Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 28636, Pages 1–11
DOI 10.1155/ASP/2006/28636

Macrocell Builder: IP-Block-Based Design Environment for High-Throughput VLSI Dedicated Digital Signal Processing Systems

Nacer-Eddine Zergainoh,[1] Ludovic Tambour,[1, 2, 3] Pascal Urard,[2] and Ahmed Amine Jerraya[1]

[1] *TIMA Laboratory, National Polytechnique Institute of Grenoble, 46 Avenue Félix Viallet, 38031 Grenoble Cedex 1, France*
[2] *ST Microelectronics, 850 Rue Jean Monnet, 38926 Crolles Cedex, France*
[3] *CIRAD, TA 40/01, avenue Agropolis Lavalette, 34398 Montpellier Cedex 5, France*

Received 3 October 2004; Revised 14 April 2005; Accepted 25 May 2005

We propose an efficient IP-block-based design environment for high-throughput VLSI systems. The flow generates SystemC register-transfer-level (RTL) architecture, starting from a Matlab functional model described as a netlist of functional IP. The refinement model inserts automatically control structures to manage delays induced by the use of RTL IPs. It also inserts a control structure to coordinate the execution of parallel clocked IP. The delays may be managed by registers or by counters included in the control structure. The flow has been used successfully in three real-world DSP systems. The experimentations show that the approach can produce efficient RTL architecture and allows to save huge amount of time.

1. INTRODUCTION

As the complexity of the high-throughput dedicated digital signal processing (DSP) systems under hardware design increases, development efforts increase dramatically. At the same time, the market dynamics for electronic systems push for shorter and shorter development times [1]. In order to meet the design time requirements, a design methodology for VLSI dedicated DSP system that favors reuse and early error detection is essential. One idea, largely widespread and applied to design DSP systems, is to adopt a modular approach based on divide-and-conquer strategy (recursive). The global complexity of the system should be divided into subsystems (i.e., elementary signal processing functions), well known and of easily accessible complexity such as filter (FIR, IIR), fast Fourier transform (FFT), Viterbi decoder, and so forth. The system can be obtained by the hierarchical assembly of these common functions of signal processing (also known as IP blocks). The intellectual-property- (IP)-based design is obviously an important issue for improving not only design productivity, but also design from the higher-level abstraction [2, 3].

However, designers encounter two major problems with the IP-block-based design approach [2–4]. The *first problem* is the difficulty in using IPs blocks for high-throughput DSP systems that require various performances (throughput) or functions with nonstandard algorithms [5]. This is because VLSI DSP system cannot be parameterized for global performance and functions; for example, necessary processing cycles cannot be adjusted for IPs blocks. The *second problem* comes from interfacing of IPs blocks between themselves. Designers have to design IPs blocks that can communicate according to the blocks' interface specification. When they connect two different IP blocks, they have to insert an extra interface circuitry in order to synchronize them. Area and delay overhead for circuitry cannot be neglected in some cases. Our goal is to find some appropriate design tactics to avoid these problems.

In this paper, we propose an efficient IP-block-based design environment for high-throughput VLSI dedicated digital signal processing (DSP) systems called DSP macrocells builder tool. The flow generates SystemC register-transfer-level (RTL) architecture, starting from a Matlab functional model described as a netlist of functional IP. To provide IPs with more reusability and flexibility, we use parameterized reusable DSP components at functional and RT levels. Thus, by setting the appropriate parameters, unnecessary functions and redundant interfaces are eliminated in our IP-based design approach. The refinement process inserts automatically control structures to treat delays induced by the use of RTL IPs. It also inserts a control structure to coordinate the execution of parallel clocked IP. The delays may be managed by registers or by counters included in the control structure. The main contribution of this paper is a prototype implementation and experimentation of the approach. The rest of the paper is organized as follows. After investigating related work in Section 2, we introduce our methodology and discuss its merits, the important issues, and how this approach handled the IP-based design problems. Section 4 details the IP-block-based design environment for high-throughput VLSI DSP systems. Section 5 describes several experiments to analyze the efficiency of the proposed design flow and Section 6 concludes the paper.

2. RELATED WORK

2.1. Standard design flow for ASIC

A standard design flow for hardware implementation of algorithms has four phases which are typically handled by four different designers. Algorithm designers conceive the chip and deliver a specification to system designers, often in the form of a floating-point simulation. The system or architecture designers begin to add structure to this simulation, partitioning the design into functional units. They must also convert the data types from floating to fixed-point and verify that finite word-length effects and pipeline depth do not compromise the algorithm. The hardware designers map the simulation RTL code and verify that the code matches the specified functionality and pipeline depth. Physical designers take standard-cell netlists synthesized from the RTL code and generate layout mask patterns. This flow requires three translations of the design, expressing the functionality as gradually less sequential and more structural with requirements for reverification at each stage. Opportunities for algorithmic modifications, to reduce power and area, are often lost due to the separation of engineering decisions. Performance bottlenecks discovered during the physical design phase are unknown to the algorithm designer. Aggressive system requirements may require new and unusual architectures, which can stall the flow, leading to uncontrolled looping back to earlier stages of the design process and extending the design time indefinitely. The main problem with this flow is that it attempts to avoid feedback information to algorithm designers.

The flow we need would allow algorithm designers to explore the design space as thoroughly as possible by creating RTL model and obtaining performance estimates. This exploration should allow refinement of fixed-point types to be constrained libraries of efficient hardware blocks, and to be carried out by automated design flow. This encourages feedback of RTL design issues to algorithm designers by allowing them to maintain ownership of the design data at all times. It also would encourage interaction with algorithm and hardware designers by reducing the design process to a single phase.

2.2. Current methods and flows for DSP algorithm implementation

Recent efforts have identified the gaps between algorithm, system, hardware, and physical design, but yet have to encompass the complete problem. Some attempt to close the gap between algorithm and hardware design by basing synthesis tools on C/C++ description [6–11]. However, these solutions require a style of code that is very similar to RTL code and it is unattractive to algorithm designers. Commercial tools from design automation companies offer RTL code generation solutions from block diagrams [12]. However, these tools are targeted mostly for hardware designers and obscure the information about the algorithm and architecture through the code generation process.

Some have proposed using high-level system design flows, such as Ptolemy [13], and POLIS [14]. These flows emphasize overall system cosimulation and cosynthesis for heterogeneous systems rather than the details required in creating and integrating DSP-ASIC into an existing system. There are also some works on system-level design flows targeted for DSP hardware systems [15–19]. Grape-II [15], Champion [16], Logic foundry [17], and MATCH [18] follow this scheme. In Grape-II, the target architecture consists of commercial DSP processors, bond-out versions of core processors, and FPGAs linked to form a powerful heterogeneous multiprocessor. The Logic foundry is system-level design flow for the rapid creation and integration of FPGA-based DSP by using predictable, preverified IP blocks that have standardized interfaces. The problem of this approach is that the area and delay overhead for standard interface circuitry cannot be neglected in some cases. Champion is IP-block-based design approach for data path of DSP-ASIC. The design automation of data path is performed using two libraries of predesigned basic blocks (functional and cells libraries). Unfortunately, the lack of flexibility of libraries (no parameterized blocks) limits the reuse of the IP blocks especially for high-throughput DSP systems which require various performances (throughput) or functions with nonstandard algorithms. This work was also limited to data paths without runtime control considerations. MATCH has attempted to compile high-level languages, such as Matlab, directly into hardware implementations (including code for DSP, embedded processors, and FPGA).

However, we believe that in all above works, design methodologies tackle some issues of DSP design but they yet have to encompass the entire problem. In fact, most of the above-mentioned approaches cannot satisfy a tradeoff between architecture quality, rapid algorithm/architecture exploration, and fast modeling and validation.

2.3. IP-based design issues

A lot of research has been carried out on the IP-based design [2–4, 20–27]. Most of the research deals with IP-based SoC [2, 20–23]. Problems on SoC synthesis are addressed in [23], where it is assumed that an external reference clock is supplied and the asynchronous communication is used. However, most of on-chip buses for SoC use the synchronous communication. IP blocks are also exploited in the application-specific instruction-set processor (ASIP) synthesis for the embedded DSP software [26]. To accelerate the execution of the software, they select an optimal set of IPs

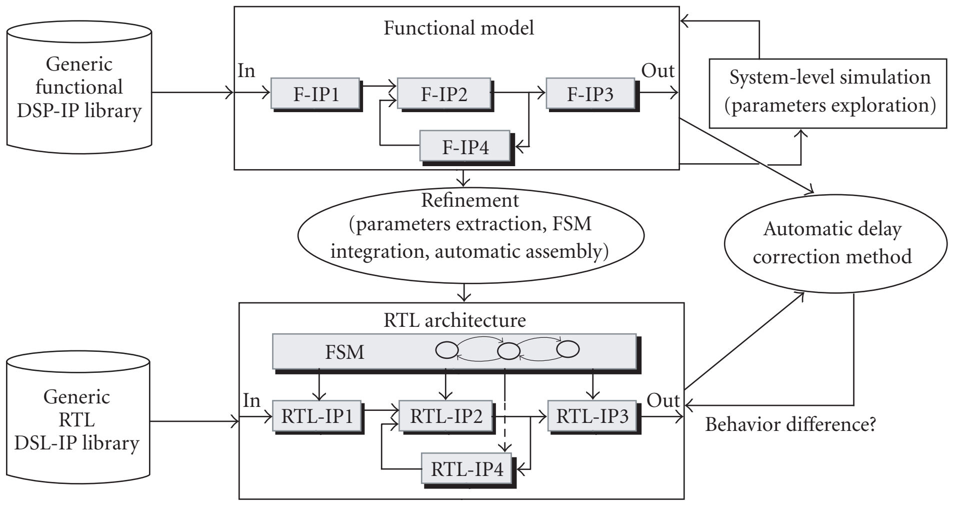

Generic functional DSP-IP library
Functional model
In
F-IP1
F-IP2
F-IP3
F-IP4
Out
System-level simulation (parameters exploration)
Refinement (parameters extraction, FSM integration, automatic assembly)
Automatic delay correction method
RTL architecture
FSM
Generic RTL DSL-IP library
In
RTL-IP1
RTL-IP2
RTL-IP3
RTL-IP4
Out
Behavior difference?

Figure 1: IP-based design methodology for VLSI dedicated DSP.

and interface types for each IP. However, the interface types for IPs are restricted to coprocessor integration style. Interrupt/trap or shared-mapped I/O memories are often used. The software called handshaking offers flexible communication between hardware and software, but it is too slow. Some researchers are trying to develop general communication interfaces in hardware. In the area of the application-specific integrated circuit (ASIC) design, communication between IPs is often conducted by shared registers or shared memories. The typical interface configuration contains multiplexes with enable signals or address decoders. The concept of a generic virtual interface has been attracting a lot of attention as a way to increase the design reuse. General virtual interfaces lead to designers believing that any IP could communicate with any other IP [27]. Some practical approaches are reported such as the automatic matching/generation/deletion of interface pins [3, 4, 24, 25]. General virtual interfaces are kinds of wrapper IPs, so they would have the area and delay overhead.

None of the above works solve the two above-mentioned problems for the high-throughput ASIC DSP systems. The main contribution in this paper is to provide some appropriate design tactics to avoid these problems.

3. OVERVIEW OF DESIGN METHODOLOGY

The IP-based design methodology is based on designer's practice [28, 29]. The methodology, described in Figure 1, generates register-transfer-level (RTL) architecture starting from a functional model, given in Matlab. The functional modeling and RTL architecture generation are performed using two libraries of predesigned DSP basic blocks (functional and RTL libraries).

In our IP-block-based design approach, the functional model is created by assembly of existing functional IP written in Matlab [30]. The refinement process keeps the same architecture and replaces each functional IP by a corresponding RTL one, according to a set of parameters given by the designer. These are present as attributes in the functional model. The choice of IP parameter values (i.e., architectural parameter values such as bit width) is made by the system designer in order to satisfy a tradeoff between signal quality and implementation constraints. To generate the architecture, IP parameter values are firstly extracted from a validated functional architecture model and then used to instantiate the predesigned RTL IP written in synthesizable hardware language (i.e., VHDL, SystemC). The architecture is then built by automatic assembly of predesigned RTL IPs (with the same assembly topology as the functional model). The connection between the RTL IPs is made by name. The design flow includes a unified verification platform used to verify both RTL and functional models.

The platform exploits directly the high-level environment used for functional validation. The results of the methodology are a safety functional and RTL models of the whole DSP application. The functional model can be used as an executable reference for the next generation of design. Overall, the final architecture takes implicit advantage of the hardware designer expertise. The RTL model is suitable for logical synthesis.

3.1. Generic DSP-IPs blocks

To provide IPs with more reusability and flexibility *(problem 1)*, we are developing parameterized reusable DSP components at functional and RTL levels called "generic F-IP" and "generic RTL IP." We define the generic F-IP as template described in Matlab hybrid representation; many details are left open, only some signals which are relevant for the quantification are implemented in quantified integer. The generic F-IP blocks are stored in library. We define the generic RTL IP as a synthesizable RTL model of a basic DSP block. Each F-IP is mapped on one or more RTL IPs. A typical RTL IP is shown in Figure 2 (where the generic

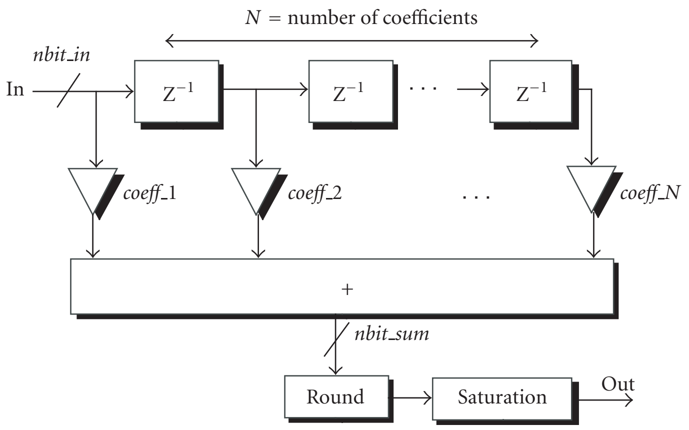


Figure 2: Generic RTL-IP description of FIR filter.

parameters are in italic). The external interface concepts (e.g., external ports-structure, functional, and timing details, generic parameters, etc.) of IPs provide how the IP block exchanges information with its environment. The F-IP interface defines the component name, I/O data stream names, and generic parameters names. The external interface information of RTL-IP block is described by the component definition (including component name, generic parameters names, ports names, ports directions, and ports data types). The ports can be data, clock, reset, control, and test ports. Figure 3 illustrates the analogy between F-IP and RTL-IP interfaces. Therefore, just by setting appropriate parameters (*problem 2*), unnecessary functions and redundant interfaces are eliminated in the IP-based assembly approach (no need to insert an extra interface circuitry). Furthermore, the designer does not have to pay attention to the communication of interface protocols.

3.2. Overview of automatic delay correction method (ADCM)

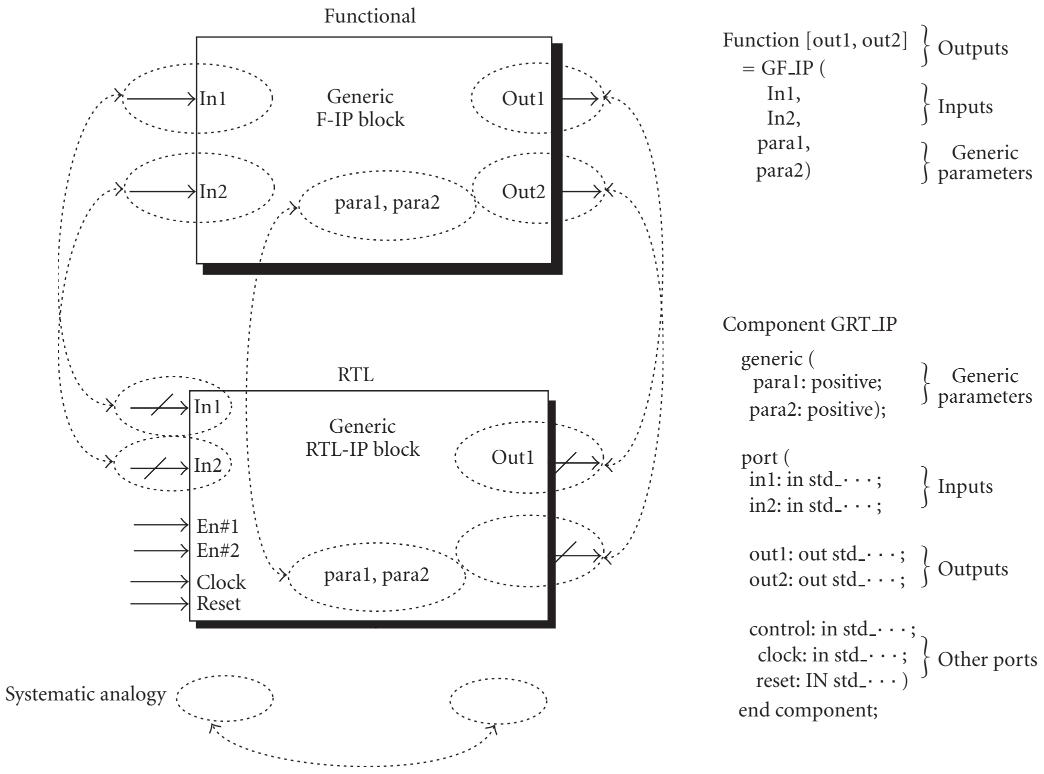
Although each functional IP and its equivalent RTL produce individually the same digital values, in some cases, the register-transfer model obtained by automatic assembly from the functional model can be wrong [31]. This can occur due to delays induced by implementation constraints (pipeline registers, output buffers, etc.). This behavioral fault is caused by the existence of delays in the RTL model, which cannot be found in the functional model. These delays occur when the DSP application contains parallel branches of IPs converging towards another IP, feedback loops of IPs, and/or time-depending IP. This problem is generally known as retiming issue.

There are three main techniques able to correct the different behavior between the two models. The first technique involves the insertion of synchronization protocol (e.g., handshake protocol) for each IP component, which indicates when the input and output data are valid. The advantage of this technique is that the delay problems are solved before the assembly of RTL IPs. The main drawback of this technique is the area and delay overhead for each IP. However, the problem occurs only in the three cases above. The second technique involves the insertion of registers between RTL IPs in order to compensate the additive delays. This technique has the advantage of being nonintrusive. However, performing the corrections manually (i.e., locating the places where the problems are, determining exactly how many registers need to be inserted, and where to insert them) is a very difficult task, increasing the number of IPs. The third technique involves the modification of the initial finite-state machine (FSM) by generating additional signals to control the IPs. These signals are time shifted of the initial signals of the global FSM. Therefore, they are able to put back or to put forward the activation of the IPs. This technique adopts various stages of the second technique (i.e., locating the places where the problems are, determining exactly how many signals are needed) and requires the FSM to be modified. Modification costs more because of the complexity of the FSM (multiplies the number of control signals and increases the number of states).

In our IP-block-based design, we have implemented a systematic approach called "automatic delay correction method (ADCM) to solve the problem without inserting an extra interface circuitry. The ADCM implements efficiently the last two techniques (register-insertion-based and FSM modification-based). We have developed two algorithms (algorithm-1 and algorithm-2) to perform ADCM [31]. The first algorithm (algorithm-1), similar to the Bellman labeling algorithm [32], determines an optimal solution in latency; whereas the second algorithm (algorithm-2), similar to the simplex algorithm [32], determines an optimal solution in the number of inserted registers (i.e., optimizing area).

4. DSP MACROCELL BUILDER: IP-BLOCK-BASED DESIGN ENVIRONMENT FOR VLSI DEDICATED DSP

Our IP-block-based design environment called "*DSP macrocell builder*" shown in Figure 4 consists of system-level validation flow, hardware design flow (including data path and



```
Function [out1, out2]   } Outputs
  = GF_IP (
    In1,                } Inputs
    In2,
   para1,               } Generic
   para2)               } parameters

Component GRT_IP
  generic (
    para1: positive;    } Generic
   para2: positive);    } parameters

  port (
   in1: in std_· · · ;  } Inputs
   in2: in std_· · · ;

   out1: out std_· · · ; } Outputs
   out2: out std_· · · ;

   control: in std_· · · ; } Other ports
    clock: in std_· · · ;
    reset: IN std_· · · )
  end component;
```

Figure 3: F-IP interface versus RTL-IP interface.

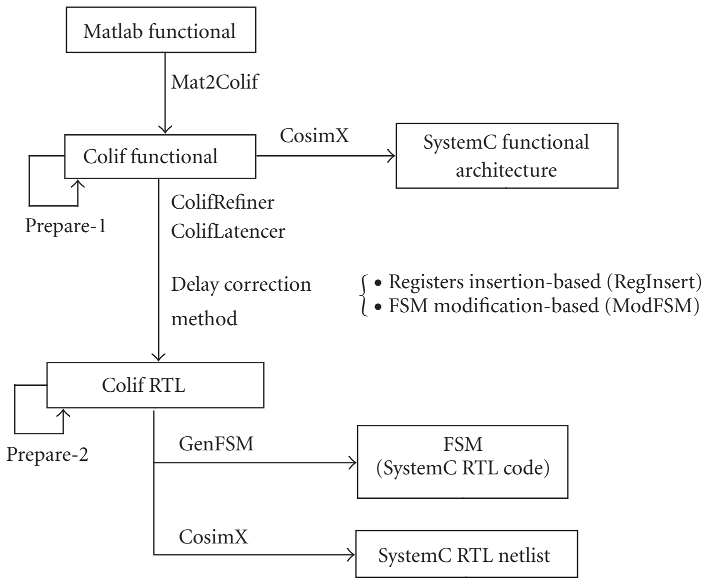


Figure 4: DSP macrocell builder.

FSM), and delay correction flow for high-throughput VLSI dedicated DSP systems. The main feature of our configuration is that the tool flow is based on a unified design model for simulation and synthesis of system-on-chip (SoC) architectures, called "*Colif*" [33]. Other tools take advantage of information from the Colif and the characteristics of the generic IPs libraries. Initially, a designer uses the generic F-IP library to describe his functional model in Matlab [30]. The next step is to explore a pure algorithm for DSP system using Matlab environment. Then, the Mat2Colif tool transforms the Matlab description into Colif description. IP parameter values are extracted from a validated functional model

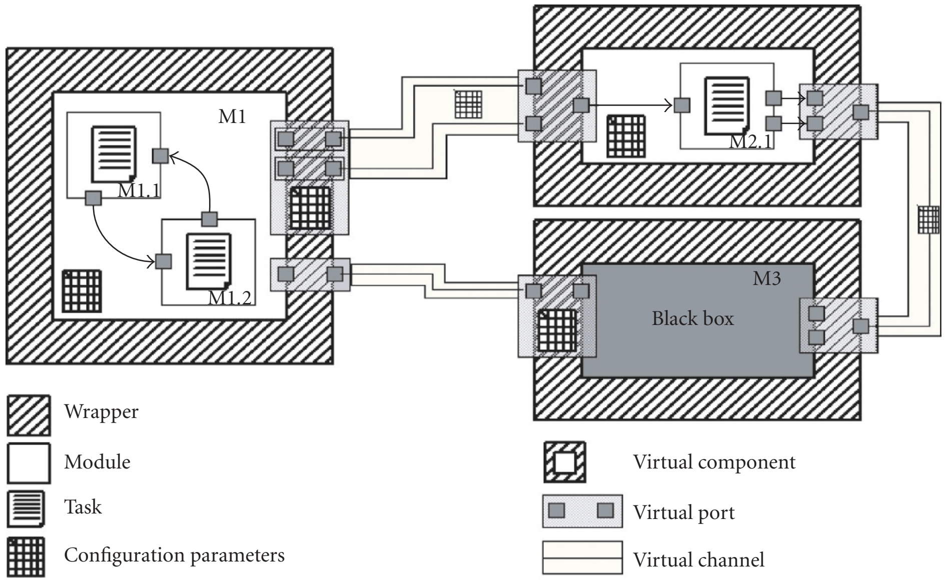


FIGURE 5: Colif representation.

and then used by Prepare1 and CosimX tools to generate the functional architecture in SystemC [11]. The delay correction flow (including ColifRefiner, ColifLatencer, and ADCM), as will be explained later, transforms the Colif functional into a *corrected* Colif RTL. Architectural parameters are used to instantiate the predesigned RTL IP written in synthesizable hardware language (i.e., VHDL, SystemC). The DSP macrocell builder includes the automatic generation of RTL SystemC of the final architecture (including data path and FSM). After cycle-level simulation, the generated architecture can be passed to a logic synthesis, automatic placement, and routing tools, in order to achieve a good performance circuit.

The following subsections detail the several automatic phases of the macrocell builder.

4.1. Colif (Codesign language-independent format)

Colif is a unified abstract model for high-level system design and refinement methodology [33]. Colif represents a system as a hierarchical network of virtual components using three basic concepts: module, port, and net. Virtual components use wrappers to separate the interface of the internal component from the interface of external nets (see Figure 5). The wrapper is the set of virtual ports of a virtual component. Virtual ports contain internal and external ports that can be different in terms of communication protocol and abstraction level. Colif uses a uniform syntax to represent systems that are described at multiple abstraction levels. A virtual port can contain multiple levels of hierarchy to represent an “$N : M$” (N and M are natural numbers) correspondence between internal and external ports. The internal ports are used to connect the internal behavior of the module to the virtual port. The external ports are used to connect the external communication channel to the virtual port. A virtual channel groups nets that are parts of the same communication protocol. Each Colif object has a list of local parameters, for example the kind of protocol used in a virtual channel and addresses of ports.

Colif is used as intermediate language for describing the design model through different phases of the DSP macrocell builder.

4.2. Mat2Colif

The *Mat2Colif* is developed to transform the functional Matlab model into a functional description in Colif language. It consists of a lexical and syntactical analyzer applied upon the Matlab description, the functions treating the different input parameters of the tool, and the functions necessary for producing the correct output file. The tool needs an intermediate variable for integrating the inputs and the outputs. This means that it is not possible to use directly the labels of the inputs and the outputs for calling these functions. After the intermediate form is explored, the tool imports the Colif objects corresponding to functional IPs. After all the objects are correctly imported into the Colif tree structure located in memory, the tool instantiates this structure in order to obtain a suitable file for visualization. This file describes the functional description in Colif language.

4.3. ColifRefiner

The *ColifRefiner* tool transforms the Colif functional architecture model into Colif RTL model. First, the Colif F-IPs are substituted by their corresponding Colif RTL IPs using the IPs libraries. Then, the module of global FSM is added and the ports-nets connections are performed. The connections between IPs are made by name, meaning that ports with same role have same name in both functional and RTL models. The output result of *ColifRefiner* is Colif RTL structure

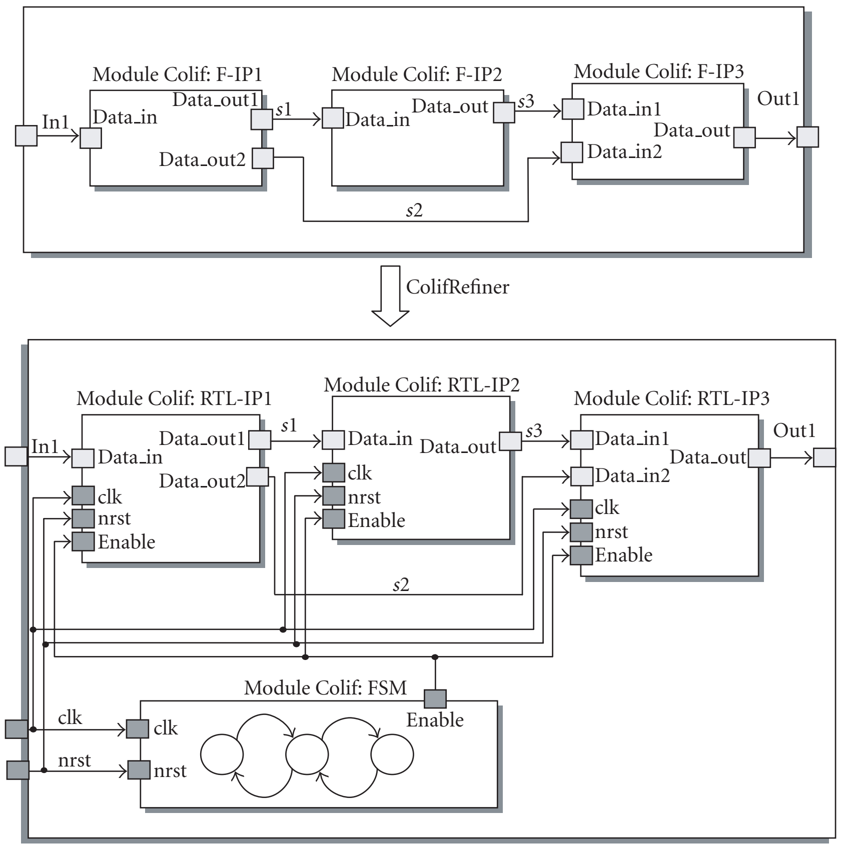


Figure 6: Input and output of ColifRefiner tool.

description of the system (including data path and FSM structure). Figure 6 illustrates an example of the input and output of *ColifRefiner* tool.

4.4. Delay correction flow

Figure 7 shows the flow of the delay correction method [31]. The inputs of this flow are Colif functional and Colif RTL descriptions of the entire system. The output of this flow is a corrected RTL-level description producing the same digital displays as the functional description. The localization and the calculation of the number of delays to be inserted require the use of an intermediate form called *differential graph of evolution*, highlighting the delays present in the RTL model and absent from the functional one. For that, the functional model (resp., the RTL model) is represented by a graph called *functional graph of evolution* (resp., *RTL graph of evolution*) describing its own delays. The *differential graph of evolution* is created by performing one by one the difference between the weight of edges of the functional graph of evolution and those of the RTL graph of evolution. This difference makes possible to see only the additive delays due to the constraints of implementation, by removing all the delays related to the functionality.

Starting from the differential graph, the ADCM determines the corrections necessary to compensate the additive delays. The ADCM uses two algorithms in performing the corrections needed by the differential graph of evolution in order to obtain a balanced graph. The first algorithm (algorithm-1) determines an optimal solution in latency, while the second algorithm (algorithm-2) gives an optimal solution in number of inserted registers, optimizing area. Finally, the step of code generation produces a corrected RTL description of the system, inserting the right number of delays into the right place. The ADCM implements efficiently two alternatives to correct the RTL description of the system: one based on registers insertion while the second is based on FSM modification. According to the implementation constraints and the target application, the designer can choose the suitable techniques to be used. In practice, the *ColifLatencer* tool inserts automatically the latency values into the Colif files (Colif RTL and functional), while the ADCM performs the correction.

4.5. Synthetic example

In order to highlight the problem of behavior difference and its solution in a real case of IP-block-based design, we have willfully selected a synthetic example composed of two parallel branches of block IPs tending towards the same IP (see Figure 8). One contains three FIR filters IPs and the others contain only one FIR filter. A behavior difference between

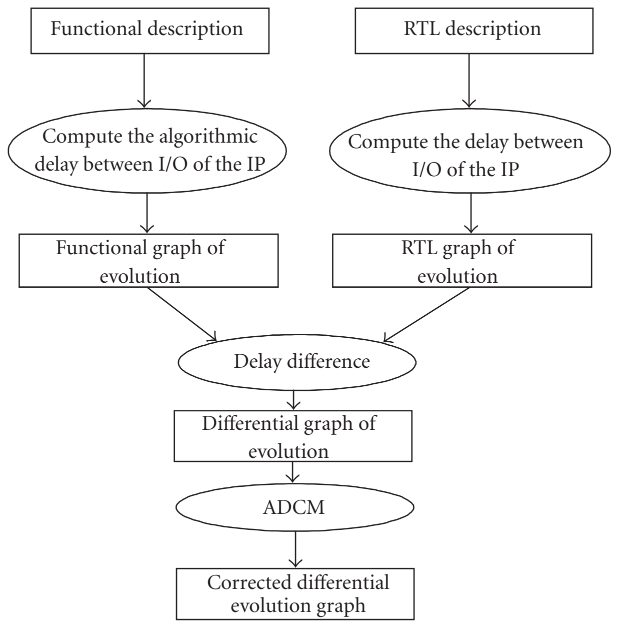

Functional description
RTL description
Compute the algorithmic delay between I/O of the IP
Compute the delay between I/O of the IP
Functional graph of evolution
RTL graph of evolution
Delay difference
Differential graph of evolution
ADCM
Corrected differential evolution graph

FIGURE 7: Delay correction flow.

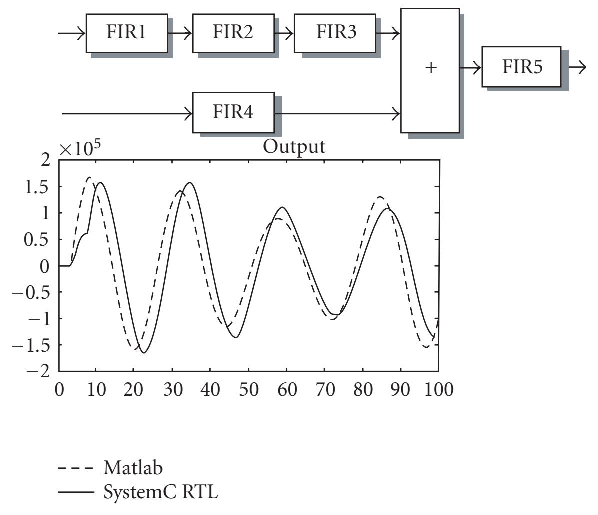


FIGURE 8: Synthetic example and problem of behavior problem between functional and RTL models.

the RTL and functional model has been detected by our ADCM as well as during both functional and cycle accurate simulations (both digital data curves in Figure 6 are different). The problem is due to an output register present in each RTL FIR filter and absent in functional filters. This register induces an additional delay in the RTL model. The differential graph of evolution is shown in Figure 9. The first path has three additional delays, whereas the second path has one additional delay. It was necessary to add two delays in the second path in order to balance the differential graph. This delay correction was translated in two ways on the RTL model. The first one consists of inserting two registers in the second path (Figure 10(a)). The other method involves modifying the initial FSM (Figure 10(b)). The initial FSM generates a control signal at each fourth clock cycle. In the case of the second correction method, the FSM has to produce a supplementary control signal, but delayed initially by two impulses of the first signal (8 clock cycles). Then, the filter in the second branch starts its computation after the same period as the filters of the first path. Both of the two techniques were applied upon this example. Independently of the used correction method, the RTL models produce exactly the same digital values as the functional one (see Figure 11).

4.6. *Discussion*

We assumed the systems are mono-rates, do not include time-varying IPs, and can be built by acyclic assembly of IPs. We assumed the model is a static data flow graph (SDF graph), that is, latency and data throughput of IPs are constant, and this model is not a limitation of our methodology. In practice, when designing data-dependency IPs, the FIFOs with parameterized sizes are placed at the outputs of IPs; it boils down to SDF graph case. In the case of cyclic graph, heuristic algorithms that build acyclic graphs from cyclic ones need to be considered, which are outside the scope of this paper. The most high-throughput DSP systems can be supported by our methodology.

5. EXPERIMENTAL RESULT

We applied our ADCM and associated IP-based design flow to synthetic example (previously presented) and three high-throughput dedicated DSP systems: digital modulation chain circuit extracted from a real design of TV digital transmission satellite application, decoder based on the soft-output Viterbi algorithm (SOVA), MP3 (MPEG-1 audio layer-3) audio compression standard. Functional and RTL models of these three applications were built by assembling the various predesigned and prevalidated IPs. The behaviors were subjected to ADCM; we used two alternatives (registers insertion and FSM modification) of implementing delay correction on these circuits. The logic synthesis was performed using Synopsys Design Compiler [34] and the resultant circuits were mapped to AMS's 0.35μ cell-based array library. The resultant gate-level circuits were compared with respect to the following metrics: *area and performance*. The area and clock period are obtained after performing synthesis and technology mapping. The performance, that is, execution time is the product of clock period and number of clock cycles (RTL simulation).

Table 1 presents the number of registers inserted after the behaviors which were performed by ADCM (i.e., algorithm-1 and algorithm-2). Save for the synthetic example, algorithm-2 improves significantly the solutions found by algorithm-1. The average registers improvement is 50% (the averages were calculated based on comparing the sum of the values in algorithm-1 and algorithm-2 columns).

In Tables 2 and 3, we present, respectively, execution time and area results. The results are obtained according to the following three cases: designs without ADCM (second column),

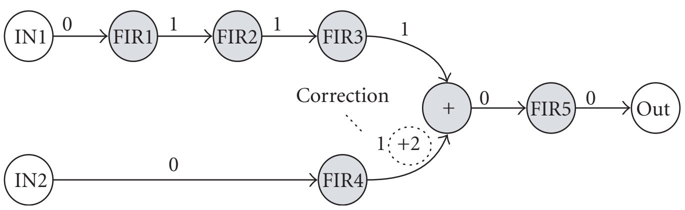
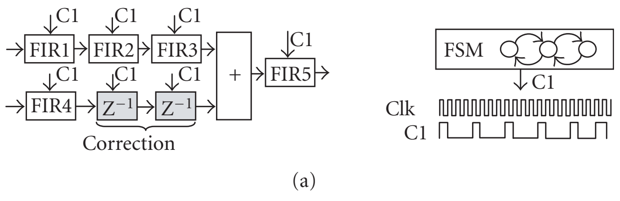


Figure 9: Balanced differential graph of evolution.



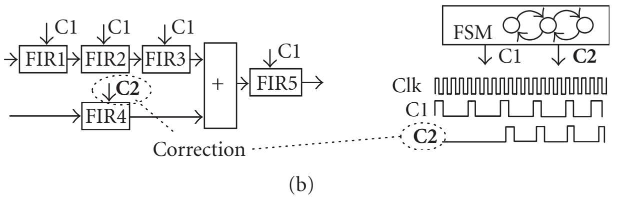


Figure 10: Two ways to correct the behavior in RTL model: (a) delay correction by registers insertion and (b) delay correction by modifying FSM.

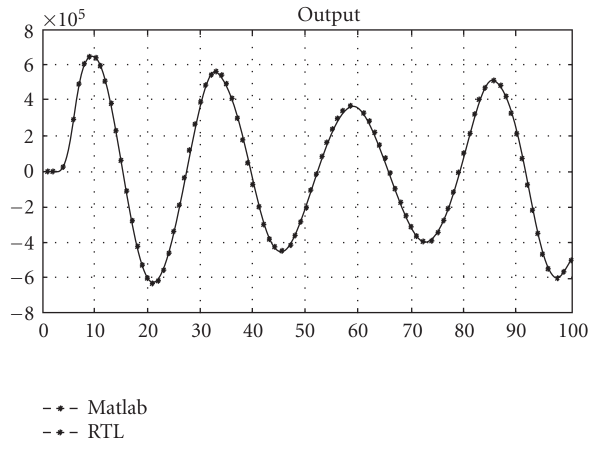


Figure 11: Output signals of functional and corrected RTL models.

Table 1: Registers insertion performed by ADCM algorithms.

System	Registers inserted no.	
	Algorithm-1	Algorithm-2
Synthetic example	2	2
Modulation chain	15	7
SOVA	9	5
MP3 decoder	12	5

designs with ADCM by registers insertion (third column), and ADCM by FSM modification (fourth column). In the first case (second column), the interfacing of IPs blocks was performed by inserting an extra interface circuitry (i.e., handshake protocol), in order to synchronize them.

Tables 2 and 3 indicate that our ADCM results in significant improvements of performance and area, the average performance improvement is 15.67%, whereas the average area improvement is 10.7%. Area and delay overhead for circuitry cannot be neglected in the first case (second column). Regarding both alternatives of ADCM (registers insertion versus FSM modification), the two corrections are equivalent in terms of performances (third and fourth columns in Table 2). With regard to area (third and fourth columns in Table 3), the two corrections give the slightly different results. We note that the difference is the area no more than 4% for the cases studied. This difference is due to the way in which the N delays correction are distributed on the balanced differential graph. The N delays correction on differential graph correspond to M ($1 \leq M \leq N$, M depends on the distribution of delays correction on the differential graph) supplementary control signals in ADCM by FSM modification, whereas they always correspond to N registers in ADCM by registers insertion. The choice of a method is closely related to the application and must be done after applying the two methods, and analyzing the area results.

6. SUMMARY AND CONCLUSIONS

In this paper, we proposed an efficient IP-block-based design environment for high-throughput VLSI Systems. The flow generates SystemC RTL architecture, starting from Matlab-based functional model of digital system. To provide IPs with more reusability and flexibility, we are developing parameterized reusable DSP components at functional and register-transfer level called “generic F-IP” and “generic RTL-IP.” Thus, by setting the appropriate parameters, unnecessary functions and redundant interfaces are eliminated in the IP-based design approach. Although each functional IP and its equivalent RTL produce the same digital displays, in some cases, the register-transfer model obtained by automatic assembly from the functional model can be wrong. We have also proposed an approach called automatic delay correction method to solve this problem without the insertion of an extra interface circuitry. The approach corrects the behavior of

Table 2: Performance results: registers versus FSM modification.

System	Execution time (ns)		
	Without ADCM	ADCM by registers insertion	ADCM by FSM modification
Modulator	4567870	3859850	3859854
MP3 decoder	2297500	1929902	1929907
SOVA*	68266	58026	58027

*Time required to decode a single 1024-bit block of information using 4-stage iterative decoding.

Table 3: Area results: registers insertion versus FSM modification.

System	Area (# kgates)		
	Without ADCM	ADCM by registers insertion	ADCM by FSM modification
Modulator	541.7	482.5	483.2
MP3 decoder	149.4	134.1	134.9
SOVA	52.5	47.5	47.1

the RTL model in a judicious way that includes locating the places where the problems occur, determining how many delays are needed, and implementing the correction. We have described two alternatives (registers insertion and FSM modification) of implementing delay correction methods and we have presented a realistic example where the delay correction method has been efficiently applied. Experimental results in real cases, also, demonstrate significant improvements in the quality of the synthesized implementations.

REFERENCES

[1] International Technology Roadmap for Semiconductors, 2003 Edition Report, http://public.itrs.net.

[2] A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, and M. Sgroi, "Benefits and challenges for platform-based design," in *Proceedings of 41st IEEE Design Automation Conference (DAC '04)*, pp. 409–414, San Diego, Claif, USA, June 2004.

[3] G. Martin, "Design methodologies for system level IP," in *Proceedings of IEEE Design, Automation and Test in Europe (DATE '98)*, pp. 286–289, Paris, France, February 1998.

[4] D. D. Gajski, A. C.-H. Wu, V. Chaiyakul, S. Mori, T. Nukiyama, and P. Bricaud, "Essential issues for IP reuse," in *Proceedings of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC '00)*, pp. 37–42, Yokohama, Japan, January 2000.

[5] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons, New York, NY, USA, 1998.

[6] Celoxica, *Handel-C Language Reference Manual*, 2003. RM-1003-4.0, http://www.celoxica.com.

[7] G. De Micheli, "Hardware synthesis from C/C++ models," in *Proceedings of IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE '99)*, pp. 382–383, Munich, Germany, March 1999.

[8] S. A. Edwards, "The challenges of hardware synthesis from C-like languages," in *Proceedings of IEEE Design, Automation and Test in Europe (DATE '05)*, vol. 1, pp. 66–67, Munich, Germany, March 2005.

[9] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauser, and S. Zhoa, *Spec C: Specification Language and Methodology*, Kluwer Academic, Boston, Mass, USA, 2000.

[10] D. C. Ku and G. De Micheli, "HardwareC: A language for hardware design," Tech. Rep. CSTL-TR-90-419, Computer Systems Laboratory, Stanford University, Stanford, Calif, USA, August 1990.

[11] SystemC Community, http://www.systemc.org.

[12] Xilinx System Generator v2.1 for Simulink Reference Guide, Xilinx, 2000.

[13] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: a framework for simulating and prototyping heterogeneous systems," *International Journal of Computer Simulation*, vol. 4, no. 2, pp. 155–182, 1994.

[14] F. Balarin, M. Chiodo, P. Di Giusto, et al., *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*, Kluwer Academic, Boston, Mass, USA, 1997.

[15] R. Lauwereins, M. Engels, M. Ade, and J. A. Peperstraete, "Grape-II: a system-level prototyping environment for DSP applications," *IEEE Computer*, vol. 28, no. 2, pp. 35–43, 1995.

[16] S. Natarajan, B. Levine, C. Tan, D. Newport, and D. Bouldin, "Automatic mapping of khoros-based applications to adaptive computing systems," in *Proceedings of Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD '99)*, pp. 101–107, Laurel, Md, USA, Septemper 1999.

[17] G. Spivey, S. S. Bhattacharyya, and K. Nakajima, "Logic foundry: rapid prototyping for FPGA-based DSP systems," *EURASIP Journal on Applied Signal Processing*, vol. 2003, no. 6, pp. 565–579, 2003.

[18] P. Banerjee, N. Shenoy, A. Choudhary, et al., "MATCH: A MATLAB Compiler for Configurable Computing Systems," Tech. Rep. CPDCTR-9908-013, Center for Parallel and Distributed Computing, Northwestern University, Evanston, Ill, USA, August 1999.

[19] W. R. Davis, N. Zhang, K. Camera, et al., "A design environment for high-throughput low-power dedicated signal processing systems," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 420–431, 2002.

[20] R. K. Gupta and Y. Zorian, "Introducing core-based system design," *IEEE Design and Test of Computers*, vol. 14, no. 4, pp. 15–25, 1997.

[21] L. Lavagno, S. Dey, and R. Gupta, "Specification, modeling and design tools for system-on-chip," in *Proceedings of 7th IEEE Asia and South Pacific Design Automation Conference and*

15th International Conference on VLSI Design (ASP-DAC '02), pp. 21–23, Bangalore, India, January 2002.

[22] W. Cescirio, A. Baghdadi, L. Gauthier, et al., "Component-based design approach for multicore SoCs," in *Proceedings of 39th IEEE Design Automation Conference (DAC '02)*, pp. 789–794, New Orleans, La, USA, June 2002.

[23] B.-W. Kim and C.-M. Kyung, "Exploiting intellectual properties with imprecise design costs for system-on-chip synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 240–252, 2002.

[24] M. Vachharajani, N. Vachharajani, S. Malik, and D. I. August, "Facilitating reuse in hardware models with enhanced type inference," in *Proceedings of IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '04)*, pp. 86–91, Stockholm, Sweden, September 2004.

[25] R. Passerone, J. A. Rowson, and A. Sangiovanni-Vincentelli, "Automatic synthesis of interfaces between incompatible protocols," in *Proceedings of 35th IEEE Design Automation Conference (DAC '98)*, pp. 8–13, San Francico, Calif, USA, June 1998.

[26] H. Choi, J. H. Yi, J.-Y. Lee, I.-C. Park, and C.-M. Kyung, "Exploiting intellectual properties in ASIP designs for embedded DSP software," in *Proceedings of 36th IEEE Design Automation Conference (DAC '99)*, pp. 939–944, New Orleans, La, USA, June 1999.

[27] VSI Alliance, http://www.vsi.org.

[28] L. Tambour, "Efficient methodology for design and validation of complex DSP system-on-chip," Ph.D. thesis, Institut National Polytechnique de Grenoble (INPG), Grenoble, France, December 2003, http://tima.imag.fr/publications/files/th/mfs_196.pdf.

[29] N. E. Zergainoh, K. Popovici, A. A. Jerraya, and P. Urard, "Matlab based environment for designing DSP systems using IP blocks," in *Proceedings of 12th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI '04)*, pp. 296–302, Kanazawa, Japan, October 2004.

[30] The MathWorks Incorporation, http://www.mathworks.com.

[31] N. Zergainoh, L. Tambour, H. Michel, and A. A. Jerraya, "Méthode de correction automatique de retard dans les modèles RTL des systèmes monopuces DSP obtenus par assemblage de composants IP," *Techniques et Sciences Informatiques*, vol. 24, no. 10, pp. 1227–1257, 2005.

[32] A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, Cambridge, UK, 1985.

[33] W. O. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, and A. A. Jerraya, "Colif: A design representation for application-specific multiprocessor SOCs," *IEEE Design and Test of Computers*, vol. 18, no. 5, pp. 8–20, 2001.

[34] Synopsys Incorporation, http://www.synopsys.com.

Nacer-Eddine Zergainoh received the State Engineering degree in electrical engineering from National Telecommunication School and the M.S. and Ph.D. degrees in computer engineering from University of Paris XI, in 1992 and 1996, respectively. Currently, he is an Associate Professor at Ecole Polytechnique of University of Joseph Fourier, Grenoble, and member of the research staff of the Techniques of Informatics and Microelectronics for Computer Architecture Laboratory, Grenoble. Prior to that, he was an R&D Engineer at ILEX-Computer Systems, Paris, France. His current research interests are hardware/software codesign, high-level synthesis and CAD issues for real-time digital signal processing, design and exploration of application-specific multiprocessor SoC (including design and analysis of on-chip communication architectures, network on-chip issues). He also maintains an active interest in parallel processing, multiprocessor architectures, and real-time operating systems. Professor Zergainoh has served on the technical program committees for several international conferences and workshops.

Ludovic Tambour received the Engineer degree in computer science from the Ecole Polytechnique de Grenoble in 2000 and the M.S. and Ph.D. degrees in computer science from the Institut National Polytechnique de Grenoble (INPG), Grenoble, France, in 2000 and 2003, respectively. In 2000, he joined the R&D SHIVA Group at ST Microelectronics and SLS Group at TIMA Laboratory where he worked on methodology and flow for design and validation of digital signal processing ASIC macrocells. In 2004, Dr. Tambour moved to hold an Engineer position at CIRAD (International Cooperating Center in Research for Agronomic Developing), Montpellier, France. His research interests include software tools for modeling, simulation and data analysis in a large field of activities including microelectronic, signal processing, agronomy, and so forth.

Pascal Urard joined ST Microelectronics in 1992 where he has worked successively in test, engineering, ASIC design, and architecture of mixed signal processing ASICs. In 2000, he joined ST R&D to work on ESLD flows. He initiated a Matlab-2-RTL flow that is now used internally in ST. In 2001, he initiated cooperations with HLS tools companies. He is now the Manager of High-Level Synthesis Group within ST Microelectronics Central—CAD.

Ahmed Amine Jerraya received the Engineer degree from the University of Tunis in 1980 and the DEA, "Docteur Ingénieur," and the "Docteur d'Etat" degrees from the University of Grenoble in 1981, 1983, and 1989, respectively, all in computer sciences. In 1986, he held a Full Research position with the CNRS (Centre National de la Recherche Scientifique). From April 1990 to March 1991, he was a member of the scientific staff at Nortel in Canada, working on linking system design tools and hardware design environments. He is the General Chair of HLDVT '02 and Coprogram Chair of CASES '02. He served as the General Chair for DATE 2001, ISSS '96, and General Cochair for CODES '99. He also served as Program Chair for ISSS '95, RSP '96, and Coprogram Chair of CODES '97. He published more than 100 papers in international conferences and journals. He received the Best Paper Award at the 1994 ED&TC for his work on hardware/software cosimulation. Dr. Jerraya is currently managing the System-Level Synthesis Group of TIMA Laboratory and has the grade of Research Director within the CNRS.