2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

TLM-based Virtual Platforms at Ericsson

Challenges and Experiences

Ola Dahl, (*ola.dahl@ericsson.com*)

Michael Lebert (michael.lebert@ericsson.com), Eric Frejd (eric.frejd@ericsson.com)

Baseband and Radio Virtual Platform Development, Ericsson AB, Sweden

***Abstract* — Development projects involving hardware and software are to an increasing extent executed in a parallel fashion, where overlap in time between hardware development and software development is the key to shorten the time to market. The use of virtual platforms, where models of hardware components are integrated into a host-based simulator, is an important enabler for such parallel development of hardware and software.**

In the Radio Base Station area, Ericsson is an active developer and user of virtual platform solutions. We have developed SystemC/TLM-based virtual platforms for digital ASICs and boards, both in the baseband and in the radio domain. The virtual platforms have been used to accelerate software development in early phases prior to board bring-up, but also in the continued development where hardware is available and the simulator is used as a tool for development and regression testing.

This paper gives a summary of our experiences, with a focus on challenges we have seen and experiences that we have made. We comment on our experiences with upscaling from a prototype to regular product development, where we today support several different ASICs and boards, and where we have users from many layers of the software stack. We describe briefly our software architecture framework and our way of working with models, with respect to model requirements for different use cases, and with respect to delivery and integration of models from different sources into a virtual platform.

Keywords—virtual platforms; use cases; TLM modeling; model integration; development flow; Radio base station

I. FROM PROTOTYPE TO PRODUCT

Our first TLM-based virtual platform was developed in 2011. It was a simulator for a baseband ASIC used in some of our Radio Base Stations (RBS). The simulator contained models for parts of an ASIC that was already produced, and its purpose was to serve as a demonstrator of the chosen simulation technologies (SystemC and TLM). The simulator has since then been extended to a fully functional model of the ASIC in question. The software that runs on the ASIC (and on the simulator) is now in maintenance mode, but nevertheless, the simulator is still used.

For the purpose of illustration, and to put the baseband ASICs into a bigger perspective, a selection of Ericsson RBS products is shown in Figure 1.

Figure 1. Ericsson Radio Base Stations

We see in Figure 1 that there is a range of different radio base stations, ranging from Macro Cell base stations to Small Cell solutions. We are today developing and maintaining virtual platforms for a variety of baseband ASICs and radio ASICs. We also develop and maintain board-level simulators where we add models for a selection of components available on the respective boards.

A board-level simulator is illustrated in Figure 2. The simulator is constructed by combining a simulator for the control and networking processor with a simulator for the baseband. The two parts come as stand-alone simulators, and we combine them using a dedicated connection. The connection implements a TLM over TCP/IP protocol, including facilities for simulated time synchronization between the two simulators. The synchronization algorithm allows each simulator to run ahead for a certain amount of simulated time before a synchronization is done, in the same way as the TLM quantum keeper does synchronization between SystemC threads.

It is also illustrated in Figure 2 how external test equipment and external hardware can be connected to the simulation. We use TCP/IP for these connections, and also here a time synchronization mechanism is needed. In contrast to the connection between the control/networking part and the baseband part, where simulated time is synchronized, the connections to the external test equipment and the external hardware require synchronization between real time and simulated time.

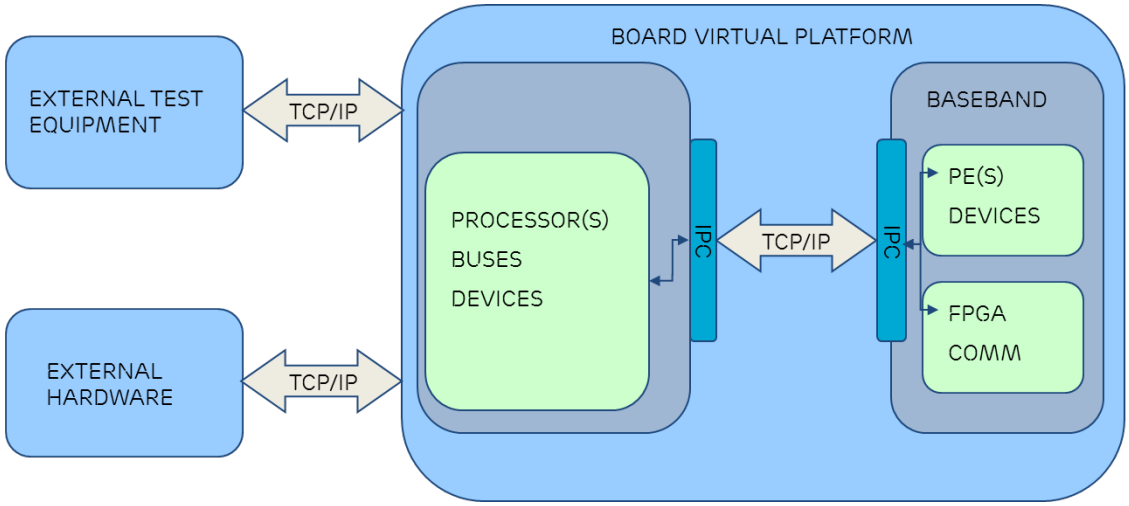

Figure 2. An illustration of connectivity, between virtual platforms and between virtual platforms and external equipment

The simulation approach depicted in Figure 2 implements a parallel simulation, where the parallelism is obtained by having more than one SystemC simulator instance. We have done work in this area, in cooperation with Cadence, where we have developed a methodology for breaking up a simulation into several pieces, running in separate processes, but keeping the communication so that when seen from an individual model it looks like a traditional TLM connection. The approach is referred to as CoMix, and it is described e.g. in [1].

From the start we have had a steadily growing user base, with representation from different SW development groups. We have users developing software that is close to the hardware, such as drivers and operating systems, but also users developing software that executes higher up in the software stack. Our users work with baseband processing software, radio processing software, and control software, and our virtual platforms contain models for hardware blocks as well as different kinds of processors. The processor models range from special-purpose cores, via DSPs, to control processors. Our current user base, taking into account also automatic simulations done in regression testing, contribute to more than 100 K simulator runs per day.

We build our virtual platforms with SystemC and TLM, and we use the open source reference implementation from Accellera [2] as well as an EDA-vendor supplied implementation. We integrate TLM models from different sources, both in-house and from external suppliers.

We have a development team of approximately 20 persons. We are divided into subteams, with 2-6 members in each. We use an Agile development methodology, and our teams work with Scrum as well as Kanban. We have a mix of competence in hardware and software development, and many of us have a background in working with embedded systems of different kinds.

Our overall strategy for using virtual platforms, and a discussion of different modeling use cases, has been presented before, in [3].

Challenges

- *Staffing – getting the right people onboard* – we have a mix of software and hardware competence, and we have found that a strong background in C++ is required, especially for the more software architectural aspects of our work.
- *User community acceptance* – we have different experiences from different user groups. We have seen challenges when supporting software layers higher up in the software stack, and we have seen challenges supporting software that relies on many interactions on board level.
- *Setting the requirements right* – in our early development, we had a focus on demonstrators and proof-of-concept, hence a focus on delivery rather than modeling accuracy. Today we have a mix of requirements, ranging from simple register models to fully functional data plane processing models.

Experiences

- *The first platform is still used* – we see that platforms tend to live also long after the hardware is available. This requires a support organization on our side that can work with new developments as well as legacy code.
- *Model creation can (and should) be distributed* – our team has taken on the role of integrator, and we receive models from different sources, both in-house and external.
- *CD/CI is a bliss* – we have adopted a scheme of continuous delivery and integration. We do releases often, and we have well-defined baselines for external dependencies, e.g. target software from our users that we also run in our own regression testing.

II. REQUIREMENTS AND MODELS

The virtual platform should represent the real hardware. It must however be decided to which extent this shall hold. We have chosen an approach where we represent all blocks that software accesses directly, and in some cases also blocks that software accesses indirectly. We model these blocks with varying degrees of accuracy, depending on the need as seen from software, and from overall simulation accuracy requirements.

There are models of different kinds, representing different kinds of real hardware. There are processor models and models for more dedicated blocks, such as hardware accelerators. There are also models for interconnects, and for interfaces to the external world.

Processor models – We use processor models from external vendors, and we use processor models developed in-house. These models are instruction accurate rather than cycle accurate, and they model timing to some extent. The models are wrapped into TLM models, and integrated into our virtual platforms. We often use cycle-accurate processor models as references, for verification of our instruction-accurate models.

Hardware accelerators – We model different kinds of hardware accelerators. In many cases, we use data plane models in the form of signal processing models, which we wrap into TLM models. Many of the hardware accelerator models are also used for ASIC verification. In this way, they serve as golden reference models, and are by definition assumed correct. Other models contain a simplified functional behavior, implemented e.g. using state machines. These models are often good enough for control plane scenarios. All models contain a bit-accurate register map. For some models, this register map is good enough – for example for blocks that are not involved in the software verification use cases of interest, but are needed e.g. for a successful system boot.

Networking blocks – We use models for networking, such as Ethernet. Many of these models are functionally correct and contain control plane as well as data plane processing. It is often also desirable to have an

external connection to these kinds of models, for the purpose of connecting other programs to the simulator. Here, the concept of user-mode networking, where no root privileges are required, is often preferred.

Challenges

- *Where can we find information?* – we mix usage of formal documentation with more informal ways of finding information about hardware designs
- *When are we* done*?* – we need to set the requirements right, so that software can be verified functionally without modeling parts of the hardware that are not seen by software
- *Who* validates *the golden reference?* – we need a mechanism for securing the quality of the reference model itself.

Experiences

- *HW specs are good, but SW User stories are also very useful* – we have found that for many models, a hardware specification driven approach is the right thing to do. For other models, a more software driven approach has shown to be fruitful. In this case, software user stories are often used as input material for the modeling.
- *Sometimes a register model with a selection of functional behavior is good enough, and sometimes we need full functionality (e.g. a complete signal processing algorithm)* – we use both kinds of models, in different use cases
- Some *models are used also in ASIC verification, others are not* – for the models that are not used in ASIC verification, we need other means of deciding that they are accurate enough.

III. PLATFORM ASSEMBLY

An overall view of the virtual platform design is shown in Figure 3. The right part of Figure 3 shows the actual virtual platform, with models for processors and baseband accelerators. It also shows a connection, in the top of the picture, to an external simulator.

In the bottom part of the right half of Figure 3, we see different interfaces to the virtual platform. There are interfaces for dynamic creation of models, a debug interface, handling of model attributes, and an interface for simulator execution control. In the left part of Figure 3, we see a selection of external tools that are connected to the virtual platform. We are able to connect external debuggers and test tools, and we are also able to connect to real hardware, for example when verifying parts of a hardware design using software running on a simulated processor to control, and evaluate, the verification.

The architecture shown in Figure 3 is common for all our virtual platforms, i.e. common for virtual platforms that simulate different ASICs and boards. This makes it possible to handle several variants, each corresponding to one of several hardware systems, of which some are being developed concurrently.

The delivery of models, and their integration into a virtual platform, is illustrated in Figure 4. We see in Figure 4 that TLM models are created. Some of the models are also used as references for RTL. The TLM models are delivered to the entity called TLM Model Adaptation in Figure 4. It is our experience that this entity is needed, for the purpose of adapting the TLM model so that it can be used efficiently inside the virtual platform. Even if the TLM standard is followed, there may be discrepancies e.g. in methods used for model configuration and control - hence the need for additional adaptation. There is also the aspect of register modeling, which may, or may not, be included in the model delivered from the model source. In our TLM model adaptation, we sometimes add functionality so that certain registers become visible when running the simulation, which may be important in a debugging scenario involving hardware as well as software. As a means for reducing, or eliminating, these TLM model adaptations, we aim for using more of the forthcoming CCI standard [4], in which we are also actively participating.

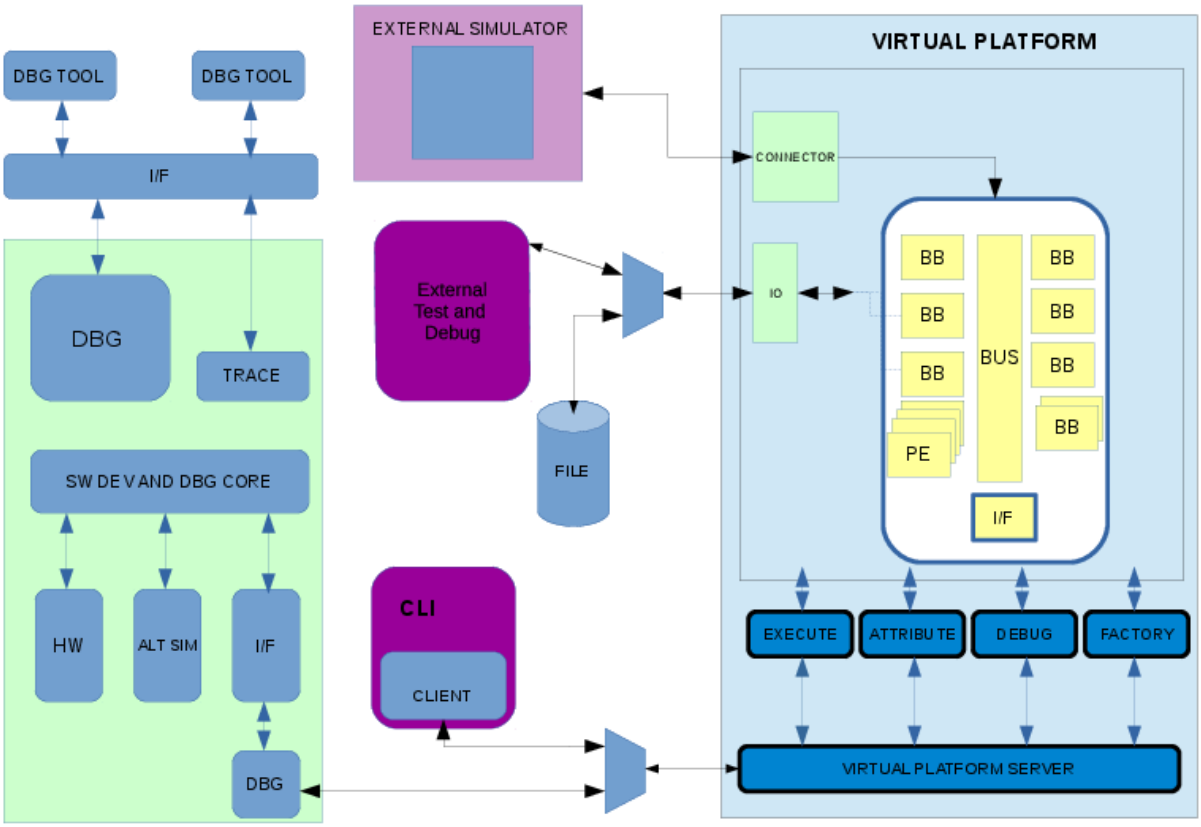

Figure 3. Virtual Platform Architecture

We use a virtual platform framework that allows us to create virtual platforms in a rapid and flexible manner. It is based on a factory, for creating models on demand, in combination with a description format for the connectivity. The simulator reads a connectivity description, which includes the models to be used as well as their interconnections, and creates, at run-time, a virtual platform. The virtual platform can be started and stopped, and interactions during simulation can be performed, using software debuggers as well as dedicated virtual platform tools. The creation of the virtual platform is managed through the interfaces shown in the lower part of the right side of Figure 3.

The dynamic configuration approach, using connectivity descriptions, gives a flexible way of assembling virtual platforms, and enables rapid prototyping of new architectures. It is based on dynamic linking, at run time, and each model linked in is represented by one shared library. For this reason, it also gives some challenges, such as shared libraries not being found, due to textual errors in the connectivity description, or linking problems based on symbol resolutions happening in a different order when a configuration is changed.

Challenges

- *We need to support several platform variants* – this is done using connectivity descriptions, as mentioned above
- *How can we formulate modeling requirements for model makers?* - we create modeling guidelines, that we distribute to our vendors, both in-house and external

- *We* need *to put the platform together, but we also need to interact with it (in many ways)* – our architecture framework contains well-defined interfaces for attaching software debuggers and other tools, e.g. test tools that are also used together with real hardware.

Experiences

- *Dynamic platform building is useful* – we can create new platform variants quickly
- *A custom module class and associated configuration and control classes have helped a lot (and CCI could make it even better)* – we are active in the CCI working group
- *A common SW architecture was created, and it is still used* – this has been a fruitful investment

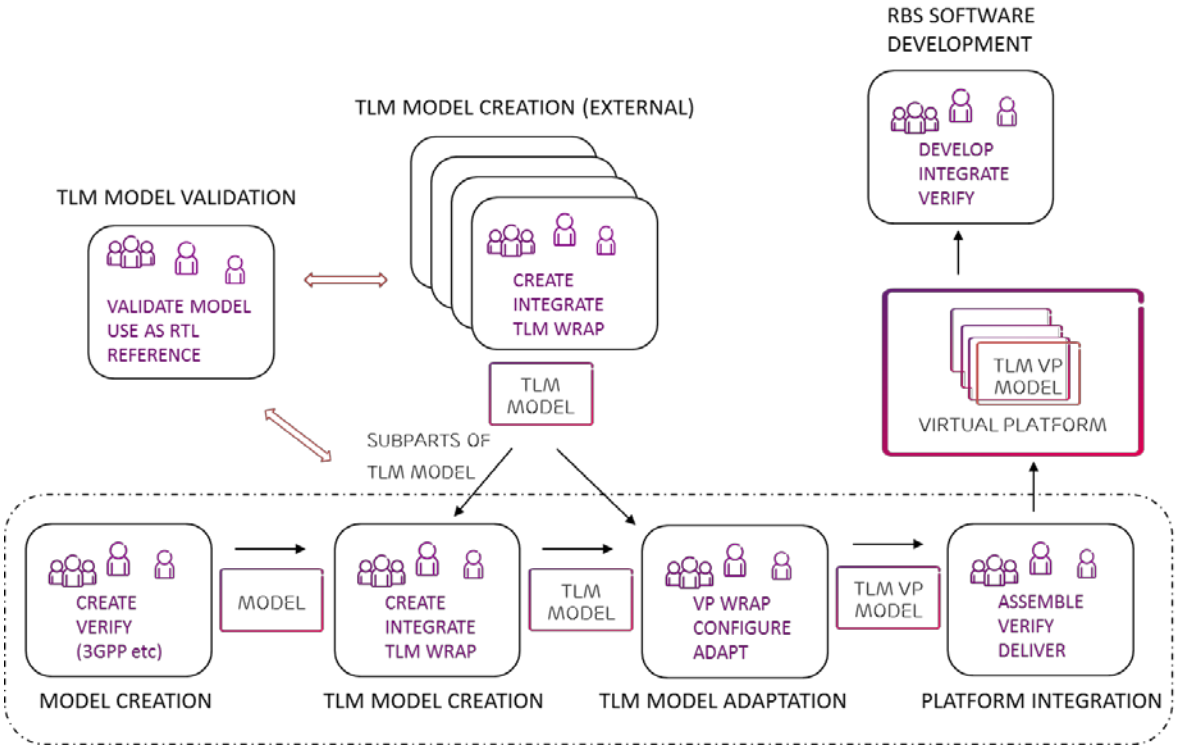

Figure 4. An illustration of the delivery flow for TLM models

IV. USAGE PATTERNS

A virtual platform is often constructed at an early phase of a project. In this phase, RTL is not ready, and we rely on specifications and/or personal communication to get the information needed for model creation. The first users are driver developers and operating system developers.

As a project proceeds, more and more becomes known about the hardware, and the models in the virtual platform are evolved accordingly. A first major milestone is often the ASIC bring-up, followed by one or more board bring-ups. At this point in time, several software layers are using the virtual platform, and we can see improvements in integration time and bring-up time, compared to earlier projects where simulators were used to a lesser extent.

Challenges

- *Supporting Control plane and Data plane – on time and on budget* – we often start with control plane modeling, and add data plane models as the project proceeds

- *How to best prepare for ASIC bring-up and Board bring-up* – we need to work tightly with software organizations, in order to understand which test cases that are most important to run in a simulated environment
- *How to support SW development and integration also after the hardware has arrived* – we have mixed experiences here, but in many projects we have seen a demand for simulators also after the hardware is available

Experiences

- *Control plane is common, but well-defined data plane scenarios can be real game changers* – it is costly to simulate all functionality, and we have selected specific use cases where we do full data plane simulation
- *Many virtual platforms tend to stay around, also long after hardware is available*
- *A virtual platform that is integrated into the SW development flow can run and test new software, but it can also serve as gatekeeper for deliveries (tests must pass on the virtual platform before delivery is allowed)*

V. SW DEVELOPMENT AND VIRTUAL PLATFORMS

When supporting software development, especially on higher layers, it is not always obvious how a virtual platform shall be used. One aspect is software dependencies, which makes the platform usable for a certain software layer only if the underlying layers are in place. We have experienced different approaches to this, such as using API stubs representing lower layers of software, which in effect disconnects the upper software layers from the virtual platform (i.e. from the hardware). Another approach is simply to wait for lower layers to be ready. In these scenarios, an early delivery of a virtual platform is necessary, so that the lower software layers can be developed and tested at an early stage of the project.

Many of our virtual platforms simulate multicore systems. We use the concept of quotas, as built into the SystemC kernel. This, in combination with the cooperative scheduling approach used by SystemC, results in the different processors taking turns in execution. We have experienced that although this type of simulation is non-accurate in the sense that true parallel execution is not possible, it can expose concurrency flaws in software. This is especially true for software that is designed with timing assumptions in mind, such as software that relies on a certain operation to finish within a certain number of cycles. It is an ongoing challenge to support software developers of these kinds of software, and in a debugging scenario where a certain piece of software works fine on hardware but not on the simulator, the burden of proof often lies with us in the virtual platform team.

Challenges

- *Driver development is straightforward, but how can we support higher layers in the software stack*?
- *My software works on hardware but not on the virtual platform – what to do next?* – we have had many debugging scenarios where it has been difficult to see if a symptom is due to an error in software or in the virtual platform (or both) – and this is especially challenging if the software works fine on real hardware
- *I want to check my software performance, can I do that?* – we have a focus on functional modeling, but using time annotation in LT models, some aspects of performance measurements can be done.

Experiences

- *Virtual platform developers often need to connect the dots – between the software layers and down to the hardware – especially when solving problems (is it a software crash or a virtual platform crash?)* – we experience this in our daily work with debugging

- *Timing-sensitive software does not run well on a virtual platform (it may run well on one specific hardware, but not on all hardware) – in this way, the presence of a virtual platform encourages robustification of software, which in the end should lead to higher software quality* – we see this as an additional selling point for a virtual platform

- *Timing annotation can give indications of performance, but cycle-accurate models and/or RTL is needed for more precise figures* – the modeling of timing can be discussed in detail, but should perhaps deserve a paper on its own

VI. CONCLUSIONS

Ericsson Digital hardware development for Radio Base Stations has been using SystemC/TLM based virtual platforms since 2011, for the purpose of accelerating the development process. Our experiences show that this is indeed possible, provided the modeling requirements are thought through, so that each model is as accurate as needed, but not more, while at the same time the complexity of multiple sources of TLM models is handled, so that integration into virtual platforms can proceed in an efficient manner. We have also seen that a well-designed underlying software architecture is of great help for support of multiple variants of platforms, representing multiple variants of ASICs and boards.

We have seen that the TLM standard is of help, and we have successfully integrated models from external sources. A significant amount of our integration effort has however been spent on areas such as module configuration and module control and inspection. We are active in the CCI working group, and we anticipate a simplified integration effort once the CCI standard is in place.

We use virtual platforms as a regular tool in our software development, for baseband as well as for radio. We have seen that early development of models, often using loosely formed requirements on behavior but specific details regarding register maps, in combination with instruction accurate processor models, can provide a head start for software development. As the project proceeds, the models are evolved, and after having secured ASIC and board bring-up, the models are kept, and used more and more in regression testing. In our current user base, where we have more than 100 K simulator starts per day, we see that also older platforms have a large percentage of the total usage.

We have seen how the usage of virtual platforms enables an early start of software development. As an example, from one of our baseband ASICs, we were able to boot the target software on a virtual platform one year ahead of the arrival of the first silicon – effectively giving us one year of software development on the virtual platform, which in turn translates into a corresponding gain in time-to-market.

We have also seen how the usage of virtual platforms enables us to shorten the bring-up time. This is true for ASIC bring-up but also for board bring-up, where a board-level virtual platform, e.g. as depicted in Figure 2, has shown to be a very useful tool for board bring-up preparation.

REFERENCES

[1] C. Sauer, H-M Bluethgen, Hans-Peter Loeb, *Distributed, loosely-synchronized SystemC/TLM simulations of many-processor platforms*, The 2014 Forum on specification & Design Languages

[2] Accellera Systems Initiative, http://www.accellera.org/

[3] TLM Use Cases at Ericsson AB, in Case Studies in SystemC, DVCon Europe, 2014.

[4] Accellera – About SystemC CCI, http://accellera.org/community/systemc/about-systemc-cci