Virtual Prototyping, Verification and Validation Framework for Automotive Using SystemC, SystemC-AMS and UVM-SystemC

Yao LI*, Zhi WANG*, Marie-Minerve LOUËRAT*, François PECHEUX*, Ramy ISKANDER*,
Philippe CUENOT†, Martin BARNASCONI‡, Thilo VÖRTLER§, Karsten EINWICH§
*LIP6 Laboratory, University Pierre & Marie-Curie, Paris, France
†Continental Automotive France SAS, Toulouse, France
‡NXP Semiconductors, Eindhoven, Netherlands
§Fraunhofer IIS, Design Automation Division EAS, Dresden, Germany
{*Email: Yao.Li@lip6.fr*}

***Abstract*—Electronics for automotive systems are characterized by an increasing complexity and a more and more tighter interaction between analog, digital hard- and software. They consist of a huge portion of embedded software, which executes on large (regarding the number of devices) digital subsystems and controlling and assisting analog devices. Furthermore, automotive electronics deeply interacts with the non-electronic environment. Due to this increasing complexity of automotive electronic devices the verification becomes more and more challenging. Today, existing verification methodologies are mostly focused on pure digital devices and are completely decoupled from analog verification. Additionally, the existing methodologies tend to focus on implementation level verification. They do not meet the requirements of state of the art automotive applications.**

Based on the powerful 3-uple (Universal Verification Methodology (UVM) principles, SystemC, SystemC AMS extensions), this paper shows how the principles of the new UVM methodology can be soundly enhanced to offer to the test designer a flexible framework for the virtual prototyping of multi-discipline testbenches that supports both digital and Analog Mixed-Signal (AMS) at the architectural level. The paper clearly details the architecture of the reusable verification IPs and the synchronization mechanisms used to simultaneously manage test at a high level of abstraction (test sequences) and communicate with the AMS DUT at the pin level. The presented techniques are applied to two operational case study, a full-fledged programmable filter testbench, and then applied to a smart power supply module on an automotive Electronic Control Unit.

I. Introduction

Virtual prototyping [1] is one of the major achievements of the past ten years. With certain success, semi-conductors companies have worked on effective solutions to model their products at the architectural level of abstraction, enabling early embedded software development. At the same time, the virtual platforms have followed More Than Moore laws. Today, manufactured products are heterogeneous in nature, for they integrate on the same die digital, analog, RF and multi-physics components.

The verification and validation of such integrated heterogeneous systems have become a real challenge today for semiconductor companies, fab-less design houses and system developers as their intrinsic complexity keeps growing: more embedded functions per μm^2 and behavior heterogeneity resulting from the combination of AMS/RF domains with the digital hardware and software domains [2].

In practice, the main issue is to dramatically improve the design efficiency and quality of such complex systems, by developing methods and tools that can tackle the design challenges in the next generations of technologies, taking into account the heterogeneous integration of different functions. It is now the time for system verification, that covers simulation-based system analysis before tape-out and system validation, which focuses on the analysis of the physical implementation using test & measurement equipment as soon as silicon is available, by defining an unified system-level verification methodology for heterogeneous products, by specifying an efficient reuse strategy for verification IP, across and inside companies for different product generations, and by defining a clear path from verification IP to validation IP, to bridge the gap between verification and validation [3].

There is actually a need for a unified system-level verification kit that could bring new methods, language extensions and tools dedicated to heterogeneous systems to continuously prove the architectural integrity and system specification throughout the whole product creation process. In order to verify and validate the architecture and specification of these systems, an end-to-end verification approach is required, based on the creation of an "executable specification" that includes the essential verification components and IP necessary to describe stimuli, test-benches, analysis algorithms, etc.

Additionally this verification kit must follow or extend known standards: new modeling methods for SystemC-based [4] verification, interoperability of tools and reuse techniques using IP-XACT [5], and using a common programming interface to test and measurement equipment through the OVM [6], AVM [7] or UVM [8] standards.

The paper is organized as follows. Section II describes the related work in the areas of methodologies and verification tools. Section III gives an overview of the Virtual Prototyping,

Verification and Validation Framework for Automotive. The UVM-SystemC concepts are introduced in Section IV. In Section V we apply our framework to a programmable filter test case, and then to a smart power supply module on an automotive Electronic Control Unit. The simulation results are shown in Section VI. Finally, Section VII draws some conclusions and future directions.

II. State of the art

As shown in Fig.1, various verification methodologies for digital systems have been developed and promoted by EDA vendors over the last decade, evolving towards UVM as part of an industry supported standardization effort in Accellera. The industry-recognized Universal Verification Methodology (UVM) [8] support design and verification engineers with an open-source class library to create verification components and models for digital test-benches. A standardized methodology which allows reuse and portability of Verification IP based on this approach. This verification methodology will become language and tool independent, supporting the creation of Verification IP in SystemVerilog [9], SystemC [4] and *e* [10].

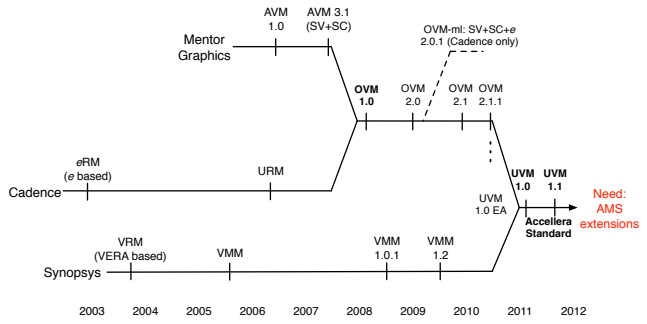

Fig. 1. *Evolution of Verification Methodologies*

However, all these initiatives do not fully comply with the methods defined in the UVM standard, primarily because they are built on the former AVM [7], OVM [6], VMM [11], technologies, were consolidation into a single standard resulted in major changes. As a consequence, the user has to deal with the incompatibilities related to simulation semantics and language constructs. Especially the move from OVM to UVM significantly changed the way how components deal with the phasing mechanism and how the end-of-test is managed. To avoid legacy concepts and constructs in modern test benches, migration to UVM standard compatible implementations should be encouraged. Furthermore, SystemC as the currently preferred language standard for system level design, needs to be extended with advanced verification concepts.

Besides, Virtual Prototyping (VP) technique has been studied and implemented in recent years in engineering design [1], [12], these recent developments have considerable potential. With the goal of replacing physical prototypes, VP has a great potential to improve the current product development process. A lot of development for utilizing VP is being realized by automotive and aerospace companies. For example, in [13], they investigate the steps needed to apply virtual reality (VR) for virtual prototyping (VP) to verify assembly and maintenance processes.

III. 3VFA Framework (Virtual Prototyping, Verification and Validation Framework for Automotive)

The proposed virtual prototyping, verification and validation framework for multidiscipline automotive designs is detailed in Fig. 2. It is entirely based on the SystemC C1 [4], its analog mixed-signal extensions called SystemC-AMS C2 [14], and the new verification library UVM-SystemC C3. In 3VFA, SystemC is used to model and simulate the digital parts of the Design Under Test (DUT), such as the processors and their associated caches, the memory banks, the interconnect networks, I/O peripherals and controllers, etc. SystemC-AMS is built on top of SystemC and proposes additional constructs which introduce new execution semantics (synchronized data-flow, multiple models of computation) and system-level methodologies to design and verify mixed-signal systems. UVM-SystemC, based on the standardized UVM library and extended for AMS, introduces a true reusable system-level verification methodology that can address software, digital hardware and AMS hardware altogether.

A. High Level Modeling of Digital System (SystemC)

SystemC is a set of C++ classes which provide an event-driven simulation interface that enables a designer to simulate concurrent processes typically found in microelectronics. It is a Electronic System-Level (ESL) modeling language used for system-level modeling, architectural exploration, performance modeling, software development, and functional verification. SystemC supports many models of computation, from synthesizable RTL to transaction-level modeling (TLM) [15]. From a pragmatic viewpoint, a SystemC module is an inherited C++ class that contains input and output ports (declared as data members) and one or several SC_METHOD or SC_THREAD that actually represent as a process the behavior of the module. SC_METHODs represent an elegant way to describe simple sequential and autonomous digital algorithms, while SC_THREADs, with the ability to insert wait() statements, allow to stop and resume a process in order to soundly synchronize with other processes. To provide a good trade-off between speed and accuracy, SystemC designers can use specific kinds of modeling like Communicating Synchronous Finite State Machines [16] to represent digital behavior as a set of synchronized SC_METHODs. Eventually, the SystemC models of RAM ④ can be powerful enough to actually load the contents of a binary file, and hence feed the processors with appropriate instructions and data. In 3VFA structure, the well-known GNU GCC tool chain can be used to generate the code that will be used both in simulation and physical prototype ⑤.

B. High level Modeling of Mixed System (SystemC-AMS)

The SystemC AMS [17] extensions are built on top of the SystemC language standard and define additional language

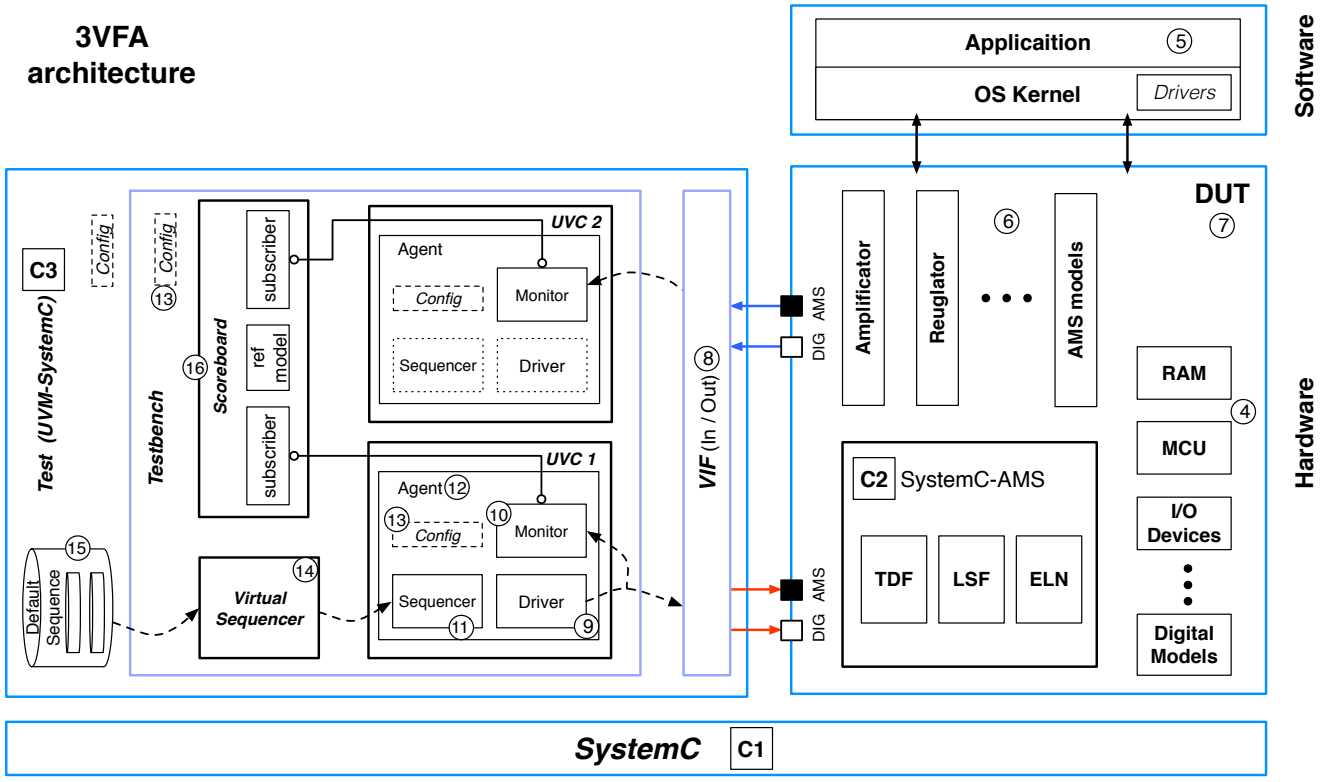

Fig. 2. *The proposed virtual prototyping, verification and validation framework for automotive*

constructs, which introduce new execution semantics and system-level modeling methodologies to design and verify mixed-signal systems. The SystemC AMS extensions provide a framework for functional modeling [18], integration verification, and virtual prototyping of *Embedded Analog / Mixed-Signal Systems*. This language allows to create discrete-time and continuous-time models at different levels of abstraction. The SystemC AMS extensions provide three different models of computation: Timed Data Flow (TDF), Linear Signal Flow (LSF), and Electrical Linear Networks (ELN). For the AMS models ⑥, such as, amplifier, regulator, the Timed Data Flow model of computation is used. TDF is a discrete-time modeling style, which considers data as signals sampled in time. These signals are tagged at discrete points in time and carry discrete or continuous values. Besides, TDF can be used with great efficiency to model complex non-conservative behaviors.

C. Managing Verification Complexity (UVM-SystemC)

UVM-SystemC is the adaptation of the SystemVerilog standard methodology to SystemC for verifying integrated circuit designs. The UVM-SystemC C++ class library brings much automation to SystemC such as the generation of reusable sequences and powerful means to configure testbenches and exchange stimuli between components. Its main objective is to provide a formalized and structured test infrastructure and reduce time and complexity associated with DUT verification and validation.

The centerpiece of a UVM scenario is the mixed-signal DUT ⑦ which is composed of digital and AMS parts, some of them eventually executing software. The DUT is connected to the rest of the testbench by means of the interface mechanism ⑧ proposed by SystemC that brings huge modularity and effectiveness. The interfaces are connected either to UVM driver ports (**uvm_driver**, that actually stimulate the DUT) ⑨ or monitor ports (**uvm_monitor**, that actually get and accumulate samples coming from the outputs of the DUT) ⑩. UVM defines an appropriate request/response protocol to manage the transfers between the sequencer ⑪ and the driver. The 3-uple (sequencer, driver, monitor) represents a **uvm_agent** ⑫. UVM defines an **uvm_environment** as a collection of consistent **uvm_agents** (i.e. acting in a coordinated manner and linked to a specific part/functionality of the DUT interface). UVM defines the virtual sequencer object ⑭ which is responsible for managing and scheduling the collection of sequences that feed the sequencer of each UVM component. As such, a virtual sequence constitutes a timed test scenario, that can be compiled and stored in a virtual sequence library ⑮. The scoreboard is based on the *"subscriber"* design pattern to efficiently collect the data from the various sources. UVM has also been conceived to take full advantage of C++ in order to provide the testbench designer with a very reusable and convenient way to replace components or agents with other and hence switch smoothly from verification to validation while keeping intact the scheduling and the sequencing of the functional test vectors.

Fig. 3 illustrates the top-down decomposition of test se-

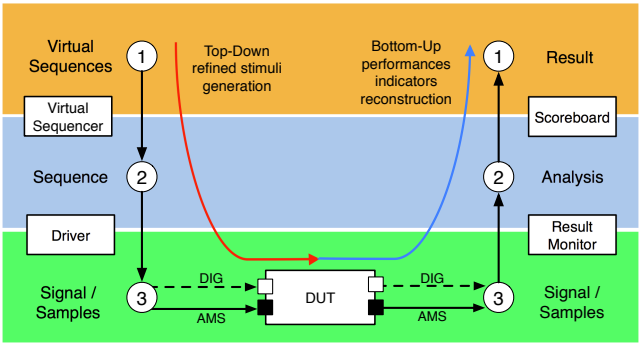

Fig. 3. *Three layers of structures*

quence stimuli in UVM, as well as the bottom-up reconstruction of performance indicators for verification. More precisely:

1) The top-down refined stimuli generation :
   - Virtual sequence level: The virtual sequences are extracted from a scenario database and propagated to the testbench.
   - Sequence level: The global virtual sequencer dispatches the appropriate sequence to the corresponding UVC component. The sequence is decomposed in sequence items that are managed by the local agent sequencer.
   - Signal level: The driver sends accurate stimuli/samples to the DUT digital and AMS ports.
2) The bottom-up performances indicators reconstruction :
   - Signal level: The monitor receives and accumulates the output signal level vectors/samples.
   - Analysis level: The UVM monitor contains utility functions that can be used to perform several analyses on the collected signal waveforms and to compute system performances.
   - Result level: The scoreboard performs end-to-end checking by comparing the golden model reference performance specifications with the extracted performances.

We will detail the basic concepts and features of UVM-SystemC in the following sections.

IV. UVM-SystemC Concepts Overview

UVM-SystemC, developed by NXP, defines all the features to create a standard verification environment. It takes advantage of the most prominent technologies used in efficient SystemC-based virtual prototyping such as interface configuration mechanism, transaction-level modeling and AMS extensions for heterogeneous modeling. At present, UVM-SystemC is still under development and contributed to Accellera for further standardization.

In the following, we elaborate the essential UVM concepts in several parts, to create a structured, modular, configurable and reusable verification environment.

A. UVM-SystemC basic elements

1) *agent*: The cornerstone of UVM is the agent. It can be seen as a convenient mean to receive sequential requests and convert them into low-level data exchanged with the DUT. If the UVM agent is active, it may contain a driver and a monitor working under the control of the corresponding sequencer. If it is inactive, an **uvm_agent** may just contain a **uvm_monitor** without sequencer and driver.
   Additionally, an agent may contain some analysis functions for coverage and checking, and some configuration capabilities. Agent can eventually be configured as master or slave according to the way they are configured.
2) *sequencer*: The sequencer works according to a pull semantics, i.e. it reacts to the orders given by the driver and offers the service of getting and delivering the next high-level test sequence item/transaction to the driver.
3) *driver*: In the UVM methodology, roles are clearly identified. The driver is responsible for creating and driving the physical signals to drive the DUT. For this purpose, the driver repeatedly requests transactions, encapsulated in a sequence, via the sequencer, and translates these to one or more physical signal(s).
   More precisely, the purpose of driver is to handle the conversion of high-level transactions into bit-accurate (for digital) or analog signal samples (for AMS) that physically (i.e. at the electrical pin level) drive the DUT. For instance, a driver controls the read/write signal, data bus and address bus for a period of clocks signal to execute a read transfer.
4) *monitor*: The monitor is a passive element that only captures the DUT signals. It extracts signal information from the interface and translates this information to abstract transactions. It will distribute this transaction to all connected elements for e.g. coverage collection and checking. The connection between the monitor and the DUT is established by using a dedicated channel, which is made available via the configuration mechanism.
5) *virtual sequencer*: A virtual sequencer is used in the stimulus generation process to allow a virtual sequence to be distributed across multiple sequencers within several agents.
6) *sequence item*: Sequence items or transactions belong to a sequence. Sequences are part of the upper scenario layer which define streams of transactions. The properties (or attributes) of a transaction are captured in a sequence item. Sequences are not part of the design hierarchy, but are mapped onto one or more sequencers. We will see later that sequences can eventually be layered, hierarchical or virtual, and may contain multiple sequences or sequence items. Sequences and transactions can be configured via the factory. High-level sequences are generally directly attached to the sequencer of each active UVC but the management of these sequences can also be delegated to a higher level, hence allowing the synchronization of UVCs by a kind of testbench conductor.
7) *scoreboard*: The role of the scoreboard is to determine whether the DUT operates correctly or not. It takes

two streams of transactions as input and compare the expected and collected performances indicators values with relational operators. For a thorough purpose of detailed verification, it is necessary to compare the actual performances computed by the monitor with the expected ones, coming from the user specifications or from a golden model. In the digital world, these comparisons are performed with simple equality tests. In the AMS world, comparisons always involve more subtle inequality tests.

B. Configuration mechanism and the Factory

UVM-SystemC provides a configuration mechanism and the factory pattern, both of these mechanisms make a convenient and effective way for reusing the existing verification, entire components or test in difference test case.

Indeed, the power of UVM comes from its very efficient configuration mechanism (⑬ in Fig. 2) that allows to propagate or retrieve hierarchically any information (value, configuration, filename, UVM components, etc) to any object of the built UVM hierarchy, and this configuration mechanism combined with the factory design pattern to dynamically instantiate or override UVM components make it powerful.

The configuration mechanism provides access to a centralized database where type specific information can be stored and retrieved. The configuration and resource classes (**uvm_resource_db** and **uvm_config_db**) provides a typed interface for object-centric configuration and resource facility.

The factory pattern is an well known object-oriented design pattern, this factory mechanism is used to create UVM objects and components. Use of the factory mechanism allows dynamically configurable component hierarchies and object substitutions without having to modify their code and without breaking encapsulation. It can provide the solutions to deal with the requirements, such as: replace the base UVM component which is deep in the hierarchy of your environment, add one more UVM component to the current UVM component list so that from the verification environment can work for another test case.

C. UVM-SystemC Phases

In order to have a consistent testbench execution flow, the UVM-SystemC uses phases to order the major steps that take place during simulation. There are three groups of phases, which are executed in the following order:

1) Pre-run phases (**build_phase**, **connect_phase**): The **build_phase** is executed at the start of the UVM testbench simulation and their overall purpose is to construct, configure the testbench component hierarchy. During the build phase UVM components are indirectly constructed using the factory pattern. The **connect_phase** is used to interconnect all the components.
2) Run-time phase (**run_phase**): The testbench stimulus is generated and executed during the run-time phases which follow the build phases. In **run_phase** and run-time phases, we execute the test scenarios by performing the configuration of the DUT and applying primary test stimulus to DUT.
   All UVM components using the run-time schedule are synchronized with respect to the pre-defined phases in the schedule.
3) Post-run phases (**extract_phase**, **check_phase**, **report_phase** and **final_phase**): where the results of the test case are collected and reported. The 4 post-run phases are used to post-precess the results after the execution of the test scenario of the DUT.

D. AMS extension

To correctly handle the AMS signals found in heterogeneous designs, the original digital UVM drivers and monitors have been substantially modified to support the Timed Data Flow (TDF) Model of Computation of SystemC-AMS. Next to the drivers and monitors required for digital verification, specific TDF analog drivers and monitors are added to the AMS UVCs, that allow to generate or monitor discrete time sampled value waveforms. The connections of the TDF drivers and monitors with the AMS-DUT form a TDF cluster that operates with its own cluster period, that is different from the driver period handling high level sequences.

E. Synchronization

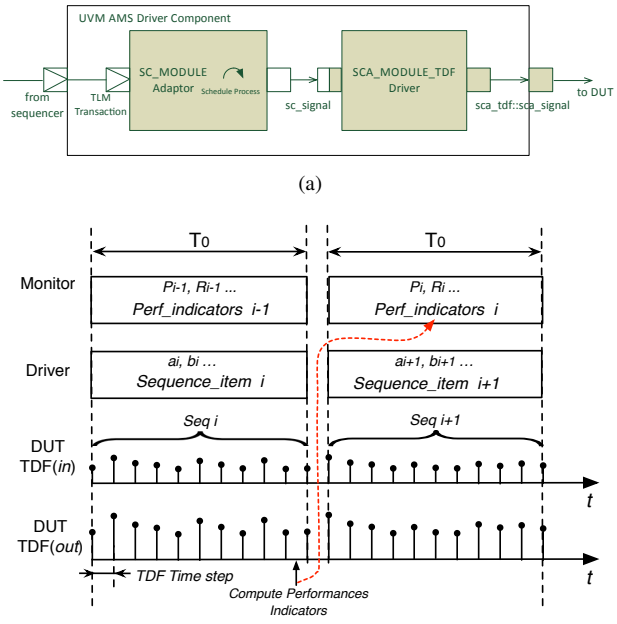

Fig. 4. *(a). UVM AMS driver component, it is composed of a SystemC TLM adaptor and a SystemC AMS TDF driver (b). Pipelined synchronization method.*

Virtual sequences, sequences, and sampled signals represent the actual data sent to or received from the DUT at different time scales. To correctly synchronize the UVCs involved in the testbench and the DUT, it is necessary to synchronize the related SystemC and SystemC-AMS. In Fig. 4(a), a UVM AMS driver component consists of two modules. The first

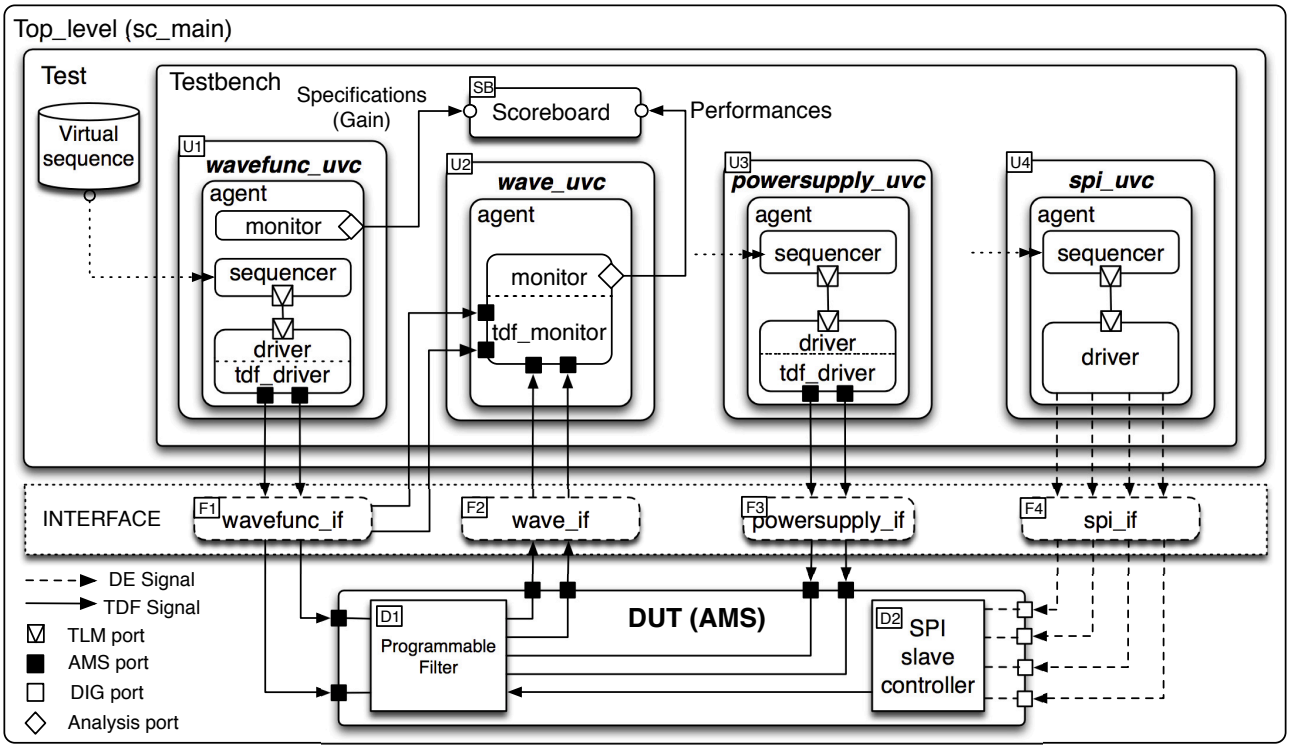

Fig. 5. *Case study: a full-fledged programmable filter testbench with UVM-SystemC (AMS)*

component is a SystemC TLM adaptor component. A process of this component reads the timestamps of the transactions and writes (schedules) the data of the transaction in the second delta cycle, one resolution time step before the timestamp of the transaction. This guarantees, that the data will be read at the correct time by the TDF module, since SystemC-AMS reads event driven SystemC signals always at the first delta cycle of the current time.

This pipelined synchronization method is presented in Fig. 4(b). Between each sequence item, the sequencer/driver waits for a period *T0*. The monitoring thread waits for the same period, so that the emission of stimuli and the reconstruction of performance indicators are kept synchronous at any time during the simulation, for the scoreboarding comparisons to make sense. Once a new sequence item has been propagated to the driver (at the very beginning of the sequence item period), the corresponding TDF driver module uses the high-level stimuli signal description to generate the TDF stimuli for the DUT. For instance, if the sequence item defines a new operating frequency for the wave generator, the corresponding TDF processing function reacts immediately (i.e. within a TDF period indicated by the TDF cluster timestep) and generates the appropriate low-level samples. The sequence item high-level information data remain valid during the whole *T0* period. At the very end of the *T0* period, all the TDF stimuli samples corresponding to a given sequence item have been sent to the DUT, and all the monitored values coming from the DUT have been aggregated in order to compute the values of the performance indicators. When scheduled, the thread associated to the monitor gains direct access to these performance values, that will remain valid for the next *T0* period.

V. Case Study

A. Programmable filter

In order to validate the framework, a verification test case has been achieved at the first step. The selected use case is an SPI-Controlled programmable filter, that is composed of two parts, as shown in Fig. 5. The first part is the Analog Mixed Signal (AMS) filter D1, and the second part is a digital SPI slave controller D2.

The filter DUT is written in SystemC and its AMS extensions. There are two power supply inputs v_plus and v_minus, two differential inputs named p_in and m_in, and two differential outputs p_out and m_out. The behavior of the filter is controlled by a 4-bit word cmd_in. The two MSB bits of cmd_in control the filter mode that can either be ideal (mode 00), one-pole (mode 01) and two-poles (mode 10). The two LSB bits of cmd_in define the amplification factor, from x1 (00) to x1000 (11). The SPI slave controller is written in SystemC and has a genuine SPI interface, with a complementary reset input named *reset_n*, a chip select input named *cs_n* the clock input named *sclk* and the 8-bit command data are shifted in using the *mosi* input.

In order to verify the DUT with UVM-SystemC (AMS), it has to be connected to 4 entities: a power supply generator, a

function generator, an SPI driver that generates SPI-compliant commands and an analyzer in order to visualize and interpret the signals produced by the DUT filter. For the interpretation of the output signals to make sense, the input signals must also be monitored and managed by the analyzer, to compute the output gain of the programmable filter.

The 4 entities are implemented as AMS UVCs. The **powersupply_uvc** U3, **wavefunc_uvc** U1 and **spi_uvc** U4 contain only active agents and the **wave_uvc** U2 contains only passive agent.

Connections between UVCs and the DUT follow the interface configuration mechanism provided by UVM-SystemC. In this case, there are 4 interfaces related to the 4 UVCs. **powersupply_if** F3, **wavefunc_if** F1, **wave_if** F2 are TDF signal interfaces and **spi_if** F4 is a digital signal interface.

During the SystemC-AMS simulation, the 4 UVCs, the TDF drivers and monitors are carefully synchronized, following the synchronization rules of the previous section. That way, the sequence items are extracted from the sequence with a slow (sequence item) period that remains compatible with the fast TDF cluster execution period. Similarly, the UVM monitor operates at the same slow rhythm and periodically asks the TDF monitor to give the last values of performances indicators.

The scoreboard SB is responsible for comparing the expected performances of the programmable filter, which are managed by the **wavefunc_uvc**, with the evaluated performances that are computed after a thorough analysis of the TDF samples coming from the filter and taken care of by the wave monitor. The expected gain is directly part of the wavefunc sequence item, and the collected gain is calculated by the wave monitor. This is where the **uvm_analysis_port** in the wavefunc sequencer and in the wave monitor come into action, as means to propagate the values to be compared to the scoreboard.

In our case, a virtual sequence consists in setting the powersupply, resetting and writing a SPI command, and providing a frequency test request to generate a series of TDF samples with a specific operating frequency.

The 4 UVCs and the scoreboard are instantiated in a UVM testbench. The testbench also connect the expected performances from the **wavefunc_uvc** and the observed performances from the **wave_uvc** to the scoreboard.

The test instantiates the testbench and defines the virtual sequence. The virtual sequence is initiated and terminated in the test. The top-level (e.g **sc_main()**) contains the DUT-AMS and the test. The interfaces to which the DUT can be connected are stored in the configuration database, and can be used by the UVCs to connect to the DUT in the top-level after a simple retrieve operation in the database.

B. Automotive use case - Smart power supply module

The selected use case from Continental Automotive France for this experimentation is a smart power supply module extracted from an automotive Electronic Control Unit (ECU) of an engine management system. It is implemented in an

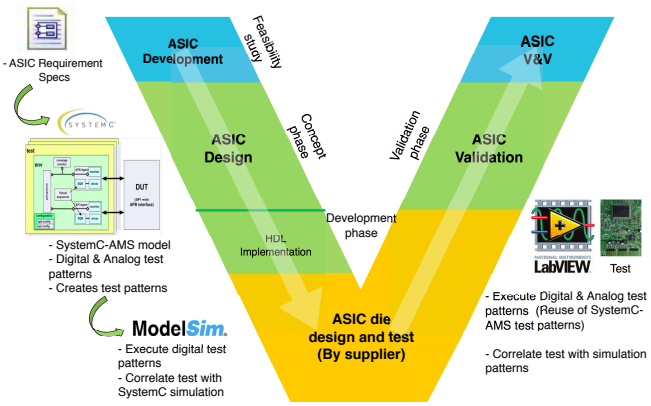

Fig. 6. *ASIC design, test, verification & validation process.*

ASIC including voltage regulator block providing several supply voltages in addition to internal and external reset chain for the complete ECU. The ECU is controlled by an SPI communication interface. Additionally, it embeds a complex state machine as digital circuitry for safety monitoring purpose, connected via the same SPI interface to the main controller. The ASIC serves as the DUT to be verified by the UVM methodology. It is written in SystemC-AMS, abstracting the gate level implementation and being functional and approximately timed for both digital and analog domain. Moreover, the DUT as tape-out silicon shall be validated in laboratory test environment controlled by LabVIEW [19] software. The simulation and test stimuli shall be unified in order to generate the equivalent test patterns and to compare both tests results. This strong requirement induces additional constraints on the UVM testbench through the implementation of its verification components.

To follow the targeted methodology as defined in the Fig. 6, the DUT verification starts by implementing a simulation testbench using the UVM-SystemC class libraries. The UVC is defined according to the DUT behavior, in order to be reused across several design verification. As shown in Fig. 7, the test-bench is organized in 4 UVCs agents. First the driver of **AMS-UVC(1)** is controlling the TDF input signal of the power voltage regulation (battery, key and main relay voltage) and its monitor is recording the regulated voltage outputs (5V, 3.3V, 1.3V.. etc) and the inputs voltage values. Another **AMS-UVC(2)** is setting / resetting the digital reset inputs and is recording the reset out and in pins. It is coupled with **AMS-UVC**$_1$ to monitor the internal supply of the ASIC built in the voltage regulation block. A digital **UVC(3)** used for SPI communication is driving and monitoring the question and answer of the DUT SPI interface. The voltage regulation and digital reset blocks use SPI only for configuration, therefore are loosely coupled. Finally, an **AMS-UVC(4)** component is driving TDF mixed signals and logical control inputs, recording the inputs and monitoring logical outputs, which are in charge of deactivation of safety critical hardware component. The monitoring unit block is controlled by its input/output and SPI command interface. Therefore they are strongly

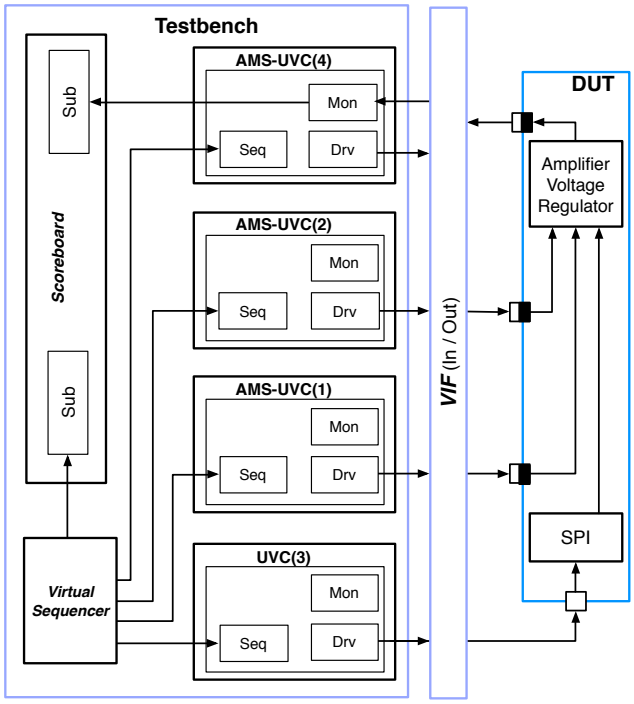

Fig. 7. *The implementation of the amplifier voltage regulator to the 3VFA architecture*

coupled. The coupling of UVC component is controlled by synchronizing the stimuli time stamp.

Each UVC implements a sequencer controlling its local time and performing TLM time stamped data transaction, while the signal timed control is ensured by the driver. The time stamp data transaction is read by the sequencer from a text file. Multiple data can be passed on the transaction for an identical time step depending on the UVC. The start of the sequencer of UVC component is controlled by the top level virtual sequencer. It gives the reference synchronization time t_0 and decides the UVC executed mode according to the test sequence: in parallel or in sequential. Each UVC component will be fully configurable depending of the stimuli defined in the text file. The scoreboard collects all recorded transaction of the UVCs and builds a log file including data and time stamp to allow later post processing verification.

The reuse of the test scenario for functional validation of the tape-out ASIC is guaranteed by the reuse of the UVM command files. As they contain the stimuli configuration with data and time stamp, the LabVIEW software will interpret them thanks to the test sequencer decoder built in LabVIEW blocks. The computer and equipment latency for controlling the input analog physical and the SPI communication are taken into consideration during the writing of the stimuli sequence. The influence of latency for the output and in particular during the recording phase will not be visible, thanks to time resynchronization on input data also recorded. The verification of the trace results is planned to be performed off line by post-processing of the log trace comparison. Thanks to the same transcript format between simulation and test, the simulation scoreboard improvement or evolution can then be reused on trace generated by the equipment.

VI. Simulation Results

A. Programmable filter

Fig. 8 presents a simulation snapshot of the programmable filter, obtained after the application of a test scenario in the designed within the UVM-SystemC (AMS) environment. The snapshot is composed of four parts. The first part *SPI commands* respect a sort of timescale, while the second part *powersupply*, the third *input/output DUT waveforms* and fourth parts *performance indicators* use a much bigger timescale. The SPI configuration orders sent once for all at the very beginning of the simulation.

The snapshot shows that the two differential *powersupplies* are set to +5V / -5V, after a period of ramp-up. The results show that the consequences of the ramp-up cause the saturation of output DUT signals at the first period of the simulation.

The *SPI commands* show *clk*, *reset*, *cs*(chip_select), and *mosi* respectively. In the **run_phase** method of UVM, a serial *clk* signal is generated for the SPI slave controller. The *reset* signal performs SPI reset on the DUT at the very beginning on the simulation. When the *cs* signal is asserted, the *mosi* signal writes 8-bit data to the interface SPI on the clock falling edge. In the example, the SPI writes an SPI command=0x9 to the DUT, telling it switch to mode 2 (10, the upper two bits of the 4 bits 0x9=0b00001001 command) with amplification factor x10 (01, the lower two bits of the 4 bits 0x9=0b00001001 command). Then we send the analog test samples to the DUT.

To test the programmable filter, the virtual sequence defined in the testbench contains a row of logarithmically spaced points between decades minimum(7000Hz) and maximum(35000Hz) frequency and provides 7 different transactions for the **wavefunc_driver**, each of which with a specific operation frequency. *fc* (cut-off frequency) is 20000Hz. The *input/output DUT* shows the frequency response of the filter, the attenuation of the signals in case of too high frequency is clearly visible. The performances indicator (Gain) is shown in the last section in Fig. 8. Another type of simulation results are mentioned in the following table. It corresponds to the comparison coming from the scoreboard.

```
#Sequence item starts
    Frequency = 9262.47Hz
#1.6 ms: test.tb.scoreboard
    Gain = 9.46642dB successfully matched ( > 8dB)
#Sequence item stops
```

B. Automotive use case

The simulation of the DUT, allows verifying the ASIC functional requirements. Thanks to the simulation trace, as visible on a general overview of Fig. 9, we can simulate power up and power down sequence of the ASIC and validate specific requirement on delay and latency of the output voltage regulation during these critical phases (not detailed and visible here). Another example, is the verification of the regulation

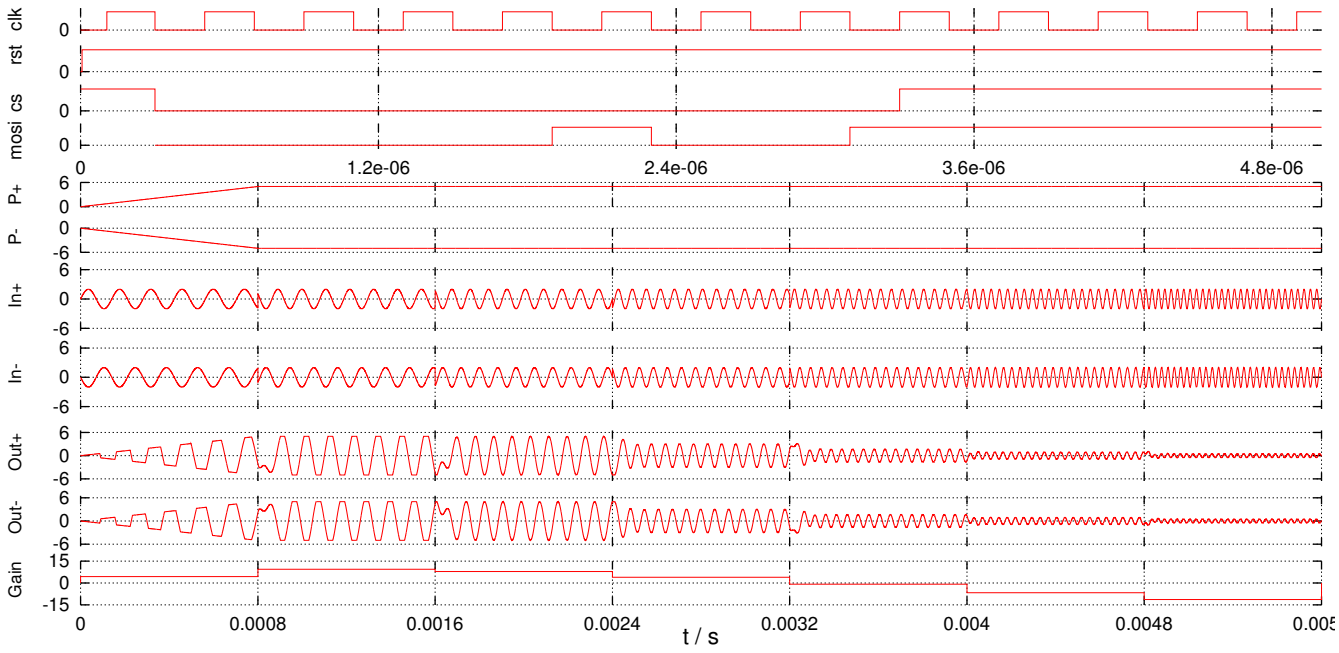

Fig. 8. *Simulation results, from top to bottom, they are SPI commands, powersupply, input/output DUT and performance indicators (Gain) respectively.*

functional performance during stabilized phase (+- 5%) as for the 3.3V (Vdd3_pin in green) still stable even if input VBD (VBD_VDR pin in purple) fluctuates within the defined range. This is not visible in this trace, as zoom and extra trace shall be performed to force the VBD range out of the component limit.

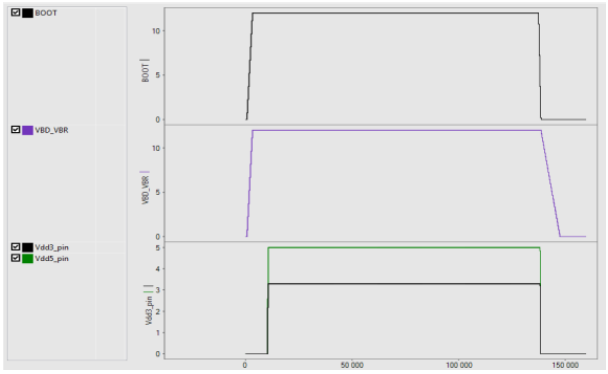

Fig. 9. *Extract of simulation.*

Moreover, for the equipment control and stimuli generation operating with LabVIEW, we are now able to regenerate the same functional scenario. As depicted on Fig. 10, we have slowed down the power-up time to make visible trace and adapt the voltage transient phase to equipment performance. The pattern is verified on a standard oscilloscope, showing that voltage sequence is correctly applied.

The integration of the SPI communication interface has also been implemented and verified. The equipment, controlled to interface the hardware, is made by a standard USB/SPI interface card from National Instrument.

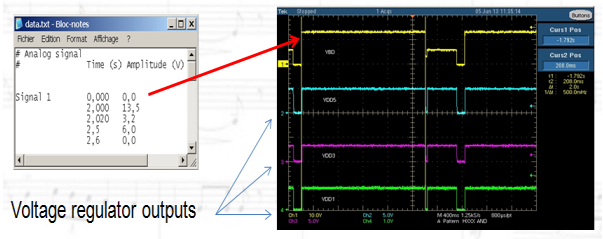

Fig. 10. *LabVIEW piloted power supply.*

VII. Conclusion

In this paper we presented a virtual prototyping, verification and validation framework for automotive using SystemC, SystemC-AMS and UVM-SystemC. We use SystemC for digital parts of modulation and simulation, SystemC-AMS for system-level design of mixed-signal systems, and UVM-SystemC for establishing a standard and reusable system-level verification environment. The combination of these three languages provide a unified, structured, configurable and modular testbench for automotive design.

We applied our framework to a smart power supply module on an automotive Electronic Control Unit for engine management, and successfully achieved the first step of integration of an amplifier voltage regulator. The next step will be the completion of the whole case study design.

Acknowledgement

This work was funded by the project Verification For Heterogeneous Reliable Design and Integration (VERDI), which is supported by the European Commission within the

7th Framework Programme for Research and Technological Development (FP7/ICT 287562).

REFERENCES

[1] S. Choi and A. Chan, "A virtual prototyping system for rapid product development ," *Computer-Aided Design*, vol. 36, no. 5, pp. 401 – 412, 2004.

[2] W. Ecker, V. Esen, R. Schwencker, T. Steininger, and M. Velten, "TLM+ modeling of embedded HW/SW systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 75–80.

[3] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. P. Pande, C. Grecu, and A. Ivanov, "System-on-chip: reuse and integration," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1050–1069, 2006.

[4] IEEE Computer Society, *1666-2005 IEEE Standard SystemC Language Reference Manual*, IEEE, 1666-2005.

[5] A. El Mrabti, F. Petrot, and A. Bouchhima, "Extending IP-XACT to Support an MDE Based Approach for SoC Design," in *Design, Automation Test in Europe Conference Exhibition*, 2009, pp. 586–589.

[6] *Cadence Design Systems Initiative, Open Verification Methodology (OVM)*, http://www.cadence.com/alliances/languages/pages/ovm.aspx/.

[7] *Mentor Graphics Initiative, Advanced Verification Methodology (AVM)*, http://www.mentor.com/go/cookbook/.

[8] *Accellera Systems Initiative, Standard Universal Verification Methodology (UVM)*, http://www.accellera.org/downloads/standards/uvm/.

[9] C. Spear, *SystemVerilog for Verification, Second Edition: A Guide to Learning the Testbench Language Features*, 2nd ed. Springer Publishing Company, Incorporated, 2008.

[10] *Cadence Design Systems Initiative, e Reuse Methodology (eRM)*, http://www.verisity.com/products/erm.html/.

[11] J. Bergeron, *Writing testbenches: functional verification of HDL models*. Kluwer Academic Publishers, 2003, vol. 2.

[12] G. G. Wang, "Definition and review of virtual prototyping," *Journal of Computing and Information Science in Engineering*, vol. 2, no. 3, pp. 232–236, 01 2003.

[13] A. G. de Sa and G. Zachmann, "Virtual Reality as a Tool for Verification of Assembly and Maintenance Processes," *Computers and Graphics*, vol. 23, pp. 389–403, 1999.

[14] *Accellera Systems Initiative, SystemC AMS 2.0 Standard*, http://www.accellera.org/downloads/standards/systemc/ams.

[15] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2003, pp. 19–24.

[16] A. Leveque, F. Pecheux, M.-M. Louerat, H. Aboushady, and M. Vasilevski, "SystemC-AMS Models for Low-Power Heterogeneous Designs: Application to a WSN for the Detection of Seismic Perturbations," in *ARCS Conference*, 2010, pp. 1–6.

[17] A. Vachoux, C. Grimm, and K. Einwich, "Extending SystemC to Support Mixed Discrete-Continuous System Modeling and Simulation," pp. 5166–5169, 2005.

[18] M. Zhou and R. van Leuken, "SystemC-AMS Model of A Dynamic Large-Scale Satellite-Based AIS-Like Network," in *Forum on Specification and Design Languages (FDL)*, sept. 2011, pp. 1 –8.

[19] *LabVIEW System Design Software*, http://www.ni.com/labview/.