ESL Design and HW/SW Co-verification of High-end Software Defined Radio Platforms

Ng, A. C.H., Weijers, J.W., Glassee, M., Schuster, T., Bougard, B., Van der Perre, L.
IMEC, Belgium
{nganthon, weijers, glasseem, schuster, bougardb, vdperre}@imec.be

ABSTRACT
Multiple wireless technologies are converging to run on personal handhelds. The plethora of communication standards next to the cost issues of deeper submicron processing require handheld platforms to shift from sets of multiple application specific ICs (ASICs) to multi-purpose Multi-Processor System-on-Chip (MPSoC) on which Software Defined Radios (SDR) are run. SDR design faces hard real-time processing and data transfer latency constraints. Designing SDR under stringent time-to-market (cost), energy and real-time processing constraints requires the help of advanced Electronic System-Level (ESL) design methodologies. This paper demonstrates an integrated ESL design flow built on advanced ESL tools to design SDR platforms for handhelds. We share the experience from creation of a high-level virtual platform model down to hardware/software (HW/SW) co-verification of a large scale SoC (5 million gates+). Incremental RTL verification based on co-simulation and co-emulation is also presented.

Categories and Subject Descriptors
B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms
Design, Verification

Keywords
ESL, Hardware/Software co-design, verification, SDR, emulation

1. INTRODUCTION
The recent trends to integrate multiple radio access technologies into a portable terminal result in a dramatically increased complexity in HW and SW platforms. Several researches have been conducted on implementing SDR platforms, and the heterogeneous MPSoC approach appears to be the most appropriate design [1]. *Figure 1* depicts a top-level partitioning of a typical SDR platform [2]. Functions related to packet detection have high duty cycles, and are mapped on processing units with high-energy efficiency and low flexibility, i.e. Digital Front-end (DFE). However, functions such as (De)modulation, forward error correction (FEC) have lower duty cycles and have a significant potential for energy-scalable implementation [3]. They are best to be partitioned on different programmable units.

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.

When combined with aggressive energy-management techniques, such partitioning enables flexibility with reasonable impact on average power consumption and standby time. However, this requires SW to be developed on heterogeneous units and makes HW/SW verification difficult.

Platform-based ESL design enables fast architecture exploration and minimize final RTL verification efforts. A crucial part of an ESL flow is the support of HW/SW co-verification. ESL flows inherently enable gradual refinement and support progressive verification. The design is first specified at high level languages (e.g. SystemC), then it is gradually refined down to RTL. Individual components on the platform can be refined to RTL incrementally while keeping the high level platform model as an executable specification. The platform model is used as testbench stimuli for functional RTL verification and this requires the support of SystemC/HDL co-simulation in the ESL environment.

To the best of our knowledge, the usage of ESL tool is mostly confined to Transaction level Modeling (TLM) [4][5] for high level platform simulation. There are still few studies demonstrate large scale design (over 5 million gates) with this new ESL design methodology to develop a complete system. In this paper, we describe an ESL design flow of an SDR platform, we also show how integrated refinement and incremental HW/SW co-verification can be achieved in the ESL environment. This methodology includes early concurrent HW and SW development based on a virtual platform reference model and the concept of *co-emulation* is introduced for simulation speed up. In this work, we carry out a complete design flow by integrating the ESL flow from the two independent tool vendors, CoWare and Mentor. We successfully *co-emulate* the two simulation environments, CoWare's TLM virtual platform environment is communicating with Mentor's emulator HDL simulation environment via the custom transactors which we developed.

The motivation of our study is to share the design experience in creating an SDR SoC with this advanced tool and to discuss the additional values which the tool brings to our design. We also evaluate the simulation speed advantage of creating a TLM platform model. Finally, we compare the two verification techniques, co-simulation and co-emulation via an experimental study of a wireless application. We identify that the simulation speed bottleneck is the SystemC/HDL interface, the complexity of the HDL implementation alone has relatively less impact on the overall simulation speed.

The remainder of this paper is organized as follows: In Section 2, we detail the ESL design methodology. Section 3 focuses on the system performance optimization. Section 4 describes RTL refinement and HW/SW validation. Section 5 demonstrates a case study in speedup evaluation. Conclusions are drawn in Section 6.

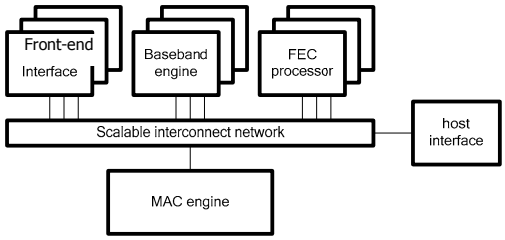

Figure 1: Opportunistically partitioned SDR platform

2. ESL DESIGN METHODOLOGY

The traditional approach in designing a chip is usually too close to implementation level, which inhibits design reuses and makes full system verification difficult. ESL is a new design approach raising the level of abstraction to system level. The aim is to introduce HW/SW verification early in the design flow.

TLM is a key technique used in ESL design, it provides a number of abstraction levels to trade off between simulation speed and accuracy [6]. In this work, CoWare Platform Creator 2005.2.1 [7] is used as our ESL design environment.

After a few months of steep learning curve, we were able to create an initial architecture of the platform with the tool. The time spent was mainly to get familiar with using the tool for platform assembling. Once we are experienced, a new platform model can be easily created within days. Although the initial investment to learn the tool is considerably large, the possibility for architectural exploration and system performance assessment brings significant values to achieve a more efficient design (See Section 3.1).

The proposed design flow starts with creating a high level SystemC TLM model (*Phase 1*). *Figure 2* outlines the design methodology of the HW/SW co-verification flow. Once the platform is assembled, HW/SW partitioning of the reference application code can be performed and mapped on the virtual platform. Processor specific information such as memory maps is now available for HW dependent SW (*Phase 2a*) development. SW development and RTL refinement (*Phase 2b*) can then be decoupled. Joint HW/SW validation is performed based on co-simulation (*Phase 3*). When more platform components are available at RTL level, *co-emulation* (*Phase 4)* is used for RTL simulation speed up on an emulator. The availability of a virtual platform enables individual block RTL refinement by using the virtual platform as testbench during verification.

2.1 Phase 1 Virtual Platform and HW Dependent SW Development

The goal is to rapidly assemble a system for SW development and performance assessment. Using TLM, we create a virtual platform, modeling some of the HW blocks at bus cycle accurate level, processing units at instruction cycle accurate level and the rest of the platform at untimed level. Some of the blocks are chosen to be at cycle accurate level because some of our IP models are only available in HDL, which is usually the case for legacy designs. The decision to start at which level of abstraction depends on IP availability, the ease of creating a new model, and the simulation speed requirements. Taking the memory model in our design as an example, we first model its implementation at the

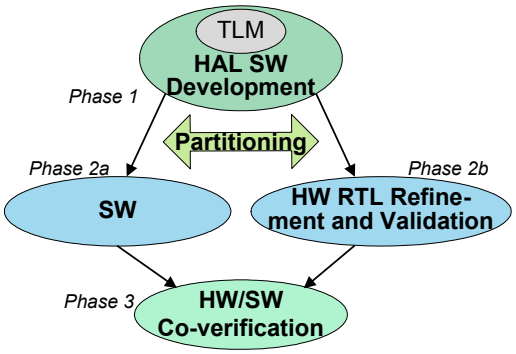

Figure 2: Design Methodology from TLM platform model to HW/SW Co-verification

highest level of abstraction (Programmer's View [6]). For SW development, we are only interested in the corresponding memory address for read/write access, accurate timing information is relatively less important, so it can be modeled with untimed implementation. The virtual platform model allows mixed platform components at different abstraction level to co-simulate via the use of transactors. Therefore, the whole design is not confined to start at a specific abstraction level but rather depends on what design decisions have been made to implement the corresponding details. Development of the HW-dependent SW e.g., the HW Abstraction Layer is started based on an early release of the virtual platform. This enables basic simulation-based HW/SW validation and provides early feedback on the HW/SW interaction performance. This feedback is used to optimize the interconnect and the handling of interrupt events (e.g., latency) deserve attention at this step. The virtual platform acts as a reference simulator for functional SW development and also as a reference model to verify with the RTL design.

2.2 Phase 2a Functional Software Development

The programmable cores on the platform are either custom processing units modeled with Application Specific Integrated Processors (ASIP) development kits e.g. LISATek [8] or third party IPs, e.g. ARM Instruction Set Simulator (ISS). Each ISS is deployed in a SystemC container provided with a bus interface (e.g. AHB). Encapsulation of the ISS generated with other retargetable compiler environment (e.g. Tensilica) is also possible with custom SystemC wrappers. The simulation of the ISS is tightly coupled with the underlying SystemC simulation environment and this means synchronization of multiple cores can be debugged concurrently. This greatly enhances the visibility of the HW/SW interface.

2.3 Phase 2b RTL Refinement and Validation

According to the gradual refinement concept, RTL refinement is performed for each platform component, while keeping the rest of the platform at TLM level. The latter acts as a testbench for the progressive verification of the refined RTL component. Simulation speed is a major concern during RTL refinement. Co-simulation of SystemC/RTL models becomes a simulation speed bottleneck which reduces achievable validation coverage. *Co-*

emulation is used to address the simulation speed problem and will be described in Section 4.

2.4 Phase 3 HW/SW Co-verification

In this phase, cycle accurate models, including power down modes are available for most components. System startup sequences can now be verified. *Co-emulation* allows application SW to be tested before all RTL models are available. Once all models are at RTL level, *co-emulation* could be replaced by emulation if the testbench is also written at RTL level. If the testbench is already written as a SystemC TLM model, which is usually the case when ESL flow is adopted, then *co-emulation* avoids the needs to rewrite testbenches again at RTL level.

3. VIRTUAL PLATFORM

Throughout the design of the virtual platform, the bus model is kept at the cycle accurate level as it provides well-defined interfaces to attach models (e.g. peripherals) at different abstraction levels to the cycle accurate bus via the use of transactors. The abstraction models can be started at behavioral level then refined with more precise timing information to match the behavior of the real design. After that, platform exploration for optimizing the system performance can be done independently from the rest of the platform development.

3.1 Platform Exploration

Platform exploration is an important step to identify system bottlenecks. The use of ESL tools to rapidly create a virtual platform brings advantages in this step. When designing a system directly at HDL level, one cannot easily evaluate system performance upon any changes in the interconnect and platform architecture. For example, it is difficult to assess whether a further segmentation of the interconnect would improve the performance or which interconnect segment is best to locate a particular component for the optimal access time.

The ability of profiling system parameters on the virtual platform using bus transaction traces and analysis tools provides early feedback of the design, especially on the understanding of complex interactions between different system components (i.e. specific bus access pattern). Stand alone HDL testbenches typically verify functional correctness of individual blocks only. It is often difficult to evaluate the complex interactions between different components before a fully functional system is available.

We present a simplified example of IP selection and interconnect optimization to demonstrate how platform exploration steps can be done on a virtual platform. The same methodology can be applied for more complex performance analysis, including non-sequential data transfer when a bus internal buffer is introduced. We present the SDR platform architecture refinement as follow:

Stringent timing constraints in baseband processing require efficient data transfers between different processing elements (PEs). Hence, interconnect has to be optimized for high performance and low power consumption (e.g., bus frequency). In our design, we consider multi-layered AMBA2 as the interconnect standard. We adopt the AMBA TLM library [9], which is a set of API which model low-level bus signals as high-level read/write transactions. Direct memory access controllers (DMACs) are used for data transfers. There are six types of data flow including, Digital Front End (DFE) to Baseband Engine (BBE) to Forward Error Correction Engine

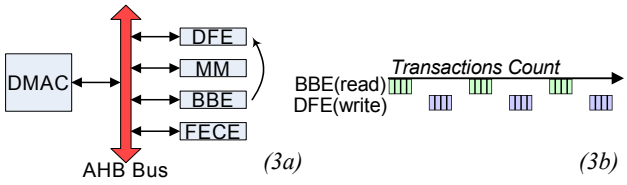

Figure 3: a) Single bus architecture (left), b) A snapshot of the corresponding transaction trace (right)

(FECE), FECE to main memory (MM) and three types are in an opposite direction at the transmission. Burst data transfer is used because it matches the wireless modem execution flow which corresponds to a regular chain of DSP kernels applied to block of samples. In the most stringent case of IEEE 802.11n MIMO, the flow between DFE and BBE has the most demanding rate of 512 Mbit/s per antenna. The total required throughput is at least the sum of these six types of transfers (1550 Mbit/s) and scaling up linearly with the number of antennas. In 2x2 MIMO case, the required data rate is 3.1 Gbit/s, so a 32-bit bus at 200 MHz offers effective rate of 3.2 Gbit/s is sufficient. However, this only holds for a fully pipelined bus without taking into account the control overhead of the bus. To guaranty real time execution, one need to implement deterministic interconnect and verify the effective bandwidth at design time.

In our platform exploration, we examine the system performance based on simulating the real behavior of the selected IPs and the bus connectivity on the platform. Interconnect optimization is performed by exploring different bus architectures and progressively searching for the optimal configurations. At each step, bus throughput and bus utilization are evaluated. A simple SW testbench is developed on the ARM ISS to perform the profiling (interconnect stress test). It emulates the interconnect request at maximum load.

We start with a simple bus architecture (see *Figure 3-a*) where all the processing elements (PEs) and a single-port DMAC (ARM PL081) are connected to a bus. The disadvantage of this architecture is the exclusive usage of the bus. This means only one data transfer is allowed at a time while other transfers are blocked. We configure a transfer from BBE to DFE. The simulated throughput is 1.8 Gbit/s (see *Table 1-a*), which is below the optimal bus throughput of 3.2 Gbit/s. The limited FIFO depth of the DMAC IP (4-word) and the control overhead of the bus protocol hamper continuous data transfers (see *Figure 3-b*).

In the initial bus architecture, bus bandwidth is the bottleneck. Hence, we double the bus capacity by splitting the bus into two segments. The transfer bottleneck is now shifted from the interconnect to the DMAC. In this architecture, we select a dual-port DMAC (ARM PL080) that bridges the two bus segments as shown in *Figure 4-a.* By overlapping the read/write transactions over time, one bus segment is performing a read transaction while the other bus segment is writing the data that was obtained from the previous read operation. Theoretically the bus throughput doubles using this interconnect configuration.

We configure a transfer from BBE to DFE. The simulation result shows a surprisingly low bus utilization of 20% (see *Table 1-1b*). Using the profiling tool, the transaction trace shows that the 4-word read/write bursts are split into a multiple of inefficient single-word transfers (see *Figure 4-b)*.

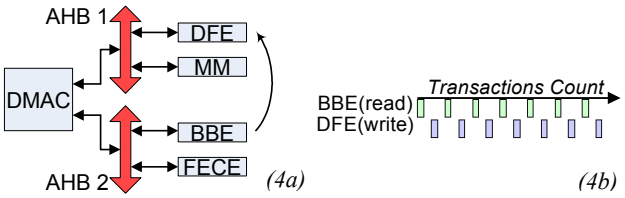

Figure 4: a) Segmented bus architecture with a dual-port DMAC (left), b) A snapshot of the corresponding transaction trace (right)

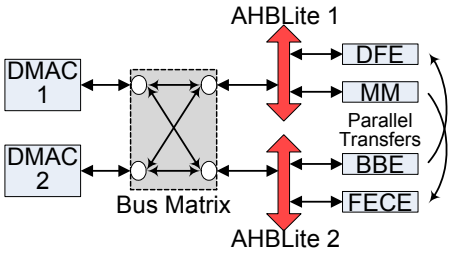

Figure 5: a) Segmented bus architecture with two single-port DMACs

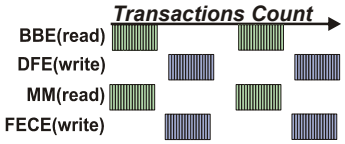

Figure 5: b) A snapshot of the corresponding transaction trace

Based on the unexpected performance of the selected DMAC in our setting, we select another DMAC with more appropriate implementation. The performance statistics show that to optimize the transfer rate, we have to reduce the bus overhead involved in switching the read/write operations and also to increase the parallelism of the transfers among the bus segments.

Next, we replace the dual-port DMAC by 2 single-port DMACs (Barco Silex BA612) with an adjustable FIFO depth. We identify that burst length of 16-word gives a reasonable tradeoff between throughput and latency as the increased burst length introduces extra delay for the access of the lower priority of the two transfers.

The adaptation of two DMACs requires modification of the interconnect as both DMACs need to have access to the bus segments to perform back to back parallel data transfers. We configure a multi-layer AHB bus as shown in *Figure 5-a*. The concurrency of multiple 16-word burst transfers between the two segments is shown in *Figure 5-b*). The bus throughput increases (see *Table 1-1c*) to 4.3 Gbit/s which is sufficient for 802.11n.

By further applying the platform exploration technique described, we obtain an SDR interconnect architecture with optimal data transfer performances (see *Figure 6*). An interrupt mechanism is used to communicate between the ARM and the rest of the platform components.

4. RTL REFINEMENT AND VERIFICATION

In this section we describe the uses of co-simulation and *co-emulation*. We also show some experimental results of the simulation speed based on the two verification techniques.

Table 1: Data transfer performance under different bus configurations.

Optimization Steps	*Bus Configurations*	*Throughput (Gbit/s)*	*Bus Utilization (%)*
1a	Single bus with single-port DMAC	1.8	57
1b	Segmented bus with dual-port DMAC	1.3	20
1c	Segmented bus with two single-port DMACs (16-word internal buffer size)	4.3	68

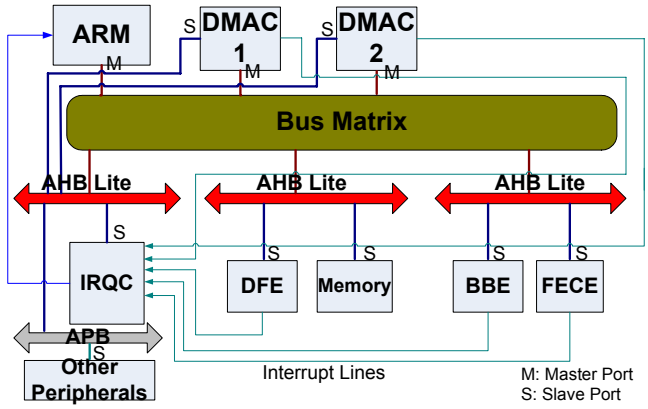

Figure 6: A platform interconnect architecture

4.1 Co-simulation

In co-simulation, RTL models are simulated together with TLM models. It is used to enable individual RTL refinement of the components while using the rest of the TLM models as testbench. For legacy IP, co-simulation is often the only option as SystemC models are not always available. However, the major drawback of co-simulation is the low simulation speed.

Co-simulation ranges from instantiating a single HDL model in a SystemC design to instantiating a single SystemC module in an HDL design. At the early stage of RTL refinement, we use co-simulation as most blocks are defined at TLM. When more blocks are simulated at RTL level, we can switch to *co-emulation*.

In our framework, the SystemC simulator is master and calls the RTL simulator to simulate the imported RTL components. This requires the generation of an "RTL proxy module". The RTL proxy module is a SystemC wrapper that interfaces through the Verilog PLI interface with the actual RTL model. The RTL proxy code is generated automatically by Platform Creator.

4.2 Co-emulation

Pure HW emulation is typically used to speed up RTL verification. However, the disadvantage is the need to develop specific synthesizable RTL testbenches. Therefore we use *co-emulation,* a new verification technique to speed up the simulation while keeping SystemC platform testbenches in the same simulation environment. We use Mentor VStationTBX 1.3.0.3 and Mentor VStationPRO 6.1.0.13 as our emulation environment.

During *co-emulation*, RTL blocks are executed on the emulator while SystemC blocks are simulated on a PC. Next, we show how the inter language function calls of SystemVerilog enable *co-emulation*.

4.2.1 Transactors

The "Direct Programming Interface" (DPI) of SystemVerilog allows a SystemVerilog component to call a SystemC function and vica versa. These communication pipes between SystemVerilog and SystemC can be seen as transactors. Recent RTL simulator versions allow SystemVerilog models to be simulated together with Verilog and VHDL models. Such transactors can be thought of as the cable(s) connecting the emulator which contains the RTL part of the simulation with the host PC running the SystemC part of the simulation.

During a transaction, the simulation control is temporarily transferred from the emulator to SystemC (see *Figure 7*). Clock gating is used to temporarily stop the HW. The SystemC simulator executes the function which is called through the transactor. This function in turn may trigger events which starts other processes or threads. Results of these processes or threads have to be sent back to the called function and this function has to wait for these results. When the function called by the transactor finishes, the HW continues. The actual time during which the HW is stopped depends on the workstation load. The HW clock was stopped during the transaction, so from the view of HW, the SystemC function returns immediately (zero execution time).

5. CASE STUDY - SPEEDUP EVALUATION

We would like to set up experiments to evaluate how much speedup can be obtained using *co-emulation* compared to co-simulation. The system under test consists of the Digital Front End (DFE Rx), DMAC and the AMBA Interconnect.

The DFE Rx core is used for wireless packet detection by searching for packet preamble in a flow of incoming sample data. It consists of an Automatic Gain Control (AGC) unit, a synchronization processor, decimations filters and a buffer for storing incoming I and Q samples. The detailed description of the DFE Rx design can be found in [2].

To compare the simulation speed, two comparable system setups are described in *Table 3*. For co-simulation, the AGC and DMAC are described in RTL, since this is the only available implementation. The synchronization processor model is an ISS and the rest of the platform are modeled in SystemC. For *co-emulation*, all components except the testbench are described at RTL level. A diagram shown in *Figure 8* describes the setup of the different data transfers between SystemC/VHDL interface under co-simulation.

As a reference application for the comparison, the reception of a packet is simulated. Input stimuli (I/Q values) are read from a file at the SystemC I/O interface.

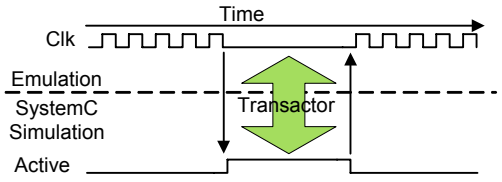

Figure 7: The transfer of control during co-emulation.

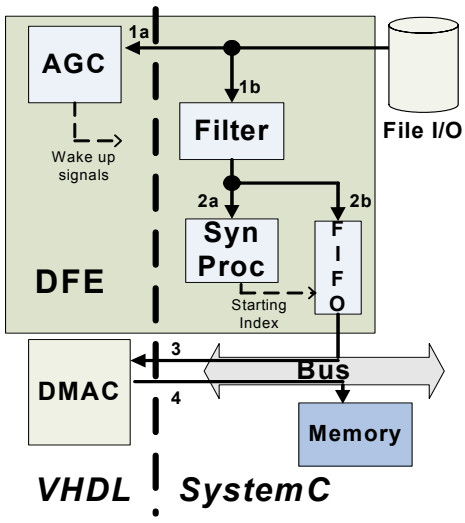

Figure 8: Data flows between SystemC/ VHDL interface

First, power calculations are done on the AGC. Based on the calculation results, AGC subsequently wakes up the synchronization processor, filters and FIFO. When a synchronization point is found after autocorrelation, a valid preamble is received. Then the packet samples are transferred over the AHB bus (TLM model) read by the DMAC (RTL side) and finally written back to the memory (SystemC side) for baseband processing. This experiment extensively tests the simulation of the AGC, synchronization processor, DMAC, the AMBA interconnect (bus transactions) and the data transfers between SystemC/VHDL interface. Therefore it is a good showcase for speed measurement when most RTL simulations are shifted to the emulator under co-emulation.

The simulation speed is measured in number of simulated clock cycles per second under the duration of a packet reception. For co-simulation, simulation speed of 1k cycles/sec is obtained, while *co-emulation* goes up to 104k cycles/sec. In our experiments, we also observed that the speed bottleneck mainly depends on the amount of traffic flowing between the SystemC/VHDL interface. The complexity of the RTL block has less impact on the simulation speed. When we place an additional DMAC RTL model onto the platform, there is only marginal reduction in speed. An alternative way to speed up co-simulation would be to temporality disable the clock input to the RTL blocks during it idle time, though it may not always be possible in certain cases.

5.1 Synergy of Co-simulation and Co-emulation

Co-simulation and *co-emulation* are essential for HW/SW co-verification but they should be used for different purposes within the design cycle. Their properties are compared under a number of different aspects. The comparisons are summarized into a Cost-Benefit matrix as shown in *Table 2*.

Speed: With the help of the HW emulator, simulation can be speeded up by a factor of 100. There are factors which determine the speed limit of the two verification techniques:

Table 2: Cost-Benefit Matrix of co-simulation and co-emulation.

	Speed	*HW Resources*	*Ease of Use*	*Debugging Ability*	*Usability for SW Development*
Co-simulation	Slow (1k cycles per second)*	Linux PC	Automatically generated SystemC proxy	Full system visibility	Concurrent usages
Co-emulation	Fast (104k cycles per second)*	Hardware emulator + Linux PC	Knowledge to create custom transactors	Difficult to set breakpoints	Single user usage

**Result obtained from our DFE Rx experiment.*

Table 3: Platform configurations with major components shown under the two verification environments.

Platform Components	*Co-simulation (Virtual Platform)*	*Co-emulation (Emulator + SystemC)*
AGC (DFE Rx)	RTL	RTL
Sync Pro (DFE Rx)	ISS	RTL
FIFO and filter (DFE Rx)	SystemC	RTL
DMAC	RTL	RTL
AMBA System	SystemC	RTL
Other system level components	SystemC	RTL
Stimuli generation and evaluation	SystemC	SystemC

Co-emulation: Mainly depends on the number of transactions passing between the emulator and the SystemC via the transactors. The critical path in synthesized design defines the emulator clock speed.

Co-simulation: The speed bottleneck is the interface between the SystemC and RTL simulators. The number of events inside the RTL design has a relatively small influence on the simulation speed.

Hardware Resources: For co-simulation, a PC is required to simulate the virtual platform. For *co-emulation*, a HW emulator is needed.

Ease of Use: To enable co-simulation, only a simple SystemC proxy is required. These proxy blocks are readily available in the Platform Creator tool. For *co-emulation*, the learning curve is steeper as considerable effort is required to create custom transactors.

Debugging Ability: Full system visibility is available on the virtual platform during co-simulation, whereas in *co-emulation,* SW debugging is more difficult. Co-simulation provides more controllability and debugging facilities of the complete system. For example, it is feasible to set a break point to stop the entire system at once, whereas it is difficult to stop the clock of the HW on the emulator at the instruction boundaries of the ARM. The Platform Creator tool has profiling functionalities to trace and analyze the status of the system such as the bus contention and cache statistics.

Usability for SW Development: Co-simulation allows multiple instances of the platform to be simulated and hence concurrent usage of the virtual platform on individual PC for SW development is possible. In case of *co-emulation*, concurrent SW development within the team is limited as there is only one instance of the platform running on the emulator.

6. CONCLUSIONS

We presented a practical use of a complete ESL design flow from high level virtual platform modeling to HW/SW co-verification of a large scale SDR SoC design. We discussed the advantages of using the ESL tool to achieve a more efficient design via the demonstration of platform exploration steps during the SDR platform architecture refinement. We successfully co-emulated our design in the CoWare's SystemC and the Mentor's emulator simulation environment via the custom transactors which we developed. We presented a cost-benefit matrix between two verification techniques, co-simulation and *co-emulation*. We also performed platform experiments based on a wireless application to evaluate the simulation speed of the aforementioned verification techniques in our ESL environment.

7. REFERENCES

[1] H-M. Bluethgen et al., "Finding the Optimum Partitioning for Multi-Standard Radio Systems," *Proc. Software Defined Radio Technical Conference*, Nov 2005

[2] B. Bougard et al., "A Low Power Signal Detection and Pre-Synchronization Engine For Energy-aware Software Defined Radio", *SDR Forum*, USA, Nov 2007

[3] A. Sinha et al., "Energy-Scalable System Design," *Trans. VLSI*, Apr 2002, pp 135-145

[4] T. Kogel et al., "Virtual Prototyping of Embedded Platforms for Wireless and Multimedia," *IEEE/ACM Conference on Design Automation and Test in Europe (DATE)*, Mar 2006

[5] G. Gailliard et al., "Towards a SystemC TLM based Methodology for Platform Design and IP reuse: Application to Software Defined Radio", *RECOSOC,* Jul 2006

[6] T. Kogel et al., "TLM Methodology Guideline", *CoWare Inc*

[7] http://www.coware.com

[8] A.. Hoffmann et al., "Architecture Exploration for Embedded Processor with LISA", *Kluwer Academic Publishers*

[9] CoWare AMBA bus TLM library manual