Behavioral IP Specification and Integration Framework for High-Level Design Reuse

Sebastien Pillement[1], Daniel Chillet[1]
[1]LASTI-ENSSAT-University of Rennes
6, rue de kerampont
22300 Lannion, France
name@enssat.fr, http://archi.enssat.fr

Olivier Sentieys[1,2]
[2]IRISA-INRIA
Campus de beaulieu
35042 Rennes, France

Abstract

Specifying virtual components at the behavioral level appears as the most promising solution to achieve a real efficiency of design reuse. In this paper we propose a methodology to specify and use Behavioral Level IP *(BL-IP). Thus,* IP *designer tasks are easier due to the unified representation offered by this level of abstraction. The genericity of a behavioral* IP *permits efficient optimizations and make application context adaptations a reality. We propose a unified framework to define an IP at the behavioral level and to tune a particular block according to designer needs. Therefore, we define the* IP *generator tool and the Universal High Level Synthesis concept.*

1 Introduction

The electronic industry is moving toward the design and implementation of entire systems on a single chip (SoC). Such types of integrated circuits are actually built around a processor core [3], that is available in the libraries of semiconductor companies. Additional functions are added to this core in order to realize an application or domain specific processor. Three types of reusable core (IP Core) can be distinguished [19]: the *Soft Core* is described using a high level description language (i.e. VHDL or Verilog), the *Firm Core* is described and synthesized for specific library and finally the *Hard Core* is described at the layout level. Because of the growing complexity of SoC designs [2], design reuse methodologies are known as the better way to bridge the gap between performance and time-to-market [15, 13] with nowadays methodologies and tools. For IP integrators, the challenges consist in understanding all the specifications of existing blocks responding to his needs, and in testing and interface development [9, 10]. Unfortunately, IP reuse lacks design methodologies and tools facing these problems. At present, all blocks are designed by vendors for one technology, and without real normalization, expecting research, validation and integration effort [1, 6].

In this context, we propose an innovative approach, consisting in defining the IP at a higher level of abstraction (i.e. Behavioral Level). This allows the designer to specify the behavior of a function, and an IP generator, based on provider know-how, generates all the descriptions of the block and the synthesis script needed to achieve user performance requests. Finally, it includes behavioral synthesis tools in the design flow of the IP integration. Internet, favored by its worldwide access and its great flexibility for both provider and integrator appears as an inescapable technology for design-reuse methodology. Related works on Web-based Framework are focusing on IP selection [12, 22], on the test and the simulation [8, 21, 5] of IP blocks. At present, methodologies for design-reuse are specific for IP provider [16, 11] or for IP integrator [4, 17, 7]. We propose a unified framework to i) define an IP at the behavioral (i.e. algorithmic) level ii) tune a particular block according to designer needs. This level of abstraction permits a great genericity of blocks and thus offers a large spectrum of implementations, and finally allows high optimizations [20].

In the next section, we define the Behavioral-Level IP (BL-IP) characteristics and objectives and introduces the BL-IP parameters and rule processing. Section 3 describes the generator concepts and the framework infrastructure. The last section describes an exemple of a BL-IP for digital filtering.

2 Behavioral Level Ip Characteristics

2.1 BL-Ip Objectives

We can define a set of objective criteria for a Behavioral-Level IP (BL-IP) that guarantees the performances of the block:

- Design methodology must permit behavioral specification entry point. The most popular behavioral languages are VHDL or SystemC, but others can be used.
- Blocks models must be uniform. This enables the extraction of generic parameters and rules to define and compare different implementations of functions.
- Methodology should be applied to all kind of blocks.

Proceedings of the International Symposium on Quality Electronic Design (ISQED'02)
0-7695-1561-4/02 $17.00 © 2002 IEEE

- Design and reuse methodology must be cheaper than *classic methodology*. Choices and refining block cycle have to be optimized.
- BL-IP overhead development cost must be reasonable.
- BL-IP use have to lead to the performance needs of the integrator.
- Flexibility is an important characteristic for BL-IP. The environmental adaptation of BL-IP is done by setting a set of parameters.
- BL-IP performance have to be tool independent. This criterion introduces the concept of Universal High-Level-Synthesis (U-HLS) tool.

To reach these goals, we propose to use an interface called a *generator* and to specify the IP at the behavioral level associated to the use of a methodology to simplify the IP integrator task.

2.2 BL-Ip Advantages

Most of reusable cores are hard cores that do not achieve real portability. Differences on the reusable core come from the level of abstraction for each style. Hard cores defined at the layout level have the greatest performance but are not portable. On the other hand, soft cores, usually described at the RT Level are very flexible but can not easily guarantee the Intellectual Property for the creators of blocks and needs high integration effort for designers [11].

Specifying the IP at the behavioral level is mainly characterized by the potential of hardware resource sharing by different IPs, by a free cycle scheduling of operations that can enable a single IP to respect different throughput constraints. Finally, high abstraction is independent of technology and performance constraints [20]. It therefore appears as the most promising solution to achieve a real efficiency of design reuse. In [15], an extension of Synopsys Design Ware concept is developed, and Behavioral Compiler is used to schedule the algorithm for respecting the constraints. In [18], the authors focus on formalizing behavioral reusable generic descriptions, and their communication interfaces to reduce buffer overhead.

From the designer point of view, the IP block is seen as a customizable function (a behavioral function) where he can parameterize some features. This point of view is based on a good description of the IP provider know-how model in the BL-IP generator, which provides to the tool (not to the designer) all informations about synthesis and optimization directive for a particular degree of performance.

2.3 Definition of Ip Parameters

IP parameters have three different levels of abstraction:

- **Algorithm selection parameters** are used to select and customize a specific algorithm. For example, Signal to Noise Ratio (SNR) in dB associated to the input/output data width can be used in order to define the datapath bit-width.
- **Integrator parameters and constraints**. For example, the throughput constraint influences the pipeline depth of the architecture, but can also influence the loop-unfolding factor or limit the size of the application (e.g. number of samples in an FFT).
- **Synthesis tool parameters**. These parameters are used as synthesis constraints. For example, if there is no time constraint then the area optimization are taken into account.

Parameters can interact with another from a different level, and can be defined by enumerations or bounds.

2.4 Ip Instance Definition

We define an IP instanciation as a characterized block that a designer can implement in his application. This instance is composed of a function described at a behavioral level associated with two interface blocks, assuming communications between the function and its environment. In all cases an IP block comes with different description files [19]:

- The synthesis script depends on the CAD tools which will be used to synthesize the design. It gives the best option set to obtain an optimal design.
- The description file contains the description of the functionality to synthesize. Generally described in VHDL at the Register Transfer Level (RTL), some recent works [20, 18, 15] start to use a behavioral level specification (VHDL or C/C++).
- The documentation part resumes all the informations about the block. This documentation must contain the function description, a set of performances and characteristics, and the interface description.
- A set of testbenches must permit evaluation of different IP block characteristics. Evaluations should be performed at different stages of integration.

3 Ip Generator Concept

The BL-IP concept achieves a real reuse of one function. Due to the high-level description of IP blocks, adaptation of the function to the designer needs and to the environment becomes easier. There are two ways to make these adaptations:

- The creator adapts the block to the designer needs. It is the current solution taken for RTL IPs. Creators are more an integration expert of a block for a particular technology. The IPDesigner tool has been developed to simplify this task.
- On the other hand, adaptations can be automatic (or semi-automatic). In this case synthesis is driven by the designer. This solution permits a better adaptation to the application context, and accelerates the refining cycle. This can be achieved, if needs and constraint specifications can be used by a CAD tool adaptable to every environment and to every technology. We call this tool IPCompiler, it is based on the U-HLS concept.

3.1 Universal High Level Synthesis Concept (U-HLS)

The Universal High-Level Synthesis is a *virtual* tool. It is a super-set of specifications that we can adapt to different behavioral synthesis tools.

3.2 Ip Generator General Architecture

A generator is specific to a BL-IP function block, from which we can extract different sub-modules (cf. figure 1).

- The interface enables the specification of constraints and parameters for a particular block, that will help the integrator in the definition of the IP.
- The estimator guides the designer to a good implementation of the function according to the required performances.
- The U-HLS generator takes the parameters defined through the interface and generates an U-HLS IP.
- Derivators transform an U-HLS IP to a tool specific description of the BL-IP.

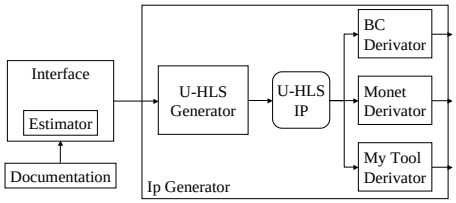

Figure 1. An IP instance generation.

To help the IP Provider and the IP Integrator, we have developed a framework which helps each of them to specify the IP block and to implement an instance of an IP block in the application context. This framework is based on classical client/server functionalities. The tools are described as java applet and can be loaded by any browser. The applet exchange informations with IP Database located on the server through TCP/IP socket.

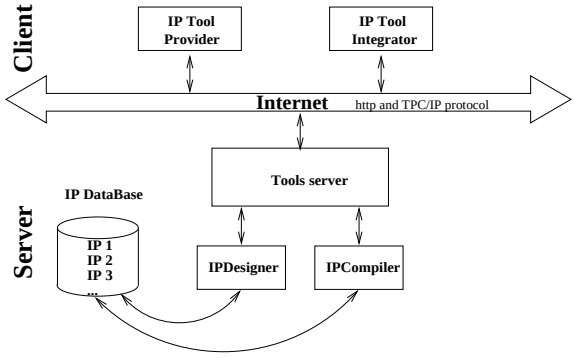

Figure 2. Framework for IP specification and use

Our approach is independent of the considered application. A high adaptation of the description and of the synthesis process depending on the user constraint is proposed. Some parameters and the way they interact are defined.

3.3 U-HLS Ip

Ideally, an optimal IP can be obtained from our U-HLS IP for all CAD tools in an automatic manner. Then, to achieve this goal the methodology should encompass:

- Different algorithms for a same function. These algorithms admit a high level of genericity, and offers a wide solution panel.
- A set of parameters with all the variation domains.
- A set of rules that will adapt a U-HLS description to a specific parameter set definition.
- A tool specific set of rules.

3.4 VHDL descriptions and script synthesis

The application context (or constraints) can impose to write several descriptions for one BL-IP, with some descriptions dedicated to a particular High-Level Synthesis (HLS) tool. Consequently, one specific derivator must be written for each HLS tool. In figure 3 we present a high-level description of a Finite Impulse Response filter BL-IP (FIR-BL-IP) specified for Behavioral Compiler from Synopsys (BC). The description is generic. According to section 4.2, size of filter is defined by the parameter N and input/ouput data is defined by Bin parameters.

```
entity  direct_fir is
  port(xn:in Integer range -2**($Bin$-1) to 2**($Bin$-1);
       yn:out ...
       clk, rst:in Std_Logic );
end direct_fir; architecture behavioral of direct_fir is
    ...
    main : Process
      Attribute dont_unroll of fir_:label is $LoopDU$;
      Variable X:vectorH;
      Variable acc:Integer range ...
      Variable mult:Integer range -2**($Bin+$Bcoeff-1)
      ...                        to 2**($Bin+$Bcoeff-1);
    main:loop
      tmp:=0;
      sig_evol:for i in 0 to x'length-2 loop
         x(i):=x(i+1);
      end loop sig_evol ;
      x(x'length-1):=xn;
      fir_:for i in 0 to h'length-1 loop
          mult:=x(i)*h(i);
          acc:=acc+mult;
      end loop fir_loop;
      yn<=acc;
      ...
```

Figure 3. Generic VHDL specification of a FIR BL-IP for BC

To optimize this description, a specific synthesis script must be written. Figure 4 shows a generic BC synthesis script. For example the effort for the scheduling or the loop unrolling factor are defined by parameters which will be fixed according to the application context.

3.5 The interface

To be efficient the interface must guide designers to converge quickly to the best solution according to specific constraints. For this reason, the designer specifies through the

```
design=$IPName$; source=design + ".vhd";
clk_periode = $Clk$;
sch_effort = "$ScheduleEffort$";
mode = "$ScheduleMode$";
schedule_option="$ScheduleOption$";
bc_chaining = true;
...
analyze -f vhdl source;
elaborate -schedule design;
...
set_cycles $Throughput$ -from_beginning design +"/main"
                        -to_end design +"/main";
schedule -effort sch_effort -extend_latency -io_mode mode;
write -hierarchy -output design+"_sch.db"; ... ...
```

Figure 4. Generic synthesis script for BC

interface the function characteristics, the expected performances, and some specific tool parameters.

Unfortunately, definition of a particular constraint reduces the validity domain of other constraints. We need then an actuator module to update the new accepted value for parameters that will guide the designer to a realistic solution. This actuator module will be based on the rule library. Using Internet technology maintains the up-to-date version of a BL-IP generator, by integrating a script in the interface (i.e. a modification of the generator is immediately accessible for all customers). After the parameter definition (made through a Java Applet) and the verification of constraints, the generator creates all the needed files for the function integration. Then, customers can download them by FTP, for example.

3.6 U-HLS Generator

U-HLS description generator (cf. figure 1) takes parameter values from the interface and realizes different tasks:

- **Parameter processing** determines in which module the current parameters is needed. This phase adapts parameters to a specific module of the design flow. A specific parameter can be used by different modules, such as the throughput parameter used in the algorithm selection module and synthesis constraint module.
- **Algorithm selection** of a specific algorithm is made according to a rule library and to current value of parameters according to the designer specifications. Then, rule resolution consists in tuning the algorithm characteristics using these parameters and a function library. The latter is a set of implementations of different algorithms proposed by the creator of the generator. The editor has to search in the library the selected algorithm and specify it with the integrator parameters. After selection, the U-HLS description of the algorithm is passed to the rule processing module, for refining the description.
- **Synthesis rule solver** enables the derivation of parameters into synthesis constraints according to the rule library. Unresolved rules or *tool-dependent* parameters are passed to specific derivator.

The rule library is the most important module of the generator. A bad parameter definition drives to a bad implementation solution. This module represents the function and performance models of a specific BL-IP. Each rule is a Boolean expression that determines parameter influence and may depend on the synthesis CAD tools used. Nevertheless, different types of rule exist and each describes the links between the parameters. Algorithm selection rules have to be treated first. Derivator rules are tool dependent and may contain rules like loop unrolling strategies, synthesizer optimization effort, etc.

3.7 Derivators

Derivators (cf. figure 1) are used to transform an universal representation of a function into a tool-specific one. There is as much derivator as tools considered by the BL-IP generator. These module are automated, and are IP independent. The different tasks assumed by the derivator are :

- The transformation of the U-HLS IP into a behavioral description for the specific synthesis tool. At this point, the description become synthesis dependent.
- The specialization of the generic script according to end-user constraints and precedent calculation.

IP Provider must specify the dependencies between parameters and how each parameter should be recomputed (function). This part is very important and must be done very precisely. The ability to correctly implement BL-IP is dependent on this specification. All the parameter dependencies are coded in a database, as shown in figure 5, in order to be treated by the IP Compiler.

```
# Begin functions
        Bacc = ComputeBacc
        Qacc = ComputeQacc
        Qin = ComputeQin ...
# Begin dependencies
        Bin > Bacc Qin
        N > Bacc Tools
        Bacc > Qacc ...
# End
```

Figure 5. Parameter derivation rules

Linked to these dependency definition, the IP Provider must write the mathematical expression of each function. These functions are grouped together in a Java Object which is dynamically loaded when a particular BL-IP is chosen. In the figure 6 we can see the method to compute the parameter $Qacc$ through the $ComputeQacc$ function.

```
public int ComputeQacc(Integer p1) {
        int bacc = p1.intValue();

        int result = bacc-1;
        return result;
}
```

Figure 6. Java method to compute Qacc

4 Example of an IP for Digital Filtering

4.1 Introduction on Digital Filters

A digital filter is usually specified using frequency and amplitude tolerance schemes. In the frequency domain, the different passbands and stopbands must be specified. The

other desired properties of a filter are the maximum passband and stopband gains. The width of the transition bands gives the selectivity of the filter, that, configures with the selected gains, will have an influence on the number of coefficients. A number of Signal Processing tools can extract from these specifications, and for different digital filter structures, the number of coefficients and their floating or fixed point values. They will be the main input to the Behavioral IP, associated to a desired structure.
A filter can have either a recursive (IIR) or non recursive (FIR) structure. The equations 1 and 2 give the linear differential equations for IIR and FIR filters, from which we can derive the behavioral specification code for the IP.

$$y(n) = \sum_{i=0}^{N} b_i.x(n-i) - \sum_{i=0}^{N} a_i.y(n-i) \quad (1)$$

$$y(n) = \sum_{i=0}^{N-1} b_i.x(n-i) \quad (2)$$

4.2 Digital Filter IP Parameters

The algorithmic parameters have an influence on the behavioral code. In this case, the filter type (RII or RIF) and the filter structures have to be defined in this category. The classical structures [14] are direct (*I and II*), cascade, parallel, transposed and lattice forms.
The functional parameters will have an influence on the generic behavioral code given by the algorithm selection. In the case of digital filters, the first parameters are the number of coefficients (N), the value of the coefficients ($ListOfCoeff$) and the bit width of the inputs and outputs ($Bin, Bout$). Thus, assuming that the IP will use fixed-point representation of numbers, another important issue in filter design is the quantization noise that is a result of the coefficient and input/output quantization, and of the processing unit precision. The last-mentioned have an influence on the round-off noise and on the scaling scheme. In order to resolve this problem we have introduced some functional parameters to modelize the quantization: coding format and scaling of coefficients ($Qcoeff$) and input/output ($Qin, Qout$), size of accumulator ($Bacc$) and Signal to Quantization Noise Ratio ($SQNR$).
Finally, the constraints of the IP can be specified. In our case the user can define a throughput and a latency constraints, but also a $SQNR$ constraint.

4.3 Interaction between IP Parameters

This part of a specific IP design is very important and will guarantees its performances. We will detail in this section the influence of throughput and SQNR for a FIR direct form filter. We assume that the input data are in the interval $[-1 \ldots 1]$ using a fixed-point representation. The generalization for all the structures has been treated and implemented in the tool.

Signal to Quantization Noise Ratio $SQNR$ will depend on the value of the coefficients and on the parameters $N, Bin, Bout, Qin, Qout, Qcoeff, Bacc$. Internal variables have to be computed: the dynamic range, and the number of guarded bits BG that will guarantee that the filter will not overflow during computation. Then, the size of accumulator $Bacc$ must respect the following rule:

$$\begin{aligned} Dynamic\,range &= \sum_{i=0}^{N-1} |b_i| \\ BG &= log_2\lceil Dynamic\,range \rceil - 1 \\ Bacc &\leq BG + Bcoeff + Bin \end{aligned}$$

Then, the $SQNR$ is given by:

$$SQNR = \frac{\sigma_x^2 \sum_{i=0}^{N-1} b_i^2}{\sigma_e^2 \sum_{i=0}^{N-1} b_i^2 + \frac{q_{out}^2}{12}(1 - 2^{-2(Bacc-Bout)})}$$

At this point the user can specify a $SQNR$ constraint and IP compiler will derive the suited size for accumulation.

The throughput of the filter, i.e. the sample period, will first have an influence on the time constraint given to the synthesis script of the high-level synthesis tool. Secondly, a real constraint will be on the unfolding factor of the filtering loop. In tools like Monet or BC, this parameter must be specified in an optimal manner to obtain efficient results. In the case of the FIR filter with a multiplier time of T_{mult} this criteria must follow:

$$Unfolding\ factor \geq \lceil \frac{N.T_{mult}}{Throughput} \rceil$$

4.4 IP Compiler for digital filters

We consider an application context which guide the IP Integrator to use a RIF BL-IP block. We can suppose that the IP integrator has some knowledge of hardware constraints (for example he knows that the input data of RIF IP are coded with 8 bits).

With these knowledges, the IP integrator can use the IP Compiler tool and can gradually design and tune his IP. The figure 7 present the IP Compiler tool running the RIF BL-IP generator. When a BL-IP is chosen, the list of parameters is printed. This list is sorted in 4 types which are: the tools parameters, algorithmic parameters, functionality parameters and constraint parameters. The latter allows the IP Integrator to specify the application context.

IP Compiler provides a solver which supports the execution of rules defined by IP Provider. This solver is manually activated by IP Integrator and gives consistency informations (the modification of one parameter have an implication on one another, or the fixed value of one parameter is in contradiction with the values of other parameters). Figure 7 shows the generation of the specification and synthesis script for a direct structure of a FIR filter activated with specific parameters.

When the BL-IP is completely customized, the IP Integrator can load the best set of BL-IP file description, that top say he obtain the best VHDL description for the constraint he have specified and the corresponding synthesis script to optimize his block (see files on figure 7).

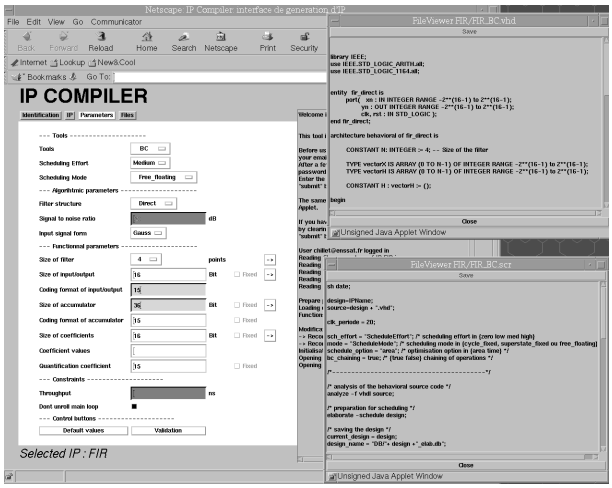

Figure 7. Parameter specification and generated VHDL file

5 Conclusion

We have presented in this paper a new IP design methodology. By the use of high level description language, we expect real reuse possibility for a particular function. The portability of a BL-IP is assumed by the decomposition of the design flow in two parts. The first one is algorithm independent and is common to all IP generators. The second one depends on the function to be implemented, and represents the IP developer experience. This methodology is technology independent, and permit a rapid cycle of development.

We have defined BL-IP architectures for Digital filters (RII and RIF), Fast Fourier Transform and Discrete Cosine Transform. They are usable through a free access by Internet. The method has also been used for the realization of a motion estimation IP for MPEG4 video compression. In this last case, the adaptation of the specification to the constraints is really important for the efficiency of the block. Many motion estimation algorithms are existing, and represent a trade-off between complexity and quality. Algorithm selection depending on user's constraints will represent a real issue in the quality of the resulting IP.

References

[1] J.-F. Agaesse and B. Laurent. Virtual components application and customization. In *DATE Conference*, pages 726–727, Munich, Germany, Mar. 9–12 1999.

[2] S. I. Association. The national technology roadmap for semiconductors. Technical report, http://notes.sematech.org, 1999.

[3] R. A. Bergamaschi and W. R. Lee. Designing systems-on-chip using cores. In *37thDesign Automation Conference*, pages 420–425, Los Angeles, June 2000.

[4] H. Choi, J. Yi, J.-Y. Lee, I.-C. Park, and C.-M. Kyung. Exploiting intellectual properties in ASIP designs for embedded DSP software. In *36th Design Automation Conference*, pages 939–944, New Orleans, USA, June 21–25 1999.

[5] M. Dalpasso, A. Bogliolo, and L. Benini. Virtual simulation of distributed ip-based designs. In *36th Design Automation Conference*, pages 50–55, New Orleans, USA, June 21–25 1999.

[6] Design and Reuse. http://www.design-reuse.com.

[7] E. Filippi, L. Lavagno, L. Licciardi, A. Montanaro, M. Paolini, R. Passerone, M. Sgroi, and A. Sangiovanni-Vincentelli. Intellectual property re-use in embedded system co-design: an industrial case study. In *Int. Symp. on Systems Synthesis*, pages 37–42, Taiwan, China, Dec. 2–4 1998.

[8] A. Fin and F. Fummi. A web-cad methodology for ip-core analysis and simulation. In *37thDesign Automation Conference*, Los Angeles, June 2000.

[9] D. Gajski, A. C.-H. Wu, V. Chaiyakul, S. Mori, T. Nukiyama, and P. Bricaud. Essential issues for ip reuse. In *Proceeding of Asian and South-Pacific Design Automation Conference*, pages 37–42, Yokohama, Japan, Jan. 25–28 2000.

[10] R. Glover, T. Inoue, and J. Teets. Panel: Challenges in worlwide ip reuse. In *36th Design Automation Conference*, pages 401–402, New Orleans, USA, June 21–25 1999.

[11] J. Haase. Design methodology for ip providers. In *DATE Conference*, pages 728–732, Munich, Germany, Mar. 9–12 1999.

[12] J. Kim, K. Kwon, Y. Lee, and C. Lee. Ip database and catalog system. In *International Workshop on IP Based Synthesis and System Design*, pages 15–22, Grenoble, France, Dec. 14-15 1999.

[13] I. Moussa, Z. Sugar, R. Suescun, M. Diaz-Nava, M. Pavesi, S. Crudo, L. Gazi, and A. Jerraya. Comparing rtl and behavioral design methodologies in the case of a 2m-transistor atm shaper. In *36th Design Automation Conference*, pages 598–603, New Orleans, USA, June 21–25 1999.

[14] A. V. Oppenheim and R. W. Schafer. *Discrete-time Signal Processing*. Prentice Hall, 2nd edition, 1999.

[15] A. Reutter, B. Mossner, and W. Rosenstiel. Design of reusable modules for high level designs. In *International Workshop on IP Based Synthesis and System Design*, pages 45–48, Grenoble, France, Dec. 15-16 1998.

[16] W. Savage, J. Chilton, and R. Camposano. Ip reuse in the system on a chip era. In *Int. Symp. on Systems Synthesis*, Madrid, Spain, Sept. 20–22 2000.

[17] F. Vahid and T. Givargis. Incorporating cores into system-level specification. In *Int. Symp. on Systems Synthesis*, pages 43–48, Taiwan, China, Dec. 2–4 1998.

[18] F. Vermeulen, F. Cathoor, D. Verkest, and H. D. Man. Formalized three-layer system-level reuse model and methodology for embedded data-dominated applications. In *DATE Conference*, pages 92–98, Paris, France, Mar. 27–30 2000.

[19] VSIA. http://www.vsi.org.

[20] K. Wakabayashi and T. Okamoto. C-based soc design flow and eda tools: An asic and system vendor perspective. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1507–1522, Dec. 2000.

[21] P. Wilsey. Web-based analysis and distributed ip. In *Proceeding of the 1999 Winter Simulation Conference*, pages 1445–1453, Dec. 5–8 1999.

[22] T. Zhang, L. Benini, and G. D. Michelli. Component selection and matching for ip-based design. In *DATE Conference*, pages 40–46, Munich, Germany, Mar. 12–15 2001.