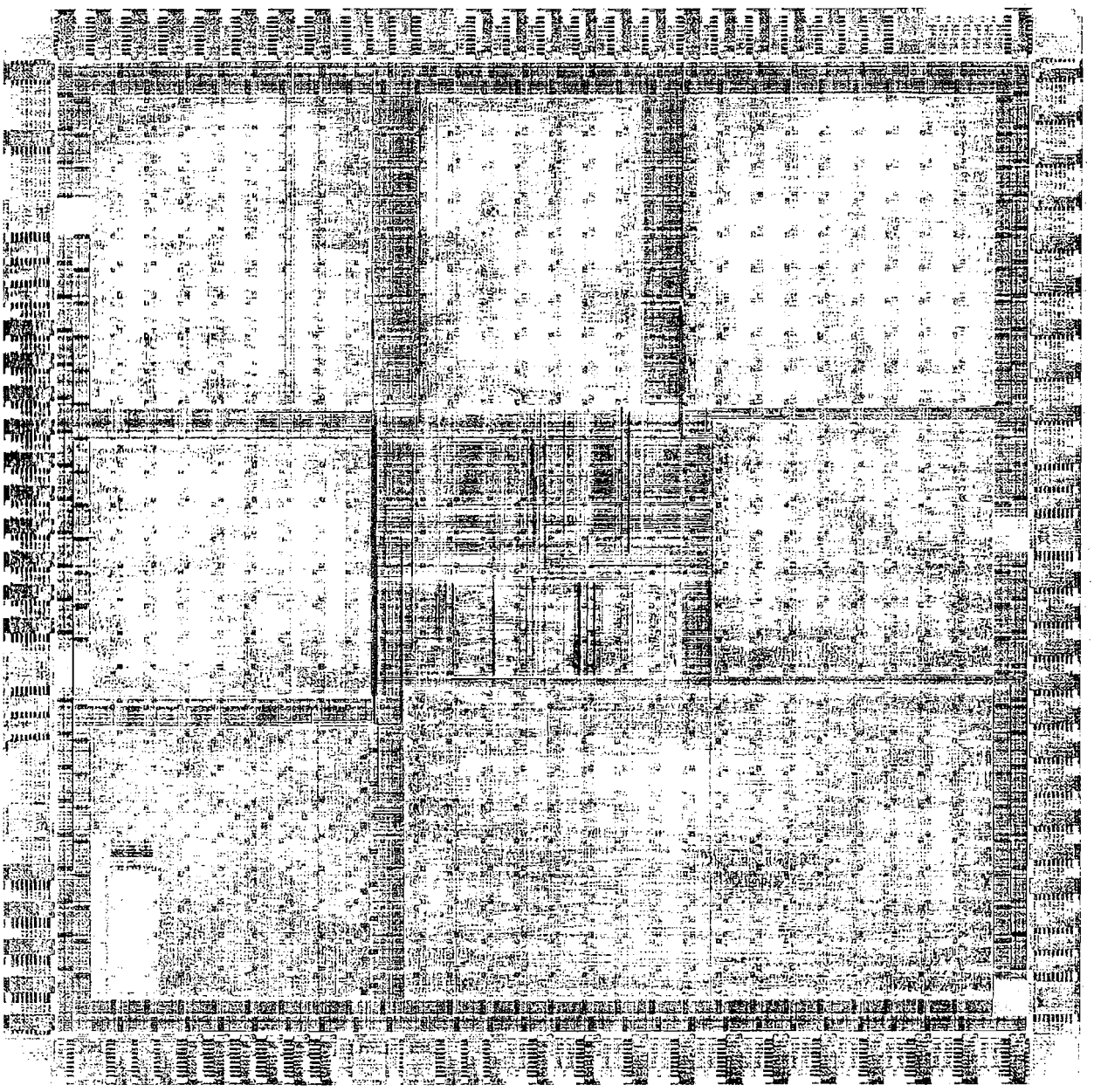
austrochip

Workshop on Microelectronics

2007



15th Austrian Workshop on Microelectronics

11 October 2007, Graz, Austria

Proceedings

TU Graz

IEEE AUSTRIA SECTION

Using Converter Channels within a Top-Down Design Flow in SystemC

Markus Damm, Jan Haase, Christoph Grimm
Institute of Computer Technology
Vienna University of Technology
{damm|haase|grimm}@ict.tuwien.ac.at

Fernando Herrera, Eugenio Villar
Microelectronic Engineering Group
TEISA Dpt. University of Cantabria
{fherrera,evillar}@teisa.unican.es

Abstract

With the advent of SystemC in HW/SW codesign, several SystemC extensions have been developed to broaden the capabilities of SystemC to mixed-signal and heterogeneous system design. The currently ongoing EU founded project ANDRES targets to integrate three such extensions, namely SystemC-AMS, HetSC and OSSS+R, within a top-down design flow of adaptive (i.e. runtime reconfigurable) heterogeneous embedded systems. These extensions implement several Models of Computation (MoCs), such that there is a need for means of converting between them. Converter channels fill this gap by being able to adapt themselves automatically to the MoCs of the system parts they are connected to. They extend the approach of polymorphic signals introduced in [3]. Moreover, they enhance the connection possibilities introduced in [8].

1 Introduction

Modern embedded systems are largely heterogeneous in the sense that they encompass the domains of digital hardware (static and/or dynamically reconfigurable), analog hardware and software. Moreover, they have to be able to adapt themselves during runtime to changing user requirements as well as varying environmental constraints. ANDRES (**AN**alysis and **D**esign of run-time **RE**configurable heterogeneous **S**ystems [1]) is a cooperative project involving DS2 and Thales Communications as industry partners, and four research institutions: OFFIS, the Technical University of Vienna, the KTH Stockholm and the University of Cantabria, whose target is the design such type of systems. The goal of the ANDRES project is to develop

- a theoretical framework for the specification of such Adaptive Heterogeneous Systems (AHES).
- a SystemC based modelling framework where the different domains are represented by specific MoCs.

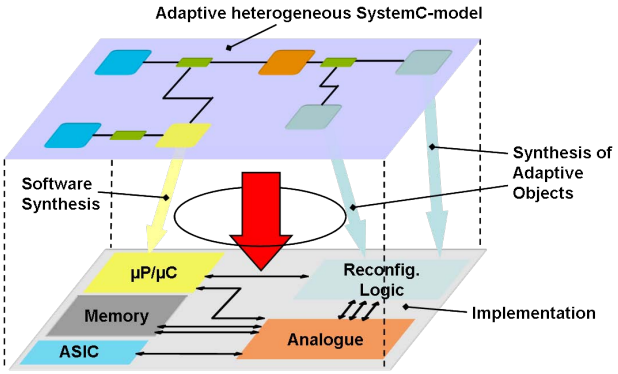


Figure 1. Mapping between different abstraction levels in ANDRES

- means to automatically synthesize the components and interfaces of AHES regarding the digital hardware and the software.

Figure 1 illustrates how an adaptive heterogeneous specification in SytemC is mapped to a runtime reconfigurable implementation.

As mentioned, a distinctive concern of ANDRES is the consideration of adaptivity in all the domains involved. However, another key and necessary subject of ANDRES is the connection and integration of the MoCs and the methodologies involved in the project. They are the following extensions (encapsulated as methodological libraries) of the SystemC language:

- **SystemC-AMS** [10] targets modelling and simulation of analog and signal processing systems. It provides the MoCs (timed) SDF and continuous time (regarding LEN - linear electrical networks).
- **HetSC** [2] is intended to model systems using different MoCs (e.g. Kahn Process Networks and Synchronous Reactive Systems), and also provides an entry point for software synthesis.
- **OSSS+R** [11] is a library for object-oriented modelling of run-time reconfigurable hardware, which

will provide direct hardware synthesis capabilities targeting dynamically reconfigurable FPGAs.

One difficulty is to bring these three libraries to work together with respect to the different MoCs involved. Since the modules in OSSS+R are basically clocked synchronous, which is a MoC provided by SystemC directly via its discrete event (DE) kernel, this is a task which mainly concerns HetSC and SystemC-AMS.

HetSC already provides the concept of *border channels* for the connection of the various MoCs defined there. Border channels make a syntactical adaptation. For instance, on one side they can offer a fifo like interface, while on the other, they offer a rendez-vous like interface. Moreover, border channels can also adapt the different semantics of connected MoCs. HetSC border channels mainly focus on the adaptations performed at the time domain. In [8], the use of border channels to connect SystemC-AMS and HetSC methodologies was explored. There, the syntactical and semantical issues of the connection of untimed (HetSC) models with the (timed) SDF of SystemC-AMS was first addressed.

For SystemC-AMS, *polymorphic signals* [3] were developed to connect modules modelled under different MoCs (LEN, SDF or SystemC-DE). Like the HetSC border channels, they have to do a syntactical and semantical adaptation. An important benefit of polymorphic signals is that they are able to select the right MoC connection in an automated fashion. This is done before the simulation starts (the *elaboration time*). The polymorphic signal detects by itself the MoCs of the modules it is connected to and provides the appropriate conversion means without any manual engagement by the designer apart from setting different options.

Converter channels are meant to unify these two approaches while also providing the feature of adaptation in the data type domain. Then, converter channels will be able to connect modules writing and reading various different data types by providing the respective data type conversion means automatically. In this way, converter channels become an advanced facility for the automatic syntactical and semantical connection of the different MoCs of the different heterogeneous specification methodologies based on SystemC.

The rest of the paper is organized as follows: Section 2 is a short introductions to HetSC and SystemC-AMS, with an emphasis on the supported MoCs. Section 3 gives a motivation for the use of converter channels driven by Mixed level Simulation and Design Space Exploration. Section 4 demonstrates how converter channels are used within a SystemC model. In section 5, it is shown how converter channels are structured internally, with additional notes on MoC and data type conversion. We conclude with section 6, where we also give an outlook on the future work.

2 HetSC and SystemC-AMS

2.1 HetSC

HetSC [2] 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 (PN), 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 [10].

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 MoCs 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.

2.2 SystemC-AMS

SystemC-AMS [10] 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 behavioural 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 enables a circuit level description, is also supported.

SystemC-AMS is extensible by other models of computation through a synchronization layer. Solvers for the supported MoCs 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 accu-

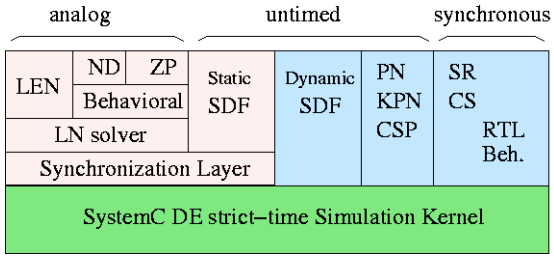


Figure 2. MoCs with SystemC-AMS-HetSC.

racy. 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 [1]. 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 communication between different MoCs (i.e. DE with SDF, SDF with LEN, etc).

As it was mentioned in the introduction, in [8] a first exploration of the connection of SystemC-AMS and HetSC was done. There it was shown that, far from overlap, both methodologies can collaborate to provide a support of a wide MoC spectrum, as it is shown in Fig. 2.

3 Mixed Level Simulation and Design Space Exploration

The converter channel's capability of *automatic* MoC and data type conversion is not simply motivated by convenience. If the only usage would be to connect system parts implemented with different MoCs and data types reflecting the heterogeneity and mixed-signal nature of the overall system, a static library of MoC and data type converters would also do the trick. Yet there are two other drivers for converter channels, namely those being indicated in the title of this section.

It is widely accepted that a top-down design approach is to be favoured over a bottom-up approach, with one benefit being the ability to do early simulation on system level. On the other hand, a pure top-down design approach is not practically applicable in every area, e.g. regarding analog systems. Here, there are too many physical factors to consider to feel save with system level simulation. Yet, a complete simulation on the physical level (e.g. using SPICE) is very time consuming and therefore costly.

This motivates the concept of **Mixed Level Simulation**. The idea is to simulate systems where different parts are specified on different levels of abstraction. That way, we get the best of both worlds: The speed of the system level simulation and the accuracy of the physical level simulation. Within SystemC-AMS, this could mean to describe a system part in the system level case with a SDF-module using transfer functions, and in the physical level case as a concrete implementation as a linear electrical network.

The downside is that these two realizations are not so easily interchanged. If, in the described setting, the system part is connected to other SDF-modules, it can only be exchanged by its LEN-variant if appropriate converters are provided, which has to be done manually. Mostly, these converters will be even part of the LEN-model, which decreases the reusability of the latter further, since if the other SDF-modules are exchanged by LEN-models themselves, the integrated converters have to be removed again. That is, a lot of work has to be invested for converters which mostly won't be present in the final design any more. Summarizing, Mixed Level Simulation requires a time consuming manual refinement of the connection, which can even perturbate the code of each part, and, thus, its reusability.

Converter channels will provide the conversion features needed to eliminate these disadvantages. The designer might just need to steer the behaviour of the converter channels with a few options, which does speed-up iterations in the refinements needed in a **Design Space Exploration** (DSE) methodology. In addition, since the conversion is hidden within the channel, it won't interfere with the reuse of partial design blocks.

The motivation for the data type conversion capabilities of the converter channels are twofold. It is a natural necessity regarding the task described above, since for example, the data type double is the standard data type of the LEN domain in SystemC-AMS. But it also comes in handy regarding DSE. For example, a designer might want to specify and simulate a design where he might want to vary the bit widths of the signals read and/or wrote by certain components, e.g. to find optimal solutions regarding the Signal-to-noise ratio when connecting parts from the analog domain to parts of the digital domain. In such a setting, a converter channel would play the role of an A/D resp. D/A converter. Once a final design decision is made, a concrete converter can be used to replace the converter channel. With regard to polymorphic signals, it was even investigated, in how far this process can be done automatically by synthesising A/D and D/A converters from polymorphic signals [4].

4 Using a converter channel

Before we describe how a converter channel works, we demonstrate it's usage. A converter channel is instantiated with up to six template parameters:
converterchannel <DT_W, DT_R1 ,..., DT_R5> name;
The first parameter, DT_W, is mandatory and denotes the data type of the writing port the signal is going to be attached to. The same time, DT_W then can also be used

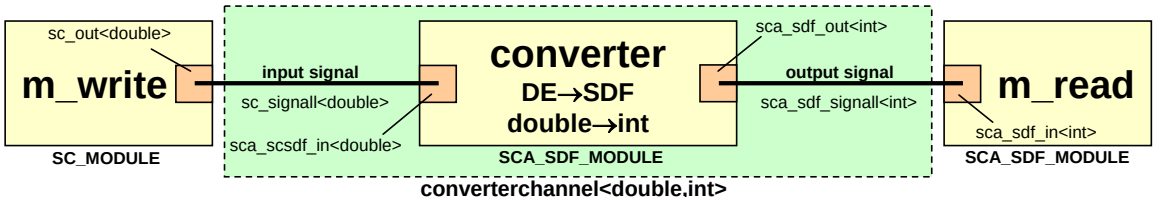


Figure 3. example of the internal structure of a converter channel

as a data type of a reading port where the signal is going to be attached to. The parameters DT_R1 to DT_R5 are optional and denote additional data types on the reading side.

The connection of the converter channel to module ports has to be done by named mapping and works like in the following example, where a SDF-module producing double values is connected to a SDF-module reading integer values and an ordinary SystemC-module (DE) reading double values:

Listing 1. Connecting a converter channel

```
converterchannel<double, int> sig;

producer_sdf_double prodsdf("prodsdf");
prodsdf.out(sig.source_sdf());

consumer_sc_double conssc("conssc");
conssc.in(sig);

consumer_sdf_int conssdf("conssdf");
conssdf.in(sig);
```

As Listing 1 shows, the usage of a converter channel differs a little bit from the usage of an sc_signal regarding the connection to the writing port, which contains a declaration of the MoC used on the writing side: instead of prodsdf.out(sig), the code line prodsdf.out(sig.source_MOC()) has to be used, where MOC denotes the MoC on the writing side. In contrast, the connection to ports of reading modules works as usual; in particular there is no need to declare the MoC of the reading module. The designer has to make manual changes at a converter channel only if

- the MoC of the writing module is changed; then the connection of the converter channel to the port has to be changed, for example from module.out(signal.source_sdf()) to module.out(signal.source_sc()) (sc represents the standard SystemC DE MoC).
- the data type of a port of a reading module is changed and is not already a template parameter in the instantiation of the converter channel; then it has to be added. So, for example, the code of the instantiation could change from converterchannel<double, int> sig; to converterchannel<double, int, sc_bv<8>> sig;

Obviously, it should be possible to set options on the converter channel. Currently, one option is implemented regarding data type conversion: If the writing side of a converter channel is of type double, and there is, for example, a sc_uint<N> port on the reading side, this situation has the potential for information loss if N is too small or if the writing side provides negative values. For such a case, the converter channel offers a method

sig.setRangeScaling(min, max)

to adapt the value range of the writing side to those of the reading side (the semantic is detailed in section 5.2). For applying this function the designer has to know (or has to have at least an idea of) the range of the input signal. However, if the values of the writing side turn out to exeed the interval [min,max] during the simulation, warnings are produced.

5 Internal structure and operation

In this section, we give some detail on how the code described above is processed, i.e. handled internally in terms of signals and converter modules. W.l.o.g., let us assume that a converter channel connects two modules m_write and m_read having a port out and in respectively. Now, four cases can occur:

1. The MoCs of the modules as well as the data types of the ports disagree.
2. The MoCs of the modules are different, but the data types are the same.
3. The MoCs of the modules agree, but the data types of the ports don't.
4. Both MoCs and port data types agree.

In case 1, the converter channel instantiates two appropriate signals (the input and the output signal), which then are connected to the ports of m_write and m_read respectively. These, in turn, are connected to an appropriate converter module. An example is shown in Figure 3: m_write is an ordinary SystemC discrete event (DE) module having a port out of type sc_out<double>, and m_read is a SystemC-AMS synchronous dataflow (SDF) module having a port in of type sca_sdf_in<int>.

To achieve conversion between SystemC-DE modules and SystemC-AMS SDF modules, the SystemC-AMS library provides sca_scsdf_in and sca_scsdf_out ports, which can be used within SDF-modules to connect to DE-signals

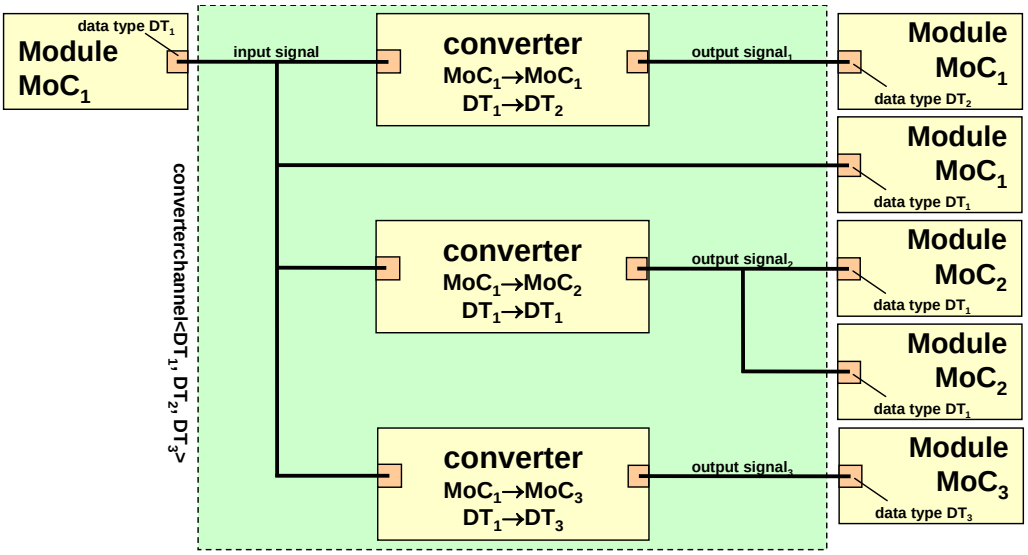


Figure 4. Internal structure of a converter channel connected to multiple reading modules

(i.e. sc_signals). Therefore, a converter module from DE to SDF can be realized by a SDF-module having an sca_scsdf_in input and a sca_sdf_out output. Then it can be connected to m_write and m_read with a sc_signal and sca_sdf_signal respectively.

Data type conversion is now done by reading the input signal, passing its value to an appropriate conversion function, and writing the converted value to the output signal. The data type conversion function here is inherited by a special data type conversion class which has templated specialisations for each data type pair. Hence, the conversion modules are implemented using two separate levels: MoC conversion and data type conversion.

In case 2, the procedure is pretty much the same; there is just no need to call a converter function for data type conversion, and the converter module just passes the value of the input signal to the output.

Case 3 is also pretty similar, with the difference that all ports and signals used are of the same type; in particular the converter module belongs to the same MoC as the two modules to be connected. Like in case 1, the input signal is read, and the converted value is written to the output signal.

Obviously, the most simple case is case 4, where there is no need for any conversion at all. Here, the converter channel simply generates only the appropriate input signal, and connects it to m_write and m_read directly without any converter module in between. E.g. for two DE-modules with ports writing resp. reading integers, an sc_signal <int> is created.

In the general case, a converter channel will be connected to several readers. Here, apart from the input signal, several converters and output signals are created. Each converter is then connected to the input signal and to an own output signal. Each reading module is connected to the appropriate output signal. If a reading module happens to agree to the writing module regarding MoC and data type, it is directly connected to the input signal. Figure 4 shows an example. The capability of the converter channel to have readers with several data types is also an extension of the concept of polymorphic signals, where data type conversion was also included, but the reading modules have to share the same data type.

We will say a bit more on data type conversion in section 5.2, but it has to be noted here that having several different data types on the reading side adds no additional complexity to the problem. This is not true, however, with having several different MoCs on the reading side. While it is no problem from the syntactical point of view (this problem is solved with the approach described above), it is apparent that having a certain mix of MoCs on the reading side won't make sense for certain writing side MoCs. An obvious example is having a non-destructive (i.e. fifo) writer and one destructive reader together with several non-destructive ones. However, we will not go into detail here on this subject. For section 5.1, we assume a one-to-one connection.

5.1 MoC-conversion

MoCs conversion performed by converter channels is a set of actions to adapt time, communication and computation domains of the MoCs connected, with time adaptation being the most important part. In [12], several types of MoC connections were distinguished with regard to the time domain. The basic observation is that when two MoCs are connected which handle time at a different detail level, there is an effect of time information injection from the most detailed one to the less detailed one. The way time is handled is often related to other aspects associated to communication semantics. For instance, it is usual to associate consuming reads and non-destructive writes to untimed MoCs, while destructive write and non-consuming reads semantic of HDL signals (like the sc_signal channel semantic) is proper of timed-clocked MoCs.

In previous works, semantical issues dealing with untimed-untimed [12] and untimed-synchronous [13] MoC connection have been addressed. Here, a first approach to the untimed-timed and the timed-timed MoC connections will be considered, where the untimed-timed case refers to the connection of untimed KPN and PN MoCs (HetSC) with the timed SDF MoC (T-SDF) of SystemC-AMS. The timed approach of SDF can actually be considered as a discrete time MoC, since each cluster execution has a specific SystemC time stamp. The timed-timed case occurs when the addition of some kind of strict-time information to the KPN/PN network connected to the T-SDF part has to be considered. We therefore have four cases to consider:

1. KPN/PN→T-SDF
2. T-KPN/T-PN→T-SDF
3. T-SDF→KPN/SDF
4. T-SDF→T-KPN/T-PN

For each case, we give the semantic of the converter channel regarding the writing and the reading side. This includes the problem of time information injection for the timed-untimed connections. It also has to be detected if the overlap of untimed/timed write and untimed/timed read semantics generate any inconsistency and, in that case, which adaptations have to be performed by the converter channel.

We start with the KPN/PN→T-SDF connection. From the writing side, the KPN/PN character of the untimed network does not carry semantical problems. The converter channel can be either configured as an unbounded fifo (like the HetSC uc_inf_fifo channel), such that the untimed part will never block. Or it can be configured as a bounded fifo (like the HetSC uc_fifo channel), such that the untimed part can block. In any case, there is no loss of data. Syntactically, this is achieved by connecting the port out of a writing module $module$ with the code line

module.out(sig . source_fifo (size));

to a converter channel sig. If the integer parameter $size$ is omitted, the internal buffer of the converter channel will be unbounded.

The time domain and communication domain adaptations are closely related. The KPN/PN not only expects to do a non-destructive write, but also a destructive read from the reading side which frees space in the internal buffer. Therefore, data consumption is needed on the reading side. This might seem like a contradiction, since, from the T-SDF side, the read is non-consuming (as well as non-blocking). However, the read is non-consuming only for the reads done at the same time stamp (t). From a sampling time stamp (t) to the next sampling time stamp (t+T), after a sampling period T, a data token is deallocated and frees space in the internal buffer, establishing a semantical coherence.

This means that on the T-SDF reading side, a token is indeed consumed, but can be read as many times as wanted, during a sampling period T. In this way, time information is injected in the untimed part. If, for instance, the converter channel has internal buffer size 2, and the sampling period is 1ms, then the original untimed part will be able to initially write up to 2 tokens and compute until the point when it tries to write the third one. Then, it will have to wait at least 1ms.

However, corner conditions can still lead to inconsistencies. In the KPN/PN→T-SDF case, the inconsistency comes when the internal buffer is empty and the read access of the T-SDF part has to consume a token. This inconsistency does not exist, for instance, when a SystemC-AMS SDF module has to read from a DE signal (sc_signal), since the internal buffer of the DE signal channel has always a valid value (the $current$ value), such that this situation can be understood as a sample & hold of the DE signal. However, the behaviour when reading from a fifo like channel is not defined. The converter channel supports several possibilities if its internal buffer is empty:

- **consume&error:** an error is raised and the simulation is stopped.
- **consume&constant:** a prefixed value is returned. This value remains constant during the simulation. A default value can be defined for each data type (i.e., '0' for boolean or 'X' for sc_logic), but can be overwritten by the designer.
- **consume&hold:** the last value read is returned. It can be seen as a variation of the previous one, where the returned value, instead being constant, is the last token which could be consumed.

That is, in the first case, the inconsistency provokes an error because the read from the T-SDF side is assumed to be allways consuming. In the last two cases the inconsistency is resolved (and a warning can be raised), since the read access from the T-SDF part is non-consuming when the buffer is empty. The designer can choose between the three cases by setting an option (a final syntax has not been fixed yet). The consume&error semantic is set by default. This ensures that the designer will be indicated and, thus aware, of that this corner situation is given in the specification. simulation.

The timed-timed T-PN/T-KPN → T-SDF connection can be treated basically like the PN/KPN → T-SDF case, including the handling of empty internal buffers. In particular, there will be no syntactical difference. The only thing that changes is the set of causes for empty internal buffer. In the untimed-timed case, a simple cause could be that the untimed part is a finite process which produces a finite amount of tokens. Then, if the simulation time is long enough, there will be a moment when the internal buffer gets empty and there is a trial to read it. Another more tricky cause in the untimed-timed case is a deadlock or a partial deadlock in the KPN/PN side involving the process which writes the converter channel. In the timed-timed case, there is also the additional situation that the production rate of the producer timed part is insufficient

compared to the T consumption throughput of the T-SDF part. The variety of reasons for empty internal buffers makes it obvious that there is no a-priori intuitive solution to this problem, and the designer has to decide which of the three options offered by the converter channel suits it best.

Let's now turn to the timed-untimed T-SDF → KPN/PN conversion, where we have to introduce the FIFO semantic from the reading side. Here, the T-SDF cluster behaves as a kind of master which triggers and transfers tokens to the untimed part at the pace of the sampling time. We decided to use an unbounded internal buffer as the default case, but the designer can also limit its size by calling a suitable method of the converter channel (syntax not decided yet).

The modification of the sense of data transfer changes the location of the semantic inconsistency. While the empty buffer does not represent a problem because the PN/KsimulationPN part remains blocked till the next data arrival, the full buffer condition can be problematic, since the T-SDF has to write at the unstoppable T pace. This inconsistency only appears when using a *bounded* internal buffer. The full buffer condition can appear due to several causes, similar to the ones presented for the KPN/PN → T-SDF case and the T-KPN/T-PN → T-SDF case. Namely, the PN/KPN part can be a finite process that finishes its execution after having consumed a finite number of tokens. It can also present a deadlock or partial deadlock preventing the consumption of tokens. For the case of a timed KPN/PN part, the consuming task can also present a throughput slower than the production throughput of the T-SDF part.

Therefore, in a T-SDF → (timed or untimed) KPN/PN connection, the write access semantic of the converter channel allocates a new token in the internal buffer whenever there is room for it. The semantic alternatives come in case there is a new write access and the internal buffer is full. Then, the following behaviours are foreseen:

- **produce&error:** an error is raised and the simulation is stopped.
- **produce&discard:** the write access results in discarding a token. In this case a warning will be produced.

This conversion can be configured attending the policy applied to decide which data token is discarded. Some immediate possibilities would be:

- **discard oldest:** the next token to be read by the consumer (at the "beginning" of the buffer) is removed, the buffer content is shifted forward and the current token (passed as parameter of the write access) is added to the "end" of the buffer.
- **discard newest:** the last token of the internal buffer is overwritten by the current token.
- **discard current:** the current token is discarded and the internal buffer remains untouched.

5.2 Data type conversion

Data type conversion is always a problematic task, since it involves potential loss of information. An obvious case is the conversion from "continuous" data types like double or float to discrete data types like int or sc_bv. Currently, the input value is rounded; future implementations will offer an option for using the floor or the ceiling function instead.

If, for example, the converter port is bound to a sc_uint <N> port on the reading side, the semantic of the range scaling function for the double to sc_uint <N> conversion is as follows: the values provided by the writing side are scaled in such a way that the value min maps to 0, the value max maps to $2^N - 1$, and everything in between maps to a value within the interval $[0, 2^N - 1]$ in a linear fashion, using the following scaling function:

$$f(x) := round((x - min)/(max - min) * (2^N - 1))$$

The round function receives a parameter in the range $[0.0, 2^N - 1.0)$. This is a real value (double type), product of the $(x - min)/(max - min) * (2^N) - 1/2)$ operations (also of double type). This value is mapped to a a natural number ($sc\_uint < N >$) through the round function. By using value range scaling when needed as described above, cases like this can be handled in a consistent and well defined manner.

Other cases are not handled so easily. This is especially true for the SystemC data type sc_logic, which has four values: SC_LOGIC_1 and SC_LOGIC_0 (representing Boolean true and false), as well as SC_LOGIC_X (representing an unknown or undefined value) and SC_LOGIC_Z (representing high impedance). How to convert the last two values to double, for example, is absolutely non-intuitive. Since SC_LOGIC_Z stands for an infinite resistance, it should be mapped to ∞, which is not possible. An alternative could be to use the highest possible value which can be represented by a double instead, but that depends on the computing environment used, and it is questionable if such an interpretation of SC_LOGIC_Z makes sense for simulating purposes. Regarding SC_LOGIC_X, there is no interpretation of this value as a double which is remotely intuitive.

Therefore it is apparant that conversions between certain data types will not be included in the standard capabilities of the converter channels. The idea of converter channels is to offer obvious and intuitive conversion means with respect to MoCs and data types. Making arbitrary or artificial assumptions regarding non-intuitively convertible data type pairs has only the potential to introduce erroneous (and also non-intuitive) behaviour to a design, which usually will be hard to debug since the source of the error will be hidden within a channel. It is planned, however, to offer the designer the possibility to pass his own conversion function to a converter channel in such non-intuitive cases. For instance, a user which wants to optime the accuracy could use a different range scaling

function for the double to $sc\_uint < N >$ conversion:

$$f(x) := round((x - min)/(max - min) * (2^N) - 1/2))$$

This enables that the designer can ensure the matching of the conversion behaviour with that of any implementation.

6 Conclusion and future work

In this paper, an overview on the ongoing work on converter channels has been given. Converter channels will provide the designer with a convenient tool to connect system parts using different MoCs and data types. They will quickly and safely solve the syntactical and semantical adaptations that are required by mixed-level simulation and design space exploration. Specifically, the syntactical solution of converter channels will save the manual refinement of those connections. In addition, converter channels will provide suitable conversion semantics, which is not always straightforward, since, they deal with adaptations at the time, communication and data type domains, which are often time consuming for a user often more worried by other aspects of the specification. Here, these adaptations have been illustrated through several details on the usage of a converter channel and it's internal structure, as well as, through a detailed explanation of the semantical issues of the (untimed and timed) KPN/PN↔T-SDF connection, and of data type conversion.

Future work will include finalizing the support for further MoC connections, e.g. the connection to linear electrical networks (LEN). Regarding the connection of LEN to SDF and discrete event modules, this is a straightforward task since SystemC-AMS provides all kind of converter facilities for this.

It has to be noted that, despite of the focus of ANDRES on run-time reconfigurable systems, converter channels aren't meant to adapt to changing communicating facilities during simulation time, e.g. to react to a port which changes its bitwidth. This would go beyond the capabilities of SystemC itself in its current state. Nevertheless, this is an aspect which could be of interest in the future, since this also would allow for an automatic support of a dynamic brand of mixed-level simulation. Here, the abstraction levels of certain system parts could even be changed during simulation time if the level of detail needed changes in certain situations.

References

[1] A. Herrholz, F. Oppenheimer, A. Schallenberg, W. Nebel, C. Grimm, M. Damm, F. Herrera, E. Villar, A-M. Fouilliart, M. Martinez. "ANDRES- ANalysis and Design of run-time REconfigurable, heterogeneous Systems" Workshop on "Adaptive Heterogeneous Systems-On-Chip and European Dimensions" in the Design Automation and Test in Europe 2007, DATE'07, Nice, France. DATE 07 Friday Workshop Notes pp. 64-71.

[2] F.Herrera and E.Villar. A Framework for Embedded System Specification under Different Models of Computation in SystemC In Proceedings of the Design Automation Conference, 2006. San Francisco. July 2006.

[3] R. Schroll. Design komplexer heterogener Systeme mit Polymorphen Signalen Dissertation, Institut fr Informatik, University Frankfurt am Main, 2007

[4] C. Grimm. Design of Analog and Mixed-Signal Systems: Still black Magic? In: Advances in Specification and Design Languages for SoC., Springer-Verlag, Wien - New York, 2006.

[5] C. Grimm, F. Brame, R. Schroll, K. Waldschmidt. Top-Down Design analog/digitaler Systeme mit SystemC-AMS In: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, 2007, pp. 131 S. 141.

[6] C. Grimm, R. Schroll, K. Waldschmidt, F. Brame. Mixed-Level Simulation heterogener Systeme In: Multi Nature Systems, S. 5, Erfurt, 2007.

[7] C. Grimm, M. Damm, F. Brame, J. Haase. Top-Down Refinement of Embedded Analog/Mixed-Signal Systems In: Proceedings of Analog/Mixed-Signal Systems Days, Silicon Saxony, Dresden, May 2007.

[8] F.Herrera, E.Villar, C.Grimm, M.Damm, J.Haase. A general approach to the interoperability of HetSC and SystemC-AMS To appear in Proceedings of the Forum on Design Languages, 2007. Barcelona. September 2007.

[9] 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.

[10] A. Vachoux, C. Grimm, and K. Einwich. Towards analog and mixed-signal SoC design with SystemC-AMS. In *IEEE International Workshop on Electronic Design, Test and Applications (DELTA'04)*, Perth, Australia, 2004.

[11] A. Schallenberg, F. Oppenheimer, and W. Nebel., Designing for dynamic partially reconfigurable FPGAs with SystemC and OSSS. Proceedings of the Forum on Design Languages, 2004, Lille.

[12] F.Herrera, P.Sanchez and E.Villar. Heterogeneous system-level specification in SystemC In Advances in Design and Specification Languages for SoC. P.Boulet. (Ed.), CHDL Series. Springer. 2005.

[13] F.Herrera and E.Villar. Mixing Synchronous Reactive and Untimed MoCs in SystemC In Applications of Specification and Design Languages for SoCs". A. Vachoux (Ed.), CHDL Series. Springer. 2006.