An Efficient Methodology and Semi-automated Flow for Design and Validation of Complex Digital Signal Processing ASICS Macro-Cells

L.Tambour, N. Zergainoh, P. Urard†, H. Michel†, A.A. Jerraya
TIMA Laboratory
46 avenue Felix Viallet 38031 Grenoble France
{Ludovic.Tambour, Nacer-Eddine.Zergainoh, Ahmed.Jerraya}@imag.fr

† ST Microelectronics
850 rue Jean Monnet, 38926 Crolles, France
{Pascal.Urard, Henri.Michel}@st.com

Abstract

We present a methodology and design flow for signal processing application specific integrated circuit macro-cells. The key features of the methodology are the mastering the complexity of design, the increasing of reuse factor and the early error detection. It takes advantages of a derivative designs, a signal processing modularity, generic modeling and combines both levels of abstraction, in order to produce an efficient architecture. The flow includes a fast verification platform that drives both algorithm and architecture validation in an efficient way. We illustrate the effectiveness of the proposed methodology by a significant industrial application. Experimental design results indicate strong advantages of the proposed schemes.

1 Introduction

Digital signal processing (DSP) is used for automotive, telecommunications, wireless, multimedia, networking aerospace applications and consumer products. The increasing complexity and data rates of DSP algorithms demand Application Specific Integrated Circuits (henceforth called DSP-ASIC macro-cells). These DSP-ASIC macro-cells will be assembled with other components (e.g. processor and domain-specific cores, peripherals etc.) to build the required Multiprocessor System-On-Chip (MP-SoC). Today, the design of DSP-ASIC macro-cell requires much time and money. It is often too rigid to follow the many modifications that occur. Indeed, the algorithm should be thoroughly tested and optimized at all design stages before final design. To verify and optimize the signal processing algorithms, they must first be specified (*floating-point specification*) and their functionality tested completely (*functional validation*). In the next step, the algorithm is transformed into macro-architecture. The optimum signal widths are determined (*fixed-point description*), balancing cost with noise and disturbances (*word-length validation*). During this step, the macro-architecture's compatibility with the original algorithm is verified (*macro-architectural verification*). Finally, the macro-architecture is transformed into Register Transfer Level (RTL) code, i.e. micro-architecture. In this final step, the designer must verify that the code matches the specified functionality and architectural constraints. This flow requires three translations of the design, expressing the functionality as gradually less sequential and more structural with requirements for re-verification at each stage. Gaps exist between languages, tools or data used at each design step. Translations are error prone and time consuming in resolving portability troubles. Additionally, opportunities for algorithmic modifications to reduce power and area are often lost due to the separation of engineering decisions. As the complexity of the DSP-ASIC macro-cells under design increases, the development efforts increase dramatically. To keep these efforts in check and at the same time meet the design time requirements, a design flow and methodology for DSP-ASIC macro-cells that favor reuse and early error detection is essential [15,16].

1.1 Related work

The productivity offered by the expressive power of RTL languages is way below critical. This level is clearly too low for complex design. Researchers have worked on developing efficient flows for mastering the DSP design by raising the abstraction level of the input system description. The flows can be classified according to the abstraction level of the input description. The high-level synthesis (HLS) [3-9] generates RTL system description starting from a behavioral system specification. However, these tools are targeted mostly for hardware designers and are unattractive to algorithm designers. The main problem with this flow is that it attempts to avoid feeding back information to algorithm designers. Some attempt to close the gap between algorithm and hardware designers by basing synthesis tools on languages derived from general programming language such C/C++ [10-14] (e.g. HardwareC [10], Handel-C, SpecC [11], SystemC [12], etc). These languages provide concepts similar to HDLs. The modeling and the validation using these languages are faster than HDL description. However, the synthesis tools starting from these languages obscure mostly the information about the algorithm and architecture through the code generation process. The generated architectures are often incompatible with industrial requirements. The authors in [1, 2] propose a design flow based on Simulink, a high level description environment. The flow converts a Simulink block diagram (i.e. structural description) into a layout. The flow uses an efficient DSP validation environment (i.e. Simulink/Matlab environment [18]). The generated architecture is very close to the input description. In spite of using the high level environment, the flow starts from a structural description similar to RTL code. The productivity offered by the expressive power of structural description is way below critical. The use of floor-plan description involves that the technology migration is very expensive in time. However, we believe that in all above works, design methodologies tackle some issues of DSP design but they have yet to encompass the entire problem. In fact, most of above-mentioned approaches cannot satisfy a tradeoff between architecture quality, rapid algorithm/architecture exploration, and fast modeling and validation.

1.2 Contribution

The main contribution in this paper is the definition of new methodology and semi-automated flow which tackle the design and validation of DSP-ASIC macro-cells in an efficient way. The flow generates RTL architecture starting from a functional architecture model of digital system. The flow is based on the automatic assembly of pre-designed generic DSP basic blocks, this allows increasing reuse factor. The flow includes a fast verification platform that drives both algorithm and architecture validation within functional validation environment. The key characteristics of the proposed methodology (that we will justify along this paper) are:

— Mastering the increasing design complexity and reducing strongly the design effort;

— Mastering the architecture generation as if it had been designed manually;

— Allowing efficient design space exploration and rapid prototyping;

— Allowing the macro-cell generation that will be easily integrated within using an existing design flow, in order to produce a complete MP-SoC.

The rest of the paper is organized as follows. Section II presents some basic concepts adopted and introduces our methodology. Section III details the design and validation steps of the methodology and the associated design flow. Section IV presents the experimental results. Finally, section V provides our conclusions.

2 Preliminaries

In our methodology a single model is defined to represent a DSP system in all phases during the design and validation process. The model consists basically of two levels: the functional level and RT level. At functional level, the system is represented as Data Flow Graph (DFG). The edges represent the flow of data and control signals between nodes. The nodes represent the pre-designed functional DSP basic blocks (F-DSP-IP, *Functional DSP Intellectual Properties*). The DFG represents the functional description written in high level language (Matlab) of the DSP system (Figure 1).

```
% DSP Functional Architecture Model
% Name of DSP Application : DSP_Appli_Example
function [out] = DSP_Appli_Example (in)

% feedback signal
  persistent r; if (isempty(r)) r = 0; end

% parameters of functional-IP are now known and fixed
  [p1, p2, p3, p4] = read_parameter ('param_file.txt');

  x = Func_DSP_IP1(in, p1);
  y = Func_DSP_IP2(x, r, p2);
  out = Func_DSP_IP3(y, p3);

% update 'r' for the next execution
  r = Func_DSP_IP4(y, p4);
```

Figure 1: Functional architecture model

At the RT level, the system is represented by two graphs written in hardware description language (VHDL): a Data-Path Graph (DPG) and State Transition Graph (STG). The DPG is analogous to a schematic diagram; it describes the design in terms of pre-designed RTL DSP basic blocks (RT-DSP-IP, *Register Transfer level DSP Intellectual Property*), and their interconnections. The STG is used to model a global finite state machine (FSM). Figure 2 shows the synoptic structure of RT level model. In our methodology, functional modeling and RT architecture generation are performed using libraries of generic pre-designed basic blocks and macros. Three libraries are supported: generic F-DSP-IP library, generic RT-DSP-IP library and state-machine macros library.

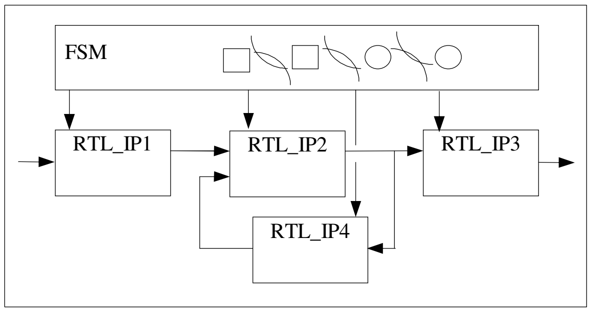


Figure 2: RTL model synoptic structure

GF-DSP-IP Block: The functional basic block is the function within the meaning of the sequential languages such as C. It describes in algorithmic form, the mathematical relation between input and output signals. In our approach, we define the GF-DSP-IP (Generic Functional DSP 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 GF-DSP-IP blocks are ranged in library by complexity and application kind. An example of GF-DSP-IP description is shown in Figure 3.

```
% DSP Library : Functional Template
% Filters library Box
% Name IP : FILTER_FIR
% Description : Finite Impulse Response Filter

function [out] = FILTER_FIR (in, nbit_in, coeff)

% compute nbit_sum and nbit_round
  nbit_sum = nbit_in + log2(sum(abs(coeff))) + 2;
  nbit_sum = ceil(nbit_sum);
  nbit_round = nbit_sum - nbit_in - 1;

% Init persistent variable : delays line
  persistent dl;
  if (isempty(dl))
    dl = zeros(1,length(coeff));
  end

% Update delay line
  dl(2:end) = dl(1:end-1); dl(1) = in;

% Compute current output
  out_filter = dl*coeff;

% quantify the output
  out_rnd = round(out_filter/(2^nbit_round));
  out = satur(out_rnd, -2^(nbit_in-1), (2^(nbit_in-1))-1);
```

Figure 3: GF-DSP-IP description of FIR filter

GRT-DSP-IP: We define the Generic Register Transfer level IP block (GRT-IP) as the architecture template describes the structural implementation of DSP basic block. Each GF-IP is mapped on one or more GRT-IPs according to the desired implementation specificity, but stilling generic like the GF-IP. Some implementation details are left open at the GRT-IP and have to be filled in by the final design step in our flow. One kind of architecture template structure (i.e. GRT-IP) is shown in Figure 4 (where the generic parameters are in italic).

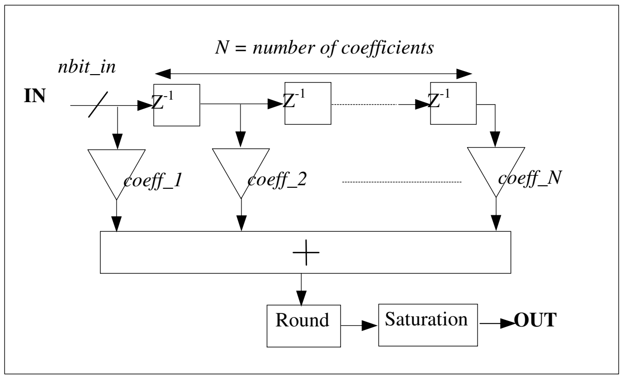
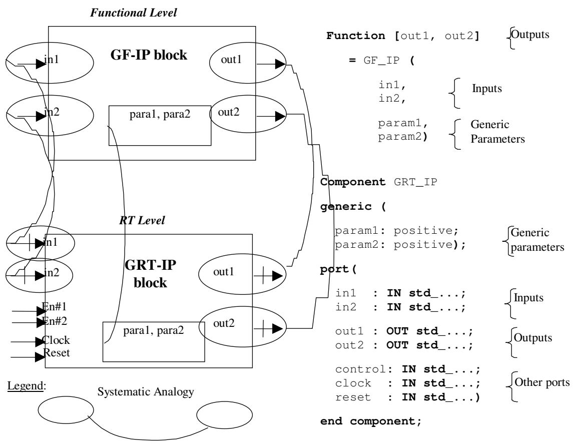


Figure 4: GRT-IP description of FIR filter

Interface Models: In order to draw an analogy and establish a relationship (in terms of interconnection issues) between RTL and functional IP blocks, we needed to study the interface models of generic IP blocks. Figure 5 draws the analogy between GF-IP and GRT-IP interfaces. The external concepts (e.g. external ports-structure, functional, and timing details, generic parameters …) of the interface model are provided to show how the generic IP component exchanges information with its environment. The GF-IP interface defines the component name, I/O data-stream names, and parameters names. The external interface concepts of GRT-IP block are 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.



```
Function [out1, out2]
   = GF_IP (
       in1,
       in2,
       param1,
       param2)

Component GRT_IP
generic (
  param1: positive;
  param2: positive);
port(
  in1   : IN std_...;
  in2   : IN std_...;
  out1  : OUT std_...;
  out2  : OUT std_...;
  control: IN std_...;
  clock  : IN std_...;
  reset  : IN std_...)
end component;
```

Figure 5: Interface concepts analogy

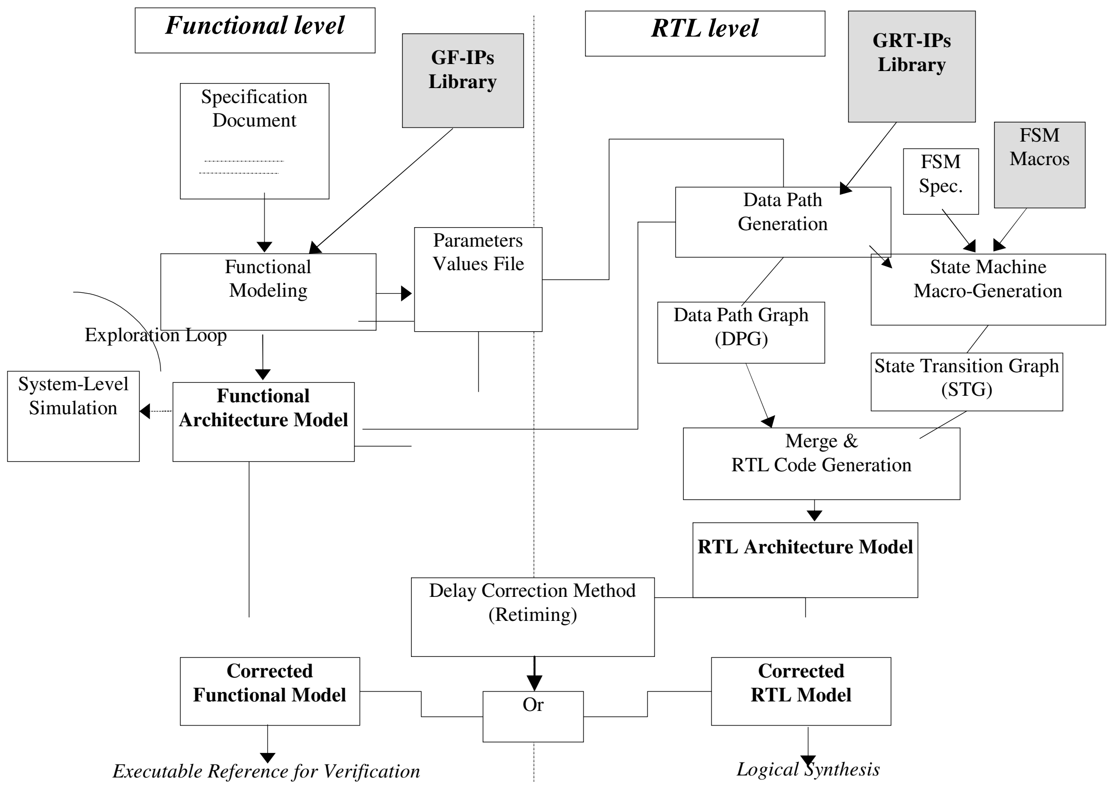


Figure 6: Proposed semi-automated design flow for DSP application

3 Design and validation methodology

An overview of our design flow is depicted in Figure 6. The input of the system is functional description in a high level language (Matlab). The output is RTL architecture (VHDL code). The functional architecture model is performed by assembly a generic GF-IPs. The choices of IPs, IP parameters values and assembly topology, are carried out by designer to satisfy a trade-off between signal quality and implementation constraints. To generate the architecture (i.e. Data Path Graph), IP parameters values are first extracted from functional architecture model, and then used to instantiate the generic RTL-DSP basic blocks. Architecture is finally built by automatic assembly of pre-designed RTL-IP blocks with same assembly topology as functional architecture model. According to implementation and functional constraints, the hardware designer can choose from different kinds of GRT-IPs by loading the appropriate GRT-IP. A global finite state machine (i.e. State Transition Graph) is generated (i.e. macro-generation-based) from a high-level FSM specification written by the designer. The composition of DPG and STG gives the final RTL architectural model. In some cases, we have identified a behavior difference between functional and RTL models. For the same test bench, both the RTL and the functional models of the whole application, produce different numerical values. This problem is generally known as retiming issue. To solve this issue, we have developed a systematic method called "delay correction method" (cf. §3.3), in order to make up for the delay and to close the gap between RTL model and functional model. The design flow includes a unified verification platform used to verify both RTL and functional model. 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 such as an executable reference for the next generation of design. The RTL model is suitable for logical synthesis.

The next sections detail the main design and validation steps of the methodology and the associated design flow.

3.1 Functional modeling

For modeling and validation of the whole DSP application, a fast functional design and validation environment is available. It includes high-level language, system-level simulator, and GF-IP library. Thanks to the modularity of DSP algorithms, the generic modeling, and the assembly technique, the methodology acts at the high abstraction level and adopts the "divide-and-conquer strategy". Starting from the specification document, the system designer decomposes the DSP system into set of subsystems. Each subsystem must be decomposed into simpler generic DSP functions of same type (i.e. DSP basic block). Once, all GF-IPs were selected and the assembly topology was defined, the system designer can easily describe the whole functionality of DSP system in functional structure form by using high level language (Matlab). In this preliminary specification, some functional constraints can be distributed on each GF-IP (e.g. noise budgeting, I/O frequencies management). This specification results in a hybrid description (*Hybrid Functional Model* [17]) of the algorithm since only some of the parameters are fixed, but the others remain generic. The next step is to determine the architectural parameters values which satisfy a trade-off between signal quality and implementation constraints. DSP algorithms are sensitive to finite word length effects. The signal-to-noise ratio is an integral part of functional modeling of DSP system. As noise is an essential aspect of the behavior of digital system, noise is the responsibility of the system designer. DSP architecture can be optimized towards area by minimizing the word size under the constraint of avoiding limit cycles and round-off noise. Parameters values exploration is performed by using functional simulation and validation platform (cf. §3.4).

The final output result of this step is functional architecture model, i.e., fixed-point functional model (including architectural parameters values file) of DSP system.

3.2 Datapath and FSM generation phases

Three major phases can be distinguished in RTL code generation of the target architecture.

The first phase is the analyzing and graph transformation phase. It consists of parsing the source program of functional model (DFG). This phase transforms the DFG graph into structural graph, identifies the GF-DSP-IPs and replaces them by their homologous RT-DSP-IPs according to systematic analogy and relationship between their interfaces. We adopt the same convention to name the interface elements. In some cases, this mapping is not a one-to-one relationship. In these cases, a mapping file is added to establish a link between both abstraction levels. One more issue will be addressed is the *unconnected ports* such control, clock, reset and test ports… These ports are lacking in functional model but must be available in RT level. The results of this phase are a *high-level structural model* of a system and an *unconnected ports list (i.e. template mapping file*). This structural model describes system in terms of GRT-IP instantiations and their topological interconnections following the physical hierarchy of the system (notice only the data-stream ports are connected, the others ports are not assigned). The unconnected ports list is a template file written in comprehensive pseudo-code format will be easily completed by designer according to the desired port connections and control constraints. Designer specifies the link between the unconnected ports and in the same time introduces the control constraints, which are added in form of sequences edges, and resolved through RTL architecture generation.

The second phase is datapath architecture generation. A sequence of generation steps transforms the high-level structural model into register-transfer description. The high-level structural model is first transformed into a new complete structural netlist; all ports are now assigned and/or connected according to the mapping file (the connection and assignment are disseminated across the hierarchy). The next step assigns to each GRT-DSP–IP one particular instance of the target architecture, according to the desired architectural constraint. The final step is the RTL code generation of the datapath architecture in accordance with coding rules for logical synthesis.

The last phase is the finite state machine generation. This phase starts from high-level FSM specification file, uses a FSM macros library and produces RTL code of global FSM of the target architecture. The designer specifies the control steps in form of transition behavior with action statements annotated as labels and added to a mapping file elaborated during the first phase. This specification is translated into synthesizable VHDL code.

3.3 Delay correction method: Retiming

As discussed earlier, in some situations we have identified a behavior difference between functional and RTL models during validation phase. In this section, we first study and classify the reasons causing the behavior difference. At last, we propose an overall solution so that the behavior difference can be systematically identified and corrected in a methodical way. We define two models as functionally equivalent when they produce the same sequence of digital values in the same order (no redundancy and no loss of value but a delay between the two sequences is allowed). Although the GF-DSP-IP and GRT-DSP-IP blocks are functionally equivalent (to the nearest delay), it happens sometimes that both models RTL and functional of the whole application (obtained by assembling the pre-designed IPs) are not functionally equivalent. Usually, this problem occurs in following cases: the model contains two convergent parallel paths, feedback-loops or time-dependent DSP-IP. In these cases, there are two reasons causing the behavior difference between RTL and functional models which are straight linked to the implementation constraints. The first one is local to each pre-designed GRT-DSP-IP block and arises from delay in the GRT-DSP-IPs. In fact the blocks are functionally equivalent to the nearest delay. The delay is due to additional clock cycles required for the implementation constraints. The second reason is global to the RTL architectural model (usually occurred in the multi-rates DSP applications) and arises from phase-difference between FSM signals.

To achieve a functional equivalence, compensation delays are cleverly added to the functional model and/or the RTL model following the case. The correction is performed in the functional model when the application contains at least one time-dependent IP block; it consists to put a delay block in front of each time-dependent IP block. These blocks are parameterized by the number of inserted delays. In the other cases, the delay correction is performed in the RTL model by inserting registers between IP blocks, in an efficient way. Conceptually, there are various steps will be performed in the delay correction method such verification of the functional equivalency by simulation, problem identification (time-dependent IP blocks, parallel IP-blocks or/and feedback loop IP-blocks), determination of the delays number and applied the correction to functional model, computation of registers number, registers partitioning/allocation and applied the correction to RTL model. The adopted method uses graph theory formalization as basis and combines several complementary algorithms (Bellman algorithm and integer linear programming) to produce the optimal solution in terms of area overhead cost and latency.

3.4 Verification platform

We propose a fast verification platform which takes advantages of the signal processing capabilities of the functional validation environment, in order to perform the RTL model verification. The use of a functional validation environment allows a better data analyze and diagnostic than the exploitation of a typical waveform viewer. The platform allows checking automatically that the RTL model products the same output values than the functional model. In our verification platform, the functional model is executed using data input stimuli produced from a pattern generator. The results can be analyzed using post-processing algorithm and save in a *result data file*. The used input stimuli are saved in a *stimuli data file* to drive the RTL simulation. The RTL compilation, simulation process, analysis and comparison between the RTL and functional results are started automatically from functional validation environment. The system produces a diagnostic file, which indicates whether the behavior difference and signals phase differences.

4 Design industrial example

In this section, we present the results that we have obtained from the design of a complex industrial DSP system of ST Microelectronics (i.e. TV digital transmission satellite application). The methodology and the design flow have been applied on the digital modulation chain of this application. The both functional and RTL models were validated. The analysis of the result shows the effectiveness of our approach. The modulation chain is typically used in a large field of telecommunication applications (cellular phone, digital radio, modem…). Hence it must strongly be reusable. It can be dedicated to be inserted with other components in a complete SOC [19].

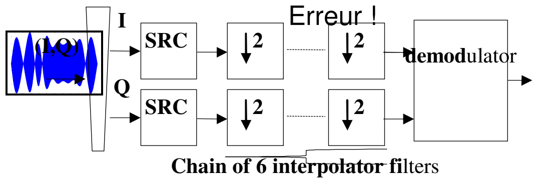


Figure 7: Digital modulation subsystem

4.1 Digital modulation chain

To verify the feasibility and effectiveness of our approach, we keep the similar specifications (i.e. functional and implementation constraints) as the real case. As shown in figure 7, the real and imaginary parts of the input signal are first separated by a demultiplexer. Then, each signal component is processing to be adapted at the output frequency. The frequency adaptation is a chain composed of a *SRC filter* and *6 interpolator filters by 2*. Finally, the real and imaginary parts are modulated by a modulator. The input frequency is 2.048 MHz. The output frequency is 200 MHz. The carrier frequency is between 5 MHz and 54 MHz. The area of the complete chip is also a constraint to be optimized. The design difficulties of this application are: the multi-frequency (I/O frequencies ratio is irrational), the high output frequency and the strict quality requirements of output signal.

4.2 Designing the digital modulation chain

From the system architecture, a functional application has been modeled and validated in Matlab [18]. Among all the architectural parameters values of the design space, we found the parameters values which allow to respect functional and implementation constraints. The parameters values have been obtained by simulation from the functional model following the proposed exploration approach. As instance, the table 1 shows the selected parameters of the 6 interpolator filters. The explored parameters are the number of coefficients and the number of bits to quantify the coefficients. From these parameters, the coefficients values can be calculated mathematically choosing an appropriate integration window. The figure 8 shows the frequency responses of each interpolator designed with the 80 dB constraint requirement.

InterpolatorFilter	Coef. Nb	Bits Nb/Coef.	Windows	Coefficients values
Filter 1	19	18	Chebyshev80	49 0 -813 0 4712 0 -17984 0 79571 79571 0 -17984 0 4712 0 -813 0 49
Filter 2	15	16	Chebyshev80	-24 0 529 0 -3487 0 19367 32768 19367 0 -3487 0 529 0 -24
Filter 3	15	16	Chebyshev80	-24 0 529 0 -3487 0 19367 32768 19367 0 -3487 0 529 0 -24
Filter 4	7	5	Hanning	-1 0 9 16 9 0 -1
Filter 5	7	5	Hanning	-1 0 9 16 9 0 -1
Filter 6	3	2	Hanning	1 2 1

Table 1: Extracted parameters values of 6 interpolators filters

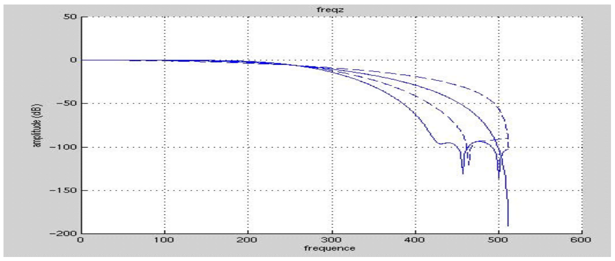


Figure 8: Frequency responses of interpolators

The final output result of this phase is complete functional architecture model of digital modulation chain in fixed-point. The obtained model perfectly respects the required constraints. In the RTL model generation phase, we applied the various steps of our methodology to design the datapath and FSM. The result of this phase is the RTL model of the digital modulation chain. The next phase is the analysis and comparison between the RTL and functional results which performed from the functional validation environment. During this phase, the system detected a behavior difference between the both models. The diagnostic file indicated that the behavior difference comes from counters in the SRC filter and the modulator (i.e. time-dependent block) as well as signals phase differences. A delays correction has been performed on the functional model to exactly obtain the same digital values. This correction is performed in adding delay blocks between the functional DSP-IPs blocks. The number of compensatory delays has been determined by system-level simulation approach. Finally a RTL model of digital modulation chain (e.g. 600 MGates) satisfying both implementation and functional constraints has been obtained as shown in Figure 9.

4.3 Result analysis

Table 2 shows the code lines number of each predesigned generic DSP-IP block. In our approach, we assumed that all IP blocks are available. In this case, the number of code lines written manually to assemble the IP blocks, in order to design functional model, is of same magnitude as the number of the calls of GF-DSP-IP blocks (i.e. a few Matlab lines). The VHDL model is automatically assembled, only some lines (pseudo-code) must be written to allow the generation of the FSM.

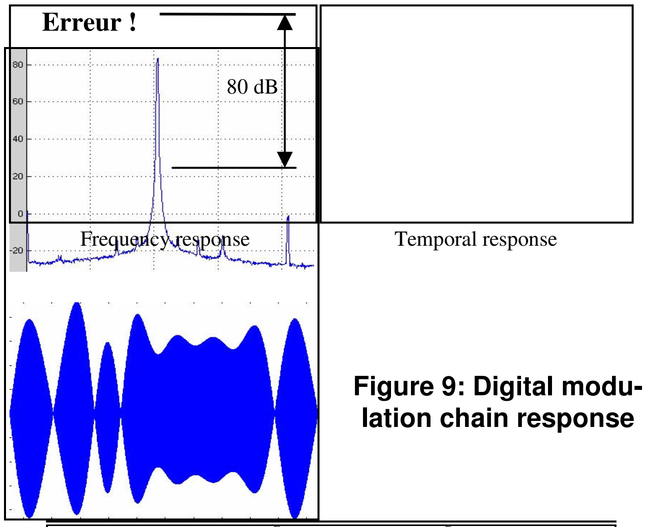


Figure 9: Digital modulation chain response

Generic DSP-IP	Matlab lines	VHDL lines
Demultiplex	6	78
Interp Filter	15	379
SRC Filter	134	711
Modulator	84	450

Table 2: Matlab versus VHDL code line numbers

Table 3 shows the cost in time to explore and to fix the all parameters values of the generic IPs. Six hours are necessary to fix the all parameters that keep to the required constraints. When the 80 dB constraint is changed, the new parameters values can immediately be reported into the RTL model. Different derivative architectures can be quickly explored changing the system architecture in the functional model. For instance, a chain of interpolator filters by 4 or by 8 can be tried. Also, the methodology allows to quickly trying different IP architectures when they are available in the library.

Generic DSP-IP	Exploration cost
Demultiplexer	(no parameters)
6 Interpolator Filters	~ 4 hours
SRC Filter	~ 1 hour
Modulator	~ 1 hour

Table 3: Cost in time to explore parameters

Considering the simulation speed, the simulation time of functional model is 40 times faster than the RTL model. The system-level simulation of digital modulation chain takes approximately 90 seconds, whereas the cycle accurate simulation takes more than 1 hour. Matlab model has been translated into C code via Matlab Compiler for fast simulation. The parameters values have been obtained by simulation from the functional model. The implementation constraints are directly specified in the functional model during the choice of the system architecture, the parameters values and IPs architectures. This allows a fast both algorithm and architecture explorations as well as to sensibly increase the reuse factor. The factor reuse is estimated at more 90% compared to a manual design. For instance, both the functional and RTL models of the chain of modulation can be obtained and validated in less than 24 hours, according to designer expertise and the availability of IPs blocks

5 Conclusion

In this paper, we presented a design and verification methodology for DSP ASIC macro-cells. The design and verification methodology is based on the assembly of pre-designed generic DSP-IPs. In our methodology, the designer is in center of the design process and masters the resulting architecture as a manual design. The RTL verification is performed from the environment used for functional validation. This allows reusing the validation environment for more efficient RTL verification. The proposed methodology allows the master of the resulting architecture, important reuse factor, quick architecture exploration and efficient verification. It has been applied on a concrete industrial.

References

[1] Da Silva, L, et al.: *Design methodology for Pico Radio networks*. Proc. IEEE/ACM Design, Automation and Test in Europe, March 2001, pp. 314–323.

[2] Davis, W. R, et al.: *A Design Environment for High Throughput Low-Power Dedicated Signal Processing Systems*. IEEE Journal of Solid-State-Circuits Vo. 37, No 3 March 2002.

[3] Jerraya. A.A, et al.: *Behavioral Synthesis and Component Reuse with VHDL.* Kluwer Academic Publishers, 1996.

[4] Hurk, J. Dilling, E.: *System Level Design, a VHDL Based Approach*. Proceedings of Euro-DAC 1995.

[5] Elliott, J.P.: *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design*. Kluwer Academic Publishers, 1999.

[6] Gajski, D., Ramacahndran, L.: *Introduction to high level synthesis*. IEEE Design and Test Computer, October 1994.

[7] Genoe, M., et al.: *On the use of VHDL-based behavioral synthesis for telecom ASIC design*. In the Proceedings of the International Symposium on System Synthesis ISSS'95, February 1995.

[8] Lauwereins R., Engels M., Ade M.: *GRAPE-lI: A System-Level Prototyping Environment for DSP Applications*. IEEE Computer, February, 1995, PP 35-43.

[9] Lee M.T., et al.: *Domain-specific high-level modeling and synthesis for ATM switch design using VHDL.* DAC'96.

[10] Ku, D.C. and De Micheli, D.*: HardwareC – A Language for Hardware Design*. Stanford University, Technical Report CSL-TR-90-419, 1988.

[11] SpecC. Available at http://www.ics.uci.edu/~specc/

[12] SystemC. Available at http://www.systemc.org/

[13] CoWare N2C. Available at http://www.coware.com

[14] De Micheli, G.: *Hardware Synthesis from C/C++ Models*. ACM/IEEE DATE'99, pp.382-383, March 99.

[15] Martin, G., Salefski, B.: *Methodology and Technology for Design of Communication and Multimedia Products via System-Level IP Integration*. ACM/IEEE DATE 1998.

[16] Koegst, M. and al.: *A Systematic Analysis of Reuse Strategies for Design of Electronic Circuits.* DATE 1998.

[17] Lee E.A., and Sangiovanni-Vincentelli, A.:"*A framework for comparing models of computation,*" IEEE Trans. CAD, vol. 17, pp. 1217–1229, Dec. 1998.

[18] The MathWorks, Inc. Matlab, Simulink and stateflow. Available at http://www.mathworks.com.

[19] Zergainoh N., et Al.: *Framework for System Design, Validation and Fast Prototyping of Multiprocessor SoCs*. Chapter in book, Edited by B. Kleinjohann, Kluwer Academic Publishers, April 2001.