Application of SystemC/SystemC-AMS in 3G Virtual Prototyping

Tao Huang
Infineon Technologies
Germany

Tao.Huang@infineon.com

Stefan Heinen
Infineon Technologies
Germany

Stefan.Heinen@infineon.com

Abstract

In this paper, we describe the application of the Timed Data Flow (TDF) feature of the recently released SystemC-AMS standard in the context of a 3G modem Virtual Prototype.

By realizing the data flow of our 3G hardware models based on SystemC-AMS TDF we demonstrate the utilization of TDF in a complex virtual prototyping platform and elaborate how a basic shortcoming currently present with TDF can be worked around.

Finally, we perform an ad-hoc comparison of the scheduling speed of SystemC-AMS TDF, plain SystemC and a commercial data flow simulation tool.

1. Introduction

System Modeling and Virtual Prototyping have been attracting growing attention in the semiconductor industry over the past years. Today, Virtual Prototyping plays an important role in pre-silicon software development and verification and allows for significantly shortened time-to-market cycles.

The initiator for the success of System Modeling was the development of SystemC as a means to describe the timed behavior of hardware using the high-level programming language C++.

With SystemC, hardware units can be efficiently modeled and meanwhile the standardization of interfaces like TLM2 allows exchanging of models in an ever-growing community.

One domain where SystemC was less successful so far is the algorithmic elaboration, taking place in a product development phase where neither timing nor architecture is defined and engineers want to concentrate only on the performance of their algorithms. For this task, developers consider SystemC's event-driven simulator usually as not efficient enough and fear that SystemC-based modeling involves a lot of simulation-related details, which distract the attendance from the actual topic of algorithm development. Tools like Matlab, SystemStudio or SPW, which allow for modeling at higher abstraction levels, are therefore dominant in this area.

So far, developers who wanted to combine the advantages of SystemC based control flow modeling with abstract data flow models, were therefore forced to resort to usually cumbersome co-simulations with aforementioned tools. The release of SystemC-AMS Timed Data Flow (TDF) now opens the chance to efficiently model the algorithmic data flow directly in SystemC, efficiently both in terms of simulation speed and modeling effort.

In our contribution, we describe the application of SystemC TDF in the complex Virtual Prototyping context of Infineon's 3G modem solution. This is a perfect test field since the 3G physical layer consists of a couple of number-crunching hardware units, whose data paths are currently realized in our Virtual Prototype by co-simulations with a commercial tool.

The rest of the paper is structured as follows. In Section 2 we describe our model-driven design flow and demonstrate in Section 3 the applicability of SystemC TDF in the Virtual Prototyping environment by integrating a SystemC TDF based data path model in the system context of the 3G Virtual Prototype.

Section 4 gives a brief ad-hoc overview about the scheduling performance of TDF compared to plain SystemC and a commercial tool.

2. System Level Modeling Flow

In the world of digital communication the development of systems is separated into several phases. The beginning of the development cycle is characterized by scientific engineering work on algorithms, including their development, evaluation, selection and optimization. Simulation at this stage takes place at a very abstract level since the focus is on algorithmic performance and high simulation speed rather than on architectural details. Therefore, stream-driven data flow modeling is widely used in this development phase.

As soon as the algorithmic exploration phase is finished, the true timing behavior of different functional units interacting with each other in the whole system gets into the focus. Naturally, timed event driven simulation, as e.g. provided by SystemC, is better suited for such kind of control flow modeling [1].

2.1 Data Flow Modeling

One main idea of the SystemC-AMS extension on top of SystemC is to provide the essential modeling formalisms required to support Analog Mixed Signal (AMS) behavioral modeling at different levels of abstraction. Timed Data Flow (TDF) is one of the models of computation, which significantly simplifies the modeling of abstract data flow graphs.

TDF uses a dedicated scheduling mechanism embedded into the SystemC simulator, which provides high simulation speed thanks to a static schedule being computed at the beginning of a simulation. I.e. other than in event-driven simulation, where a dynamically growing event list for each time instant determines what has to be done next, in a static schedule the order of execution is predetermined right from the beginning and does not change throughout the simulation. This becomes possible by restricting to fixed rate relationships between the TDF processing entities.

As any type of C++ data can be communicated over TDF nets, the above fixed rate limitation can be relaxed in the following way: if the dynamically growing/shrinking *vector*<> from C++'s stan-

dard template library is used, also variable data rates become possible. Each time a TDF module is activated, it receives and processes a given number of data items provided by the previous TDF module and sends the processed data to the next module.

In Figure 2-1, a schematic block diagram of a simple linear data path graph is shown. Each data processing step is implemented in a dedicated TDF functional module. The whole data processing chain can be instantiated in a top level SystemC module.

The scheduling constraints are given by the modeler in terms of the parameters "time step" and "rate" for each module / port of the data flow graph. The top level module provides standard SystemC ports as interfaces to the outside world such that it can seamlessly be embedded in an overall SystemC simulation.

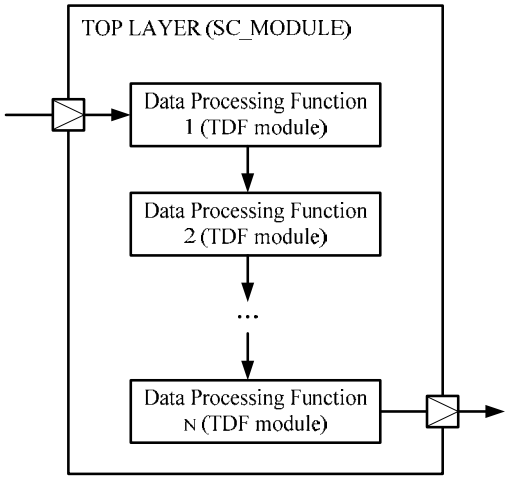

Figure 2-1: Example of data path models

2.2 Time Behavior Modeling

Developing embedded software on virtual hardware implies more than just functional correctness of the models. Also the time behavior modeling is an important issue due to the real time interaction of different hardware units as well as between software and hardware.

Event-driven, time aware simulation as provided by SystemC is an adequate basis for such modeling tasks. A mandatory requirement to an overall development flow is that the functional models from Section 2.1 can be reused one-to-one in order to minimize modeling effort and to avoid the risk of conversion errors.

In our research case, the reuse is achieved by integrating the TDF data flow graph into a SystemC control flow model. Figure 2-2 illustrates the structure of such an integrated hardware model. The unchanged data flow model from Figure 2-1 serves now as the hardware unit's data path, which exchanges data and register information with the surrounding event driven simulation domain via buffered ports.

Since the data path model is an outcome of the algorithmic exploration phase, where timing is not yet considered, usually no meaningful time annotation comes along with such models. As pure functional models they would basically run in zero time from the SystemC simulator's point of view. Therefore, methods of calling and controlling a data flow simulation under consideration of the temporal dimension are required.

Two essential components are added to make the model applicable in a timed system model context. First, a virtual bus interface providing a time-aware transaction level interface for register read/write accesses as well as controllers for reset handling, clocking and interrupt generation, and second, a state machine controlling the timely reading/writing of inputs/outputs and the invocation of the TDF simulation as indicated by the dashed arrows in Figure 2-2.

Figure 2-2 schematically sketches the structure of a state machine having four states. In the *Idle* state, the module waits for being activated. Once triggered by a register access or some other specific event, a transition into the *Read* state takes place. At this state, the required input data and register information are extracted from the bus interface and stored in a buffer which serves as the input to the TDF model.

As soon as the scheduling conditions for the data path processing are fulfilled, the state machine transits to the *Run* state, in which the TDF simulation is activated. The inputs in the buffer are processed and the results of the data path computations are written to an output buffer.

Finally, in the *Output* state, the data processed by the TDF model is transferred to the outside world and status registers are updated. Here is where time annotation comes into play: As the model has to reflect the actual computation time of the hardware's data path, the *Output* state is taken only after the annotated computation time, such that the output data becomes present in coincidence with the real hardware. This is important e.g. to detect any premature software accesses to the data path results.

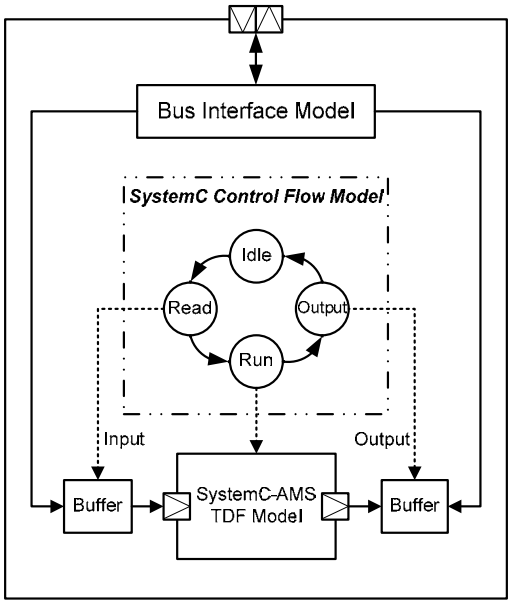

Figure 2-2: Structure of functional units

2.3 TDF Controlling

One problem comes along with using a TDF model in the above way. As the TDF simulator is realized on top of the SystemC simulator kernel, there is only one unique simulation context, which means when *sc_start* is executed, both SystemC and TDF simulations start simultaneously. This has to be avoided because the TDF modules are serving in our application as data processing

units that only may be started when the control state machine allows it.

Either a start/stop controlling mechanism or an independent simulation context for TDF modules would be desirable. Unfortunately, in release 1.0beta1 of SystemC-AMS, which we used as the basis for this work, such features were not yet implemented.

In the next two subsections, we investigate two workarounds to overcome this current shortcoming of SystemC-AMS TDF.

2.3.1 Dynamic Time Step (DTS)

When integrating TDF into SystemC a mapping between TDF ports and SystemC discrete event (DE) ports has to be accomplished. The SystemC-AMS simulation kernel is using its own simulation time t_{tdf}, which is different from the SystemC discrete event simulation time t_{DE}. If a pure SystemC-AMS TDF model is used, the SystemC-AMS simulation kernel is always blocking the DE kernel, so t_{DE} does not advance at all. For this reason, SystemC-AMS provides converter ports, named

sca_tdf::sca_de::sca_out<T>,
sca_tdf::sca_de::sca_in<T>

for the connection and synchronization between TDF ports and DE ports. When converter ports are accessed from the *processing()* method of a TDF module, the SystemC-AMS simulation kernel stops the execution of the static schedule and gives control to the SystemC simulation kernel so that the SystemC model can execute until t_{DE} equals t_{tdf}, so that the DE and the TDF models get synchronized with respect to their interface behavior [2].

The main idea of Dynamic Time Step (DTS) is to "sleep" and "wakeup" the TDF simulation by dynamically adjusting the time step of the TDF module. Table 2-1 shows a template of a DTS-enabled TDF model. At the instantiation phase the method *set_attributes()* is executed, which sets the time step of the module initially to a value much greater than the actual simulation time, 10000 seconds here. This lets the TDF module virtually "sleep" at the beginning of the simulation.

Table 2-1: TDF module with dynamic time step settings

```
/* tdf_module.h */

SCA_TDF_MODULE ( tdf_module ) {
    ...
    sca_tdf::sca_de::sca_in<int> vdd;

    int vdd_current;
    ...

    void set_attributes () {
        //sleep
        set_timestep ( 10000,SC_SEC ) ;
        vdd_current = 0;
    }

    void data_processing_function () {
        //actual data processing part
        ...
    }
```

```
    void processing () {
        if ( vdd.read() == 1
                && vdd.read() != vdd_current) {
            //wake up
            sc_time nstep( 1,SC_PS ) ;
            sca_next_max_time_step ( nstep ) ;

            data_processing_function ();
            vdd_current = vdd.read();
        }
        else {
            //sleep again
            sc_time nstep( 10000,SC_SEC ) ;
            sca_next_max_time_step ( nstep ) ;
        }
        sca_synchronize_on_event ( vdd ) ;
    } //end processing
}; //end module
```

To wakeup the TDF module, a dedicated converter port *vdd* is introduced and the model is synchronized to its default event using *sca_synchronize_on_event()*. When the converter port changes its value, the module will be started and the default processing method *processing()* is executed. If *vdd* carries a "1", the TDF time step is set with *sca_next_max_time_step()* to the smallest possible value, 1 pico second here, in order to let it virtually finish in "zero time". Next time the TDF scheduler invokes the model, the module time step is set back to 10000 seconds so that the module "sleeps" again.

The above procedure works under the assumption that the considered TDF model has to be executed exactly once per invocation of the data path processing. If the rate relations inside the data path chain require multiple executions of certain TDF models, additional measures have to be taken.

Another less attractive point of this method is that the functions *sca_next_max_time_step()* and *sca_synchronize_on_event()* are declared as *private* methods in *sca_module.h* of the SystemC-AMS version 1.0beta1. To make them callable them from the *processing()* method of the TDF model, it is necessary to patch them to *protected*.

All together makes the handling of TDF in our application case somewhat cumbersome. However, TDF-DTS definitely offers the user clear advantages. First is the simplicity of the connections between TDF and SystemC modules in terms of the TDF converter ports. This means that no additional interfaces and functions are required for the information exchange. Second is the integrity of system level simulations, which means that the TDF simulation and the SystemC simulation are compiled into one executable without the need of inter-process communication or third party library linking, which would be necessary with a co-simulation approach.

2.3.2 TDF in a Nutshell

Besides the DTS, we investigated also to put the TDF data path simulation in a separate compilation entity, namely a Dynamic Linked Library (DLL). The main idea of this approach is to package the TDF module together with its own SystemC-AMS simulation kernel into a DLL, which is then controlled from the master SystemC model. In this way, the simulation context of the SystemC-AMS simulation is decoupled from that of the controlling SystemC simulation.

One way of starting/stopping of the DLL-based TDF simulation can now be easily achieved by setting the SystemC-AMS simulation duration such that one full data path processing cycle is executed by calling *sc_start* via the DLL interface. However, this mechanism suffers from much additional computation load because the elaboration phase is repeated each time the data path is invoked.

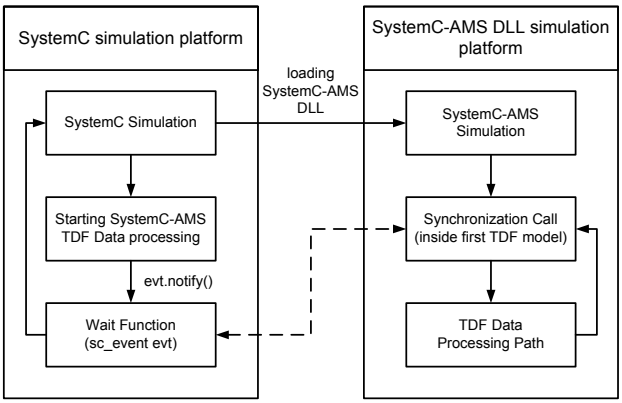

Figure 2-3: Controlling mechanism using a TDF DLL

In order to overcome this shortcoming, we investigated another start/stop controlling mechanism. Figure 2-3 presents our solution. At SystemC's instantiation phase the SystemC-AMS TDF DLL is loaded and the SystemC-AMS simulation is started. To

make the TDF start/stop controllable, we put a synchronization function in the first TDF model of the data path chain. This synchronization function is in fact implemented on the SystemC side of the master model and basically executes a *wait()* for the start event *evt* that typically would be triggered in the *Run* state of the control state machine.

When this happens, the TDF synchronization call resumes and gives the execution back to the TDF simulation. As soon as the TDF chain enters the next processing cycle it stalls itself again by executing the synchronization call.

Table 2-1 shows the implementations of both the TDF and the corresponding SystemC parts. There are three main functions defined in *main.cpp* of the SystemC-AMS DLL: *Initialize_TDF()*, *SendToTDF_Source()* and *Run_TDF()*. The function *Initialize_TDF()*, which is called in the constructor of the SystemC module, is used to instantiate the modules of the TDF simulation and also to transmit the address of the SystemC synchronization function *sc_wait_addr()* through the constructor to the first TDF model. The purpose of *SendToTDF_Source()* is to transport the input data to the data buffer of the SystemC-AMS simulation (*source*). *Run_TDF()* is called by a root *SC_THREAD* on the SystemC side in order to start the TDF simulation right at the beginning and setting it into standby mode. The simulation duration of the TDF simulation may be set to infinity as it stops together with the controlling SystemC simulation anyway.

Table 2-2: TDF in a nutshell / DLL approach

```
/************ SystemC Module ************/

/**** sc_model.h ****/
...
class sc_model: public sc_module {
public:
    ...
    TYPE source_data;
    sc_event evt;
    void _Wait_Function ();
    void Call_TDF ();
    ...
};

/**** sc_model.cpp ****/

#include "sc_model.h"

sc_model* sc_model_obj;

void Wait_Function () {
    ( *sc_model_obj )._Wait_Function () ;
}

void Start_SystemC_AMS_Simulation {
    SendToTDF_Source ( source_data ) ;
    evt.notify () ;
}

sc_model::sc_model ( sc_module_name nm ) {
    SC_THREAD (Invoke_TDF ) ;
    sc_model_obj = this;
    dllHandle = LoadLibraryA ( "TDF.dll" ) ;
    ...
    Initialize_TDF (&::Wait_Function) ;
}
void sc_model::_Wait_Function () {
    wait ( evt ) ;
}
void sc_model::Invoke_TDF () {
    Run_TDF () ;
}
```

```
/******* SystemC-AMS TDF Module *******/

/**** main.cpp ****/
...
#include "First_tdf_model.h"

First_tdf_model* tdf_model_1;
input_t source;

void Initialize_TDF ( WaitFunction_t w ) {
    tdf_model_1 = new First_tdf_model(
                        "tdf_model_1", w ) ;
}
void SendToTDF_Source ( input_t  src ) {
    source = src;
}
int Run_TDF () {
    sc_start ();
    return 0;
}

/**** First_tdf_model.h ****/

extern input_t source;

SCA_TDF_MODULE ( First_tdf_model ) {
    ...
    First_tdf_model ( sc_core::sc_module_name
                        nm, WaitFunction_t w_ )
        : sc_wait_addr ( w_ ) { }

    void processing () {
        ( *sc_wait_addr ) () ;
        // data processing function
        ...
    }
private:
    WaitFunction_t sc_wait_addr;
};
```

In the master SystemC module, the function controlling the TDF simulation is *Start_SystemC_AMS_Simulation()*. By notifying the event *evt*, the TDF simulation will get out of the SystemC *wait* function in method *sc_model::_Wait_Function()* and execute the TDF modules for one processing cycle. Note that for better readability we omitted the c-syntax and glue code necessary for communicating and calling the function pointers provided by the DLL.

Compared to the DTS approach, packaging the TDF model into a DLL avoids the somewhat cumbersome dynamic switching of time steps and patching of SystemC-AMS source code. On the other hand one major advantage of the TDF, namely the seamless interfacing with SystemC has to be sacrificed and additional code needs to be added for the transport of data between the SystemC and TDF domains as well as for the start/stop control.

So in fact none of the considered solutions to the TDF control issue is really satisfying and our hope is that future SystemC-AMS releases will have better mechanisms already on board to accomplish the task considered here.

3. Integration into a SoC Virtual Prototype

The final step of our system modeling flow is to integrate the time behavioral models into a system architecture model. Figure 3-1 displays the overall structure of this Virtual Prototype (VP). Besides the signal processing peripheral models with their algorithmic TDF kernels, the system model also contains pure SystemC models, which cover control functionalities of the chip, such as interrupt, memory or clock control.

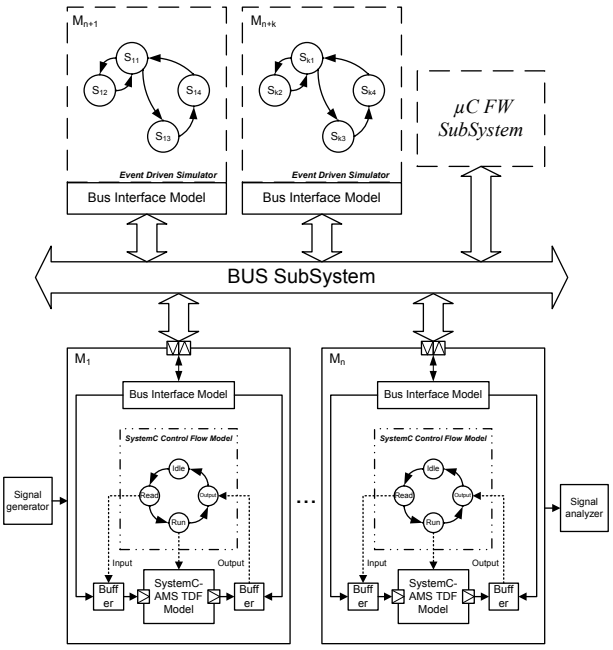

Figure 3-1: Structure of Virtual Prototype

From the firmware programmer's point of view, the VP functionally and temporally behaves (in the limits of the chosen approximations) like real hardware, also known as Programmers View, Timed (PVT). Pre-developed embedded software, can now be integrated on the VP and incrementally feature by feature can be brought up and running, typically several months before the real hardware is available.

4. Assessment of the Scheduling Performance

In order to assess the performance of the TDF scheduler, we performed simulations with (a) plain SystemC (in data flow emula-

tion mode), (b) SystemC-AMS TDF, as well as (c) a commercial tool supporting both dynamic and static stream-driven scheduling.

To make sure that the same defined number of context switches occur in (a), (b) and (c), we choose the sample system shown in Figure 4-1 and identically implemented it for the different simulation environments. The feedback loop forces the scheduler to call the models *Delay*, *Module_1*, *Module_2* and *Module_3* in sequence for each data token being transported from the source to the sink. Without this feedback, it would in principle be possible for the scheduler to apply optimization strategies, e.g. to do a block processing for multiple data tokens per module invocation, which would reduce the number of context switches significantly and hence weaken the comparability of the results.

Since the test is focusing on the speed of the scheduling mechanism the models of the simulation chain do not have any functionality, i.e. they are just feeding through their input values to the outputs.

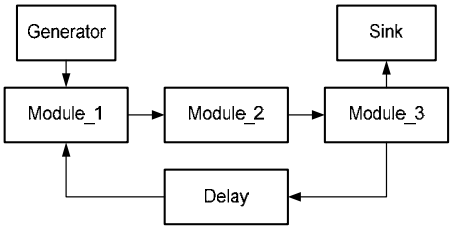

Figure 4-1: Structure of speed measurement system

The average simulation durations of the different schedulers (a)-(c) are displayed in Table 4-1.

Table 4-1: Measured Simulation durations (ms)

Tools \ Data Items		1000	10000	100000
(a) SystemC DE		1225	12384	124706
(b) SystemC-AMS TDF		360	3602	35831
(c) Commercial Tool	Dynamic Scheduling	103	1087	10250
	Static Scheduling	<1	10	103

SystemC-AMS TDF gives roughly a gain of 3.5 compared to SystemC's event-driven simulation mechanism (the event-driven scheduler in this case was abused to mimic a stream-driven behavior of the simulation).

However, as an open source tool TDF still has a large improvement space in the efficiency domain compared to the commercial tool we investigated as a reference. Considering static scheduling the commercial tool outperforms TDF by roughly a factor of 30 and even dynamic scheduling is about 3 times faster than TDF.

Of course some care is necessary in interpreting these figures. In a meaningful simulation the modules are not just feeding through but involve possibly highly complex computations, which usually by far dominate the overall simulation speed in the end.

5. Conclusion and Future Work

In this paper, we describe a model-driven system design flow based on SystemC and its AMS extension. In this context we focused on the data flow modeling using Timed Data Flow (TDF) and the integration of it into traditional SystemC control flow models.

Our experience is that TDF is in fact well suited for algorithmic design even in such complex algorithm-dominated domains like a 3G system. Reasons making TDF attractive for algorithm design are its open-source availability, its higher simulation speed compared to plain SystemC, the seamless SystemC embedment and its high level of abstraction.

We also describe our experiences with the integration of TDF simulations into a SystemC control flow model. Two approaches, Dynamic Time Step (DTS) and Dynamic Linked Library (DLL)-packaging, are introduced to realize dynamic start/stop control of an embedded TDF simulation.

Finally, we successfully applied the TDF models as data path kernel simulations of some of our 3G hardware models, and in this way proved the applicability of SystemC-AMS TDF in a complex Virtual Prototyping platform.

Future work will focus on two aspects: First, research the effects of a broader rollout of SystemC-AMS simulations in even larger and more sophisticated systems. Second, foster the development of new features of the SystemC-AMS kernel, which support an easy-to-use start/stop controlling mechanism for embedded TDF simulations.

6. References

[1] Heinen, S. and Joost M. 2009. *Firmware Development for Evolving Digital Communication Technologies*. In *Hardware-dependent Software*, pp. 151-171, Springer Science, ISBN 978-1-4020-9435-4

[2] Damm M., Grimm C., Haase J. and Herrholz A. 2008. *Connecting SystemC-AMS Models with OSCI TLM 2.0 Models Using Temporal Decoupling*. In *Forum on Specification, Verification and Design languages*, pp. 25-30, Stuttgart, ISBN 978-1-4244-2264-7