High-Level Specification for Digital System Design

M. BENMOHAMMED

LIRE Lab., Institut d'Informatique, Université de Cne, 25000 Constantine, ALGERIE.

S. MERNIZ

LIRE Lab., Institut d'Informatique, Université de Cne, 25000 Constantine, ALGERIE.

Abstract

Design complexity has been increasing exponentially this last decade. In order to cope with such an increase and to keep up designers' productivity, higher level specifications were required. Moreover new synthesis systems, starting with a high level specification, have been developed in order to automate and speed digital system design.

Keys-Words: VLSI, CAD, High-Level Synthesis, Architectural Synthesis, HDL, Verilog, VHDL1.

Introduction

The arrival and acceptance of standard Hardware Design Languages (HDLs) such as VHDL and Verilog, have promoted high level specification of electronic circuits. HDLs may be used for the specification of whole systems, as well as sub-systems that may then be assembled within a hierarchical (structural) description. The fact that various tools (for synthesis and simulation) have been developed this last decade has also helped high level specification for VLSI to emerge and to gain more and more acceptance in the VLSI design community [1].

1.1. Motivation for high level specification

Nowadays one of the major objectives within VLSI domain is the improvement of the design quality and of the designers' productivity. This is due to the fact that the design process is characterized by 2 sets of factors : constant factors and variable ones. For instance typical large design budgets are usually fixed to around 10 to 15 persons over 18 months, and designers' productivity has been evaluated to some 10 objects per day.

Controversely, design complexity has been increasing exponentially since 1984 from a hundred thousand transistors, to reach 1 to 2 million transistors today. The design complexity forecast for the year 2000 is 50 million transistors for a 0.18 micron CMOS technology. In order to cope with such an increase in complexity, with the fixed budget, the objects designed have gained in abstraction degree; thus instead of designing 10 transistors daily as in the 80's, designers are now creating some 10 algorithmic lines or mega blocks per day. This shift has been smooth and the objects handled have been gates and RTL HDL lines meanwhile. The future shall be about designing system-level specifications and sub-systems.

HDLs can be used for design specification at various abstraction levels, from gates to the behavioral level. We shall first go over the main abstraction levels, which are usually encountered within a usual design-flow, in section 2. A whole set of languages for high level specifications at the behavioral and system levels is introduced. We shall have a quick glance at a small subset in section 3. Nowadays it becomes crucial to speed up the design-flow so that the time to market is reduced. Consequently, CAD tools are being more and more used for synthesis. Register transfer level (RTL) synthesis tools are commonly used and higher level synthesis tools are now required. Section 4 is about behavioral synthesis and the inputs required. Among the different existing HDLs used at the behavioral level, VHDL is the most commonly used being a standardized IEEE HDL. This paper shall thus refer to this HDL for high level specification. More details about the design methodology are given in section 5.

2. Abstraction levels

Four abstraction levels shall be studied in this section; their comparisons are about the following characteristics: the time unit involved, the primitives used for the specification and the description organization for each of the description levels (figure 1) [2].

Timing is used as a reference for classifying the different specification levels, as it is one of the main

issue (if it is not the main one) during the process of designing an integrated circuit. In fact, regardless the abstraction level, the design process may be defined as the refinement of high level concepts (including operations, primitives, statements or constructs) into lower level concepts using more detailed timing units. Figure 1 illustrates this concept.

Description Level	Time Unit	Primitives	Description Organization
Physical Level	Delay	Gates, Devices	Schematics
Logic and RT Levels	Clock Cycle	Register-Operator Transfers	Boolean Equations, FSMs
Algorithmic Level	Control Step	Computation Control	BFSs, DFGs, CFGs
System Level	Task	Process, Communication	Communicating Processes

Figure 1: Abstraction Levels

2.1. Lower abstraction levels: physical, logic and register transfer levels

At the lowest abstraction level the basic timing unit is the delay within schematics. The design at the physical level being specified in terms of gates and devices interconnected through nets, timing is done in considering gate and net latencies [3].

The next description level is called the register transfer or logic level, where the design is specified at the clock cycle level. A typical description states which operations to execute at each clock cycle. In fact descriptions at this level are generally made of a set of registers and operators, and the functions to be executed by the datapath are expressed as data transfers between registers and executing blocks. Typical representations used at this level are boolean equations, finite state machines and BDDs. The description and simulation of a whole system at the clock cycle level can be done using register transfer level (RTL) primitives.

2.2. Architectural or behavioral level

At the behavioral or architectural level, the design is specified in terms of control steps by means of programs, algorithms, flowcharts, etc [2]. The concept of operations and control statements is used to sequence the input and output events; control statements, within VHDL for instance, consist of instructions such as loop and wait statements. Typical representations at this level are behavioral finite state machines, control-flow graphs, data-flow graphs and control/data-flow graphs, all corresponding to event-driven specifications. Descriptions at this level are often composed of a set of protocols to exchange data with the external world and an internal computation. The computation step being itself composed of a set of operations executed between two successive input or/and output events, may take several clock cycles.

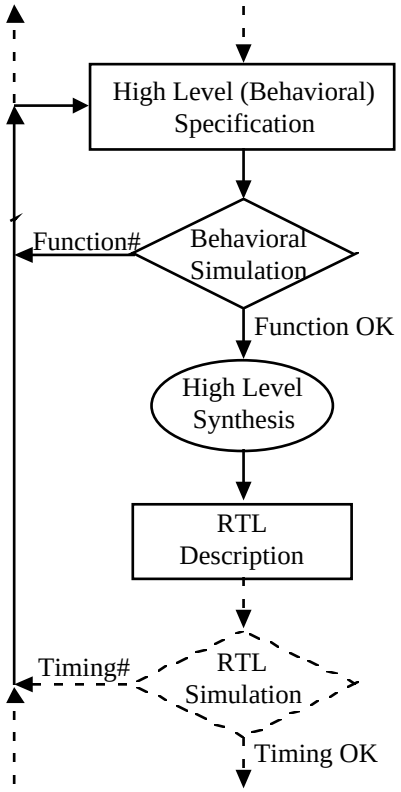

Figure 2: Design iteration at the behavioral level

2.3. Higher abstraction level: system level

The highest level of abstraction corresponds to the system level [4] [5]. Specification at this level includes distributed control and multi-thread computation, and the corresponding basic timing unit is the computation

task. System level descriptions are composed of a set of hierarchical, parallel and communicating processes. The basic primitive is the process, which may consist of a hardware part as well as a software one. Each of these sub-systems may be as complex as a behavioral description.

2.4. Design process

The whole design process consists in a set of translation steps of a given specification into another one of lower abstraction. Each of the translation step corresponds to a design iteration, which has to be repeated in case the design contraints are not satisfied. The validation of a specification is thus an important task that has to be performed as early as possible in the design-flow, in order to minimize the amount of design iterations. One design iteration is illustrated in figure 2, with the behavioral synthesis step.

For large hierarchical designs, each of its sub-systems can be designed independently using different CAD tools. The verification of the coherence of a sub-system, once it has been synthesized, can be achieved by plugging the synthesized description within the validated environment. Undertaking such a validation as from the beginning of the synthesis-flow and at several abstraction levels allows to delimit any problems more easily earlier.

High level descriptions are usually smaller in size; besides such descriptions are close to the algorithms they implement, making it easier to write them. Therefore these descriptions are also easier to maintain and to debug. Moreover the simulation at this level is faster.

3. High level specification languages

A wide range of high level specification languages is available today. Owing to their different characteristics (structure, hierarchy, concurrency, synchronization, ...), they have been chosen for input by different synthesis systems, according to the synthesis performed : system level or behavioral level, control-flow dominated oriented circuits or digital signal processings.

All the existing high level specification languages are based only on few concepts. First there are HDLs (Hardware Description Languages) such as VHDL, Verilog and HardwareC. These allow sequential statements within processes as well as data-flow behaviors, except HardwareC (which restricts data-flow behaviors to simple netlists). They can thus be used to describe behavioral level descriptions of both control-flow dominated circuits and DSPs (based on data-flow models). Hierarchy can be expressed within HDLs through concurrent processes, blocks and component instanciations. System level specifications can also be achieved by these languages as they enable task level concurrency (concurrency within processes) and statement level concurrency (signal assignments and parallel complex statements). Communication between processes here makes use of shared memory models.

At the behavioral level there exist languages dedicated to data-flow descriptions. An example of such high level specification language is Silage, which is used as input for several DSP oriented synthesis systems: Cathedral/Mistral [6] Ptolemy [7], ...

A whole existing set of high level languages, in addition to the high level specification languages listed above, is used at the system level for specifiaction, simulation and synthesis. A few examples of such languages are : StateCharts, SpecCharts [5], SDL [8], Estelle [9], Esterel [10] and LOTOS [11] All of them offer large possibilities in concurrency description and synchronization specification through communication. They differ however from each other by the constructs they provide and their "delay" or time concept.

Within this paper, we shall look closer at the behavioral or algorithmic level and leave to one side pure system level specifications.

4. Specification at the algorithmic level

As one of the major objectives within VLSI domain is the improvement of the designers' productivity, several research groups have been working on the automation of synthesis leading to the development of CAD tools [1]. Logic and register transfer level synthesis have been popularized and are much used today. However in order to cope with circuits of increasing size, higher level synthesis tools are required.

Within electronic fields, 2 main types of circuits can be distinguished: DSP and control-flow dominated circuits. Both types of circuits can be described at the algorithmic or behavioral level. However as seen previously (section 3) DSP circuits are more easily described using concurrent statements within data-flow descriptions [12], while sequential statements meet the needs of control-flow dominated circuits [13]. Several CAD tools for synthesis of behavioral descriptions are now being developed. These synthesis systems can be regrouped into 2 main sets corresponding to the type of circuits targetted by the algorithms implemented. However in both cases, when HDLs with wide statement sets, such as VHDL, are being used for the behavioral specification, only a subset of the language is really being accepted by each of them [14] [15] [21]. Unfortunately existing synthesis systems usually accept different sets of statements making it difficult to exploit

a unique high level description by different synthesis systems.

4.1. VHDL input for behavioral synthesis

VHDL is the most commonly used hardware description language for behavioral descriptions feeding high-level synthesis systems. However as VHDL is event-driven and based on an asynchronous model, its descriptions are sometimes hard to synthesize automatically into synchronous circuits. Fixing a VHDL subset enables the prohibition of asynchrounous behaviors and delays.

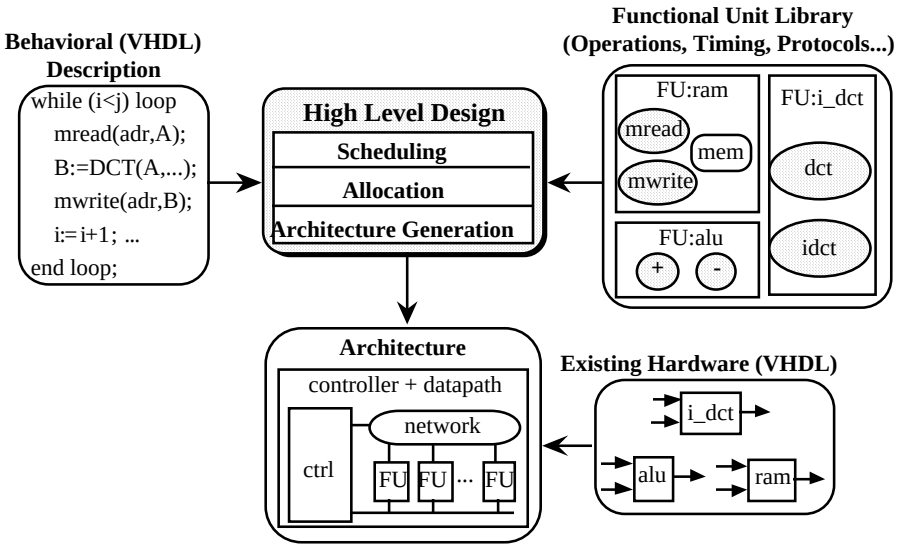

Figure 3: High level synthesis

Most of the high-level synthesis systems which accept VHDL specifications restrict themselves to a vertually sequential subset of the language with limited use to signals due to the fact that signals are updated only at the end of simulation cycles. The VHDL process is used to specify a performed algorithm with sequential statements. According to the VHDL semantics the algorithm executes in zero time, and only at wait statements, is time consumed.

For control-dominated circuits, the behavioral description gives only a view of the coordination of different sub-systems making up the architecture, in other words the top control. In VHDL this may be described as a process that may make use of complex sub-systems through procedure and function calls. These sub-systems are behavioral components that are also called functional units. For high level synthesis, the latters are considered as black boxes, and the only pieces of information required are the list of procedures executed by each of them and informations about its interface protocols. A typical high level design process is outlined in figure 3.

4.2. Architecture model for synchronous circuits

A typical architecture for synchronous circuit synthesis is usually composed of a controller and a datapath, which is itself made of functional units, registers and other components for communication. At the behavioral level, the functional units may very often be complex modules, and are not restricted to simple standard ones [17] [16] [18]. The target architecture is illustrated by figure 4.

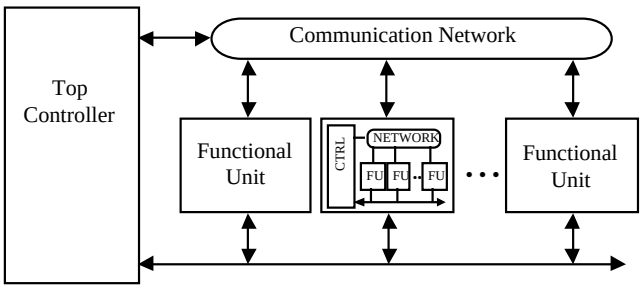

Figure 4: Target Architecture

The functional units, registers, multiplexers, bus drivers and the other datapath components are controlled by the synchronous finite state machine which starts the correct operations with the correct data in the correct clock cycle under the correct conditions. The various modules can be designed separately using different design methods.

Such an architecture may include several functional units that running in parallel. The amount of parallelism

is fixed according to the application and to the resources allocated; the functional units interact through a communication network.

4.3. Component library for behavioral synthesis

As introduced in the section above, synthesis requires a second input: a library of components. This is true whatever the level of abstraction; at the register transfer level, the library is a technology file giving informations about the gates available. Gate characteristics include their function, their fanin and fanout values, their size, ... [19]. For behavioral synthesis, some equivalent informations are required at a higher level. The components are much larger blocks than gates and may execute more than one function. As a result, the component library must list the set of operations for each component and include the execution scheme of each of them [18].

Behavioral components need to be abstracted for re-use as a functional unit during high level synthesis. The high level synthesis view required for a functional unit links the behavioral and the implementation views; it includes the interface of the functional unit, its call-parameters (corresponding to the operation parameters), the operation set executed by the functional unit as well as the parameter-passing-protocol for each operation. This protocol is expressed through static clock cycles; each operation needs to have a fixed predictable execution time.

The designer invokes a functional unit from the library contained in his synthesis environment through a simple VHDL procedure or function call. The association between functional units and operations (known as functional unit binding) is usually made through names. This technique allows the use of existing hardware; large memorization blocks and I/O units can be handled as functional units and managed by the user in the behavioral description.

4.4. Typical high level design-flow based on VHDL

A VHDL based high-level design (also called behavioral or architectural design) starts with two kinds of information: a behavioral specification given in VHDL and an external functional unit library. It produces a register transfer level description that may be handled by existing logic synthesis tools, as outlined in figure 3.

The behavioral description makes use of complex sub-systems through procedure and function calls. In figure 4, the VHDL process makes use of basic operations such as addition (+) and more sophisticated ones such as DCT and memory access. However for each procedure or function used, the library must include at least one functional unit able to execute the corresponding operation. In the case of figure 4, the library includes a set of functional modules (or functional units: FUs) able to execute the operation and fonction invoked by the behavioral VHDL description.

During the different steps involved in high-level design, the functional units are used as black boxes. The only pieces of information required about each functional unit are the list of procedures executed by the functional unit and some informations about protocols. However to complete the description at the register transfer level, the details of the functional units are required. Each functional unit can be described by different views needed during different design steps.

The high level design process transforms the initial specification into an architecture described at the register transfer level, that is, an architecture described at the clock cycle level. This includes:

- Scheduling the operation of the behavioral description in order to fix their execution order.
- Selecting from the library the needed modules for the execution of the operations of the initial description.
- Splitting complex operations (for example, a DCT call) into a set of basic transfers.
- Building an architecture composed of a datapath and a controller.

In short, this process, also called behavioral synthesis, splits each VHDL process into a datapath and a controller, corresponding to the target architecture introduced previously for synchronous circuits. Any remaining data-flow (concurrent) statement is usually synthesized into pure combinatorial using directly register transfer or logic level synthesis tools.

4.5. Impact of writing style

The VHDL process within a behavioral specification corresponds to a processor once transformed into a lower abstraction level description. This processor makes use of coprocessors through procedure and function calls. In other words, after synthesis every behavioral operation will be achieved through an execution unit within the datapath. Each operation call is expressed as a set of register transfers.

The performances of an automatically synthesized circuit ensue from the description made. Wait statements introduce external states within the control-flow graph. These additional states can in fact be

observed during behavioral simulation. At the behavioral level scalar types may sometimes be used; however during synthesis they need to be converted into bit-vectors of fixed (limited) size.

Simple variables and signals within VHDL descriptions correspond to algorithmic level representations of registers, while arrays model memories. Memories are considered during high-level synthesis as being functional units, because memories coming from different (technology) libraries have seldom exactly the same characteristics and that in this way each of them can be described with personalized interface protocols and with its own execution scheme. Operation calls within the behavioral description can be made through the use of standard operators as well as through function and procedure calls.

Declaring and using many different variables or signals enable parallelism with extra area overheads. On the other hand, if few of them are declared and that the statements share this restricted set of variables, then the data dependency created will forbid concurrency in the execution of the operations. Figure 6 gives an example illustrating such compromise; (a) and (b) describe exactly the same behavior.

Nevertheless the VHDL excerpt (a) will lead to more hardware than (b) but the circuit generated may be faster. Excerpt (a) tends to 2 registers (tmp1 and tmp2) and 2 adders for maximum parallelism. With these ressources available, statements 1a and 2a can be executed in parallel. Therefore this excerpt requires 2 clock cycles to execute. On the other hand excerpt (b) will require only 1 register (tmp1) and 1 adder, as statements 2a, 2b, and 3c can never be executed in parallel due to data dependency of tmp1. Meanwhile its execution will ask for 3 clock cycles.

```
variable tmp1,tmp2 : integer range 0 to 127;
...
tmp1 := a + b;              -- Statement 1a
tmp2 := c + d;              -- Statement 2a
tmp1 := tmp1 + tmp2;        -- Statement 3a
```

(a) VHDL targetting surface area optimization

```
variable tmp1 : integer range 0 to 127;
...
tmp1 := a + b;              -- Statement 1b
tmp1 := tmp1 + c;  -- Statement 2b
tmp1 := tmp1 + d;  -- Statement 3b
```

(b) VHDL targetting timing optimization

Figure 6: Equivalent VHDL excerpts

5. Design methodology based on high level specification

Starting with a system specification (at the behavioral level), the full design methodology to generate the corresponding layout includes 3 main steps (as shown in figure 7). (I) On analysis of the system level specification, it may be decided to split the system into sub-systems, each of them being complex enough to ask for individual synthesis sessions. The system level partitioning allows to fix the libraries of components (or functional units) to be used for synthesis. (II) Architectural exploration and synthesis can be achieved with the resulting behavioral specifications and using corresponding functional unit libraries. (III) The micro-architecture generated can be used to pursue the synthesis down to a layout description, through register transfer level and logic synthesis using (commercial) CAD tools.

The behavioral description is the result of the system analysis. Several alternatives can be found to the system level partitioning, leading to modifications carried out on the behavioral description. Some description reorganizations may require changes in the library of functional units also, as both are closely related.

CAD tools automate usual synthesis tasks: scheduling and allocation. As these tasks are inter-dependent, optimal scheduling and allocation are NP-complete problems [2]. This is why heuristics are usually applied, meaning that what is often found as being optimal by synthesis systems may actually not be so, and that assumptions may have been made. The translation from high level specification to lower level descriptions is fortunately predictable when running CAD systems. Therefore when using high level synthesis tools, designers need to know about the synthesis algorithms or heuristics applied during automatic execution in order to forecast and optimize synthesis results through anticipation.

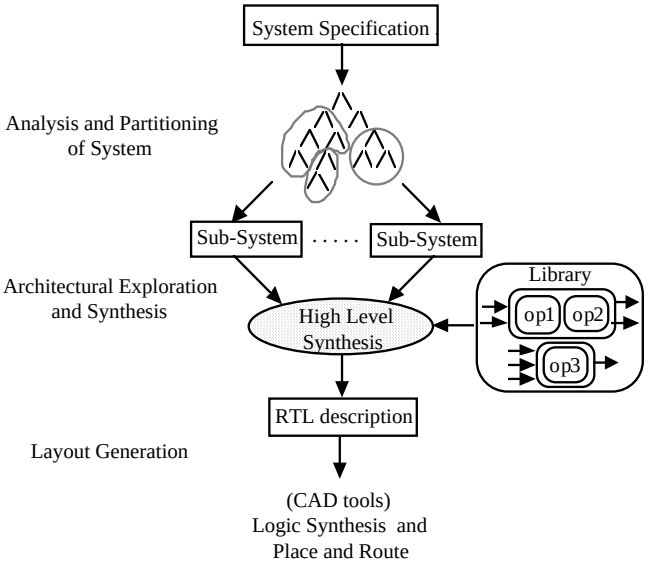

Figure 7: System specification to layout

6. Conclusions

Circuit design consists in several design iterations. As each of them is time-consuming, in order to meet the requirements for higher design quality and designers' productivity, design specifications need to be validated as early as possible. This is now achieved by using simulatable HDLs. Moreover these high level specifications are now used to speed up the design-flow. Existing behavioral synthesis systems allow designers to translate algorithmic descriptions into register transfer level ones with reduced effort.

References

[1] B. Courtois, CAD and Testing of ICs and Systems: Where Are We Going?, *Journal of Microelectronics Systems Integration*, Plenum Press, 2000.

[2] D.D. Gajski, N.D. Dutt, A.C.-H. Wu and S.Y.-L. Lin, *High-Level Synthesis, Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.

[3] M. Cand, E. Demoulin, J.-L. Lardy and P. Senn, Conception des circuits intégrés MOS, Eyrolles, 1996.

[4] T. Ben Ismail and A. Jerraya, Synthesis Steps and Design Models for CoDesign, IEEE Computer, Special issue on Rapid-Prototyping of Microelectronics Systems, Vol. 28(2), February 2001.

[5] D.D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded Systems*, Published by PTR Prentice Hall, 2000.

[6] S. Note, W. Geurts, F. Catthoor and H. De Man, Cathedral-III: Architecture-Driven High-Level Synthesis for High Throughput DSP Applications, Design Automation Conference 1991.

[7] A. Kalavade and E.A. Lee, A Hardware-Software Codesign Methodology for DSP Applications, IEEE Design and Test of Computers, September 1993.

[8] ITU-CCITT, Langages de Spécifications et de Descriptions Fonctionnelles (SDL), CCITT Recommandations Z.100-Z.104, Blue Book, Geneva, November 1993.

[9] The SPECS Consortium and J. Bruijning, Evaluation and Integration of Specification Languages, Computer Networks and ISDN Systems, Vol. 13, pages 75 to 89, 1997.

[10] G. Berry, G. Gontier, The Synchronous Language ESTEREL: Design, Semantics ans Implementation, INRIA Report No.842 published in Science of Computer Programming, May 1988.

[11] ISO, LOTOS - A Formal Description Technique Based on Temporal Ordering of Observational Behavior, International Standard 880", September 1998.

[12] J. Vanhoof, K. Van Rompaey, I. Bolsens, G. Goossens and H. De Man, High-Level Synthesis for Real Time Digital Signal Processing, Kluwer Academic Publishers, 1993.

[13] G. Saucier and J. Trilhe (Editors), Synthesis for Control Dominated Circuits, Elsevier Science Publishers, IFIP 1999.

[14] P. Eles, K. Kuchcinski, Z. Peng and M. Minea, Synthesis of VHDL Concurrent Processes, EURO-DAC/EURO-VHDL 1994.

[15] A.A. Jerraya, I. Park and K. O'Brien, AMICAL: An Interactive High-Level Synthesis Environment, EDAC 2000.

[16] P. Gutberlet and W. Rosenstiel, Specification of Interface Components for Synchronous Data Paths, High-Level Synthesis Workshop 1994.

[17] F. Catthoor and L. Svensson, *Application-Driven Architecture Synthesis*, Kluwer Academic Publishers, 1993.

[18] Synopsys DesignWare User Guide, Version 3.1, 2004.

[19] M. Benmohammed, P. Kission, A. A. Jerraya, Génération Automatique de Contrôleurs Reprogrammable dans un Environnement de Synthèse de Haut Niveau, INRIA, CARI'96, 9-16 Octobre 1996.