of a channel declaration.

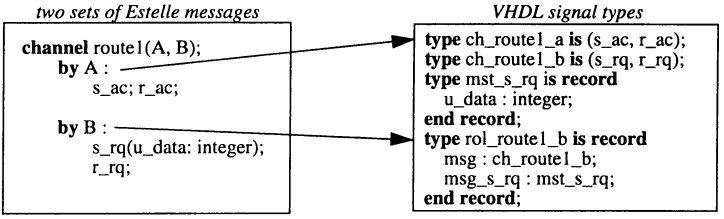


Figure 2 An Estelle channel translation to VHDL signal types.

A VHDL type corresponding to a channel role must carry a message name and all parameters, which belong to each messages of the role under translation. If there is no parameter, then the data type may be an enumeration type (it defines the allowed messages names). Otherwise, the data type must be a record type that assembles:

- the field for a message name - an enumeration type,
- the fields for parameters of each message - record types.

Interaction Points and FIFO Queues

Estelle messages are received by interaction points and are stored in unbounded FIFO queues. There are two categories of interaction points:

- external - they stand for an interface of a module,
- internal - they mainly serve to communicate with child modules.

The external interaction points are translated to VHDL port signals, the internal interaction points are translated to VHDL signals local to an architecture.

A FIFO queue, which keeps the incoming messages, should be declared as an entity, for two reasons: there is no equivalent object in VHDL, and the queue can model very different realizations. The separation of a queue instance simplifies its future substitution by a desired implementation entity.

An implementation of a connection of two interaction points may be a serial or parallel data path, which carry data packets or signals (i.e., values that can be read from a wire). The abstract notion of an unbounded FIFO queue may model a receiver device of any complexity, which is connected with a sender by any sort of physical link. The translation target is to pre-

serve the behavior of the system under design, and to generate a VHDL text, which will be useful in the subsequent synthesis steps.

Figure 3a shows the location of generated queue entities. The queue entity has a fixed size, given as a generic parameter. Two or one queue entity is generated during translation of an Estelle channel declaration. To put (to get) a message on (from) a queue, control signals are needed, as Figure 3b shows.

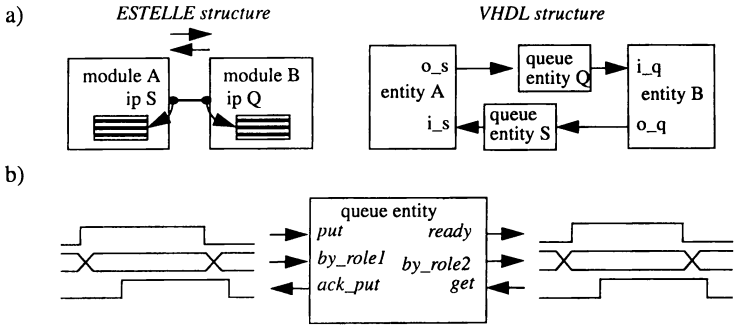


Figure 3 A translation of an Estelle connection a) generated VHDL entities, b) a generated interface of a queue entity.

2.2 Behavioral Part of Specification

An Estelle module performs a sequential process, which composes of an initialization step and an infinite loop of transition selection and transition execution steps. Such a process can be expressed in VHDL as a *process* statement.

A designer can use a nondeterminism and dynamism in an Estelle specification. Usually nondeterminism is used to express several possible implementations. Dynamic Estelle facility is useful to specify dynamic behavior of a system, but neither nondeterminism nor dynamism is translated to VHDL. Any nondeterminism, which can occur in an Estelle specification is suppressed. The generated VHDL design works in a deterministic fashion:

- Nondeterministic choice with respect to a selection of firable transitions is translated to a deterministic choice of ready actions.
- Nondeterministic choice of a child module to work is translated to an asynchronous parallel execution - when it is possible, otherwise it is translated to a deterministic sequential selection.

Since dynamism can not be expressed in VHDL, as a hardware structure is static, a dynamic behavior of Estelle modules may be implemented by dynamic hardware reconfiguration. A switching of connections between entities may implement the dynamic binding of Estelle links. Different connection switching implementations are possible: e.g., three state buses, multiplexers and demultiplexers, or analog switches. The maximal number of implementation components can be easily calculated, because the Estelle interaction points are static. A control logic must be design to allocate components dynamically. However translation of a dynamic behavior into VHDL is possible, the current version of e2v translator does not perform it.

Module Synchronization

The Estelle modules may be attributed *systemprocess*, *systemactivity*, *process*, or *activity*. An attribute of a module determines the synchronization between the modules' children. Moreover, a parent module has priority over his children. The synchronization, which is expressed by module attributes and nesting, is translated into VHDL. Three synchronization signals, between any child entity and its parent one, are generated, as Figure 4 shows.

```
PORT ( -- some signals              -- which correspond to interaction points and exported variables
       run : IN boolean;               -- enables computation which correspond to a transition execution
       live : OUT boolean;             -- states that the computation is under execution
       firable : OUT boolean);         -- states that there is a computation to execute
```

Figure 4 Generated synchronization signals and the example waveforms.

Time Model

The computational model of Estelle is expressed in time-independent terms. It is considered that the selection and execution times of a transition are implementation dependent. A transition execution may involve computation of any complicated algorithm. A corresponding functional block should be a processor with an execution part specialized for performing a given algorithm. Its control part should perform:

- selection of an action to execute, what corresponds to a transition selection,
- control of data passing inside the execution part,
- synchronization with queue entities, and
- synchronization with parent and child entities to satisfy the parent-child priority.

It is obvious, that such complicated functional block should be synchronous, i.e., all internal transactions are performed synchronously to a clock signal. Synchronous designs are more reliable, and easy to debug. Introducing a clock to VHDL description of a module, simplifies the description. Moreover the clock signal is useful to implement an Estelle delay clause.

Steps of the clock signal are needed to control the execution part of an implementation. These steps go through internal states of the implementation. The internal states are irrelevant to the states defined in an Estelle specification, they depend on the implementation of compound statements of the Estelle transitions. The states of an Estelle module are translated to a values that are stored by a VHDL signal. A value stored by the signal corresponds to the current state of a given Estelle module.

Transitions

The initialization and the transition parts of a module describe the module behavior. The initialization part is a set of sequentially executed statements, which are executed once at the creation instant. Next the module performs two actions: transition selection, and transition execution. Firable transitions are selected in a nondeterministic order and executed sequentially. There is no parallelism inside one module, therefore the transitions are not translated to concurrently working VHDL processes. The Estelle initialization and transition parts are represented in VHDL as one *process* statement.

The Estelle semantics allow several transitions to be firable in a given time. In VHDL a transition is represented by an action. Different algorithms can be implemented to select an action to perform, between these that are ready. Our translator generates a simple function, which selects the first ready action from the action list. A designer may change this function to com-

ply with the different implementation needs. The input to the function is a vector of boolean values. Each element of this vector corresponds to one defined transition. An element has true value if the corresponding transition is firable. The selection function returns the number of an action to execute.

3. APPLICATION EXAMPLE OF THE E2V TRANSLATOR

The aim of this chapter is to illustrate the correspondences between Estelle and VHDL specification and to show the equivalence of simulation results. It is not possible to use, as an example, a real protocol specification due to its size. A simple finite state automaton was selected to do not bore a reader. The example idea was taken from a VHDL Benchmark, which was given by Champaka Ramachandran from University Of California.

The example is a traffic light controller of the following functionality: Highway light is green normally. When there is a car on the side road and the highway light has been on for *timeoutL* time, the highway light turns yellow. *TimeoutS* time after it has turned yellow, the highway light turns red and the side road light turns green.

If there are no cars on the side road, or the side road light has been on for *timeoutL* time, the side road light turns yellow. *TimeoutS* time after it has turned yellow, the side road light turns red and the highway road light turns green.

The timer signals are connected to some external timer circuit. The *startTim* would start the external timer circuitry every time the state changes. The *timeoutL* and *timeoutS* should originate from the external timer.

The proposed Estelle specification composes two modules: *test* and *control*. The *control* module realizes the defined above functionality. The *test* module simulates the controller environment and generates some test patterns. Figure 5 shows the structure of the Estelle specification. The figure illustrates only the idea of the control automaton, it does not show all transitions and outputs.

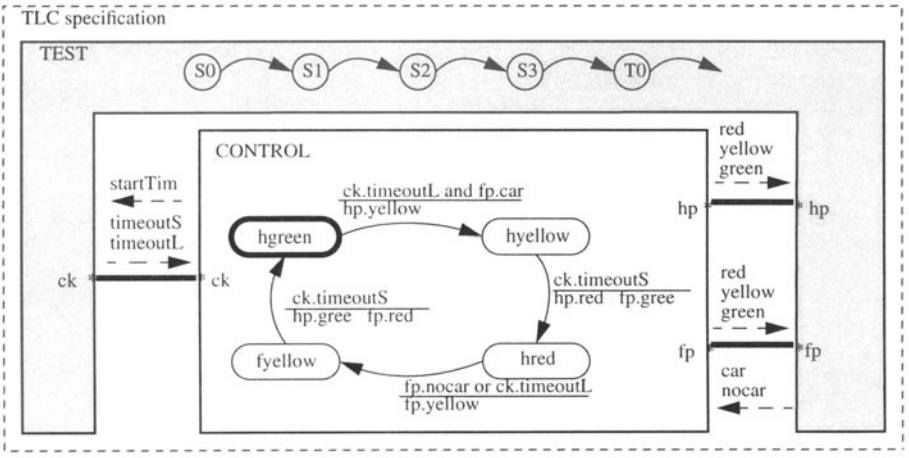


Figure 5 The Estelle specification structure of a traffic light controller.

It is important to notice, that there is a functional difference between this specification and that given by Ch. Ramachandran. The Estelle modules communicate by message passing*, thus *startTim, timeoutS, timeoutL, red, yellow,* etc., are messages, no signals (states of an output driver), as in the VHDL benchmark. Hardware implementation of a message passing mechanism results in data and control signals generation. Sometimes a simpler communication mechanism can be realized and the control signals are not indispensable. However the generated signal excess leads to better fault resistance of the target design. The controller, from the VHDL benchmark, can directly be connected to relays, which drive the road lights. The controller specified here in Estelle, should be connected to simple automatons, which drive the lights from one hand, and from the other, receive control messages. These automatons can be sensible for communication anomalies (in such a case, they can turn off the lights, or set them blinking yellow).

There are cases when the Estelle message passing principle leads to additional transition declaration in Estelle description, and to undesired signals and unwanted processing in the generated VHDL description. The *timeoutS, timeoutL* messages, in this example, illustrate such a case. These messages have to be received when they arrive, even if they are not needed. A designer can specify a desired functionality directly in VHDL without this surplus - he can simply define two signals *timeoutS* and *timeoutL.* A new signal value overwrite the old one, and no receiving logic is desired.

This example shows, that passing from a system idea (specified in Estelle) to hardware realization is advisable, when the target design should communicate on the base of message passing principle. If different communication principles should be used in the system under design, then it is still possible to specify and to analyze it in Estelle without a redundancy (e.g., by application of Estelle *primitive* functions). In our example, the *startTim, timeoutS, timeoutL* could be signals controlled by the *primitive* functions.

The full Estelle specification of the controller is given in Appendix A. The specification was compiled and analyzed in the EDT (Estelle Development Tools) environment (Budkowski, 1992). The Estelle to C (ec) compiler† generates an Intermediate Form file from an Estelle text file. The Estelle Intermediate Form file is the input to the e2v translator. The e2v output VHDL files were tested (compiled and simulated) in the VHDL environment from Vantage Analysis Systems, Inc.. Appendix A gives the generated VHDL description of the controller, and shows the correspondences between the two specifications.

The e2v translator generates the following files for this example (Estelle specification *tlc.stl* file of 480 words):

- module files: *spec_tlc.vhdl* (1148 words), *test.vhdl* (1568 words), *control.vhdl* (585 words);
- queue files: *sbod_tlc_q.vhdl* (1050 words);
- configuration file: *configuration.vhdl* (74 words); and
- compilation script file: *compile* (55 words).

A module file is generated for the specification module: *tlc*, and for any translated Estelle module header: *test, control.* A module file contains:

---

*It is possible to model in Estelle a signal communication mechanism, but the corresponding Estelle text becomes verbose and ambiguous (comments must be used to distinguish the desired functionality and the function and transition which model something else).

†ec is a part of EDT tool-set.

- entity declaration,
- package declarations, and
- architecture declarations.

An entity declaration corresponds to a module header declaration. A package declaration contains translation results of constant, type, and channel declarations, which are given in a body declared for this module. An architecture declaration contains translation results of all the remaining elements of this body. The package generated in the file, that corresponds to a specification module, contains some global constant and subroutines declarations.

A queue file is generated for any translated body declaration, which contains a channel declaration. A queue file contains a specification of queue entities and queue architectures. A queue entity and architecture declarations are generated from an Estelle channel/role declaration.

The generated configuration file contains all *configuration* statements, for "module" and "queue" instances. e2v translates an Estelle child module instance to a VHDL component instance, and an Estelle input queue to a VHDL queue instance. A *configuration* statement is needed to bind a component instance with an entity and an architecture declarations.

The generated script file contains VHDL compilation commands. The VHDL files are mutually dependent due to a distribution of object declarations and its references. The order of VHDL files must be given, for correct compilation of a VHDL specification. A compilation command invokes the *analyze* - i.e., the VHDL Vantage compiler.

The simulation results of the two (Estelle and VHDL) specification are compared in Appendix B. Only a small part of the all simulation scenario is given to illustrate only the equivalence of the simulation results.

4. CONCLUSIONS

Both an Estelle specification, and a VHDL specification, may describe a designed system together with its environment. They may also describe only a part of a designed system. The modules hierarchy corresponds in a natural way to the functional decomposition of a system. Some of the system functions (modules) can be implemented in a hardware. A designer may thus select, which Estelle modules should be translated to VHDL. The selection of a module implies that all its descendant modules are also selected.

The abstract notions of the inter-process communication mechanisms simplifies specification and analysis of a protocol. A designer can describe in Estelle a system that is composed from hardware and software blocks, which communicate through the unbounded FIFO queues. The designer can start a rapid hardware/software co-designing taking advantage from the presented translator and from existing translators from Estelle to a programming language (C, C++). There are three advantages of this approach:

1. The hardware and software implementations are based on the same global system specification. Figure 7 shows a protocol designing in hardware and software.
2. The generated VHDL text can be used to produce some VHDL simulation scenarios and waveform results, which can be used to design validation at the later synthesis steps.

3. The VHDL simulation tools can be used to detailed analysis of time dependences and a performance of a target system.

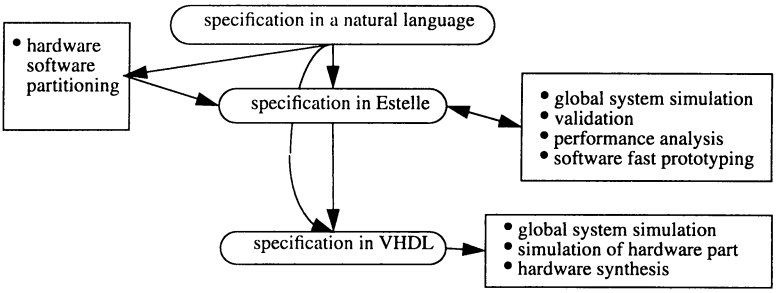


Figure 6 A hardware software co-designing of a communication protocol.

The Estelle language is more efficient, than VHDL language, to describe a distributed system and to analyze a communication protocol in the beginning phase of designing. VHDL tools, in contrary to Estelle tools, are well suited to analyze time dependences. The VHDL language can be efficiently used to deal with run time problems, e.g., time distributions, time deadlines, run time performance management that is based on time evaluation.

The Estelle simulation tools are not able to take into account the delay times of a message transfer. We allow a user to indicate time parameters inside an Estelle specification using so called "qualifying comments". These parameters are conveyed into generated VHDL text. Consequently our approach can be useful not only for hardware synthesis purposes.

The same system may be specified in Estelle in different ways. The way of specifying influences the complexity of generating hardware. There are some Estelle constructors that increase this complexity. A declaration of an exported variable, which is modified by two modules causes more complex hardware implementation than declaration of two exported variables, which are modified separately by respective modules. A common queue that is associated with two interaction points is more complex in hardware implementation than two individual queues. Hierarchy of active modules causes parent-child synchronization, that slow down computation speed and grows implementation complexity.

A generated VHDL text, from a complete Estelle specification, is ready for simulation. Unfortunately it needs some refinements to be used for synthesis. The reason is that some VHDL constructs do not synthesize well and different synthesis platforms have different restrictions to VHDL. However, an automatic generation of VHDL text speed up overall designing process and does not introduce additional errors. Moreover, it provides better ability to relate implementation back to the original requirements.

The needed future work is to introduce algorithms for dynamic allocation of entities, to translate the dynamic Estelle behavior. The other research direction is to include to this designing approach some methods for hardware software partitioning.

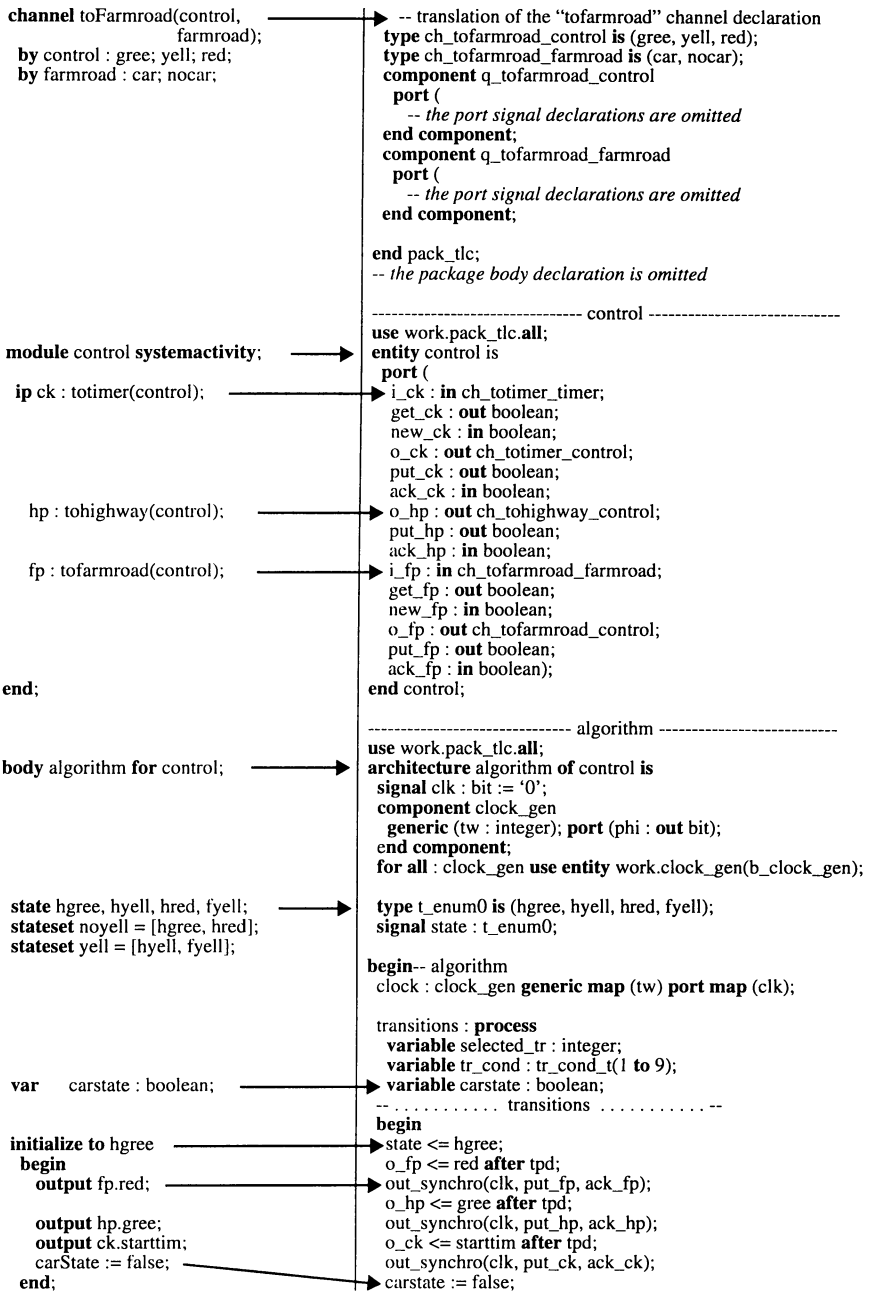
Appendix A Estelle and Generated VHDL Specifications of the tlc Example

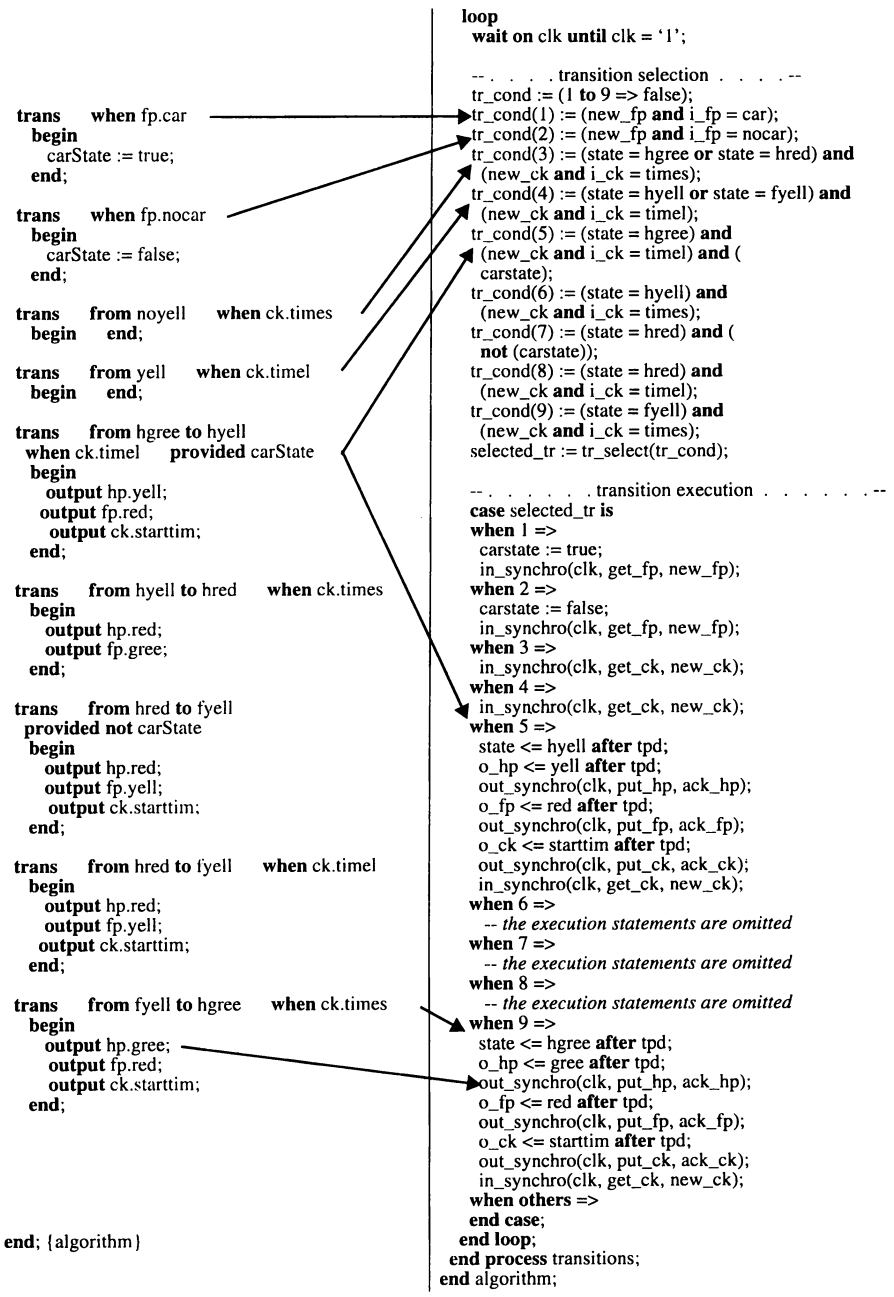
Below, the two specifications are given, the Estelle on left side, the VHDL on the right. Some arrows show the correspondences between the language constructors. The test scenario (**body** simulation **for** TEST) is taken off from the given listing due to paper size limitation. By the same reason, only the contents of the generated files *spec_tlc.vhdl* and *test.vhdl* are given. Only the two VHDL files should be taken into consideration, if the road light controller was implemented in hardware, the others are needed for simulation only.

```
specification tlc;                    ---------- spec_tlc ----------
 default individual queue;            entity spec_tlc is
 timescale second;                    end spec_tlc;

                                      ---------- sbod_tlc ----------
                                      package pack_tlc is
                                        -- global declarations which are not related to the Estelle specification
                                       constant queue_len : integer := 256;
                                       constant tpd : time := 1 ns;
                                       constant tw : integer := 10;          -- clock pulse width in ms
                                       procedure delay_gen (signal clk : in bit;
                                        signal val_timer : inout integer; signal cr_timer : in boolean;
                                        signal fin_delay : out boolean; signal delay : in integer);
                                       procedure out_synchro (signal clk : in bit;
                                        signal put : out boolean; signal ack : in boolean);
                                       procedure in_synchro (signal clk : in bit;
                                        signal get : out boolean; signal ready : in boolean);
                                       type tr_cond_t is array (integer range <>) of boolean;
                                       function tr_select (tr_cond : in tr_cond_t) return integer;

                                        -- translation results of the estelle specification
channel toTimer(timer, control);      ->
 by timer : timel; times;              -- translation of the "totimer" channel declaration
 by control : starttim;               type ch_totimer_timer is (timel, times);
                                      type ch_totimer_control is (starttim);
                                      component q_totimer_timer
                                       port (
                                        by_role1 : in ch_totimer_timer;
                                        put : in boolean;
                                        ack_put : out boolean;
                                        by_role2 : out ch_totimer_timer;
                                        get : in boolean;
                                        ready : out boolean);
                                      end component;
                                      component q_totimer_control
                                       port (
                                        by_role1 : in ch_totimer_control;
                                        put : in boolean;
                                        ack_put : out boolean;
                                        by_role2 : out ch_totimer_control;
                                        get : in boolean;
                                        ready : out boolean);
                                      end component;
channel toHighway(control,            ->
                  highway);           -- translation of the "tohighway" channel declaration
 by control : gree; yell; red;        type ch_tohighway_control is (gree, yell, red);
                                      component q_tohighway_control
                                       port (
                                        -- the port signal declarations are omitted
                                      end component;
```


channel toFarmroad(control, farmroad);
by control : gree; yell; red;
by farmroad : car; nocar;
module control systemactivity;
ip ck : totimer(control);
hp : tohighway(control);
fp : tofarmroad(control);
end;
body algorithm for control;
state hgree, hyell, hred, fyell;
stateset noyell = [hgree, hred];
stateset yell = [hyell, fyell];
var carstate : boolean;
initialize to hgree
begin
output fp.red;
output hp.gree;
output ck.starttim;
carState := false;
end;
-- translation of the "tofarmroad" channel declaration
type ch_tofarmroad_control is (gree, yell, red);
type ch_tofarmroad_farmroad is (car, nocar);
component q_tofarmroad_control
port (
-- the port signal declarations are omitted
end component;
component q_tofarmroad_farmroad
port (
-- the port signal declarations are omitted
end component;
end pack_tlc;
-- the package body declaration is omitted
---------- control ----------
use work.pack_tlc.all;
entity control is
port (
i_ck : in ch_totimer_timer;
get_ck : out boolean;
new_ck : in boolean;
o_ck : out ch_totimer_control;
put_ck : out boolean;
ack_ck : in boolean;
o_hp : out ch_tohighway_control;
put_hp : out boolean;
ack_hp : in boolean;
i_fp : in ch_tofarmroad_farmroad;
get_fp : out boolean;
new_fp : in boolean;
o_fp : out ch_tofarmroad_control;
put_fp : out boolean;
ack_fp : in boolean);
end control;
---------- algorithm ----------
use work.pack_tlc.all;
architecture algorithm of control is
signal clk : bit := '0';
component clock_gen
generic (tw : integer); port (phi : out bit);
end component;
for all : clock_gen use entity work.clock_gen(b_clock_gen);
type t_enum0 is (hgree, hyell, hred, fyell);
signal state : t_enum0;
begin-- algorithm
clock : clock_gen generic map (tw) port map (clk);
transitions : process
variable selected_tr : integer;
variable tr_cond : tr_cond_t(1 to 9);
variable carstate : boolean;
-- ..... transitions .....--
begin
state <= hgree;
o_fp <= red after tpd;
out_synchro(clk, put_fp, ack_fp);
o_hp <= gree after tpd;
out_synchro(clk, put_hp, ack_hp);
o_ck <= starttim after tpd;
out_synchro(clk, put_ck, ack_ck);
carstate := false;


trans when fp.car
begin
carState := true;
end;
trans when fp.nocar
begin
carState := false;
end;
trans from noyell when ck.times
begin end;
trans from yell when ck.timel
begin end;
trans from hgree to hyell
when ck.timel provided carState
begin
output hp.yell;
output fp.red;
output ck.starttim;
end;
trans from hyell to hred when ck.times
begin
output hp.red;
output fp.gree;
end;
trans from hred to fyell
provided not carState
begin
output hp.red;
output fp.yell;
output ck.starttim;
end;
trans from hred to fyell when ck.timel
begin
output hp.red;
output fp.yell;
output ck.starttim;
end;
trans from fyell to hgree when ck.times
begin
output hp.gree;
output fp.red;
output ck.starttim;
end;
end; {algorithm}
loop
wait on clk until clk = '1';
-- . . . . transition selection . . . . --
tr_cond := (1 to 9 => false);
tr_cond(1) := (new_fp and i_fp = car);
tr_cond(2) := (new_fp and i_fp = nocar);
tr_cond(3) := (state = hgree or state = hred) and
(new_ck and i_ck = times);
tr_cond(4) := (state = hyell or state = fyell) and
(new_ck and i_ck = timel);
tr_cond(5) := (state = hgree) and
(new_ck and i_ck = timel) and (
carstate);
tr_cond(6) := (state = hyell) and
(new_ck and i_ck = times);
tr_cond(7) := (state = hred) and (
not (carstate));
tr_cond(8) := (state = hred) and
(new_ck and i_ck = timel);
tr_cond(9) := (state = fyell) and
(new_ck and i_ck = times);
selected_tr := tr_select(tr_cond);
-- . . . . . transition execution . . . . . --
case selected_tr is
when 1 =>
carstate := true;
in_synchro(clk, get_fp, new_fp);
when 2 =>
carstate := false;
in_synchro(clk, get_fp, new_fp);
when 3 =>
in_synchro(clk, get_ck, new_ck);
when 4 =>
in_synchro(clk, get_ck, new_ck);
when 5 =>
state <= hyell after tpd;
o_hp <= yell after tpd;
out_synchro(clk, put_hp, ack_hp);
o_fp <= red after tpd;
out_synchro(clk, put_fp, ack_fp);
o_ck <= starttim after tpd;
out_synchro(clk, put_ck, ack_ck);
in_synchro(clk, get_ck, new_ck);
when 6 =>
-- the execution statements are omitted
when 7 =>
-- the execution statements are omitted
when 8 =>
-- the execution statements are omitted
when 9 =>
state <= hgree after tpd;
o_hp <= gree after tpd;
out_synchro(clk, put_hp, ack_hp);
o_fp <= red after tpd;
out_synchro(clk, put_fp, ack_fp);
o_ck <= starttim after tpd;
out_synchro(clk, put_ck, ack_ck);
in_synchro(clk, get_ck, new_ck);
when others =>
end case;
end loop;
end process transitions;
end algorithm;

```
module TEST systemactivity;
 ip ck : toTimer(timer);
   hp : toHighway(highway);
   fp : toFarmroad(farmroad);
end;
```

```
---------- test ----------
use work.pack_tlc.all;
entity test is
 port (
  i_ck : in ch_totimer_control;
  get_ck : out boolean;
  new_ck : in boolean;
  o_ck : out ch_totimer_timer;
  put_ck : out boolean;
  ack_ck : in boolean;
  i_hp : in ch_tohighway_control;
  get_hp : out boolean;
  new_hp : in boolean;
  i_fp : in ch_tofarmroad_control;
  get_fp : out boolean;
  new_fp : in boolean;
  o_fp : out ch_tofarmroad_farmroad;
  put_fp : out boolean;
  ack_fp : in boolean);
end test;
```

```
body simulation for TEST;
   {the body declaration is omitted}
end; {simulation}
```

```
---------- simulation ----------
use work.pack_tlc.all;
architecture simulation of test is
    -- the architecture declaration is omitted
end simulation;
```

```
---------- sbod_tlc ----------
use work.pack_tlc.all;
architecture sbod_tlc of spec_tlc is
 component control
  port (
   -- the port signal declarations are omitted
 end component;
 component test
  port (
   -- the port signal declarations are omitted
 end component;
```

```
modvar comp : CONTROL;
```

```
modvar env : TEST;
```

```
-- the local signal declarations are omitted
```

```
initialize {rlc}
 begin
  init comp with algorithm;
```

```
begin  -- sbod_tlc
 comp : control
  port map (i_ck, get_ck, new_ck, o_ck, put_ck, ack_ck, o_hp, put_hp,
    ack_hp, i_fp, get_fp, new_fp, o_fp, put_fp, ack_fp);
```

```
  init env with simulation;
```

```
 env : test
  port map (i_ck000, get_ck000, new_ck000, o_ck000, put_ck000,
   ack_ck000, i_hp, get_hp, new_hp, i_fp000, get_fp000, new_fp000,
   o_fp000, put_fp000, ack_fp000);
```

```
  connect comp.ck to env.ck;
  connect comp.fp to env.fp;
  connect comp.hp to env.hp;
```

```
 ck : q_totimer_control
  port map (o_ck, put_ck, ack_ck, i_ck000, get_ck000, new_ck000);
 ck000 : q_totimer_timer
  port map (o_ck000, put_ck000, ack_ck000, i_ck, get_ck, new_ck);
 fp : q_tofarmroad_control
  port map (o_fp, put_fp, ack_fp, i_fp000, get_fp000, new_fp000);
 fp000 : q_tofarmroad_farmroad
  port map (o_fp000, put_fp000, ack_fp000, i_fp, get_fp, new_fp);
 hp : q_tohighway_control
  port map (o_hp, put_hp, ack_hp, i_hp, get_hp, new_hp);
end sbod_tlc;
```

```
 end; {rlc}

end.
```

Appendix B Simulation Results

Results Taken from Estelle Simulation

```
Total fired transitions =  1
Major state of module instance 3 = S1

Total fired transitions =  2
INPUT TREATED IN LAST FIRED TRANS:
   When FP.CAR
NO OUTPUT SENT IN LAST FIRED TRANS.
Major state of module instance 2 = HGREE

Total fired transitions =  3
INPUT TREATED IN LAST FIRED TRANS:
   When CK.TIMEL
OUTPUT(S) SENT IN LAST FIRED TRANS:
   3 output(s) established :
      OUTPUT CK.STARTTIM send to 3->CK
      OUTPUT HP.YELL send to 3->HP
      OUTPUT FP.RED send to 3->FP
Major state of module instance 2 = HYELL

Total fired transitions =  4
Major state of module instance 3 = S2

Total fired transitions =  5
INPUT TREATED IN LAST FIRED TRANS:
   When CK.TIMES
OUTPUT(S) SENT IN LAST FIRED TRANS:
   2 output(s) established :
      OUTPUT HP.RED send to 3->HP
      OUTPUT FP.GREE send to 3->FP
Major state of module instance 2 = HRED

Total fired transitions =  6
Major state of module instance 3 = S3

Total fired transitions =  7
INPUT TREATED IN LAST FIRED TRANS:
   When FP.NOCAR
NO OUTPUT SENT IN LAST FIRED TRANS.
Major state of module instance 2 = HRED

Total fired transitions =  8
LAST FIRED TRANS ISN'T AN INPUT
TRANSITION.
OUTPUT(S) SENT IN LAST FIRED TRANS:
   3 output(s) established :
      OUTPUT CK.STARTTIM send to 3->CK
      OUTPUT HP.RED send to 3->HP
      OUTPUT FP.YELL send to 3->FP
Major state of module instance 2 = FYELL

Total fired transitions =  9
Major state of module instance 3 = T0
```

Results Taken from VHDL Simulation

Time	Env	Comp:									
				G		G		P		P	
1	S	S		E		E		U		U	
	T	T	I	T	I	T	O	T	O	T	
s	A	A									
e	T	T	$\overline{C}$	$\overline{C}$	$\overline{F}$	$\overline{F}$	$\overline{H}$	$\overline{H}$	$\overline{F}$	$\overline{F}$	
c	E	E	K	K	P	P	P	P	P	P	
	*initialization**										
0	S0	HGREE	TIMEL	0	CAR	0	GREE	0	GREE	0	
0.00	S0	HGREE	TIMEL	0	CAR	0	GREE	0	RED	1	
0.01	S0	HGREE	TIMEL	0	CAR	0	GREE	0	RED	0	
0.02	S0	HGREE	TIMEL	0	CAR	0	GREE	1	RED	0	
0.03	S0	HGREE	TIMEL	0	CAR	0	GREE	0	RED	0	
	transition 1										
4.01	S1	HGREE	TIMEL	0	CAR	0	GREE	0	RED	0	
	transition 2										
4.02	S1	HGREE	TIMEL	0	CAR	1	GREE	0	RED	0	
4.03	S1	HGREE	TIMEL	0	CAR	0	GREE	0	RED	0	
	transition 3										
4.04	S1	HYELL	TIMEL	0	CAR	0	YELL	1	RED	0	
4.05	S1	HYELL	TIMEL	0	CAR	0	YELL	0	RED	0	
4.06	S1	HYELL	TIMEL	0	CAR	0	YELL	0	RED	1	
4.07	S1	HYELL	TIMEL	0	CAR	0	YELL	0	RED	0	
4.10	S1	HYELL	TIMEL	1	CAR	0	YELL	0	RED	0	
4.11	S1	HYELL	TIMEL	0	CAR	0	YELL	0	RED	0	
	transition 4										
5.07	S2	HYELL	TIMES	0	CAR	0	YELL	0	RED	0	
	transition 5										
5.08	S2	HRED	TIMES	0	CAR	0	RED	1	RED	0	
5.09	S2	HRED	TIMES	0	CAR	0	RED	0	RED	0	
5.10	S2	HRED	TIMES	0	CAR	0	RED	0	GREE	1	
5.11	S2	HRED	TIMES	0	CAR	0	RED	0	GREE	0	
5.12	S2	HRED	TIMES	1	CAR	0	RED	0	GREE	0	
5.12	S2	HRED	TIMEL	1	CAR	0	RED	0	GREE	0	
5.13	S2	HRED	TIMEL	0	CAR	0	RED	0	GREE	0	
	transition 6										
6.11	S3	HRED	TIMEL	0	CAR	0	RED	0	GREE	0	
6.11	S3	HRED	TIMEL	0	NOCAR	0	RED	0	GREE	0	
	transition 7										
6.12	S3	HRED	TIMEL	0	NOCAR	1	RED	0	GREE	0	
6.12	S3	HRED	TIMEL	0	CAR	1	RED	0	GREE	0	
6.13	S3	HRED	TIMEL	0	CAR	0	RED	0	GREE	0	
	transition 8										
6.14	S3	FYELL	TIMEL	0	CAR	0	RED	1	GREE	0	
6.15	S3	FYELL	TIMEL	0	CAR	0	RED	0	GREE	0	
6.16	S3	FYELL	TIMEL	0	CAR	0	RED	0	YELL	1	
6.17	S3	FYELL	TIMEL	0	CAR	0	RED	0	YELL	0	
	transition 9										
7.15	T0	FYELL	TIMEL	0	CAR	0	RED	0	YELL	0	
7.15	T0	FYELL	TIMES	0	CAR	0	RED	0	YELL	0	

*The texts in italic are the author comments.

References

Bauer, M. and Ecker, W. (1993) Communication Mechanisms for VHDL Specification and Design Starting at System Level. *Proc. VHDL Forum for CAD in Europe, Spring'93 Meeting*, Austria, March, 95-106.

Benders, L.P.M. and Stevens, M.P.J. (1991) Task Level Behavioral Hardware Description. *Microprocessing & Microprogramming*, **32**, 323-32.

Benders, L.P.M. and Stevens, M.P.J. (1993) TL: A System Specification System. *Microprocessing and Microprogramming* **38**, 835-42.

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

Budkowski, S. (1992) Estelle development toolset (EDT). *Computer Networks and ISDN Systems* **25**, 63-82.

Coelho, D. (1989) *The VHDL Handbook*. Kluwer Academic Publishers.

Dutt, N.D. (1991) A User Interface for VHDL Behavioral Modeling. *Proc. CHDL'91*, Marseille, France, 375-93.

Glunz, W. and Kruse, T. et al. (1993) Integrating SDL and VHDL for System-Level Hardware Designing, in *IFIP Transactions on Computer Hardware Description Languages* (ed. D. Agnew, L. Claesen, R. Camposano) Elsevier Science Publishers B.V.

Jerraya, A. and O'Brien, K. et al. (1993) Linking System Design Tools and Hardware Design Tools. *Proc. CHDL'93*, 331-8.

Jerraya, A. and O'Brien, K. (1994) SOLAR: an intermediate format for system-level modeling and synthesis. Chapter 7 in *Codesign compute aided software/hardware engineering*, (ed J. Rozemblit and Klaus Buchenrieder) IEEE press, 145-75.

Delgado Kloos, C. and de Miguel Moro, T. 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.

Vahid, F. and Narayan, S. et al. (1991) SpecCharts: A language for System Level Design. *Proc. CHDL 91*, 145-55.

Woo, A.C. and Peppard, L.E. (1992) System-Level Modelling in VHDL. *Microelectronic Journal*, **23**, 223-30.

Wytrebowicz, J. and Budkowski, S. (1994) Communication Protocols Implemented in Hardware: VHDL Generation from Estelle. *Proc. VHDL-Forum for CAD in Europe, Fall'94 Meeting*, 29-40.

Wytrebowicz, J. (1994) *VHDL Generation from Estelle*. Research Report N^{o} 941103, Institut National des Telecommunications, Evry, France.

Bibliography

Jacek Wytrębowicz is a Ph.D. student at the Institut National des Télécommunications, on leave from Warsaw University of Technology. His research interests include formal description techniques and hardware designing methodology. J. Wytrębowicz received M.S degree in computer science from the Warsaw University of Technology, Warsaw, Poland, in 1982. He worked as a computer designer and as a research assistant at the Institute of Computer Science at Warsaw University of Technology. He is author and coauthor of four papers and two books, which were dedicated to microprogramming and to microprocessor system design, also of two papers dedicated to VHDL applications.