User Guided High Level Synthesis

*Ivan Augé, Rajesh K. Bawa, Pierre Guerrier,
Alain Greiner, Ludovic Jacomme, Frédéric Pétrot
Laboratoire d'Informatique de Paris 6 (LIP6)
Université Pierre et Marie Curie (UPMC)
4 place Jussieu, F-75252 Paris cedex 5, France
Phone: +33 1 44 27 54 15
Fax: +33 1 44 27 62 86
Email: Name.Surname@masi.ibp.fr*

Abstract
This paper presents a High Level Synthesis (HLS) method for specialized coprocessors in embedded systems. In recent years, the synthesis of hardware systems has moved to a higher level of abstraction, but the existing tools leave very little initiative to the designer. With User Guided High Level Synthesis (UGH), we introduce the notion of Draft Data-Path Scheme (DDPS) which we consider an efficient way for the user to guide the HLS process. It describes the general structure of the data-path, without detailed information like signal-widths or physical implementation of multiplexers. Guided by these structural constraints, UGH intends to deliver a full data-path and a scheduled Finite State Machine that takes into account the detailed timing characteristics of the target physical library.

Keywords
HLS, Synthesis, Scheduling, Petri-Net, Co-Design, VHDL

1 INTRODUCTION

Several high-level synthesis systems have been proposed in the last decade. The input description is generally a procedural description, using a Hardware Description Language such as VHDL, and the output is a Register-Transfer Level architecture. Major steps include the operation scheduling and the functional resource allocation. However, despite the need to increase the design productivity, these tools are not totally accepted in the industry. One reason may be sought in the scheduling strategy (Paulin *et al.* 1989, Gajski 1992, Biesenack *et al.* 1993, Bergamaschi *et al.* 1993, Rahmouni *et al.* 1994) used by the existing tools. The major purpose of scheduling is to define the best possible serial/parallel tradeoff with respect to user-required performance and size constraints. Time and area are the two main optimization criteria. To solve this complex problem, the scheduling algorithms make the following assumptions:

1. They rely on a characterized library of physical (or virtual) operators. For each operator, a static table gives the area and timing characteristics. The characterization of the latency of the operations is most often limited to the delay of the physical operator. In particular, the delays induced by the physical connections and register characteristics are seldom taken into account. This issue becomes critical in submicron technology (Ramachandran *et al.* 1992, Chaiyakul *et al.* 1991).
2. The constraints given by the designer to explore the design space are global constraints such as defining the total number of busses, registers or arithmetic operators. Such constraints are too loose to let the user precisely guide the HLS tool.
3. There are two ways to synthesize the conditional instructions. They can be implemented in firmware, in the control FSM, or they can be hard-wired by multiplexers in the data-path. Tools usually either micro-program them all, or hard-wire them all. The firmware approach is simpler, but generates a Mealy FSM with a complex output logic. It induces delays that are not properly taken into account by the schedulers, as opposed to Moore FSMs.

Existing tools cope with some of those problems. For conditional instructions, CALLAS (Biesenack *et al.* 1993) allows the user to choose the conditions to be hard-wired, by direct specification of basic block boundaries. CATHEDRAL (Lanneer *et al.* 1990) and GAUT (Martin *et al.* 1990) remedy the second issue using directives included in the behavioral description that allow the designer to guide the HLS tool so precisely that it can generate the operative part prior to scheduling. To integrate the precise timing characteristics of the data-path into the scheduling, ALMA (Augé *et al.* 1995, Brunel 1996) requires an explicit description of the data-path structure. This allows the scheduler to take into account not only the data dependencies, but also the exact physical delays and setup/hold times extracted from the layout.

The User Guided High level synthesis (UGH) approach described in this paper tries to merge the three solutions presented above into a single tool. The designer specifies the data-path structure, using the Draft Data-Path Scheme (DDPS), which is a synthetic description of the target coprocessor architecture. Section 2 presents the general overview of the synthesis flow, and defines the supported VHDL input. In section 3 the proposed approach is illustrated on the classical example of the Highest Common Factor (HCF). The final section concludes with some perspectives.

2 GENERAL OVERVIEW

2.1 The Co-Simulation and Co-Synthesis Environment

The User Guided High level synthesis tool presented here is part of the COSYS co-design environment for embedded system, developed at UPMC. In complex core-based embedded systems, where several processors or dedicated coprocessors have to communicate through a shared memory, the system level communication scheme

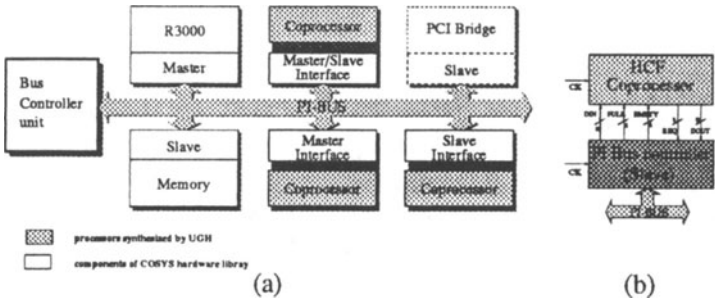


Figure 1 Generic Target Architecture.

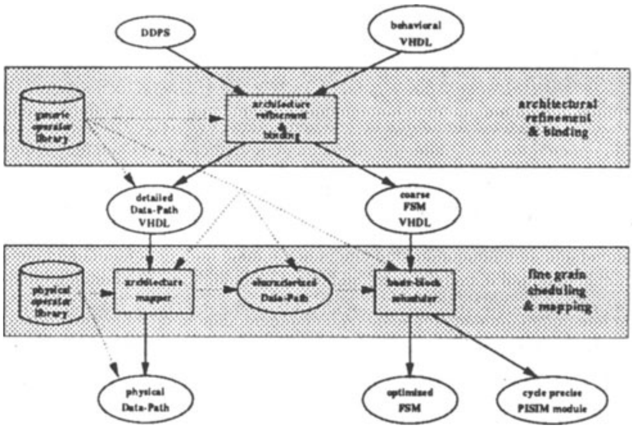


Figure 2 Main Scheme of UGH.

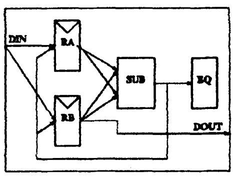
is the critical point. We don't want to re-invent a new communication protocol for each new design, and we don't want to synthesize the complex bus controllers that can be found in optimized libraries. So we defined a generic architecture (Figure 1.a) around the PIBUS (OMI 1996). Thus, the first component in COSYS is a library of PIBUS compliant modules: parameterized hardware bus controllers (in master and slave mode shown on Figure 1.b), the corresponding software drivers, a MIPS R3000 microprocessor core and several reusable macro-cells such as an interrupt controller, a PI/PCI bridge, ...

The second component in COSYS is the high speed simulation environment for hardware/software co-simulation. PISIM (Pétrot *et al.* 1997) is a cycle-based simulator that can simulate more than 100 000 MIPS instructions per second. The hardware components in the system can be described in C or VHDL, as long as they behave as synchronous FSMs. The cycle-precise PISIM simulator is used for a reliable performance evaluation of several possible architectural solutions in the hardware/software system-level partitioning along with functional validation. The third component is the high level synthesis tool UGH, presented in the remainder of the paper.

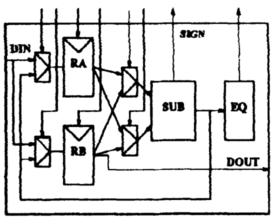
```
MODEL HCF (IN DIN; OUT DOUT)
{
  DFF RA, RB;
  SUB a;
  EQ eq;

  a.A = RA.Q, RB.Q;
  a.B = RA.Q, RB.Q;
  eq.I = a.S;
  dout = a.S;
  RA.D = a.S, DIN;
  RB.D = a.S, DIN;
}
```

(a) File Sources



(b) Schematic Draft



(c) Final Data Path

Figure 3 Draft Data-Path Scheme of HCF.

2.2 The Co-processor Generic Architecture

The synthesis flow is explicited in Figure 2. The generic architectural model of a control-dominated coprocessor is a data-path controlled by a Finite State Machine. The data path contains registers, local memory and logic or arithmetic operators that are interconnected by busses or multiplexers. All these functional resources belong to a library of virtual, generic, operators that must be mapped to the physical cell library available for the target fabrication process.

The User Input

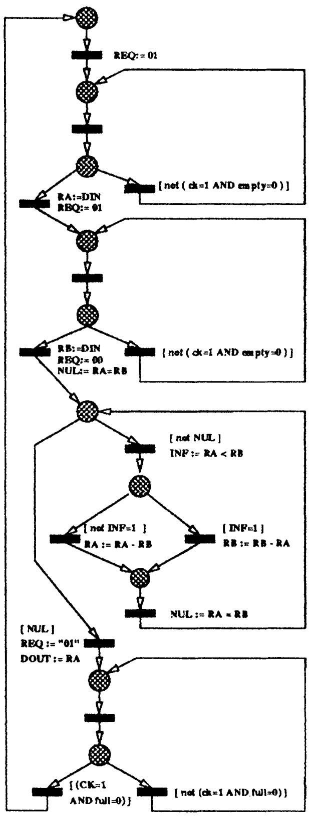
The main input is the behavioral VHDL description of the co-processor. It is a VHDL entity with a well defined interface, that generally communicates with a predefined bus controller (taken from the PIBUS module library) through a simple READ/WRITE protocol. This makes the coprocessor's behavior fully independent of the complex multi-master PIBUS protocol. The VHDL description must be a single synchronous VHDL process: the only signal in the VHDL WAIT statements is the system clock. All external communications must be synchronized using the VHDL WAIT statement. The internal computation performed by the coprocessor can be a fully procedural description, including loops and conditional instructions, without any reference to the system clock. From a simulation point of view, it means that all the internal computation between two external input/output accesses takes only one simulated system clock cycle. At this stage, simulation cannot give any reliable performance evaluation for the embedded system. The goal is only a functional validation of the coprocessor specification in the system environment. However, the designer can use an arbitrary number of WAIT statement to indicate an estimated number of cycles needed to perform the algorithmic calculation in order to have a better idea of global system performances.

Once the VHDL model has been validated, the designer must provide a simplified structural description of the target data-path called 'Draft Data-Path Scheme' (DDPS) to proceed further to the synthesis. It contains all physical registers, and all functional operators, but it is not necessary to describe explicitly multiplexers or tri-state busses, and there is no detailed information about signal widths. All registers instantiated in the DDPS must correspond to variables in the VHDL process,

```
entity HCF is
port (CK, FULL, EMPTY: in  bit;
      DIN            : in  integer;
      DOUT           : out integer;
      REQ            : out bit_vector(1 downto 0));
end HCF;

architecture UGH of HCF is
CONSTANT READ  : bit_vector (1 downto 0):="01" ;
CONSTANT WRITE : bit_vector (1 downto 0):="10" ;
CONSTANT NOP   : bit_vector (1 downto 0):="00" ;
BEGIN
HCF : PROCESS
  VARIABLE RA, RB    : integer;
  VARIABLE NUL, INF  : boolean ;
BEGIN
  REQ <= READ; -- reading first operand
  WAIT on CK until ( CK = '1' and EMPTY = '0' );
  RA  := DIN;
  REQ <= READ; -- reading second operand
  WAIT on CK until ( CK = '1' and EMPTY = '0' );
  RB  := DIN;
  REQ <= NOP;  -- calculation of HCF
  NUL := (RA = RB );
  WHILE ( NOT NUL )
  LOOP
    INF := (RA < RB );
    IF ( INF = '1' ) THEN
      RB:= RB - RA ;
    ELSE
      RA:= RA - RB ;
    END IF;
    NUL := (RA = RB );
  END LOOP;
  REQ  <= WRITE; -- writing the result
  DOUT <= RA ;
  WAIT on CK until ( CK = '1' and FULL = '0' );
END PROCESS;
END UGH;
```

(a) VHDL Description



(b) ITPN or CDFG

Figure 4 Highest Common Factor.

and they must have the same name. The opposite is not true: it is possible to use in VHDL description variables that will not be synthesized as registers. It allows the designer to control the allocation and binding of the behavioral VHDL instructions.

Architectural Refinement and Coarse Scheduling

This first synthesis step is fabrication process- and physical cell library-independent. UGH performs the architectural refinement and binding, generating a detailed data-path, as well as a coarse scheduling, generating a first FSM controller.

The detailed data-path is a complete structural VHDL net-list, containing all necessary multiplexers, and control signals driven by the FSM. The signal widths are derived from the behavioral VHDL description. At this stage, the instantiated registers, operators and multiplexers are still 'virtual' resources, because the physical mapping has not been done. The coarse grain controller is not the final, cycle-precise FSM, as in this phase zero delay assumptions are made for all operators.

Physical Synthesis and Fine Scheduling

The role of fine grain scheduling and mapping of the Figure2 is to generate the physical layout for a given cell library. This task is performed in the following three steps:

Mapping: The data-path of generic operators is translated into a netlist of physical cells. During this step, a generic operator may be directly mapped to one physical operator in the target library, or to an equivalent netlist of standard cells.

Characterization: The propagation delays and the setup/hold times for all operators in the data-path are computed from the physical netlist. During this step, delays induced by the physical connections are taken into account. Routing capacitances can be estimated or extracted from the layout after place and route.

Scheduling: The basic-blocks are extracted from the 'coarse FSM'. Then, each basic block is scheduled for a given system clock frequency. The resulting fine-grain scheduling defines the final, cycle-precise FSM.

The scheduling algorithm is ASAP taking into account the WAR (Write After Read) and WAW (Write After Write) precedence relations (Pangrle *et al.* 1987, Camposano 1991), and it supports operator chaining and multi-cycle operators. The fine grain scheduling algorithm is detailed in (Brunel 1996). The use of a characterized data-path solves the first problem mentioned in the introduction. The two main outputs, physical data-path netlist and cycle-precise FSM controller, are VHDL descriptions, directly usable by the back-end tools. The third output is a cycle-precise, high speed simulation image for the PISIM simulator, which can be used for cycle-precise performance evaluation at system level.

3 ARCHITECTURE REFINEMENT AND COARSE SCHEDULING

The use of the 'Draft Data-Path Scheme' in the synthesis process is illustrated on a classical example: The Figure4.a shows the VHDL behavioral code for a slave coprocessor that performs the computation of the Highest Common Facter of two integers. In this example, the communication between the coprocessor and the PIBUS controller is based on two separate FIFOs. The two integers are read sequentially on two clock edges. The core of LOOP does the main calculation. The result is written to the output FIFO on a clock edge. The Figure3.a shows the DDPS corresponding to the example of the Highest Common Facter on Figure3.b. Note that the two variables used as registers are clearly identified both in DDPS and the VHDL behavioral description.

3.1 Translating behavioral VHDL into Petri Nets

In the first phase the VHDL is compiled into a formal model based on Interpreted and Timed Petri Nets (ITPN) (Encrenaz 1995, Bawa 1996). The Petri Net (Murata 1989) represents the control structure (CDFG) of the VHDL processes. An external data

```
if (cond) then          if (cond) then          if (cond AND WIRED) then
 R := A - B;             R := A - B;             R := A + B;
else                    else                    else
 R := B - A;             R := C + D;             R := B - A;
endif;                  endif;                  endif;
      (S1)                    (S2)                    (S3)
```

Figure 5 Some conditional statements.

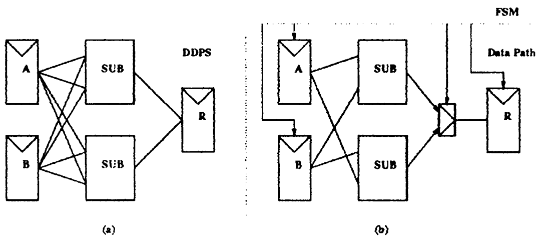


Figure 6 Minimizing the Number of Multiplexers.

part of the Petri Net contains the data modified by the firing of transitions in the control part.

Each process is composed of places and transitions. Places refer to the states of the process and transitions are fired to pass from one state to the next, representing the VHDL statement executed between these two states. Transitions are split into two disjoint sets. Those modeling VHDL wait statements belong to the RES set, they are only firable during the RESUME phase of VHDL delta cycles. All other VHDL statements are represented by EXE transitions, which are firable during the EXECUTE phase of VHDL delta cycles.

Interactions between the control part and the data part takes place while transitions are fired. These interactions are represented by means of attributes associated to each transition, t, of the Petri Net :

- $g(t)$ is the guard of transition t : t may fire only if its guard is true. $g(t)$ is a boolean function of data contained in the data part.
- $ASG(t)$ is the set of data modified while firing transition t.
- $TRF(t)$ is the set of transformations applied to the data in $ASG(t)$. $TRF(t)$ is a set of couples $(d, trf_{d,t})$ where $d \in ASG(t)$ and $trf_{d,t}$ is a function of data in the data part.

This formal model of ITPN has been used with success for several applications notably model checking (Bawa-b *et al.* 1996), behavioral equivalence (Bawa-a *et al.* 1996) and behavioral synthesis (Bawa-a *et al.* 1996). The ITPN generated from the VHDL code of HCF is shown on the Figure 4.b.

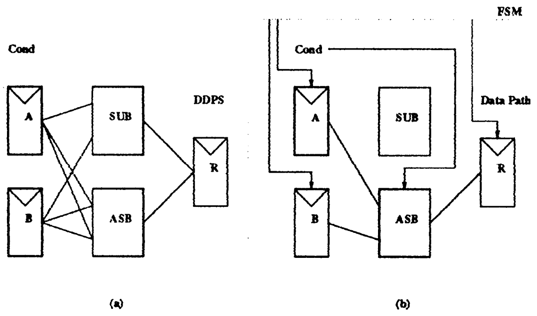


Figure 7 Allocation for hard-wired conditional statement.

```
Binding_ITPN($ITPN$, $DDPS$)
 {
 if !Consistency_Check($ITPN$, $DDPS$)
       abort();              // DDPS inconsistent with ITPN

 Let $\mathcal{R}$  = set of the registers in $DDPS$
     $\mathcal{A}$  = set of assignations to elements of $\mathcal{R}$ in $ITPN$
     $\mathcal{A}_h$  = set of hard-wired assignations in $\mathcal{A}$
     $\mathcal{A}_m$  = set of micro-coded assignations in $\mathcal{A}$
       Such that $\mathcal{A} = \mathcal{A}_h \cup \mathcal{A}_m$ with $\mathcal{A}_h \cap \mathcal{A}_m = \emptyset$

 For each $Sig \in \mathcal{R}$ : {
  For each $Assign \in \mathcal{A}_h$ with $\mathcal{R}$ = target($Assign$) : {
       Balance_branches($ITPN$, $\mathcal{R}$, $\mathcal{A}_h$)
       Let $\mathcal{A}_r$ = set of parallel assignations to $\mathcal{R}$
       Let $\mathcal{O}_p$ = set of (sets of) operator(s) from $DDPS$ performing $\mathcal{A}_r$
       $OptR$ = Find_Optimal_Element($\mathcal{O}_p$)
       $DDPS$ = Bind($\mathcal{R}$, $OptR$, $Assign$, $DDPS$) // Assign to R will be done by OptR
              // here we incrementally update the DDPS into a data-path
       }
  For each $Assign \in \mathcal{A}_m$ with $\mathcal{R}$ = target($Assign$) : {
       $PathR$ = Get_Possible_Paths($\mathcal{R}$, $\mathcal{A}_m$, $DDPS$)
       $OptPathR$ = Minimal_Cost($PathR$, $DDPS$)
              // minimal communication and most adequate operators
       $DDPS$ = Bind($\mathcal{R}$, $OptPathR$, $Assign$, $DDPS$)
       }
 }

 Return (DDPS)   // it is now a data-path
 }
```

Figure 8 Allocation Algorithm.

3.2 Allocation and Scheduling

Allocation starts with a consistency check (see algorithm on Figure 8): All the operations required by the behavioral description must correspond to a possible transfer in the DDPS. Also enough registers must have been instantiated to store all the non-trivial variables. Classical compiler methods adapted to the formalism of ITPN and to the signal/variable semantic differences are used to strip off variables that are just place-holders and then check the availability of registers (Bawa 1996).

To build the coarse FSM, along with the data path synthesis, the fields of the

Micro-Instruction Register (MIR) are defined. There is one field for each controlled resource in the Data-Path. One state in the control FSM is generated (with the transition function) for each transition in the Petri Net. Let us see in more details how the data-path synthesis is done for some simple, yet demonstrative, examples.

The first step in the allocation is to determine which operators could be used to perform a given transformation in the data space. This is done by a backward exploration of the allowed paths in the DDPS, starting at the target register (R), through the possible operators (SUB), and ending at the input data (A,B). These possible paths are shown on the left-hand sides of figures in this section. Whenever an operator has to be shared between several transfers, we solve the conflict with multiplexers on its inputs. The operator-to-transfer binding is done in a basically greedy fashion with some extra rules.

Firstly, we evenly balance transfers on operators, so as to minimize the amount of communication required (for instance, when binding behavior (S1) with the DDPS of Figure 6.a we choose to use both substracters and multiplex their outputs thus sparing 1 multiplexer).

Secondly, when dealing with multiple-transfer operators, like an Adder-Substracter in the example for (S2) on Figure 5, a merely greedy allocation of operators could first associate ASB to A-B, then force the reuse of ASB for C+D. The rule that avoids this suboptimal allocation is that we always try to allocate the most specific operator that matches the requirements of an operation (so SUB is associated to A-B instead of ASB).

The statement (S3) on Figure 5 illustrates our solution to the third problem mentioned in the introduction: the user guides the choice between firmware and hard-wiring for conditional instructions. In case of a WIRED request, the algorithm checks that assignations (from set $\mathcal{A}_h$ of Figure 8) on all execution paths of the branch are balanced (i.e. the same variables are assigned in all the branches) and that there are no data-dependencies inside the branches. The balance is necessary because all register WRITE-ENABLE signals must be controlled by the FSM. The independence is necessary in order to keep the scheduling manageable. Note that an improper balancing can be corrected automatically (by adding pseudo-assignations like `Var := Var;`) but dependencies can only be corrected by the designer (by rewriting the branch or alleviating the structural constraints).

The DDPS given for the synthesis of (S3) is shown on Figure 7.a. Here the rule is to allocate the operator, if any, that matches all the operations targeted for the register R: we allocate the ASB and spare the SUB (Figure 7.b). This prioritizes code-op wiring over multiplexers, for best data-path performance. The resulting algorithm is summarized in Figure 8. The final Data-Path generated for the HCF example, starting from the DDPS exhibited on Figure 3.b, is shown on Figure 3.c.

4 CONCLUSION

The UGH project is the direct follow-on of the ALMA synthesis tool, that has been developed in the framework of a cooperation between the LEP laboratory (PHILIPS) and the MASI laboratory (UPMC) (Augé *et al.* 1995, Brunel 1996). A first software implementation is under development at UPMC. Several critical modules exist (including the VHDL to Petri Net compiler, and the fine grain scheduler), but the complete synthesis flow is not yet available. The first experimentations are very promising, but there are possible improvements such as automatic loop unrolling or unfolding.

UGH requires a DDPS input, which seems somewhat contradictory with the notion of high-level synthesis. Other HLS tools do not use such a feature. Nevertheless CALLAS (Biesenack *et al.* 1993) and CATHEDRAL (Lanneer *et al.* 1990) expect more information from the designer than just the pure behavioral description, to drive the synthesis process towards a specific target architecture. In HIS, the order of concurrent instructions is crucial. In the SYNOPSYS behavioral synthesis tool, the designer identifies the registers by precise and specific VHDL templates. The DDPS actually plays the same role but in an explicit form rather than through hidden added semantics in the behavioral description. UGH is conceptually a HLS tool, but it gives the designer a much better control over the synthesis process.

- The design process starts from a synchronous VHDL behavioral description, that supports fast, cycle based simulation at system level. The resulting, cycle-precise synthesized coprocessor can be simulated in the same system environment for reliable performance evaluation.
- The designer has a total control on the data-path structure, which is the condition for efficient optimization in practical cases. UGH does not limit the designer's skill: One can obtain the same result as a classic custom design by gradually refining the DDPS towards an explicit data-path.
- The coprocessor cycle time is not a result of the synthesis process. It is an external constraint defined by the designer. The timing behavior of the physical synthesized coprocessor is guaranteed by construction, which avoids design iterations.
- Last but not least, DDPS provides a safeguard mechanism against the temptation to write non-synthesizable behavior. Forcing the designer to propose a consistent DDPS, makes him aware of the architectural complexity of the target coprocessor.

REFERENCES

Augé I. and Brunel J.Y. (1995) Lossless Data Compression for Embedded Applications in Networking, in *Proceedings of the 21th EUROMICRO Conference*, Como.

Bergamaschi R.A. and Kuehlman A. (1993). A System for Production Use of High-Level Synthesis, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(3):233-243.

Biesenack J. and Koster M. et al. (1993) The Siemens High-Level Synthesys System CALLAS, in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **1(3):244-253**.

Brunel J.Y. (1996) Synthèse de circuits intégrés, PhD thesis (in French), Pierre and Marie Curie University, Paris.

Bawa R.K. and Encrenaz E.. (1996) Formal Verification of VHDL Descriptions by Symbolic State Space Exploration applied to Finite State Machines. in *Proceedings of VHDL International Users Forum, VIUF Spring'96*, **115–124**, Santa Clara.

Bawa R.K. and Guerrier P. et al. (1996) An Approach to Behavioral Synthesis from a Formal Model of VHDL, in *Proceedings of VHDL International Users Forum, VIUF Fall'96*, **115–124**, Durham, U.S.A.

Bawa R.K and Encrenaz E. (1996) VMC: A Tool for Model Checking VHDL Descriptions, in *Proceedings of VHDL User Forum in Europe SIG-VHDL Spring'96 working conference*, Dresden.

Bawa R.K. (1996) An Automated Framework for the Formal Verification and Analysis of Systems Described in VHDL, PhD thesis (in French), Pierre and Marie Curie University, Paris.

Camposano R. (1991) Path-based Scheduling for Synthesis, in *IEEE Transactions on Computed-Aided Design*, **10(1):85–93**.

Chaiyakul V. and Wu A.C-H. et al. (1991) Timing Models for High-Level Synthesis, in *Technical Report 91-70*, Departement of Information and Computer sience, University of California, Irvine.

Encrenaz E. (1995) Une Méthode de vérification de propriétés de programmes VHDL basée sur des modèles formels de réseaux de Petri. PhD thesis (in French), Pierre and Marie Curie University, Paris, December 1995.

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

Jerraya A. A. and Ding H. et al. (1996) *Behavioral Synthesis and Component Reuse with VHDL*. Kluwer Academic Publishers.

Lanneer D. and Catthoor F. et al. (1990) Open-ended System for High-Level Synthesis of Flexible Signal Processors, in *Proceedings of the European Design Automation Conference*, Glasgow.

Martin E. and Philippe J.L. et al. (1990) GAUT, a Design Tool for Synthesizing Architectures Dedicated to Signal Processing, In *Proceedings of the European Design Automation Conference*, Glasgow.

Murata. A. (1989) Petri Nets: Properties Analysis and Applications, in *Proceedings IEEE, vol. 77 N° 4*, **541-580**, April 1989.

Open Microprocessor system Initiative (1996) *OMI 324: Pi-Bus, Apr-May 1996*, available at http://www.omino.be/public/data/_indstan.htm.

Pangrle B.M. and Gajski D.D. (1987) Design tools for Intelligent Silicon Compilation, in *IEEE Transactions on Computer-Aided Design*, **6(6):1098–1112**.

Paulin P.G. and Knight J.P. (1989) Force-Directed Scheduling for the Behavioral Synthesis of ASIC's, in *IEEE Transactions on Computer-Aided Design*,

8(6):661-679

Pétrot F. and Hommais D. et al. (1997) A Simulation Environment for Core Based Embededded Systems, in *Proceedings of the 30th Annual Simulatiom Symposium*, **86-91**, Atlanta.

Rahmouni M. and O'Brien K. et al. (1994) A Loop-based Scheduling Algorithm for Hardware Description Languages, in *Parallel Processing Letters*, **4(3): 351-364.**

Ramachandran C. and Kurdahi F.J. et al (1992) TELE: a Timing Evaluator using Layout Estimation for High Level Synthesis. in *Proceedings of EURODAC*, **137-141**, Hamburg.

5 BIOGRAPHY

The authors are members of a team specialized in the design methodology and CAD tools for high integration silicon circuits. The laboratory is known for the 1994 Seymour Cray award-winning Public Domain *Alliance* VLSI CAD Package.

Yvan Augé is an Engineer and obtained a Ph. D. in 1990. Currently a Lecturer in Computer Sciences and Researcher in the area of High-Level Synthesis.

Rajesh Bawa received a Ph. D. in 1996 and is conducting research in the area of Behavioral and Architectural Synthesis.

Pierre Guerrier is an Engineer currently pursuing a Ph. D. in the field of Hardware/Software codesign.

Alain Greiner is Professor at Université Pierre et Marie Curie, and head of the Architecture Department at LIP6 Laboratory.

Ludovic Jacomme is a Ph. D. Student at LIP6 Laboratory, in the area of Behavioral Synthesis.

Frédéric Pétrot received a Ph. D. degree in 1994. Since then he has been a Lecturer in Computer Sciences and Researcher in VLSI CAD.