LECTURE NOTES IN ELECTRICAL ENGINEERING 10

Eugenio Villar
Editor

Embedded Systems Specification and Design Languages

Selected Contributions from FDL'07

Springer

Contents

Part II
Analog, Mixed-Signal, and Heterogeneous System Design

Chapter 8
Heterogeneous Specification with HetSC and SystemC-AMS: Widening the Support of MoCs in SystemC

F. Herrera[1], E. Villar[1], C. Grimm[2], M. Damm[2] and J. Haase[2]

Abstract This chapter provides a first general approach to the cooperation of SystemC-AMS and HetSC (*He*terogeneous *S*ystem*C*) heterogeneous specification methodologies. Their joint usage enables the development of SystemC specifications supporting a wide range of Models of Computation (MoCs). This is becoming more and more necessary for building complete specifications of embedded systems, which are increasingly heterogeneous (they include the software control part, digital hardware accelerators, the analog front-end, etc.). This chapter identifies the syntactical and semantical issues involved in the specifications which include facilities from both, SystemC-AMS and HetSC methodologies. This work, which is an extension of the paper presented in FDL'07 [7], considers the availability and suitability of the MoC interface facilities provided by both methodologies, especially those of SystemC-AMS, which will be proposed for future standardization. Some practical aspects, such as the current set of MoCs covered by the methodologies and the compatibility on the installation of their associated libraries are also covered by this chapter. A complete illustrative example is used to show HetSC and SystemC-AMS cooperation.

Keywords Heterogeneity, Models of Computation, System-Level Design, SystemC.

8.1 Introduction

Support for heterogeneity has become an important feature for specification methodologies that aim to cope with the current complexity of embedded systems. In this context, heterogeneity is the ability of the specification methodology to enable the building of models with parts specified under different MoCs [1].

[1]University of Cantabria, Spain

[2]Technical University of Vienna, Austria

Each design domain adopts a specification methodology which usually corresponds to a specific model of computation (MoC). One of the most characteristic points associated with the MoC is the handling of time. For instance, analog models (Continuous Time (CT) models [2] handle strict-time information, that is, specification events have an associated time tag representing physical time and fixing strict order relationships among them. In contrast, concurrent software models often neglect such detail in the time domain and consider only partial order relationships among the events associated to the code.

The development of a system-level heterogeneous specification methodology is, to a great extent, a unification work. Some works developed interfaces between different languages, i.e., to connect hardware description languages (HDLs) with high-level programming languages [3]. This enabled certain decoupling between different design teams, which can fix the connection points and work separately. However, a system-level specification methodology has to enable the generation, understanding, edition and simulation of the specification of the whole system. This is a unification work which involves finding common points for the specification and simulation methodologies handled by the different design communities.

An effort to develop a common specification and simulation framework was done. Relevant examples are Metropolis [4] and Ptolemy II [5]. These frameworks enable specification under different MoCs, approaching the separation of computation and communication in different ways. Both provide support for graphical specification, while Java adopts the role of underlying implementation language.

Up to now, the focus of this unifying work has tended to be the language itself. The lack of a unified system specification language has been identified as one of the main obstacles bedevilling SoC designers [6]. A common specification language is a major aid in generating a specification methodology which aims to combine and achieve coherence among traditionally different and separated design approaches. SystemC has started to play a role as unifying system-level language for embedded system design. Becoming an IEEE standard is a symptom of its acceptance and of a stated syntax and unambiguous semantic for the language constructs which are used by SystemC-based methodologies.

In this context, several proposals have appeared for building heterogeneous specifications in SystemC. This chapter shows how two of them, HetSC and SystemC-AMS can be jointly used to enable models based on the SystemC language and comprising a wide spectrum of MoCs. This work is based on [7], which is improved and extended here. After this introduction, Section 8.2 reviews previous work on heterogeneous specification in SystemC. The main focus is on the HetSC and SystemC-AMS specification methodologies. Section 8.3 deals with general issues about the interoperability of these methodologies. First, some practical issues concerning the installation and the scope of the libraries are discussed. Then, how the SystemC-AMS and HetSC constructs are mixed in the same specification is explained. Reviewing and understanding how the existing facilities provided by the two methodologies for MoC connection can be used and combined serves later to propose improved connections. Section 8.4, provides an illustrative example of the previous concepts. Last section ends with the main conclusions and advances further steps of the research on this topic.

8.2 Heterogeneous Specification in SystemC

Although the SystemC core language supports hardware specification (RTL and Behavioural) and a generic Discrete Event (DE) modelling, there is a set of MoCs which are not sufficiently supported by the core language. Such support must include new specification facilities, MoC rule checkers, report tools, etc. Several works have attempted to cover such deficiencies. In the following subsections these works are overviewed. Most of these methodologies are supported by an associated library; however, they extend SystemC in different ways.

8.2.1 SystemC-AMS

SystemC-AMS [8] is a specification methodology developed by the OSCI SystemC-AMS working group which provides support for analog and mixed-signal specification. This involves supporting the Synchronous Dataflow (SDF), discrete-time (DT) and continuous time (CT) MoCs. Among the CT MoCs, it is possible to specify linear behavioral models by means of transfer functions (TF). Currently, two views are supported for TFs: the numerator-denominator (ND) view and the zero-pole (ZP) view. In addition, the specification of linear electrical networks (LEN), which enable a circuit level description, is also supported.

SystemC-AMS is extensible by other models of computation through a synchronization layer. Solvers for the MoCs supported are layered over the synchronization layer. The design of the synchronization layer of SystemC-AMS and the MoCs provided are oriented to a system-level modelling where simulation speed is a more important factor than a very fine simulation accuracy.

The synchronization layer supports directed communication and only a simple synchronization; on user specified events or in fixed time steps. In this way, the simulation of linear networks with SystemC-AMS can be orders of magnitudes faster than the more general numerical integration for non-linear networks [9]. From the specification point of view, SystemC-AMS offers a new set of facilities, such as new kinds of modules (*SCA_SDF_MODULE*), ports (*sca_sdf_in*, *sca_sdf_out*, etc.), channels (*sca_sdf_signal*), and other MoC specific facilities, such as the *sca_elec_node*, *sca_elec_port*, etc. Linear behavioural models are embedded in SDF modules, while LENs are enclosed in SystemC modules. SystemC-AMS provides converter ports and facilities to enable different MoCs to communicate (i.e. DE with SDF, SDF with LEN, etc).

8.2.2 HetSC

HetSC [10] is a methodology for enabling heterogeneous specifications of complex embedded systems in SystemC. MoCs supported include untimed MoCs, such as Kahn Process Networks (KPN), its bounded fifo version (BKPN), Communicating

Sequential Processes (CSP) and Synchronous Dataflow (SDF). Synchronous MoCs, such as Synchronous Reactive (SR) and Clocked Synchronous (CS) and the timed MoCs already supported in SystemC are also included. HetSC aims at a complete system-level HW/SW codesign flow. Indeed, the methodology has been checked in terms of system-level profiling and software generation [11].

The HetSC methodology defines a set of specification rules and coding guidelines for each specific MoC, which makes the designer task more systematic. The support of some specific MoC requires new specification facilities providing the specific semantic content and abstraction level required by the corresponding MoCs. The HetSC library, associated with the HetSC methodology, provides this set of facilities to cover the deficiencies of the SystemC core language for heterogeneous specification. In addition, some facilities of the HetSC library help to detect and locate MoC rule violations and assist the debugging of concurrent specifications. One of the main contributions of HetSC is its efficient support of abstract MoCs (untimed and synchronous). This is because they are directly supported over the underlying discrete event (DE) strict-time simulation kernel of SystemC. New abstract MoCs do not require additional solvers since the new MoC semantic is embedded in the implementation of the new specification facilities (usually channels) related to the abstract MoC. When the new MoC can be abstracted from the DE strict-time MoC, then, it is possible to find a mapping of internal events of the new specification facility, i.e., a channel, over the strict-time axis of the DE base MoC. This makes it feasible to write the implementation of such a channel by using SystemC primitives, such as SystemC events, which control when things happen within the channel and, therefore, in the processes related by the channel.

8.2.3 SystemC-H

SystemC-H [12] is a methodology that proposes a general extension of the SystemC kernel for the support of different MoCs. This methodology proposes the extension of the SystemC kernel by including a solver for each MoC. The current scope of the SystemC-H library covers the SDF and CSP untimed MoCs. For instance, SystemC-H provides a solver for static scheduling of SDF graphs which enables schedulability analysis and provides a 75% speed-up respect to DE [12]. However, this extension is not always worthwhile. Indeed, the speed-up for some abstract MoCs can be negligible [13]. In addition, the effects of Amdahl law can make simulation speed-ups vanish. For instance, in [12] the speed-up of a mixed DE-SDF example decreases to 13%. This suggests that providing a specific solver for each MoC can be not always worthy. In cases like these, it can be more efficient to let several MoCs to share the same simulation kernel. This is the approach of HetSC, where similar speed-ups to those of [12] were reported for the dynamic approach to SDF for large-grain SDF specifications [14]. Another problem of this approach is that the way the extension is proposed requires modifying the standard kernel of the library. In contrast, SystemC-AMS and HetSC methodological libraries rely on the SystemC standard library, which remains untouched.

8.2.4 SysteMoC

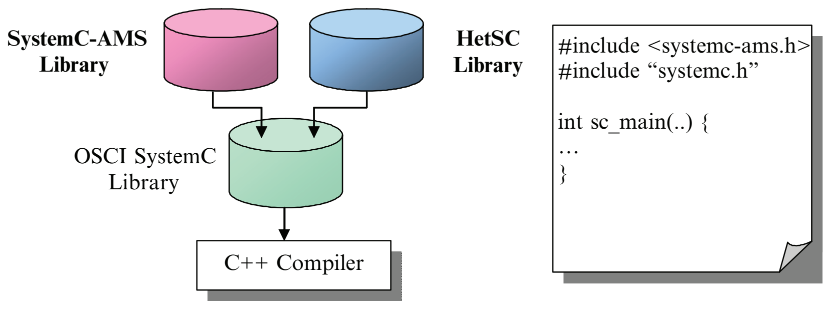
SysteMoC [15] focuses on providing a methodology with the ability to extract and analyze the MoC employed in the SystemC design. This is understood to be a pre-requisite for the rest of the design activities. In order to achieve this, the SysteMoC library provides support for a basic MoC called *Funstate*. Specifications written under this MoC express their communication behaviour under the finite state machine (FSM) MoC. This enables the automatic extraction and analysis of the MoC employed, only by analyzing communication FSMs together with the topology of the specification.

8.3 HetSC/SystemC-AMS Interoperability

8.3.1 Installation and Scope

Figure 8.1 describes the installation requirements of the SystemC user. Apart from the SystemC core library, the SystemC-AMS and HetSC libraries have to be installed on top of the SystemC core library. There is flexibility with respect to the development platform (i.e., Linux, Unix and Windows-Cygwin are supported).

There is no compatibility problem in the installation of HetSC and SystemC-AMS libraries. In this work, the HetSC library is extended with some specific facilities for enabling an easier connection of HetSC and SystemC-AMS parts. These HetSC facilities use some SystemC-AMS facilities through forward declarations. This prevents obliging an installation order between HetSC and SystemC-AMS libraries, making the installation procedure easier. Once such an installation has been done, the development system is ready for compiling and executing SystemC specifications written under a wide range of MoCs. The user only has to include the SystemC-AMS and HetSC libraries in the source code of the heterogeneous specification.



```
#include <systemc-ams.h>
#include "systemc.h"

int sc_main(..) {
…
}
```

Fig. 8.1 SystemC-AMS and HetSC libraries are installed over the SystemC library

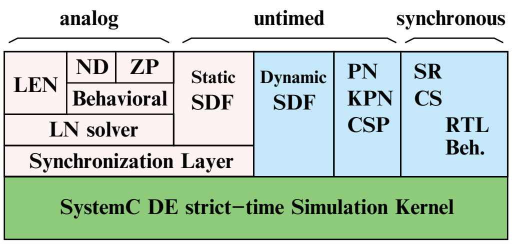


Fig. 8.2 MoCs spectrum provided by the cooperation of SystemC-AMS and HetSC

Figure 8.2 shows the supported MoCs. The cooperation of SystemC-AMS and HetSC provides a complementary MoC support. While SystemC-AMS provides support for analog MoCs and static synchronous data flow, untimed and synchronous MoCs are supported by HetSC and SystemC core facilities.

This is also an efficient configuration for the support of a wide spectrum of MoCs. The reason is that specific solvers are provided only for a set of MoCs where the simulation speed up is significant. This set corresponds to analog MoCs, where the simulation speed ups can be of orders of magnitude. Bearing in mind the limited speed-ups reported in [10, 12, 13], untimed and synchronous MoCs can be satisfactorily supported directly over the SystemC kernel. The exception would be fine grain SDF specifications, where the speed up of a static SDF compared to a dynamic SDF could be significant. Specifications without CT parts but with synchronous hardware (RTL or behavioural) could also justify a cycle-accurate simulator. However, the study of these exceptions is not in the scope of this work.

8.3.2 *Syntactical and Semantical Issues*

There are some basic issues to consider in a general discussion of the connection between HetSC and SystemC-AMS. In terms of the resulting structure, two parts can be distinguished in the specification. One corresponds to the AMS part, while the other corresponds to the HetSC part.

From the syntactical point of view, the SystemC-AMS part will be identified by *SCA_SDF_MODULEs* and/or SCA hierarchical modules. This part presents a hierarchical heterogeneity where the underlying MoC is the static synchronous dataflow (SDF) MoC. The HetSC part is characterized, in general, by an amorphous heterogeneity. This means that the HetSC specification permits mixing MoC facilities in a flat hierarchy. Nevertheless, the HetSC specifier will often make use of module hierarchy for separating parts of the system under different MoCs. Thus, in many cases, module partition will correspond with MoC boundaries.

From the semantical point of view, there is a basic consideration. While HetSC directly relies on the DE strict-time simulation kernel, SystemC-AMS relies on a synchronization layer, which provides support for the solvers. In SystemC-AMS, CT descriptions are always embedded in dataflow clusters [8]. That is, the most important solver is the SDF one which, from the point of view of time semantics, is the basis for the analog MoCs. The time axis in SystemC-AMS is actually sliced by each SDF cluster in strict-time delays called cluster periods ($T_{cluster}$), which depends on the sample period (T) and rates of the cluster SDF graph. Thus, with respect to the premises of [14], the SDF approach of SystemC-AMS is not an untimed SDF. Internally, modules of the cluster can be viewed as a strict-time timed approach to the SDF MoC (denoted as T-SDF here), which enables a static execution of the AMS processes at each cluster period. More important for the purpose of this work, from an external point of view, the cluster can be conceived as a timed-clocked synchronous (CS) block which triggers at each cluster period. Thus, the cluster period must be taken into account to synchronize the DE part with the SystemC-AMS part.

Since every MoC supported by HetSC is abstracted over the DE strict-time simulation kernel and every SystemC-AMS MoC is clustered in the T-SDF MoC, the problem is reduced to providing a SystemC/SystemC-AMS connection, which is basically a DE/T-SDF connection. In SystemC-AMS, this connection is done by means of SystemC signals (*sc_signal* channels) and a set of SystemC-AMS connection facilities (*sca_scsdf_in*, *sca_scsdf_out*, *sca_sc2v*, *sca_sc2r*, etc.). Each of these connection facilities are based on the sampling and/or update of a SystemC signal at each cluster time. Therefore, an immediate conclusion is that these elements can be directly employed to combine HetSC and SystemC-AMS.

Such direct usage of the *sc_signal* and the DE/SystemC-AMS connection facilities is immediate in some HetSC/SystemC-AMS connections. On the left hand side of Fig. 8.3, a HetSC part under a synchronous reactive MoC (SR MoC) is represented. Both in Figs. 8.3 and 8.5 the graphical representation used in HetSC methodology is employed. There is a simple reactive chain composed of a generator process (GP) which triggers a reactive process (RP). This RP is also a border process (BP), since it writes to a SystemC signal channel (*sc_signal*), which is connected to a SystemC-AMS part. Its connection with the SystemC-AMS part by means of a signal channel is syntactically and semantically compatible with the SR MoC rules. These rules and, specifically, perfect synchrony, are respected since the write access to the SystemC signal is non-blocking. This enables the reactive chain to be computed consuming one or more simulation delta cycles, but without requiring a SystemC time advance. This is the way in which perfect synchrony is implemented in HetSC.

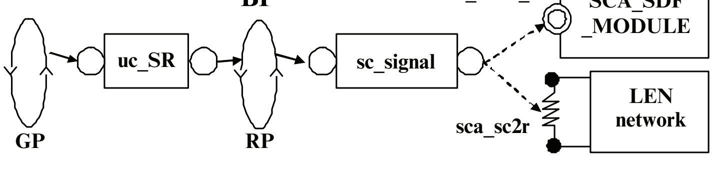


Fig. 8.3 Connection of HetSC and SystemC-AMS parts by means of a border process

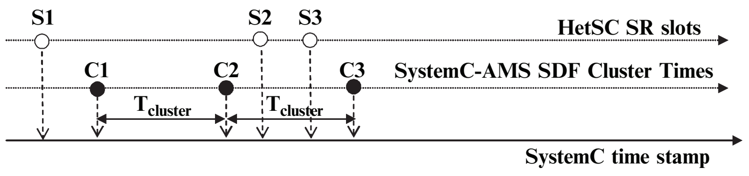


Fig. 8.4 Strict time information in the HetSC/SystemC-AMS connection

From the SystemC-AMS part, the connection is coherent too. In the connection to a *SCA_SDF_MODULE*, the value of the *sc_signal* is read at each cluster time. This value can be read as many times as necessary, as the consumption rate of the *sca_scsdf_in* port determines. Another possibility is the connection to a linear electrical network (LEN MoC of SystemC-AMS) by means of a converter facility, for instance, a *sca_sc2r* in Fig. 8.3. This facility enables the update of its associated resistance value whenever the *sc_signal* channel is written. Then, this updated resistance value is employed by the LEN solver in the following cluster times to solve the differential equation corresponding to the electrical network the converter facility belongs to.

The time stamp information of the HetSC SR slot is irrelevant to the effect of the HetSC SR MoC itself (the only necessary condition is that each slot has to happen at different time stamps). However, it is important to the effect of the (HetSC) SR MoC/(SystemC-AMS) LEN MoC connection, since it tells when the differential equation system is updated, before or after a given cluster time. For instance, in Fig. 8.4, the time stamps of the SystemC-AMS cluster computations are represented as black dots. Their time stamps are equally spaced. The time stamps of the SR time slots are represented as white dots. As mentioned, there is still consistency in the SR part if slot time stamps are not equally separated. However, actual time stamps of SR slots affect the relationship of the SR part with the timed SystemC-AMS part. For example, the first two cluster computations, C1 and C2, use the signal value updated in the slot S1, while the cluster computation C3 uses the signal value updated in the slot S3. If the S2 slots moves to a time stamp before C2, then, although this is no relevant to the effects of the SR part, it affects the SystemC-AMS part, since C2 takes the value updated in S2.

The set of MoCs abstracted from the DE MoC and supported by HetSC is rich enough to consider specific connections which cannot be directly handled by only a SystemC signal plus SystemC-AMS connection facility. For instance, the connection of a KPN MoC with a LEN MoC involves fifo channel semantics on one side and electrical nodes on the other side. It would be convenient to count on some connection facility which enables such direct connection, without the explicit intermediation of the SystemC signal (*sc_signal*).

In order to get such a direct connection, both in syntactical and semantical terms, the SystemC signal-SystemC-AMS connection facility can be conveniently complemented and wrapped by one of the basic concepts employed in HetSC for the connection of MoCs, the border process. Externally, the connection facility can take the shape of a HetSC border channel (BC). Figure 8.5 shows a border channel (*uc_inf_fifo_sca_sdf*), which enables a direct connection between a KPN MoC and a T-SDF MoC. It is built as a hierarchical channel which on the one hand exports the write interface of an *uc_inf_fifo* channel, while on the other hand offers a T-SDF port (*sca_sdf_out*) port. Internally, it uses a border process which consumes fifo tokens, whose values are used to update the internal SystemC signal. The signal is connected to a converter port (*sca_scsdf_in*) of an inner SystemC-AMS module. In addition, BCs provide a scalable way to construct these direct connections since it does not require the SystemC-AMS kernel to be changed.

BCs provides a semantical solution for the untimed/timed connection which arises when untimed MoCs of HetSC are connected to SystemC-AMS MoCs. In the (HetSC) SR-(SystemC-AMS) T-SDF example the solution was based on sampling (read) and updating (write) signals and considering the relationship of the actual time stamps of HetSC SR slots and the cluster period of the SystemC-AMS part. The connection of SystemC-AMS with untimed HetSC MoCs is more complex because of the differences in terms or communication semantics. HetSC untimed MoCs handle a different behaviour in terms of the destructive and non-destructive semantics of the write and read accesses.

For instance, a KPN part, expects that writing to a (fifo) channel provokes the accumulation of tokens within the channel in case they cannot be immediately transferred, thus they are never lost. This is a non-destructive write semantic. It also expects to consume instead of peeking or sampling the data present in the channel, that is, a destructive read. This communication semantic, typical from untimed MoCs has to be coherently connected with the T-SDF part, which writes and reads at a fixed pace (determined by the cluster period and port rates) with a non-accumulative (destructive) write and sampling (non-destructive) read semantic. Then, some kind of adaptation has to be introduced to convert consumption in sampling (and vice versa) and production in writing (and vice versa). Actually, this type of adaptation is not comprised by any of the SystemC-AMS connection facilities. Such adaptation can be explicitly written, i.e. as a HetSC border process. The BC enables the packaging of such adaptation in a specification primitive. For instance, in the *uc_inf_fifo_sca_sdf* is a BC. In this BC, when to consume fifo tokens is defined by means of a sampling period, which, in general, can be different from the cluster period. The BC can also raise an error if the internal fifo gets empty when a new sampling is given.

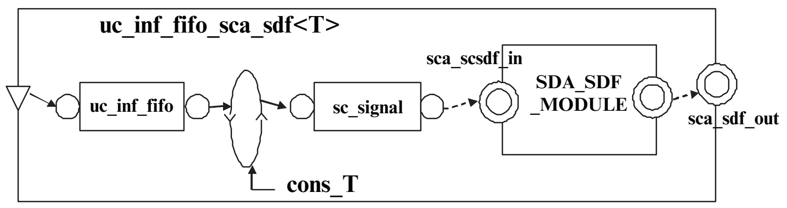


Fig. 8.5 Structure of a uc_inf_fifo_sca_sdf channel

8.4 Example

In order to demonstrate the previous general concepts, an example has been developed. This example is available in [16]. It consists of a soundboard, which is shown in Fig. 8.6. The system has an audio input and an audio output. The audio input undergoes three stage filtering. The first filter is a noise filter to remove any signal component over 22 kHz. The second one is a 10-channel equalizer. The last one is an integrator, which, at the same time, controls the general volume and filters the DC component of the audio output. The system has other inputs, as well as the audio input. A dial enables selection of the equalizer channel, while another dial tunes the gain of the selected channel in dBs (in a [-10 dB, 10 dB] range). A state display shows the current state of the equalization, while an edition display shows the currently selected channel and the currently edited equalization profile. This profile is not applied till the *set* button is pressed. Then, the state display changes to reflect this equalization profile. If the *cancel* button is pressed instead, then the edition display and the edition equalization profile return to the initial state (0 dB for every channels). Another dial controls the general gain of the system (also in a [-10 dB, 10 dB] range). It does not depend on the *set* button. That is, its change immediately updates the system gain.

Figure 8.7 depicts how this has been solved using HetSC and SystemC-AMS together. The system is enclosed in a SystemC module (*soundboard*). This top module contains another SystemC module (*panel_control*), which contains the

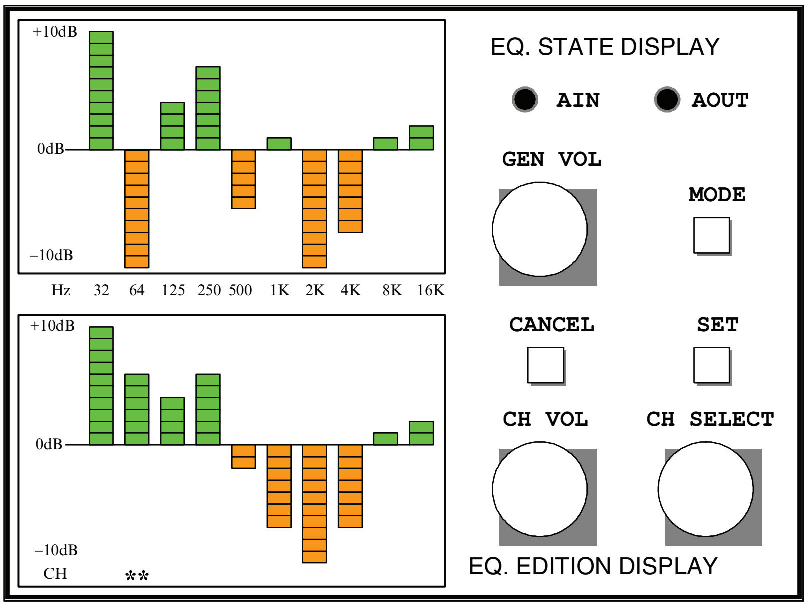


Fig. 8.6 Soundboard system

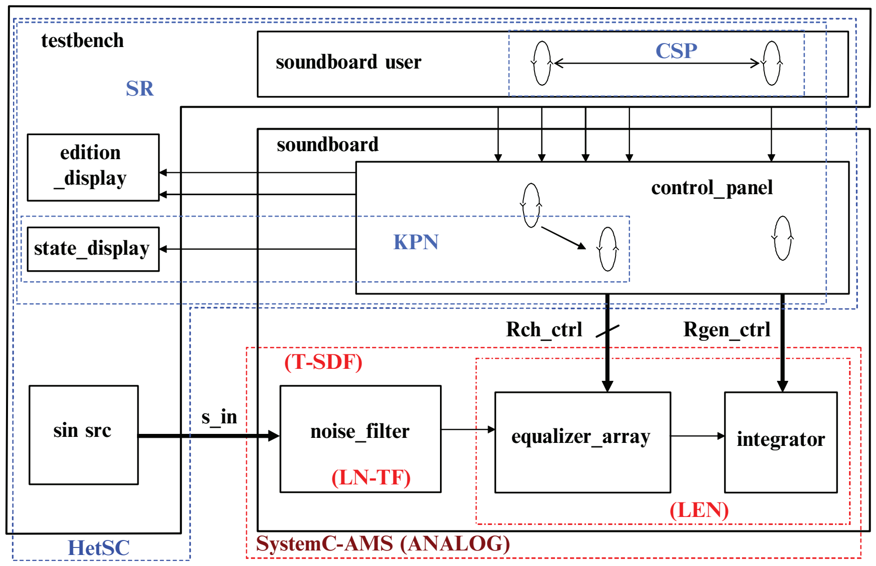


Fig. 8.7 SystemC-AMS-HetSC specification

HetSC part of the system and uses the HetSC library specification facilities. The *soundboard* module also contains three modules which use SystemC-AMS facilities. In this case, the testbench model (*testbench* module) is composed of four modules which only use HetSC facilities. In another version, part of the testbench (the audio input) was specified using SystemC-AMS specification facilities). In this sense, several combinations were possible leading to the same result.

In Fig. 8.7, the correspondence with the MoCs employed is depicted with dashed lines. In the testbench module, two processes (*left_hand* and *right_hand*) model the handling of dials and buttons of the soundboard. The two processes are synchronized through a rendezvous channel, to ensure the left hand edits the equalizer profile before the right hand pushes the *set* button and raises the general volume. Because of this, this part is a CSP network. In addition, each of the processes is an autonomous process generating a SR reactive chain. Dial turn and button press are modelled as writes to *uc_SR* channels. The reactive chain which controls the general volume is pure in that it is composed only of generator and reactive SR processes. The reactive process converts the dial events (turning left or right), which mean plus 1 dB or minus 1 dB, considering the bounds of the [-10 dB, 10 dB] range, in a control SystemC signal which affects the value of a resistor composing the integrator module. A similar thing happens with the channel equalization control. However, here there is not a pure reactive chain, since the two reactive processes are border processes, as they also write to infinite fifo HetSC channels (*uc_inf_fifo*), proper of the KPN MoC. For instance, one is used to pass the new equalization profile to the state display when the *set* button is pressed.

In the analog part, the noise filter is modelled through a SystemC-AMS SDF module (*noise_filter*). This module has an input port, to read the *s_in* external signal which provides the audio samples. It is designed as a second order Butterworth low pass filter with a cut frequency of 22 kHz, which is modelled under the LN-TF MoC of SystemC-AMS, using the ND view. The other two blocks are modelled at a circuit level, under the LEN MoC. The *equalizer_array* module encloses an array of ten equalizer cells. Each of them is an active band pass filter centred at the channel frequency. This filter is described as a circuit with three resistors, two capacitors and a model of operational amplifier (OA) which considers the gain, the input and output resistance. Each equalizer cell is instantiated taking the capacitor values as the parameter for centring each filter at the channel frequency (32 Hz for channel 0, 64 Hz for channel 1 and so on till 16 kHz for channel 9). The output of each equalizer cell is connected to a resistor instance of type *sca_sc2r*, controlled by one of the signals of the *Rch_ctrl* signal array. These resistors are connected to the same electrical output node, where the contribution of each equalizer cell is added. This node is used as input to the *integrator* module. This module is also described as a circuit which also instantiates the previously mentioned OA model, a capacitor, and a resistor controlled by the *Rgen_ctrl* signal, to control the gain of the integrator and, thus, of the whole system.

In both, the HetSC and SystemC-AMS parts, elements are employed to connect MoCs. For instance, BPs connect KPN and SR MoCs in the HetSC part, and a *sca_sdf2v* instance connects the noise filter to the equalizer array. In Fig. 8.7, the connections between the HetSC and the SystemC-AMS part have been highlighted with thicker arrows. Specifically, the audio input samples are transferred to the *soundboard* module through an instance of the *uc_inf_fifo_sca_sdf* channel introduced in the previous section. This border channel enables a direct connection between the untimed part, which generates the samples, and the SystemC-AMS input converter port of the noise filter. The connection of the SR reactive chains to the LEN part of the model is placed between the lower part of the *control_panel* module and the *equalizer_array* and *integrator* analog modules. For instance, the reactive process triggered by the turn events of the general volume dial is indeed a border process which writes the *Rgen_ctrl* SystemC signal. A similar thing happens with the non pure reactive chain, which drives an array of ten signals (each one for its corresponding equalizer channel). Each of these signals controls the value of a SystemC-AMS *sca_sc2r* primitive.

A time domain simulation and two frequency analyses have been performed. The time domain simulation is dumped to data and waveform files. The first frequency analysis is done in the middle of the time domain simulation. At this time, the soundboard response corresponds to that of the initial state (0 dB gain for every channel and for the general volume). The second frequency analysis is done at the end of the time domain simulation, once a manual configuration has been performed and the *set* button pressed. Additional results of the simulation are two data files, with the frequency response of the equalizer (thus, the equalization profile) at different points of the simulation time. The result has been post-processed with *Octave specification execution*, just to reflect the change on the equalization profile (Fig. 8.8). Other outputs of the system are two log files which reflect the activity of the displays.

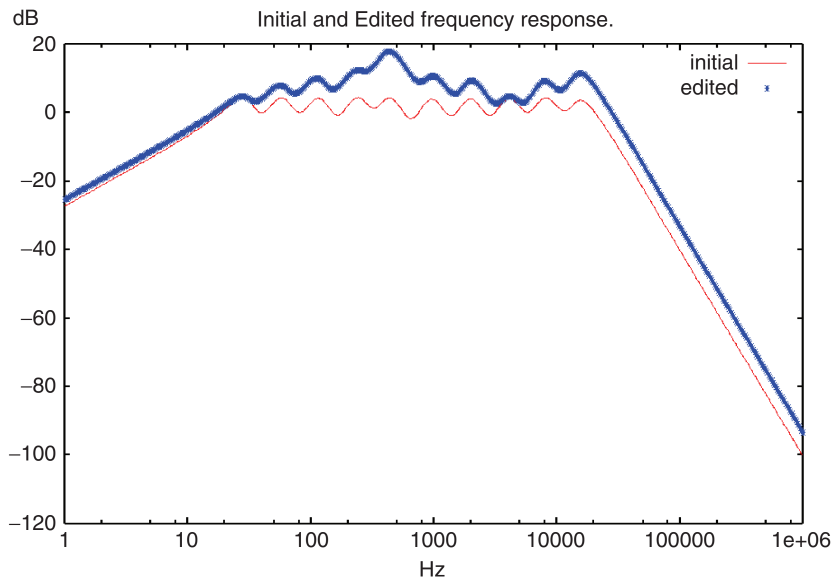


Fig. 8.8 Initial and edited frequency spectrum

Table 8.1 Host configurations where the example has been compiled and executed

OS	GCC	SystemC	SystemC-AMS	HetSC
Linux 2.6.3/32 bits	3.3.2	2.1v1	0.15RC1/RC2	1.2
Linux 2.6.3/32 bits	4.0.0	2.2.0	0.15RC4	1.2
Linux 2.6.3.2/64 bits	4.1.2	2.2.0	0.15RC4	1.2

This specification took around 2,500 SystemC code lines including test bench modules and around 30 man-hours (ignoring learning time). The simulation time was less than 53 s in an Intel PIV 2.8 GHz, Linux 2.6.3 development platform. This illustrates how fast the system-level specification and analysis of such heterogeneous system can be done using HetSC and SystemC-AMS. The example has been checked for three configurations of development platforms, reflected in Table 8.1.

8.5 Conclusions and Future Work

This work addresses how the HetSC and SystemC-AMS specification methodologies can be used together. With their cooperation, a wide range of MoCs, from untimed to analog ones, are efficiently covered. This is a key feature in enabling the early system-level specification of embedded systems. The installation and compatibility of the HetSC and SystemC-AMS libraries has been checked. Furthermore, the syntactical and semantical issues related to the connection have been discussed. SystemC-AMS is based on a timed SDF MoC, where AMS clusters can be conceptually seen as timed-clocked synchronous blocks from the DE part. SystemC-AMS provides facilities for this AMS/DE connection which are based on the sampling and update of the SystemC signal. Since HetSC MoCs are abstracted from the underlying DE strict-time MoC, the connection of any HetSC MoC with any System-AMS MoC can be reduced to a SystemC DE/SystemC-AMS connection. Thus, SystemC-AMS facilities for the DE/AMS connection can be used. Moreover, the HetSC border channel can be conveniently used to provide direct connections among specific untimed and synchronous (HetSC) MoCs and analog (SystemC-AMS) MoCs, hiding the intermediation of DE signals in the connection of MoCs that do not employ such specification primitives and encapsulating the detection of error situations which consider the cluster period, the time conditions of HetSC part, etc. The immediate evolution of this work can be found [17], where converter channels are introduced. These channels incorporate concepts of polymorphic signals [18], releasing from any manual engagement in the system refinement. As well as adaptations on the time and communication domain, converter channels also introduce adaptations at the data type domain. Finally, this work implicitly states the need for a formal environment in order to obtain a common understanding of the interoperation of this kind of methodologies.

Acknowledgments Work supported by the FP6-2005-IST-5 European project.

References

1. E.A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17(12), December 1998.
2. A. Jantsch. Modelling Embedded Systems and SoCs. Morgan Kaufmann, San Francisco, CA, June 2003. Morgan Kaufmann Publishers An imprint of Elsevier Science 340 Pine Street, Sixth Floor San Francisco, California 94104-3205 www.mkp.com
3 R. Gupta. HDL/C Interface Exploration. Tech. Report, ICS Dpt., University of California, California, 2002.
4. A. Davare et al. A Next-Generation Design Framework for Platform-Based Design. In DVCon 2007, San Jose, CA, USA, February 2007.
5. C. Brooks et al. Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java. Tech. Report, University of California, Berkeley, CA, July 2005.
6. L. Geppert. Electronic Design Automation. IEEE Spectrum, 37(1), January 2000.
7. F. Herrera, E. Villar, C. Grimm, M. Damm and J. Haase. A General Approach to the Interoperability of HetSC and SystemC-AMS. In Proceedings of the Forum of Design Languages 2007. FDL'07, Barcelona, Spain, 2007.
8. A. Vachoux, C. Grimm, and K. Einwich. Towards Analog and Mixed-Signal SoC Design with SystemC-AMS. In IEEE DELTA'04, Perth, Australia, 2004.
9. A. Herrholz et al. ANDRES – Analysis and Design of Runtime Reconfigurable Heterogeneous Systems. In Proceedings of DATE'07, Nice, France, April 2007.

10. F. Herrera and E. Villar. A Framework for Embedded System Specification Under Different Models of Computation in SystemC. In Proceedings of DAC'06, San Francisco, CA, July 2006.
11. H. Posadas, F. Herrera, V. Fernandez, P. Sanchez, and E. Villar. Single Source Design Environment for Embedded Systems Based on SystemC. Transactions on Design Automation of Electronic Embedded Systems, 9(4):293–312, December 2004.
12. H.D. Patel and S.K. Shukla. SystemC Kernel Extensions for Heterogeneous System Modeling: A Framework for Multi-MoC Modeling. Springer, July 2004.
13. H.D. Patel, D. Mathaikutty, and S.K. Shukla. Implementing Multi-Moc Extensions for SystemC: Adding CSP and FSM Kernels for Heterogeneous Modelling. Tech. Report, FERMAT, Virginia Tech., June 2004.
14. E.A. Lee and D.G. Messerschmitt. Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE Trans. on Computers, C-36(1):24–35, 1987.
15. J. Falk, C. Haubelt, and J. Teich. Efficient Representation and Simulation of Model Based Designs in SystemC. In Proceedings of FDL'06, Darmstad, Germany, September 2006.
16. http://www.teisa.unican.es/HetSC/downloads.html
17. J. Haase, M. Damm, C. Grimm, F. Herrera, E. Villar. Using Converter Channels within a Top-Down Design Flow in SystemC. The 15th Austrian Workhop on Microelectronics, Graz, Austria, October, 2007.
18. R. Schroll, C. Grimm, and Waldschmidt K. Verfeinerung von Mixed-Signal Systemen Mit Polymorphen Signalen. In Analog'05. VDE-Verlag, Berlin, Germany, 2005.