Requirements specification
for
SystemC Analog Mixed Signal (AMS) extensions

Version 2.1
March 8, 2010

Bugs and Suggestions
Please report bugs and suggestions about this document to
http://www.systemc.org

Contributors

Karsten Einwich	Fraunhofer EAS/IIS Dresden
Christoph Grimm	Vienna University of Technology
Wolfgang Granig	Infineon Technologies
Gerhard Noessing	Infineon Technologies
Wolfgang Scherr	Infineon Technologies
Serge Scotti	STMicroelectronics
Martin Barnasconi	NXP Semiconductors
Giorgia Zucchelli	NXP Semiconductors
Alain Vachoux	Ecole Polytechnique Fédérale de Lausanne

History

Version	Editor	Update comments	Date
0.1	Martin Barnasconi	Initial version	29/11/2006
0.2	Martin Barnasconi	Update based on AMSWG review	17/01/2007
1.0	Martin Barnasconi	Approved internal version by AMSWG	05/03/2007
1.1	Martin Barnasconi	Introduction, numbering of requirements, short explanation for each requirement added	29/07/2008
1.2	Martin Barnasconi	Added Annex A. comparison table, reviewed version AMSWG f2f meeting October 15/16 2008.	21/10/2008
2.0	Martin Barnasconi	Approved version by AMSWG	25/11/2008
2.0	Martin Barnasconi	Version for public release	02/12/2008
2.1	Martin Barnasconi	Version for public release AMS 1.0 Standard	08/03/2010

Introduction

On February 15 2007, the Analog/Mixed-Signal (AMS) Working Group adopted the following charter:

“The Analog/Mixed-Signal (AMS) Working Group develops and recommends techniques and provides AMS extensions to the SystemC language standard, promoting the modeling of heterogeneous systems including both continuous-time and discrete-event behaviors at architectural level.
Main purpose is to enable specification, simulation, verification, implementation and evaluation of embedded heterogeneous systems containing digital, analog, mixed-signal and radio frequency functions, which are characterized in electrical or non-electrical domains.”

This document covers the requirements for the definition of the OSCI AMS standard, which is being developed by the OSCI AMS Working Group.

The AMS working group has identified several objectives for Analog/Mixed-Signal Modeling:

- Analyze and standardize extensions of SystemC with a semantic for describing:
  - non-conservative and conservative systems
  - continuous value/signal and time descriptions
  - electrical or non-electrical domains and quantities
- Extend SystemC with C/C++ class libraries for analog, mixed-signal and radio frequency language constructs, which require additional Models of Computation (MoC) and dedicated solvers. For these extensions, techniques for continuous value/signal/time representations and time synchronization mechanisms are explored.
- Disseminate the SystemC AMS extensions by providing examples and application-specific libraries, such as wired and wireless communication systems, automotive and sensor applications.

The OSCI AMS standard will focus on extending the SystemC standard with semantics and language constructs for AMS application modeling. We also provide a set of predefined primitives to model AMS components and systems at different levels of abstraction.

Part I of this document describes the motivation, AMS applications and use cases for analog/mixed-signal modeling. This informative section also explains the positioning and scope of the standardization of the SystemC AMS extensions within the design and simulation environment.

Part II defines the detailed implementation requirements for the OSCI AMS standard.

Acronyms and Terms

The OSCI AMS Working Group has created a separate document (OSCI AMS Glossary) that describes in detail the terms and acronyms used in this document. Please refer to that document.

Part I: Motivation, applications and use cases

Part I provides the motivation, applications and use cases for the SystemC AMS extensions.
The goal of this section is to outline the purpose of AMS modeling. The detailed implementation requirements, which are described in the next chapter, should be applicable for the application domains listed below and should satisfy the use case requirements defined in this section.

1 Why include AMS and/or RF in ESL design

The analog and/or RF content on a Systems-on-a-Chip, including its environment, will play an even more dominant role in the definition and verification of the embedded system as the number of RF interfaces to support (multi-mode) wireless and cellular standards will increase. Similarly, the number and the speed of analog serial interfaces both on-chip and off-chip will increase (e.g. NoCs) significantly. Also EMC/EMI related effects due to these very high-speed and RF interfaces will influence the performance of the integrated circuits and need to be modeled as early as possible in the design cycle.

Heterogeneous systems require an Integrated Design and Simulation Environment. In particular, the increasing complexity of sensor-actuator-systems in industry and automotive requires a modeling interface to non-electrical domains. Also modeling of highly complex heterogeneous systems requires different Models of Computation (MoC) to improve simulation speed.

2 Heterogeneous System IC Design

If one considers the currently accepted abstraction levels for a system, it is clear that there is no modeling language for the AMS domain at the architectural level (Fig 1).

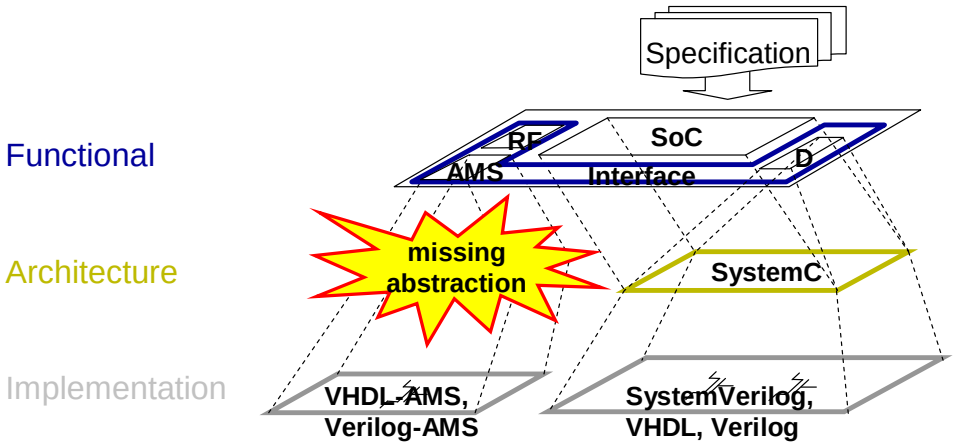

Fig 1: Missing AMS modeling language at architecture level

At architecture level, for purely digital systems, SystemC is being the widely accepted language. Therefore, it be logical to extend SystemC with an analog/mixed-signal capabilities by developing the SystemC AMS extensions. The combination of SystemC with the SystemC AMS extensions would provide a powerful simulation environment for the design and evaluation of heterogeneous systems at the architecture level. However, it would be also very advantageous to ensure that this complete SystemC environment can be easily coupled to both the functional and the implementation simulation domains to provide a common integrated design environment (IDE) for both system and circuit designers.

Fig 2 shows how the abstraction levels would look when the SystemC AMS extensions become available.

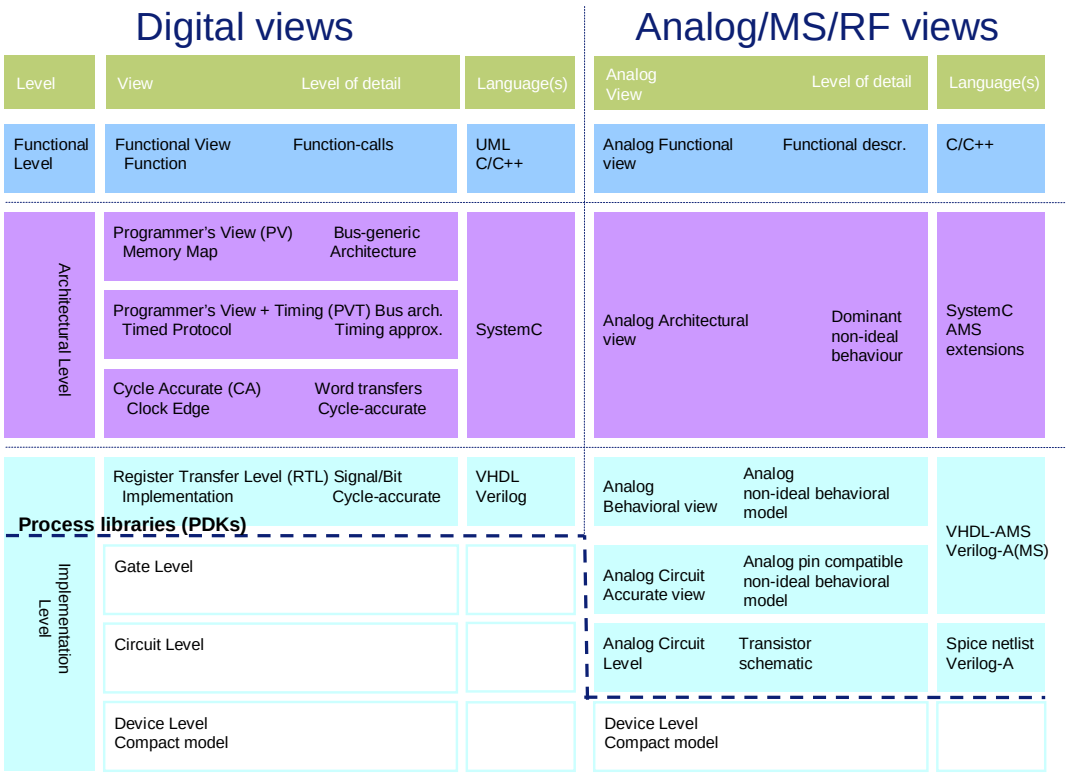

Fig 2: Abstraction levels for digital and analog/mixed-signal and RF views

As can be concluded from Fig 2, the SystemC AMS extensions will not be a replacement of circuit level simulation for analog, mixed-signal or RF circuitry (using Spice, FastSpice, ...) nor a replacement of behavioral modeling and HDL languages such as Verilog-AMS or VHDL-AMS.

3 Application domains

The SystemC AMS extensions will be applied in multiple application domains. This chapter gives an overview of the different application domains which need to be supported by methodology and technology libraries (see also Fig 7). Main focus is on modeling of the physical layer (PHY) or subsystems, within the context of the complete system architecture or application (e.g. ISO/OSI protocol stack, architecture HW/SW or application SW).

Wireless & Cellular communication systems

Examples of standards:

- WLAN (802.11a/b/g, 802.11n, ...) (see Fig 3)
- WiMAX (802.16d/e), WiBro
- Bluetooth, Zigbee, Wibree, ...
- GSM/EDGE, UMTS, WCDMA
- Wireless USB (Ultra Wide Band)
- Wireless sensor networks

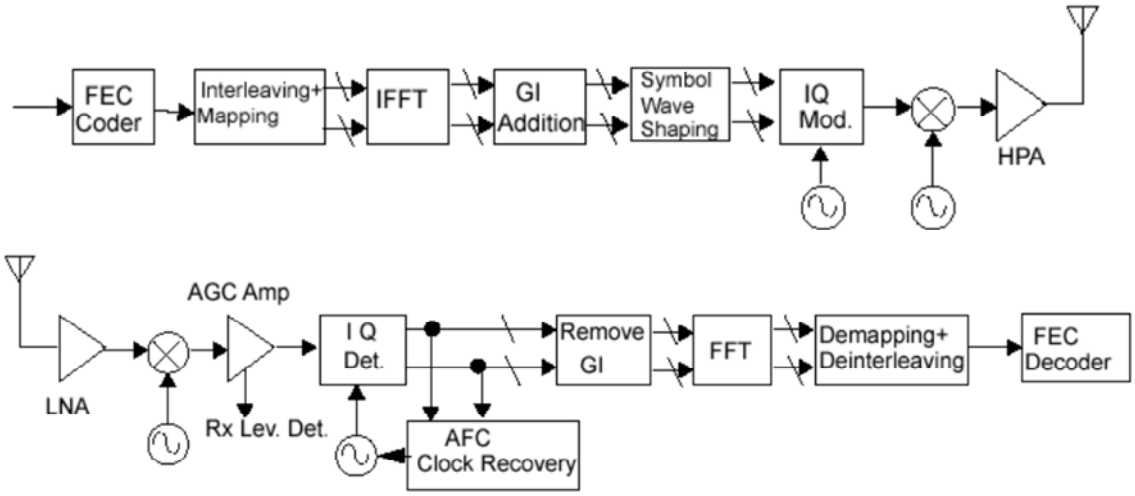

Fig 3: Wireless LAN Physical layer

Broadcast

Examples are:

- FM radio
- DVB-T, DVB-H (TVoM), ...

High-speed serial (physical) interfaces

Examples are PCI express, MIPI, USB, HDMI, ...

Automotive systems
Examples are:

- Power drivers (switching)
- Sensor interfaces
- Control systems (see Fig 4)

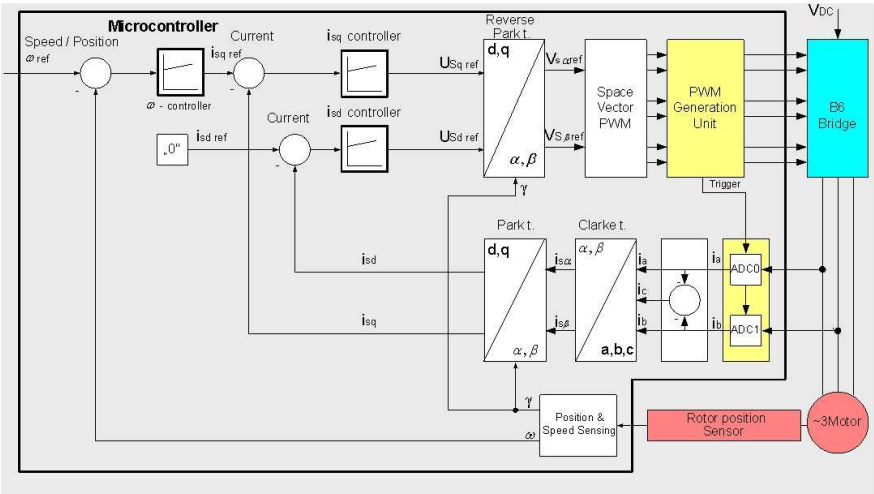

Fig 4: Power driver for Automotive system

Wire-line communication systems and interfaces
Examples are:

- POTS, ADSL, VDSL, SHDSL, … (see Fig 5)
- Line/channel models

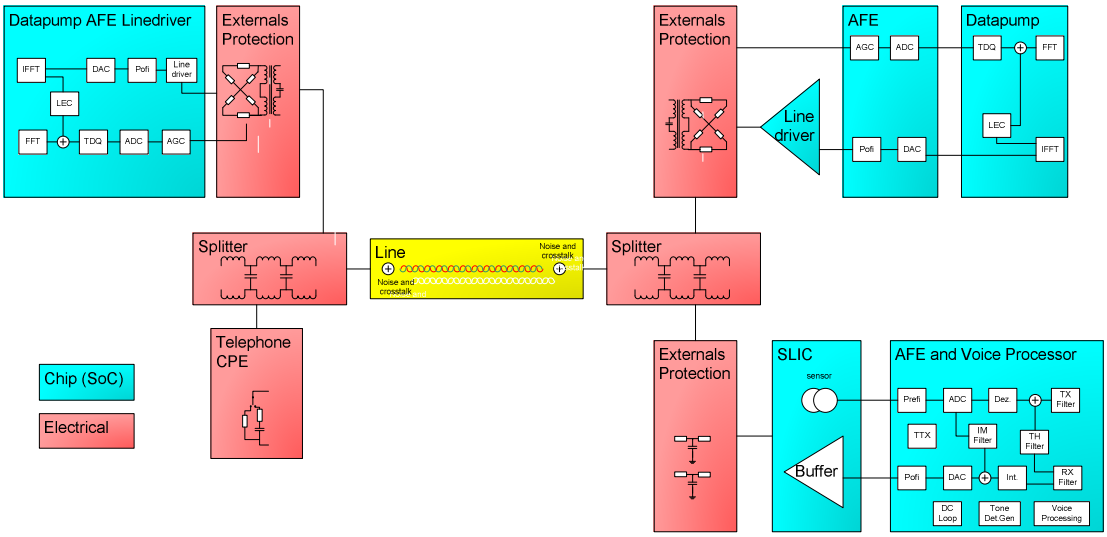

Fig 5: Wire-line communication system

Physical interfaces
Examples are:

- Sensors for magnetic field, pressure, temperature
- Power lines for actuators

4 Use Cases and Use models

The SystemC AMS extensions will be used for different tasks in the design process. This chapter gives an overview of the different use cases and use models in which these should be used.

4.1 Executable specification

- Dimensioning/specification of the system function
- Validation of the protocol (communication protocol, OSI/ISO layers, ...)
- Algorithm design (MPEG4, H264, Viterbi, OFDM puncturing, ...)
- Mapping functional specification onto a reference architecture

4.2 Architecture exploration

- Architecture/IP definition and partitioning (Busses, IP blocks, voltage domains, CPU elements, ...)
- Architecture technology trade-offs (SW, Digital HW, Analog/RF HW)
- Performance optimization, SW optimization, HW optimization

4.3 Integration validation

- IP verification including mixed-abstraction level (co-)simulation (e.g. Functional/Architectural co-verification)
- Digital HW IP Creation from C via SystemC (TLM) to VHDL/Verilog (RTL)
- Analog/RF HW IP Creation from C to SystemC AMS extensions to Verilog-AMS and transistor level
- IP verification and mixed-domain (co-)simulation (e.g. HW/SW co-verification)
- System validation and testing based on HW prototyping platform

4.4 Virtual prototyping

- Embedded Software/Firmware definition
- Middle layer (driver) and application SW development
- Software integration validation on the platform
- Network and protocol (OSI-model) simulation

5 Positioning of an environment for the SystemC AMS extensions

SystemC has become the accepted modeling language for digital architectures, while there is an increasing need to integrate complex AMS functions into these systems. In this context, the AMS extensions should be seen as an addition to the SystemC environment. It should address the following design needs:

Design complexity is increasing

- A higher level design abstraction is required for the analog domain which
  - must cope with (new) architectural elements (e.g. RF-baseband serial busses)
  - be able to undertake top-level system and functional verification
  - be able to provide architectural exploration of multiple SoC designs
- Need to develop reference architectures
  - to map different functional requirements onto a single AMS and/or RF architecture
  - to explore different AMS and/or RF architectures fitting to the functional requirement

Need for analog/digital co-design

- Protocol specification has become an integral part of the definition and implementation phase of the physical layer
- Increased interaction between analog and digital design implementation
  - digital assisted analog components require a common integrated design environment for system and circuit design

Need for (AMS) hardware/software co-design

- towards adaptive, reconfigurable systems and software defined radio
  - more MODEM functions implemented in firmware (DSP)
  - standardized interface requirements (API) to PHY function

Overcome EDA tool limitations

- There is no true architecture design tool available for AMS/RF system design & verification
- There is no interface between the AMS/RF architectural design and the digital architecture design approach
- There is no agreed system modeling language to include AMS/RF architectures

Platform for improved AMS and/or RF model exchange

- Documented and open model interfaces independent of a single tool vendor
- Based on C/C++
- IP protection capability (compiled code)
- Enable virtual prototype (executable spec) approach

5.1 Design and simulation environment

SystemC extended by AMS capabilities should become available as a complete simulation environment for modeling at architectural level and should be embedded in the overall design environment, as shown in Fig 6.

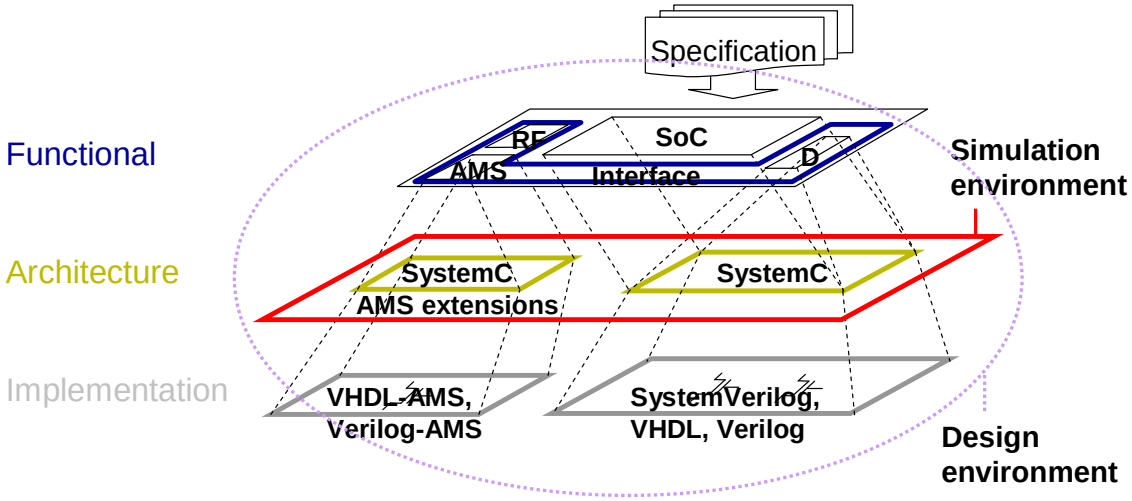

Fig 6: Design and simulation environment for SystemC and the AMS extensions

It is essential that the overall design environment should have the capability of simulating a complete system, with components modeled in all possible abstractions.

5.1.1 Design environment requirements

5.1.1.1 User-friendly environment for the Architectural evaluation of Heterogeneous Systems

- Need for a flexible and adaptable framework is favorable over a (complex) specialized framework

5.1.1.2 System design through all abstraction levels and behavioral views

- Environment enabling descriptions with multiple abstraction levels
- System design with hierarchical structure

5.1.1.3 Library consistency with different abstraction levels / tools in the flow and simulation domains

- Libraries based on standard languages or specific library interfaces
- Availability of source code of the models

5.1.1.4 Model re-use, develop behavioral models for bottom-up verification

- Non-linear frequency-dependent models
- Interface to third-party behavioral languages

5.1.2 Simulation Environment Requirements

5.1.2.1 Evaluation of system performances at all abstraction levels

- Link between abstraction levels and MoCs enabling hierarchical simulation
- Heterogeneous system environment: specify MoC appropriate for the analysis of each task
- Simulation of system specs to generate/evaluate sub-blocks specs

5.1.2.2 Analysis of synchronization and timing issues for system performance optimization

- Cope with fast correction loops
- Mixed-signal simulation: synchronized digital (discrete-event) solver and analog (continuous-time) solvers

5.1.2.3 Impact of implementation specific non-linear impairments on the system

- Co-simulation of architecture level and implementation

5.1.2.4 Class libraries for SystemC AMS extensions should comply to the SystemC build environment

- Support the same operating systems and platforms
- Support the same versions of compiler/linker tools (e.g. gcc, etc)

Part II: AMS Implementation requirements

The definition of the SystemC AMS extensions should be seen as an addition to the SystemC standard, without any change to the SystemC standard itself or its reference implementation. Also a potential proof-of-concept implementation should use Standardized IEEE Std 1666-2005 compliant features of SystemC, and should not rely on implementation details of the SystemC implementation.

The objective is to build a single executable of the architectural description, including a synchronized digital discrete-event solver and a continuous-time analog solver. Therefore, the SystemC AMS extensions should comply to the layered SystemC structure defined in the Language Reference Manual (LRM) Annex A (see also red colored elements in Fig 7 below), providing an efficient object-oriented extension of IEEE Std 1666-2005.

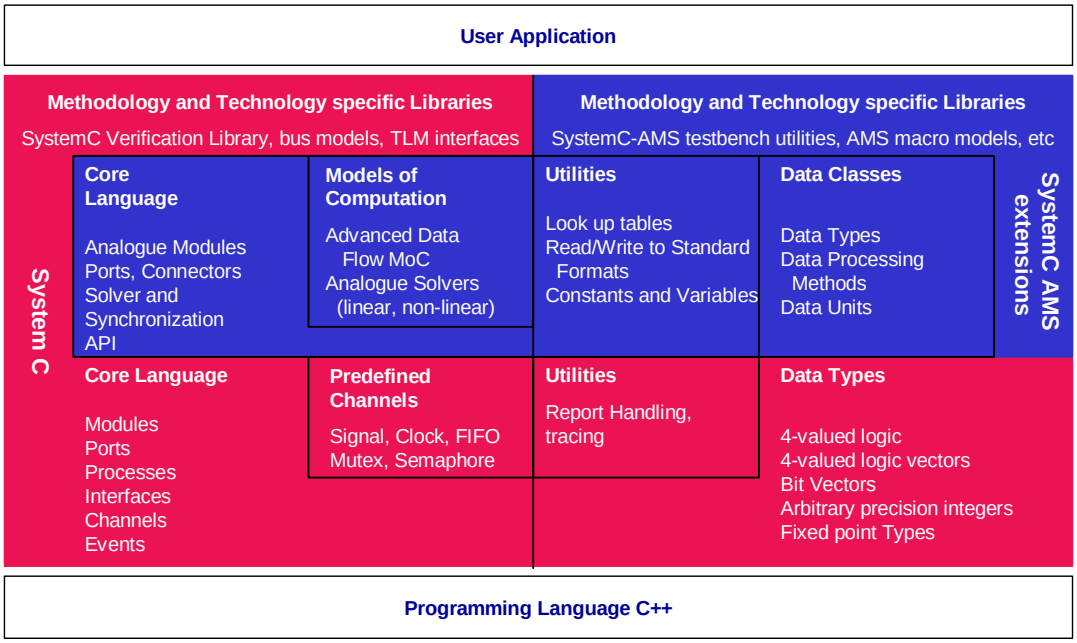

Fig 7: SystemC and the AMS extensions

6 Implementation requirements

The implementation requirements are grouped in the following domains:

- Methodology and Technology Specific Libraries
- Core Language
- Models of Computation and Solvers
- Utilities
- Data classes
- Application Programming Interface (API)

The implementation requirements have been prioritized with High (H), Medium (M) and Low (L) priority, according to the following definition:

High (H)	Essential requirement. Needed as basic infrastructure (foundation) for the SystemC AMS extensions, to show functionality and its capabilities
Medium (M)	Valuable requirement, but not essential to show basic functionality. Should be considered in next LRM release (if not covered already)
Low (L)	"Nice-to-have" requirement. Not essential as part of the AMS Standardization and/or could be defined/standardized elsewhere

6.1 Methodology and Technology Specific Libraries

6.1.1 Refinement methodology

6.1.1.1 Language construct supporting a design refinement methodology (H)

The language constructs and modeling formalism introduced by the SystemC AMS extensions should facilitate a design refinement methodology to define AMS behavior at different levels of design abstraction. The level of design refinement depends on the use case in which the models are used. A top-down design and bottom-up verification methodology should be supported where mixed abstraction-level modeling is facilitated.

6.1.2 Basic communication library

6.1.2.1 open source library of non-differentiating system models and IP to facilitate the creation of test-benches and verification suites (L)

Library of higher level models with algorithmic or procedural descriptions, to develop reference architectures and/or test-benches at system level. Examples of such models are down-converters, filters, modulators and demodulators, etc.

6.1.3 Basic primitives library

Library of predefined primitive modules as foundation to create a variety of applications or user-specific models. The basic set of models should offer access to all required computational techniques available in the implementation.

6.1.3.1 Resistors, inductors, capacitors (H)

Predefined models of electrical linear primitives for macro modeling to be able to construct simple electrical networks for filters, loads, matching circuits, etc.

6.1.3.2 Stimulus generators, voltage sources, current sinks, etc (H)

Predefined models for sources and sinks to create analog signals.

6.1.3.3 Basic non-electrical types and interfaces, for example, for actuator and sensor modeling (M)

Predefined models of non-electrical primitives see also requirement 6.1.4.1.

6.1.3.4 Ability to describe linear dynamic functions and non-linear static functions (H)

Define model semantics and predefined models for embedded linear dynamic functions which can be instantiated in a time- and frequency-domain simulation. Support of linear transfer functions and state-space functions should be solved in the continuous-time domain. Non-linear static functions should be supported for time-domain simulations.

6.1.3.5 Non-linear dynamic models, e.g. diode, simple bipolar model, simple MOS model (M)

Predefined models of electrical non-linear primitives for macro modeling to be able to construct simple electrical networks including only the dominant non-linear component(s), e.g. the final output stage in a power amplifier.

6.1.3.6 Consistent naming conventions with implementation languages (e.g. Verilog-AMS, VHDL-AMS) – where applicable (M)

Use same naming conventions as used in other behavioral languages to improve readability of netlists and circuits for users familiar with these other languages.

6.1.3.7 Basic pre- and post-processing library (L)

Library of higher level models with algorithmic or procedural descriptions, for signals generation and signal analysis. Will be used in combination with the architecture design and test-bench models defined in requirement 6.1.2.1.

6.1.3.8 Performance monitors (e.g. EVM, BER) (L)

Library of higher level models with algorithmic or procedural descriptions for signal analysis such as Bit-Error-Rate (BER) or Error-Vector Magnitude (EVM)

6.1.3.9 Post-processing functions, i.e. FFT (L)

Library of higher level models with algorithmic or procedural descriptions for special computational tasks, like Fast Fourier Transforms (FFT).

6.1.3.10 Output (pseudo) real-time simulation values (M)

Output simulation results using streams, which can be picked-up by other software for immediate processing or analysis. Example is to output (stream) simulation results directly to vector signal analyzer software [1, 2].

6.1.4 Extend the domain of applicability

To support heterogeneous system design to a full extend, extensions to non-electrical domains is essential. Also simulation should be possible in time and frequency domain.

6.1.4.1 Electromechanical, fluidic, thermal domain behavior (M)

Examples of domain which need to be supported to include components like MEMS, and temperature sensors.

6.1.4.2 Allow time- and frequency-domain behavior (H)

Support both time- and frequency descriptions for all Models of Computation.

6.2 Core Language

6.2.1 Single model definition for different simulation domains

To support both discrete-time and continuous-time descriptions, different Models of Computation are introduced. Objective is to have unified language constructs and views for the user/application which are detached from the underlying execution semantics.

6.2.1.1 Model of Computation independent assignments and methods (M)

Avoid different assignments and methods for the predefined primitives developed for each model of computation.

6.2.1.2 Introduce common ports and nodes or channels for conservative and non-conservative elements (M)

The concept of ports and channels should be applied for all primitives developed for each Models of Computation and should be the same where applicable.

6.2.2 Modules, ports and channels to represent hierarchy

6.2.2.1 IEEE Std 1666-2005 compliant modules should be suitable for use in a SystemC AMS extensions environment without modifications. (H)

Module hierarchy in an AMS application should use concepts as defined in IEEE Std 1666-2005.

6.2.2.2 Define converter modules/ports between different Models of Computation (H)

In order to connect modules defined in/for different Models of Computation, connect modules or ports should be introduced to facilitate an easy way to link these computational domains together.

6.2.3 Synchronization mechanisms

6.2.3.1 Support of asynchronous events (M)

In order to support flexible interaction (in time) between threads, such as asynchronous events, a modeling approach based on fixed time steps has the disadvantage that these asynchronous events are ‘missed’. An asynchronous event mechanism should be introduced, but also requires the implementation of variable time stepping (see requirement 6.2.3.2)

6.2.3.2 Variable time steps (M)

Introduce variable time steps for the synchronization between the discrete-time and continuous-time analog solvers.

6.2.3.3 Usage of channels as communication and synchronization mechanism (H)

In line with the foundations of SystemC, the use of channels as communication / synchronization mechanism is strongly recommended.

6.2.3.4 Pass values from the discrete-event domain to the analog Models of Computation (H)

The digital control must send information to the continuous solver, for instance, to switch equations: the power down signal is usually driven by a digital control, and the analog solver must take it into account.

6.2.3.5 Pass analog "events" to the discrete-event domain (H)

The analog solver must create events not yet in the event queue of the digital (discrete-event) simulator: for instance, a threshold crossing can create a "digital" event, to be taken into account by the discrete-event simulator.

6.3 Models of Computation and Solvers

6.3.1 Non-conservative domain

6.3.1.1 Support data/signal flow descriptions (H)

Dataflow and Signal flow Models of Computations should be supported.

6.3.1.2 Support static scheduling for efficient algorithm design (H)

Introduce principles of Synchronous Data Flow MoC as this modeling paradigm is often used for algorithm/DSP design.

6.3.1.3 Support for equivalent baseband and pass-band models (M)

To gain simulation efficiency, wide-band, high-frequency models could be abstracted to narrow-band, low-frequency models or equivalent baseband or pass-band models. Different concepts are available for this [3]. The basic primitives introduced by the SystemC AMS extensions should allow this type of modeling.

6.3.1.4 Support dynamic changing behavior (M)

Support for dynamically changing behavior like changed data/sample rates of adaptive and reconfigurable systems.

6.3.1.5 Support multi-rate simulations (H)

Allow multiple sampling domains and data rates, even combining them into a single stream.

6.3.2 Conservative domain

6.3.2.1 Ability to embed dedicated solvers (M)

The SystemC AMS extensions should allow user-defined extensions of dedicated solvers (see also API, section 5.3).

6.3.2.2 Solver for linear switching networks (H)

Introduce solver dedicated for linear switching networks.

6.3.2.3 Solver for strongly non-linear and stiff DAE systems (M)

Introduce solver dedicated for strongly non-linear and stiff DAE systems.

6.3.2.4 Large-signal non-linear analog/RF analysis (M)

Support of large-signal non-linear multi-tone analysis.

6.3.2.5 Small-signal non-linear analog/RF analysis (M)

Support of small-signal non-linear multi-tone analysis.

6.4 Utilities

6.4.1 File formats and variables

6.4.1.1 Support table-based models (look-up tables) (M)

Use of multidimensional tables with interpolation (e.g. AM/AM, AM/PM curve of power amplifiers).

6.4.1.2 Read/Write to Standard Formats (M)

Support to read/write to multidimensional tables and S-parameter files (e.g. Touchstone format).

6.4.1.3 Constants and Variables (M)

Include methods to accommodate shared variables and constants.

6.4.2 Tracing and Report Handling

6.4.2.1 Tracing of AMS signals (H)

Tracing functionality similar to SystemC, to trace AMS/RF electrical and non-electrical signals.

6.4.2.2 Support AMS file format (H)

Creation of standard file formats which can be easily read by post-processing tools and viewers used by the AMS community (e.g. Matlab, VCD format, comma-delimited-file, etc).

6.5 Data Classes

6.5.1 Data Types

6.5.1.1 Support standard (SystemC) data types (H)

Support standard data types such as integer, floating point, etc.

6.5.1.2 Introduce analog-specific data types (H)

Introduce complex numbers.

6.5.1.3 Introduce (efficient) data processing methods (H)

Support vector and matrix processing (arbitrary type and size).

6.5.1.4 Data Units (M)

Usage of the International System of Units (SI).

6.6 Application Programming Interface (API)

6.6.1 Compatibility

6.6.1.1 Compatible with IEEE Std 1666-2005 (M)

The API definition should be defined in terms of class definitions, in a similar way as done for SystemC (part of the Language Reference Manual).

6.6.2 Co-simulation

6.6.2.1 Permit co-simulation environment for multi-domain simulation (M)

Ability to link the SystemC AMS extensions to other simulation frameworks, or use the SystemC AMS extensions as simulation backplane by incorporating other computation processes.

6.6.2.2 Easy plug-in for analog solvers (M)

Should provide a simple mechanism to allow analog solvers to be modified or replaced.

The comparison table in Annex A in this document gives an overview of all the requirements including their priorities. In addition, it shows what is covered in the AMS 1.0 standard and what is defined in the AMS Language Reference manual. Some requirements are labeled with Not Applicable (N/A); in this case the requirement is not considered as essential part of the standardization and AMS LRM definition itself.

Requirements summary

References

[1] Agilent Technologies, 89600 Series Vector Signal Analysis Software, www.agilent.com/find/89600
[2] Rohde & Schwarz, R&S®GX430 PC-Based Signal Analysis and Signal Processing, http://www2.rohde-schwarz.com/en/products/radiomonitoring/product_categories/Signal_Analysis/GX430.html
[3] Gerd Vandersteen, Piet Wambacq, Stephane Donnay, Yves Rolain and Wolfgang Eberle "FAST - an efficient high-level dataflow simulator of mixed-signal front-ends of digital telecom transceivers," chapter in book "*Low-power design techniques and CAD tools for analog and RF integrated circuits*", edited by Piet Wambacq, Georges Gielen and John Gerrits, Kluwer Academic Publishers, 2001.

Annex A Comparison: requirements specification and AMS 1.0 standard

The table below gives an overview of the requirement specification and the definition as part of the SystemC AMS Language Reference Manual.

Req ID	Domain	Sub-domain	Requirement	Prio	AMS 1.0 standard	Description
6.1.1.1	Methodology and Technology Specific Libraries	Methodology	Language construct supporting a design refinement methodology for different use cases	H	✓	Discrete-time and continuous-time abstractions defined supporting conservative and non-conservative system description. Namespaces defined for the different MoCs.
6.1.2.1		Basic communication library	Open source library of non-differentiating system models and IP to facilitate the creation of test-benches and verification suites	L	N/A	User-defined library can be created using TDF primitive modules. Not necessarily a standardization topic.
6.1.3.1		Basic primitives library	Resistors, inductors, capacitors	H	✓	Electrical primitives available in ELN
6.1.3.2			Stimulus generators, voltage sources, current sinks, etc	H	✓	Available as predefined primitives for ELN and LSF. Advanced stimuli can be made with TDF primitives
6.1.3.3			Basic non-electrical types and interfaces, for example, for actuator and sensor modeling	M	✗	Non-electrical physical domains not part of AMS 1.0 standard (see also 6.1.4.1)
6.1.3.4			Ability to describe linear dynamic functions and non-linear static functions	H	✓	Linear transfer functions supported in TDF and LSF.
6.1.3.5			Non-linear dynamic models, e.g. diode, simple bipolar model, simple MOS model	M	✗	Non-linear solver not included in version 1.0. Static non-linear behavior available in TDF
6.1.3.6			Consistent naming conventions of primitives with implementation languages (e.g. Verilog-AMS, VHDL-AMS) – where applicable	M	✓	Aligned with Verilog(-AMS), VHDL(-AMS), Spice and SystemC naming conventions
6.1.3.7			Basic pre- and post-processing library	L	N/A	User-defined library can be created using TDF primitive modules. Not necessarily a standardization topic.
6.1.3.8			Performance monitors (e.g. EVM, BER)	L	N/A	User-defined library can be created using TDF primitive modules. Not necessarily a standardization topic.

Req ID	Domain	Sub-domain	Requirement	Prio	AMS 1.0 standard	Description
6.1.3.9			Post-processing functions, i.e. FFT	L	N/A	User-defined library can be created using TDF primitive modules. Not necessarily a standardization topic.
6.1.3.10			Output (pseudo) real-time simulation values,	M	✓	Streaming of trace files supported (see also 6.4.2.2)
6.1.4.1		Extend the domain of applicability	Electromechanical, fluidic, thermal domain behavior	M	✗	Non-electrical physical domains not part of AMS 1.0 standard (see also 6.1.3.3)
6.1.4.2			Allow time- and frequency-domain behavior	H	✓	Time and frequency analysis supported
6.2.1.1	Core Language	Single model definition for different simu-lation domains	Model of Computation independent assignments and methods	M	✓	Classes, methods and functions aligned across the different MoCs using namespaces
6.2.1.2			Introduce common ports and nodes or channels for conservative and non-conservative elements	M	✓	Ports and channels introduced in different namespaces
6.2.2.1		Modules, ports and channels to represent hierarchy	IEEE Std 1666-2005 compliant modules should be suitable for use in a SystemC AMS extensions environment without modification.	H	✓	IEEE Std 1666-2005 needs no modification. AMS extensions will rely on hierarchy concepts defined in IEEE Std 1666-2005.
6.2.2.2			Define converter modules/ports between different Models of Computation	H	✓	Converter modules/ports available to make a connect between MoCs
6.2.3.1		Synchronization mechanisms	Support of asynchronous events	M	✗	Synchronous synchronization mechanism implemented. Fall-back to SystemC for asynchronous event handling
6.2.3.2			Variable time steps	M	✗	Fixed time step defined in AMS 1.0 standard
6.2.3.3			Usage of channels as communication and synchronization mechanism	H	✓	Channels and nodes introduced. Communication defined by the MoC
6.2.3.4			Pass values from the discrete-event domain to the analog Models of Computation	H	✓	The value is read at the next sampling time-point defined by TDF.
6.2.3.5			Pass analog “events” to the discrete-event domain	H	✓	Note: Analog “events” (zero-crossings, thresholds) can only reported back to SystemC at fixed time steps.
6.3.1.1	Models of Computation and Solvers	Non-conservative domain	Support data/signal flow descriptions	H	✓	Timed Data flow (TDF) and Linear Signal Flow (LSF) introduced
6.3.1.2			Support static scheduling for efficient algorithm design	H	✓	TDF is based on principles of well known synchronous data flow MoC

Req ID	Domain	Sub-domain	Requirement	Prio	AMS 1.0 standard	Description
6.3.1.3			Support for equivalent baseband and pass-band models	M	N/A	User-defined library for baseband models can be created using TDF primitive modules.
6.3.1.4			Support dynamic changing behavior	M	✗	Static scheduling defined in AMS 1.0 standard.
6.3.1.5			Support multi-rate simulations	H	✓	Multi-rate data flow supported.
6.3.2.1		Conservative domain	Ability to embed dedicated solvers	M	✗	See API definition, 6.6.2.2
6.3.2.2			Solver for linear switching networks	H	✓	Linear solver defined
6.3.2.3			Solver for strongly non-linear and stiff DAE systems	M	✗	No generic non-linear solver for electrical networks defined in AMS 1.0 standard
6.3.2.4			Large-signal non-linear analog/RF analysis	M	✗	No harmonic-balance-like solver defined in AMS 1.0 standard
6.3.2.5			Small-signal non-linear analog/RF analysis	M	✗	No non-linear AC solver defined in AMS 1.0 standard
6.4.1.1	Utilities	File formats and variables	Support table-based models (look-up tables)	M	N/A	Use standard C++ capabilities to read/write from files
6.4.1.2			Read/Write to Standard Formats (e.g. Touchstone)	M	N/A	Use standard C++ capabilities to read/write from files
6.4.1.3			Constants and Variables	M	N/A	Standard #include for variables can be used
6.4.2.1		Tracing and Report Handling	Tracing of AMS signals	H	✓	Tracing mechanism defined for multi-rate.
6.4.2.2			Support AMS file format	H	✓	Tabular trace file format defined for VCD and tabular file format.
6.5.1.1	Data Classes	Data Types	Support standard (SystemC) data types	H	✓	Template classes support standard SystemC data types
6.5.1.2			Introduce analog-specific data types (complex numbers)	H	✓	Class implemented for complex numbers
6.5.1.3			Introduce (efficient) data processing methods (for vectors, matrices)	H	✓	Class implemented for vectors and matrices
6.5.1.4			Usage of the International System of Units (SI).	M	N/A	Standard #include for variables can be used.
6.6.1.1	API	Compatibility	Compatible with IEEE Std 1666-2005	M	✓	AMS class library derived from SystemC classes where possible
6.6.2.1		Co-simulation	Permit co-simulation environment for multi-domain simulation	M	✗	No co-simulation API defined in AMS 1.0 standard

Req ID	Domain	Sub-domain	Requirement	Prio	AMS 1.0 standard	Description
6.6.2.2			Easy plug-in for analog solvers	M	✘	No simulation “backplane” defined in AMS 1.0 standard

Legend comparison table

✓ Can be used in an applications using the SystemC and SystemC AMS extensions and/or defined in AMS LRM

✘ Can not be used in an application and/or AMS LRM definition missing

N/A Not applicable for AMS Standardization

Priority

H High, essential requirement. Needed as basic infrastructure (foundation) for the SystemC AMS extensions, to show functionality and its capabilities.

M Medium, valuable requirement, but not essential to show basic functionality. Should be considered in next LRM release (if not covered already)

L Low, “nice-to-have” requirement. Not essential as part of the AMS Standardization and/or could be defined/standardized elsewhere