UML/XML-based approach to hierarchical AMS synthesis

I. O'Connor, F. Tissafi-Drissi, G. Révy, F. Gaffiot
Laboratory of Electronics, Optoelectronics and Microsystems
Ecole Centrale de Lyon, 36 avenue Guy de Collongue, F-69134 Ecully, France
ian.oconnor@ec-lyon.fr

Abstract

This paper explores the suitability of UML techniques for defining hierarchical relationships in AMS (analogue and mixed-signal) circuit blocks, and XML for storing soft AMS IP design rules and firm AMS IP design data. Both aspects are essential to raising the abstraction level in synthesis of this class of block in SoCs. The various facets of AMS IP are discussed, and explicit mappings to concepts in UML are demonstrated. Then, through a simple example block, these concepts are applied and the successful modification of an existing analogue synthesis tool to incorporate these ideas is proven. The central data format of this tool is XML, and several examples are given showing how this meta-language can be used in both AMS soft-IP creation and firm-IP synthesis.
Keywords: analogue and mixed-signal, synthesis, IP, UML, XML

1. Introduction

Cost, volume, power and pervasivity are all difficult constraints to manage in the design of new integrated systems (smart wireless sensor networks, ubiquitous computing…). Along with increasingly complex functionality and person-machine interfaces, they are driving the semiconductor industry towards the ultimate integration of complete, physically heterogeneous systems on chip. The coexistence of sensors, analogue/mixed and radio frequency systems (multi-physics part commonly called AMS) with digital and software IP blocks causes significant design problems.

The difficulty centres on the concept of abstraction levels. To deal with increasing complexity (in terms of number of transistors), SoC design requires higher abstraction levels. But at the same time, valid abstraction is becoming increasingly difficult due to physical phenomena becoming first order or even dominant at nanometric technology nodes. The rise in analogue, mixed-signal, RF and heterogeneous content to address future application requirements compounds this problem. Efficient ways must be found to incorporate non-digital objects into SoC design flows in order to ultimately achieve AMS/digital hardware/software co-synthesis.

The main objective of such an evolution is to reduce the design time in order to meet the time to market constraints. It is widely recognized that for complex systems at advanced technology nodes, mere scaling of existing design technology will not contribute to reducing the "design productivity gap" between the technological capacity of semiconductor manufacturers (measured by the number of available transistors) and the design capacity (measured by the efficient use of the available transistors). Since 1985, production capacity has increased annually by between +41% and +59%, while design capacity increases annually by a rate of only +20% to +25%. The 2003 ITRS Roadmap clearly states that "cost [of design] is the greatest threat to continuation of the semiconductor roadmap". Only the introduction of new design technology (such as, historically, block reuse or IC implementation tools – each new technology has allowed design capacity to "jump" to catch up with production capacity) can enable the semiconductor industry to control design cost. Without design technology advances, design cost becomes prohibitive and leads to weak integration of high added value devices (such as RF circuits). One of the next advances required by the EDA industry is a radical evolution in design tools and methods to allow designers to manage the integration of heterogeneous AMS content.

2. Definition of AMS IP element requirements for synthesis tools

Most analogue and RF circuits are still designed manually today, resulting in long design cycles and increasingly apparent bottlenecks in the overall design process [GIE2005]. This explains the growing awareness in industry that the advent of AMS synthesis and optimisation tools is a necessary step to increase design productivity by assisting or even automating the AMS design process. The fundamental goal of AMS synthesis is to quickly generate a first-time-correct sized circuit schematic from a set of circuit specifications. This is critical since the AMS design problem is typically under-constrained with many degrees of freedom and with many interdependent (and often-conflicting) performance requirements to be taken into account.

Synthesisable (soft) AMS IP is a recent concept [HAM2003] extending the concept of digital and software IP to the analogue domain. It is difficult to achieve because the IP hardening process (moving from a technology-independent, structure-independent specification to a qualified layout of an AMS block) relies to a large extent on the quality of the tools being used to do this. It is our belief that a clear definition of AMS IP is an inevitable requirement to provide a route to system-level synthesis incorporating AMS components.

Table 1 summarizes the main facets necessary to AMS IP. For the sake of clarity, a reference to VHDL-AMS concepts is shown wherever possible.

Property	**Short description**	**VHDL-AMS equiv.**
Function definition	Class of functions to which the IP block belongs	entity, behavioural architecture
Performance criteria	Quantities necessary to specify and to evaluate the IP block	generic
Terminals	Input/output links to which other IP blocks can connect	terminal
Structure	Internal component-based structure of the IP block	structural, architecture
Design variables	List of independent design variables to be used by a design method or optimisation algorithm	subset of generic map
Physical parameters	List of physical parameters associated with the internal components	generic map
Evaluation method	Code defining how to evaluate the IP block, i.e. transform physical parameter values to performance criteria values. Can be equation- or simulation-based (the latter requires a parameter extraction method).	(partly) process or procedure
Parameter extraction method	Code defining how to extract performance criteria values from simulation results (simulation-based evaluation methods only).	
Synthesis method	Code defining how to synthesize the IP block, i.e. transform performance criteria requirements to design variable values. Can be procedure- or optimisation based.	
Constraint distribution method	Code defining how to transform IP block parameters to specifications at a lower hierarchical level.	

Table 1 AMS IP block facets

Figure 1 shows how these various facets of AMS IP should be brought together in an iterative single-level synthesis loop. Firstly, the *performance criteria* are used as specifications to quantify how the IP block should carry out the defined function. Performance criteria for an amplifier will include gain, bandwidth, PSRR, offset, etc. They can be considered to be the equivalent of **generics** in VHDL-AMS. They have two distinct roles, related to the state of the IP block in the design process:

1. as *block parameters* when the IP block is a component of a larger block, higher up in the hierarchy, in the process of being designed;
2. as *specifications* when the IP block is the block in the process of being designed (such as here). This role cannot be expressed with VHDL-AMS generics, although language extensions have been proposed [DOB2003] [HER2004].

It will be shown in section 3.2 that this dual role requires the definition of a new data type.

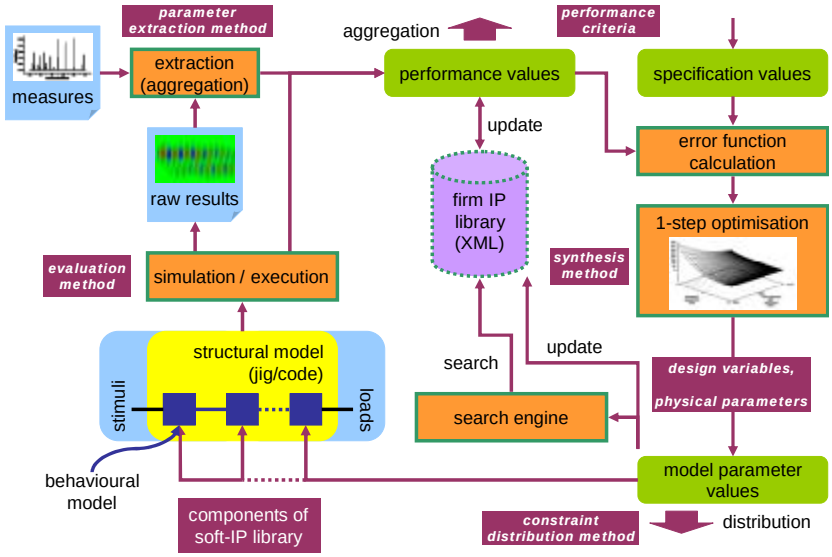

Figure 1 Single-level AMS synthesis loop showing context of AMS IP facet use

The comparison between specified and real performance criteria values act as inputs to the *synthesis method*, which describes the route to determine design variable values. It is possible to achieve this in two main ways:

1. through a direct procedure definition, if the design problem has sufficient constraints to enable the definition of an explicit solution;
2. through an iterative optimisation algorithm. If the optimisation process cannot, as is usually the case, be described directly in the language used to describe the IP block then a communication model must be set up between the optimiser and the evaluation method. A direct communication model gives complete control to the optimisation process, while an inverse communication model uses an external process to control data flow and synchronization between optimisation and evaluation. The latter model is less efficient but makes it easier to retain tight control over the synthesis process.

The synthesis method generates combinations of *design variables* as exploratory points in the design space. The number of design variables defines the number of dimensions of the design space. The design variables *must* be independent of each other and as such represent a *subset* of IP block parameters (i.e. performance criteria, described above) in a structure definition. For example, a differential amplifier design variable subset could be reduced to a single gate length, bias voltage, and three transistor widths for the current source, matched amplifier transistors and matched current mirror transistors. *Physical variables* are directly related to design variables and serve to parameterise all components in the structure definition during the IP block evaluation process. These are represented by the **generic map** definitions in structural architecture component instantiations in VHDL-AMS. In the above example, the design variable subset would be expanded to explicitly define all component parameters.

The *evaluation method* describes the route from physical variable values to the performance criteria values and completes the iterative single-level optimisation loop. Evaluation can be achieved in two main ways:

1. through direct code evaluation, such as for active surface area calculations;
2. through simulation (including behavioural simulation) for accurate performance evaluation (gain, bandwidth, distortion...). If the IP block is not described in a modelling language that can be understood by a simulator, then this requires a gateway to a specific simulator and to a jig corresponding to the IP block itself. For the simulator, this requires definition of how the simulation process will be controlled (part of the aforementioned communication

model). For the jig, this requires transmission of physical variables as parameters, and extraction of performance criteria from the simulator-specific results file. The latter describes the role of the *parameter extraction method*, which is necessary to define how the design process moves *up* the hierarchical levels during bottom-up verification phases.

Once the single-level loop has converged, the *constraint distribution method* defines how the design process moves *down* the hierarchical levels during top-down design phases. At the end of the synthesis process at a given hierarchical level, an IP block will be defined by a set of physical variable values, some of which are parameters of an IP sub-block. To continue the design process, the IP sub-block will become an IP block to be designed and it is necessary to transform the block parameters into specifications. This requires a definition of how each specification will contribute to an error function for the synthesis method and includes information additional to the parameter value (weighting values, specification type: constraint, cost, condition ...).

3. UML in AMS design

3.1. Reasons for using UML in analogue synthesis

UML (Unified Modeling Language) is a graphical language enabling the expression of system requirements, architecture and design, and is mainly used in industry for software and high-level system modelling. UML 2.0 is due to be adopted as a standard by OMG (Object Management Group) in 2005. The use of UML for high-level SoC design in general appears possible and is starting to generate interest in several research groups [RIC2005]. A recent proposal [CAR2004] demonstrated the feasibility of describing AMS blocks in UML and then translating them to VHDL-AMS, building on other approaches to use a generic description to target various design languages [CHA2004]. This constitutes a first step towards raising abstraction levels of evaluatable AMS blocks. Considerable effort is also being put into the development of "AMS-aware" object-oriented design languages such as SystemC-AMS [VAC2003] and SysML [VAN2005]. However, further work must be carried out to enable the satisfactory partitioning of system-level constraints among the digital, software and AMS components. At the system level, the objective in SoC design is to map top-level performance specifications among the different blocks in the system architecture in an optimal top-down approach. This is traditionally done by hand in an *ad hoc* manner. System-level synthesis tools are lacking in this respect and must find ways of accelerating the process by making reasoned architectural choices about the structure to be designed, and by accurately predicting analogue/RF architectural specification values for block-level synthesis.

Therefore, to be compatible with SoC design flows, top-down synthesis functionality needs to be added to AMS blocks. Our objective in this work is to demonstrate that this is possible. Since UML is a strong standard on which many languages are based (SysML is directly derived from UML, and SystemC as an object-oriented language can be represented in UML also), it should be possible to map the work to these derived or related languages.

3.2. Mapping AMS IP requirements to UML concepts

In order to develop a UML-based approach to hierarchical AMS synthesis, it is necessary to map the AMS IP element requirements given in the previous section to UML concepts.

UML has many types of diagrams, and many concepts that can be expressed in each - many more, in fact, than are actually needed for the specific AMS IP problem. Concerning the types of diagram, two broad categories are available:

1. structural diagram, to express the static relationship between the building blocks of the system. We used a class diagram to describe the properties of the AMS IP blocks and the

intrinsic relations between them. The tenets of this approach and how to generate UML-based synthesisable AMS IP will be described in this section, with an example in section 5.

2. behavioural diagram, showing the evolution of the system over time through response to requests, or through interaction between the system components. We used an activity diagram to describe the AMS synthesis process. This will be described in further detail in section 3.3, and extensions to an existing AMS synthesis tool to incorporate these concepts will be shown in section 4.

3.2.1. Class relationships

Firstly, it is necessary to establish a clear separation of a *single* function definition (entity and functional behavioural model for top-down flows) from *n* related structural models (for single-level optimisation and bottom-up verification). Each structural model will *contain* lower-level components, which should be described by another function definition. It is also necessary to establish functionality and requirements common to *all* structural models whatever their function. By representing all this in a single diagram (Figure 2), we are in fact modelling a library of system components; not the actual system to be designed itself. This can be done using an object diagram - however, in this work we will focus on the broader class diagram.

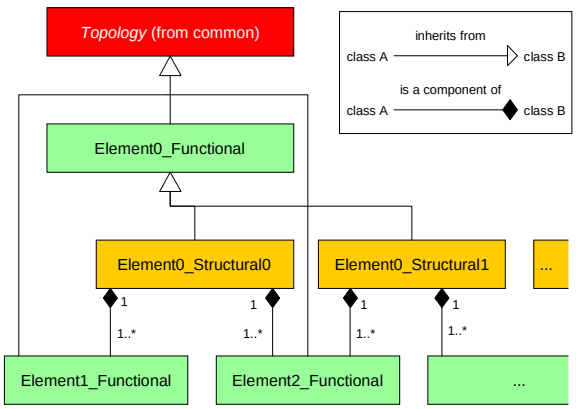

Figure 2 UML class diagram showing representation of hierarchical dependencies between AMS IP blocks

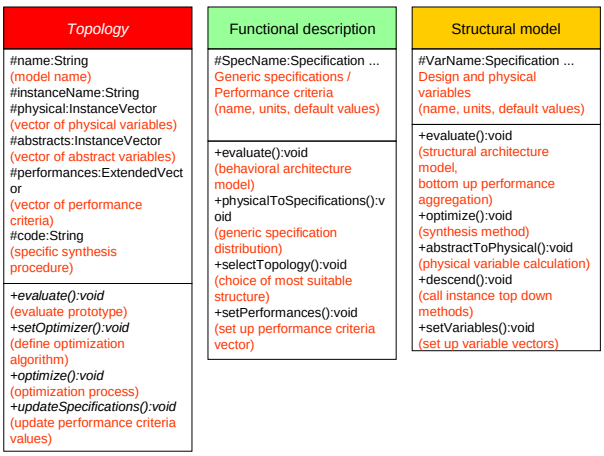

Figure 3 UML class definitions for hierarchically dependent AMS IP blocks

A class diagram constitutes a static representation of the system. It allows the definition of classes among several fundamental types, the class attributes and operations, and the time-invariant relationships between the various classes. From the above analysis, we require (cf. Figure 3):

1. a single, non-instantiable (*abstract*) class representing common functionality and requirements, in a separate publicly accessible package. We called this class Topology.
2. a single class representing the function definition, which *inherits* from Topology. An alternative solution would be to separate "Evaluatable" functionality and "Synthesisable" functionality through the use of *interfaces*. This is certainly a debatable point, but our view is that it would tend to overcomplicate the description process. Another point is that one can also be tempted to separate the *entity* aspect from the *behavioural model* aspect, which would then allow the entity class to become abstract. Again, this also appears to be somewhat overcomplicated to carry out.
3. n classes representing the structural models, which all *inherit* from the function definition class. Each structural variant is *composed* of a number of components at a lower hierarchical level, represented by a single function definition class for each component with different functionality. As the structural variant cannot exist if the component classes do not exist, this composition relationship is strengthened to an *aggregation* relationship.

3.2.2. AMS IP requirement handling through definition of class attributes and methods

Having established how to separate particular functionality between common, functional and structural parts of an AMS hierarchical model, it is now necessary to define how to include each facet of the AMS IP requirements set out in section 2. This is summarized in Table 2.

Property	Class	Attribute Type	Method	Access
Function definition • entity name • behavioural architecture	 Functional Functional	 String	constructor evaluate()	public private public
Performance criteria	Functional	Specification	 setPerformances()	protected public
Terminals	Functional	DomainNode		protected
Structure • structural architecture name	 Structural	 String		 private
Design variables	Structural	Specification	 setVariables()	protected public
Physical parameters	Structural	Specification		protected
Evaluation method	Structural		evaluate()	public
Parameter extraction method				
Synthesis method	Structural		optimize() abstractToPhysical()	public public
Constraint distribution method	Structural Functional		descend() physicalToSpecifications()	public public

Table 2 Mapping of AMS IP requirements from section 2 to class structure in Figure 2

Thus the performance criteria and variables are defined with type Specification. This is a specific data type, which plays an important role in the definition of AMS IP. It requires a name String, default value and units String as minimum information. When used as a performance requirement in a base class, it can also take on the usual specification definitions (<, >, =, minimize, maximize).

3.3. Modelling the analogue synthesis process with activity diagrams

In UML, a behavioural diagram complements structural diagrams by showing how objects or classes interact with each other and evolve over time to achieve the desired functionality. Among these, the activity diagram is useful for showing the flow of behaviour (objects, data, control) across multiple classes as a sort of sophisticated dataflow diagram.

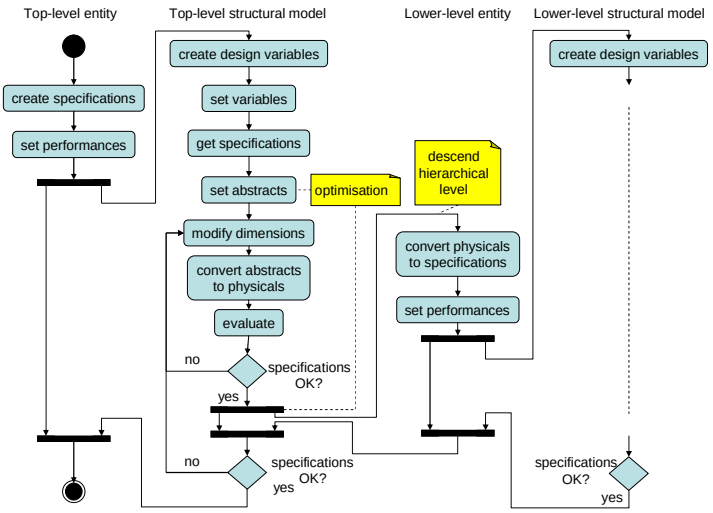

Figure 4 Activity diagram representing hierarchical AMS IP synthesis process for TIA block

Figure 4 shows an example flow for two hierarchical levels. For a given hierarchical level, the process begins with specification definition – either from an external point (e.g. user) or from the design process at the hierarchical level immediately above. It then calls a number of internal methods (set performances, variables and abstracts), all of which must have been explicitly defined by the IP creator prior to synthesis. The optimisation process can then begin with dimension (or variable) value modification according to the optimisation algorithm used, determination of the physical parameter values from the new design variable set, evaluation and comparison of achieved performance values with requirements. If the requirements are met, then the process can go down through the hierarchy to determine the parameters of lower hierarchical blocks, or if there are no lower levels then the verification process can begin. It should be noted that the sequence of events for a functional/structural model pair maps to the iterative loop shown in Figure 1.

4. Description of extensions to existing analogue synthesis tool (runeII)

We have incorporated these concepts into an existing in-house AMS synthesis framework, *runeII*. This builds on a previously published version of the tool [TIS2004]. The main motivation behind this evolution was to improve the underlying AMS IP representation mechanisms, and to enhance the input capability of the tool. A schematic showing the various inputs and data files is given in Figure 5.

From the user's point of view, there are two main phases to AMS synthesis: AMS soft-IP definition, which can be done via UML, XML or through a specific GUI[1]; and AMS firm-IP synthesis, which can be run from the GUI or from *scenarios*. XML (eXtensible Markup Language) is a text markup language for interchange of structured data specified by W3C. The Unicode Standard is the reference character set for XML content, and because of this portable format and ease of use, it is fast becoming a *de facto* standard for text-based IP exchange.

4.1. AMS soft-IP definition

The aim of the first point is to create executable and synthesisable models (here, in the form of java .class files). We consider the central, portable format to be XML, which can be generated directly from the GUI and from .java source files.

A screenshot of the GUI enabling creation of such files from graphical format is shown in Figure 6. The various zones in the figure have been numbered and the corresponding explanation follows:

1. menu bar
2. database tree explorer (top nodes = entity/functional models; nested nodes = structural models). The user is able to process several actions: new, open, export, import, delete, rename, cut, copy, paste. These actions operate on the currently selected structure or function and are also available in the main frame toolbar
3. entity/functional model editor. Here, the IP creation process starts in earnest in defining the various performance criteria
4. structural model editor. This window allows the creation of design variables, physical parameters, evaluation procedures …
5. preset design plans (sequences of optimisation algorithms)
6. technology data files
7. message window. This is to output log information, e.g. detailed information about the operation which is being made, error descriptions etc.

[1] Graphical User Interface

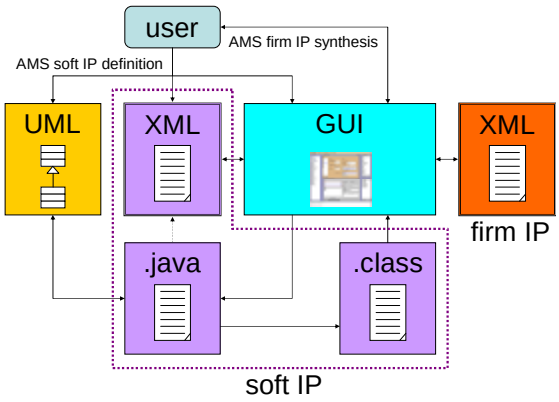

Figure 5 UML/XML use flow in runeII

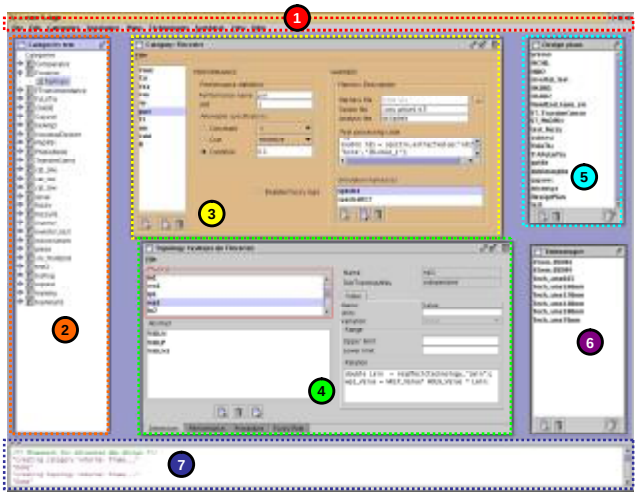

Figure 6 Screenshot of runeII GUI

4.2. AMS firm-IP synthesis

The second point exploits the created executable, synthesisable models in an iterative process aiming to determine the numerical parameter values necessary to optimally realize numerical performance requirements. Again, the database format was chosen as XML for reasons of portability. Here though, apart from capture of the numerical information itself in an XML document, a definition of the legal building blocks necessary to interpretation of the XML document structure is required. This is the purpose of a DTD (Document Type Definitions), which can be declared inline in the XML document, or as an external reference. We have chosen the latter approach, shown in Figure 7.

```
<!ELEMENT FunctionName (Structure1,
Structure2*)>
  <!ATTLIST FunctionName
    PerformanceName1 CDATA ""
    …
  >
```

```
<!ELEMENT StructureName (Component1,
Component2, ...)>
  <!ATTLIST StructureName
    VariableName1 CDATA ""
    …
  >
```

Figure 7 Entity/Functional and Structural model DTD template

As mentioned previously, the synthesis process can be run either from the GUI or through the creation of *scenarios*. Scenarios are another type of class which instantiate and setup all the components necessary for synthesis in their constructor, much as in a traditional netlist, and then define the optimisation process in the *main* method. The scenario actually represents the final executable and, while more difficult to generate, avoids any constrictions imposed by the GUI.

5. Example

We now introduce an example circuit to illustrate the concepts previously described. We focus on the representation of a resistive feedback TIA (consisting of a non-differential inverting amplifier with feedback resistance) as part of a CMOS photoreceiver front-end (Figure 8) [TIS2003].

It is important to understand how a TIA is specified in the link. The main performance criteria for the TIA itself are the in-band transimpedance gain Z_{g0}, angular resonant frequency ω_0, quality factor Q, quiescent power dissipation and occupied surface area. The first three quantities express the capacity of the TIA to convert an input photocurrent variation to an output voltage variation according to a linear second-order transfer function. The latter two criteria (power and area) can only be

accurately determined by synthesising down to transistor level, constituting the main difficulty in AMS IP formulation. To reach this level, the specific TIA structure (resistive feedback) is considered. The physical parameters consist of the feedback resistance value R_f, and the internal amplifier performance criteria (voltage gain A_v, output resistance R_o…). Concerning the design variables, it has been shown in [OCO2003] that only one is necessary: M_f, the ratio between R_f and R_o.

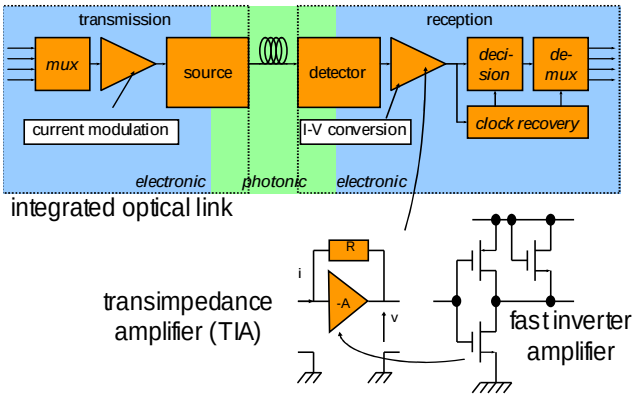

Figure 8 Context of TIA and internal amplifier in an integrated optical link

5.1. Class diagram example

This information suffices to start building a class diagram for the TIA structure (Figure 9, Poseidon®).

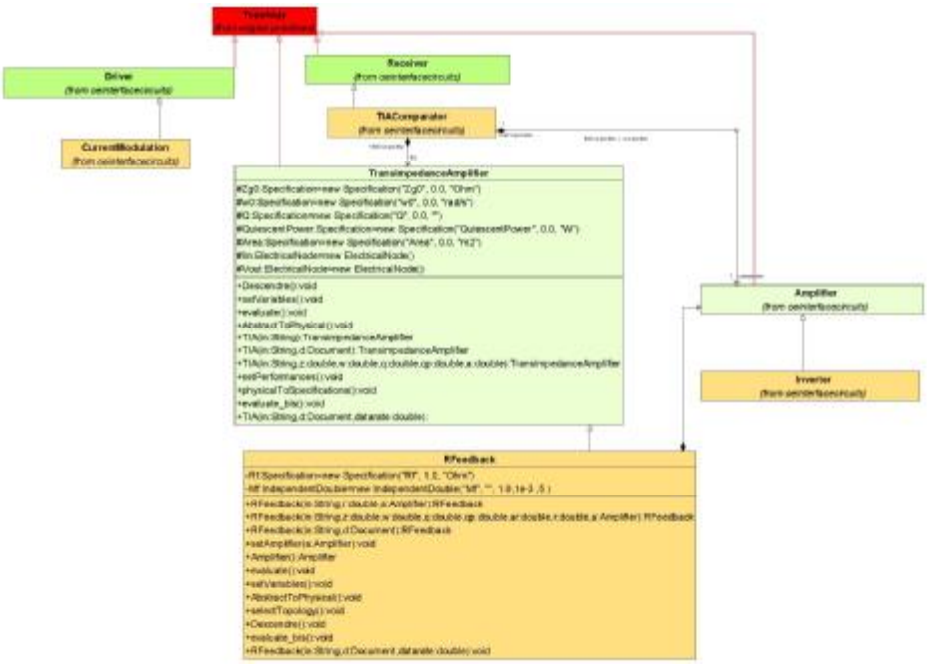

Figure 9 Class diagram showing (in expanded form) TIA function and resistive feedback structure

For clarity, only the TransimpedanceAmplifier functional model class, defining the TIA performance criteria, and its derived RFeedback structural model class, defining the physical and design variables, have been expanded. It should be noted that the physical variables related to the internal amplifier are defined in an aggregation relationship between the RFeedback class and the Amplifier functional model class. The other classes show their context in a class diagram representing an optical receiver circuit hierarchy. Some of the class methods shown are related to the specific implementation. In particular, some constructors require XML document inputs. These represent firm-IP data allowing the block to retrieve previously stored information, in the format described in section 5.4.

5.2. Soft-IP XML file example

An example of an XML file representing (i) an entity/functional model is given in Figure 10, and (ii) a structural model is given in Figure 11. Both are based on specific DTD rules corresponding to the concepts set out in section 4.2 and illustrate the various facets of AMS IP defined in section 2.

```
<category name="TransimpedanceAmplifier">
- <template name="Zg0" units="Ohm">
    <definitions constraint=">" cost="maximize" condition="0.1"/>
  - <harness simulator="spectre" file="input.scs" options="-env artist4.4.5" analysis="ac"
selected="true">
    - <code>
        Zg0 = spectre.gainMax("ID:p","vo");
      </code>
    </harness>
  + <harness simulator="eldo" file="input.cir" options="" analysis="ac" selected="false">
    </harness>
  </template>
+ <template name="QuiescentPower" units="W"></template>
…
```

Figure 10 Entity/Functional model description output in XML

```
- <topology name="TransimpedanceAmplifier-RFeedback" instanceName=""
categoryName="TransimpedanceAmplifier">
  + <physical type="dependent" name="Amplifier" instanceName="A1" categoryName="Amplifier">
    </physical>
  + <physical type="independent" name="Resistance" instanceName="Rf"></physical>
  - <abstract type="independent" name="Double" instanceName="Mf">
    - <dimension name="Value" units="" lower="0.0010" upper="100.0" variation="linear">
      </dimension>
    </abstract>
…
  - <performance name="Zg0" units="Ohm" heuristic="false" enabled="false">
    - <equation>
        Zg0  =  ((Rf_Value * A1.Av())- A1.Ro())/(1 + A1.Av());
      </equation>
    </performance>
  + <performance name="QuiescentPower" units="W" heuristic="false"
enabled="false"></performance>
…
</topology>
```

Figure 11 Structural model description output in XML

5.3. Optimisation scenario example

As a simple example, Figure 12 shows the scenario (java source) to optimise the RFeedback object.

```
package scenarios;
import basic.*; ...
public class S_RFeedback extends TestTIA {
  public S_RFeedback() {
    try {  // load specifications
           Document TIADoc = ReadXML.loadDocument("/home/work/xmlFiles/TIA_specs.xml", true);
           // create RFeedback object with specifications. Sizing is done in the constructor.
           // Assign it to tia object (defined in TestTIA base class)
           tia = new RFeedback("Rf",TIADoc);
        } catch (Exception e) { e.printStackTrace(System.err); }
    } // end constructor
  public static void main(String[] args) {
    try {  // create scenario object. Design process defined and executed in constructor.
           S_RFeedback scenario = new S_RFeedback();
           // evaluation of resulting RFeedback object
           scenario.getTIA().evaluate();
           // store results in firm IP database
           Document outputDocTIA = new Document(WriteXML.XMLTopology(scenario.getTIA()));
           WriteXML.save("/home/work/xmlFiles/outxml/S_TIA_perfs.xml",outputDocTIA);
        } catch (Exception e) { e.printStackTrace(System.err); }
    } // end main
} // end S_RFeedback
```

Figure 12 TransimpedanceAmplifier/RFeedback optimisation scenario description (java source)

5.4. Firm-IP XML output file example

The partial results, in the output XML format, of this synthesis process achieved for a 0.35µm CMOS technology and with specifications given in the first line of the file, are shown in Figure 13.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE GenericLink SYSTEM "Link_dtd.dtd">
<GenericLink BER="< 1.5E-18 Bit/s" DataRate="> 1.5e9 Gbit/s" Abstract="True"
toOptimize="True">
  - <OPPLink>
    + <Driver BiasCurrent="= 2e-3 A" ModulationCurrent="= 25e-6 A" Abstract="False"
toOptimize="True">
      </Driver>
    + <WaveguideStructure Loss="2e-2 U" Length="2e-3 U"/>
    - <Receiver>
      + <Detector extinctionRatio="1.0 U" currentNoise="1.0 U" Responsivity="0.8 A/W" />
      - <TIAComparator>
        ...
        <TransimpedanceAmplifier Cd="= 400.0E-13 F" Cl="= 150E-13 F" Zg0="= 1E3 Ohm" Q="=
0.7017" Abstract="True" toOptimize="True">
          <RFeedback Rf ="1390 Ohm">
            <Amplifier Av="10" Ro="500 Ohm" Cm="8.0 E-14 F" Co="5.0 E-13 F" Ci="7.0E-13 F"
QuiescentPower="0.5E-3 W" Abstract="True"/>
          </RFeedback>
        </TransimpedanceAmplifier>
        ...
        + <Comparator BW="= 3 GHz" QuiescentPower="= 164E-6" Latence="= " refVoltage="= "
Vl="= 0.1 V" Vh="= 0.8 V" Lmin="= 0.35E-6 m" Abstract="True" toOptimize="True"/>
      </TIAComparator>
    </Receiver>
  </OPPLink>
</GenericLink>
```

Figure 13 Firm IP synthesis results in XML format

6. Conclusion

In this paper, we have proved the feasibility of the use of UML for the representation of synthesisable hierarchical AMS IP blocks. A parallel between UML concepts and widely used concepts in AMS behavioural modelling languages (we used the VHDL-AMS example) was established, in particular:

- class diagrams to represent the various ways (structural architectures) of realizing a given function (entity and behavioural architectures)
- inheritance relations to identify the relationship between an entity/behavioural model (base class) and one or more structural architectures (derived classes)
- aggregation relations to identify the sub-components in a structural architecture.

We have successfully used these concepts to build class diagrams for a variety of AMS soft-IP blocks. While the approach is quite straightforward, the resulting diagrams can be quite large and unwieldy. Further work is necessary to determine how to make better use of package diagrams in soft-IP library management.

Several methods have to be written to render these model classes synthesisable (we associate UML with Java for this development task, but there is no technical reason why the same concepts cannot be developed with other OO languages such as C++). We used this in the context of extending an existing AMS synthesis flow and as such have used it for low-level AMS blocks (TIA, amplifiers, filters and duplexers). XML was used in this respect to formulate soft-IP information and to store all generated numerical firm-IP. Future work will include the use of Pareto-sets to optimally reduce the amount of information stored, and data mining techniques to retrieve useful information. Application of this approach to more complex discrete-time and RF blocks is also a goal.

Acknowledgements

This work was partially funded by the European FP6 IST program under PICMOS FP6-2002-IST-1-002131, and by the French Rhône-Alpes region under OSMOSE (PRTP 2003-2005). The authors gratefully acknowledge discussions with Y. Hervé for valuable comments and insights.

References

CAR2004 C.T. Carr, T.M. McGinnity, L.J. McDaid: Integration of UML and VHDL-AMS for analogue system modelling. BCS Formal Aspects of Computing, **16**, 80; 2004.

CHA2004 V. Chaudhary, M. Francis, W. Zheng, A. Mantooth, L. Lemaitre: Automatic generation of compact semiconductor device models using Paragon and ADMS. Proc IEEE International Behavioral Modeling and Simulation Conference, 107; 2004.

DOB2003 A. Doboli, R. Vemuri: Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, **22** (11), 1504; 2003.

GIE2005 G. Gielen *et al.*: Analog and Digital Circuit Design in 65nm CMOS: End of the Road?. Proc. Design Automation and Test in Europe (DATE), 36; 2005.

HAM2003 M. Hamour, R. Saleh, S. Mirabbasi, A. Ivanov: Analog IP design flow for SoC applications. Proc. International Symposium on Circuits and Systems (ISCAS), IV-676; 2003.

HER2004 Y. Hervé, A. Fakhfakh: Requirements and verification through an extension of VHDL-AMS. Proc. Forum on Specification and Design Languages (FDL), 91; 2004.

OCO2003 I. O'Connor, F. Mieyeville, F. Tissafi-Drissi, G. Tosik, F. Gaffiot: Predictive Design Space Exploration of Maximum Bandwidth CMOS Photoreceiver Preamplifiers. Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS); 2003.

RIC2005 E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio: A SoC Design Methodology Involving a UML 2.0 Profile for SystemC. Proc. Design Automation and Test in Europe (DATE), 704; 2005.

TIS2003 F. Tissafi-Drissi, I. O'Connor, F. Mieyeville, F. Gaffiot: Hierarchical synthesis of high-speed CMOS photoreceiver front-ends using a multi-domain behavioral description language. Proc. Forum on Specification and Design Languages (FDL), 151; 2003.

TIS2004 F. Tissafi-Drissi, I. O'Connor, F. Gaffiot: RUNE: Platform for automated design of integrated multi-domain systems. Application to high-speed CMOS photoreceiver front-ends", Proc. Design Automation and Test in Europe (DATE) - Designer's Forum, 16; 2004.

VAC2003 A. Vachoux, C. Grimm, K. Einwich: SystemC-AMS requirements, design objectives and rationale. Proc. Design Automation and Test in Europe (DATE), 388; 2004.

VAN2005 Y. Vanderperren, W. Dehaene: UML 2 and SysML: An Approach to Deal with Complexity in SoC/NoC Design. Proc. Design, Automation and Test in Europe (DATE), 716; 2005.