Case Study on Universal Verification Methodology(UVM) SystemC Testbench for RTL Verification

KEVIN SKARIA CHACKO

MASTER´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Case Study on `Universal Verification Methodology(UVM) SystemC` Testbench for `RTL` Verification

Kevin Skaria Chacko
`ke4067fn-s@student.lu.se`

Department of Electrical and Information Technology
Lund University

Supervisor: Liang Liu

Examiner: Erik Larsson

June 11, 2019

© 2019
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

This Master's thesis aims to conduct a case study on using Universal Verification Methodology (UVM) in SystemC for Register-Transfer Level (RTL) verification. Verification of ASICs is very important nowadays especially in terms of production costs, time to market and the sustainability of products. As Moore's Law is in motion, verification gets large, complex, and time-consuming. Re-usability and simulation performance of testbenches are key areas for improvement. Accellera Systems Initiative, a standard organization that supports user and vendor standards in the field of EDA and ICs has released UVM in SystemC as a Beta version. Since SystemVerilog UVM testbenches have been around in industries for several years in different performance optimized versions, one of these versions is implemented in UVM SystemC to study if the same optimizations are possible. This will also enable to have a common language platform for SystemC Register-Transfer Level (RTL) and RTL testbench development. In order to simulate two different hardware modeling languages in an EDA simulator, 3 different methods of interfacing UVM SystemC with SystemVerilog RTL have been explored: UVM-Multi Language (ML), Beta over Legacy (βoL), and Standard Co-Emulation Interface (SCE-MI). Based on simplicity and simulation time the βoL interface has been characterized for a better option. Moreover, knowing the relationship of SystemC simulation time with timing accuracy, a performance comparison with SystemVerilog UVM has helped to understand the bottleneck of SystemC testbench with cycle-accurate interfaces as compared to SystemVerilog interfaces where the simulator tool perform optimizations in context switching. From the results of the performance comparison, this study proposes a direction in using a hybrid testbench model where the stimulus and response data parts of the testbench are still in SystemC TLM whereas the cycle accurate interfaces with the RTL are in SystemVerilog.

Acknowledgments

To begin with, I would like to thank my supervisor, Professor Liang Liu and my examiner, Professor Erik Larsson for their guidance in this project. They have helped me to bring up the scientific aspect of this thesis all along the way. Similarly, I am thankful to my supervisor at Ericsson, Ole Kristoffersen for sharing his vast experience in ASIC verification and always encouraging me to think outside the box. I would like to show my gratitude to the Application engineers at Cadence, David Spieker and Hans Martin for providing constant support throughout the thesis. I am also thankful to my Manager at Ericsson, Torsten C Larsson, from inspiring me from the beginning being a guest lecturer at LTH to giving me an opportunity to do this thesis in his team.

To my colleagues in Ericsson: Grzegorz Swiecanski, Deepak Yadav, Arun Jeevaraj and Dimitar Dikov, I am always grateful for assisting me from the basic technical problems in the starting days to the end of this thesis work.

To my family, who have given me the motivation in pursuing my passion and interest and been the foundation of my success. I can never thank you enough.

To my friends in Lund: Aishwarya, Michail, Ghanashyam for bearing my thesis talks during lunch and after work, trying to understand and providing unconditional support in achieving my goals. Thank you for helping me in your best effort.

To my friends from National Institute of Technology India, who have helped me realize my potential to complete this thesis. This thesis has your imprints all over it. To my high school teacher, Ms. Seena, without you I would not have remembered my OOP concepts needed for this work.

To every person that I forgot to mention here, those who have played even the smallest part in my life. I thank you from the bottom of my heart.

Popular Science Summary

You need to give up something to achieve something

As the digital electronics and semiconductor industries tries to follow the Moores Law: *"The number of transistors and resistors on a chip doubles every 24 months"*, the verification process of these chips gets large and more complex. The need for verification engineers in an ASIC/IC development project has surpassed the design engineers since a few years for the same reason.

Nowadays electronic industries compete with each other to reach the market first and give the most innovative and efficient products. It can be seen with the new standards released every decade in mobile communications: GSM, 3G, 4G and now 5G. To make sure these industries meet the consumers demands for high speed communication, they need to find faster and efficient ways of designing and verifying their products. This would be the context of this thesis work: to find a direction on how to reduce the verification time of large and complex digital hardware designs.

Digital Circuits are designed with Hardware Description Language (HDL) which is like a programming language to implement a Register-Transfer Level (RTL) abstraction model of data flow and timing of a circuit. These HDL's can be simulated to check their correctness with testbenches. The traditional testbenches are usually written with HDL languages in different standards or methodologies to interface better with the Design Under Test (DUT). Therefore if an HDL testbench can be adapted to a more abstract language such as SystemC based on C++, it will be able to compile and sequentially run them much faster and efficiently.

HDL Testbench languages typically vary based on their size and complexity in verifying hardware designs, smaller designs use VHDL/Verilog with no specific standard or methodology whereas bigger designs uses SystemVerilog in Universal Verification Methodology (UVM) framework. These languages are tools of a trade like carpentry and the methodologies explains how efficiently the trade can build something big like a house.

This thesis work will verify a RTL hardware design of Ericsson with a testbench on SystemC Universal Verification Methodology(UVM) framework and compare its simulation time to already existing SystemVerilog UVM framework. The study of comparing simulation performances will help to understand the bottleneck of using UVM SystemC and provide a direction on how to improve the current best performance.

Acronyms

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BFM	Bus Functional Block
CMAFIR	Cascadable Multiplexed Asymmetric FIR
CPU	Central Processing Unit
CTSR	Clock to Sample Ratio
DUT	Design Under Test
EDA	Electronic Design Automation
FE_AFIR	Filter Engine Asymmetric FIR filter
FF	Flip-flop
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
GDB	GNU Debugger
GSM	Global System for Mobile Communication
HDL	Hardware Description Language
HiL	Hardware in the Loop
HLS	High Level Synthesis
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IFS	Integrated Functional Verification Script Environment
IP	Intellectual Propriety
ML	Multi Language
OA	Open Architecture

OOP	Object-oriented programming
RCP	Rapid Control Prototyping
RTL	Register-Transfer Level
SCE-MI	Standard Co-Emulation Interface
SoC	System on Chip
TLM	Transaction Level Modelling
UVC	Universal Verification Component
UVM	Universal Verification Methodology
VHDL	VHSIC Hardware Description Language

Table of Contents

List of Figures

List of Tables

Chapter 1
Background

In this chapter, the focus is to introduce several terms and concepts related which acts as pillars on which the thesis stands. The first section explains the role and importance of verification in digital ASIC design. It also describes how a verification plan is developed for the design flow. The second section describes about the design or Intellectual Property (IP) that is used for verification. It will give a representation of the hardware architecture showing the data and control path which is very important for functional verification. Along with the architecture, details about the design parameters, signal interface, HW/SW interface are given. The third section explains the basics of UVM testbench using System Verilog and their difference mainly from the traditional HDL testbenches. It shows how testbenches have evolved from the traditional HDL approach to using System Verilog testbench tools and then using a UVM standard to build an efficient environment for the System Verilog testbenches. The last section of this chapter has a collection of previous works on SystemC which are relevant to this thesis. It gives a good amount of information on using SystemC for verification in UVM standard.

1.1 Importance of Verification

ASIC verification is a process in which an ASIC design is verified against a provided design specification before tape out. The main focus is to achieve functional correctness of the design before the tape out. The verification process is considered very critical part of the design flow because any bugs in design not detected before tape out can lead to increase in overall cost of the design process. The increasing cost associated with the development of high end ASIC's has forced an important shift in the development process: ASIC's are more generalized and are no longer used for just one product but have a longer life span and will be used across multiple generations of the same product. This has led to spending more than 75% of the time on verifying the ASIC design than on designing the ASIC itself.

A verification plan lists the procedures and methods to be used for verification, development of testbenches and automation. They are usually constructed incrementally at the end of each iteration of the development cycle. In fig 1.1, it shows a basic verification plan along side of the ASIC development.

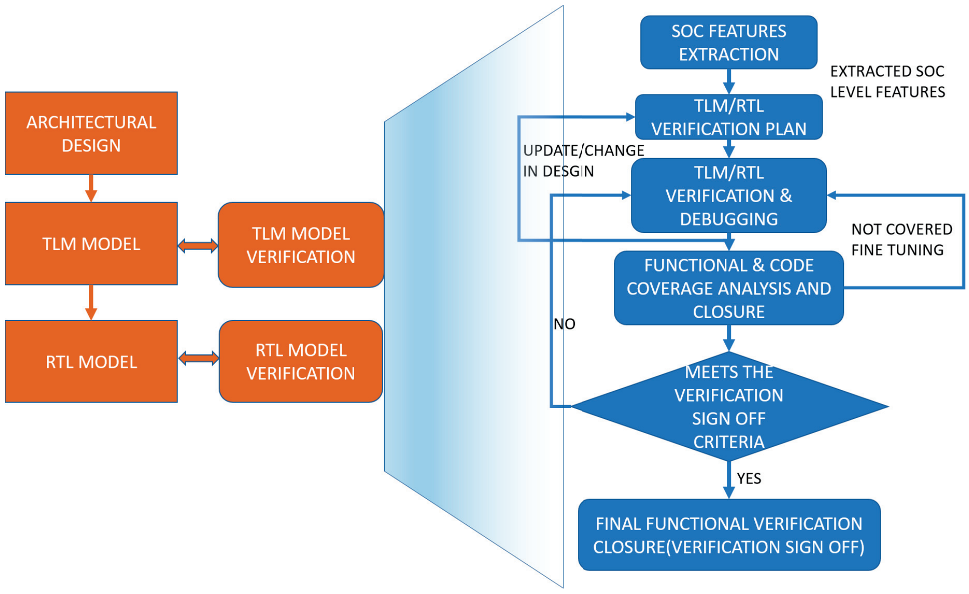

ARCHITECTURAL DESIGN
TLM MODEL
TLM MODEL VERIFICATION
RTL MODEL
RTL MODEL VERIFICATION
SOC FEATURES EXTRACTION
EXTRACTED SOC LEVEL FEATURES
TLM/RTL VERIFICATION PLAN
UPDATE/CHANGE IN DESGIN
TLM/RTL VERIFICATION & DEBUGGING
NOT COVERED FINE TUNING
FUNCTIONAL & CODE COVERAGE ANALYSIS AND CLOSURE
NO
MEETS THE VERIFICATION SIGN OFF CRITERIA
YES
FINAL FUNCTIONAL VERIFICATION CLOSURE(VERIFICATION SIGN OFF)

Figure 1.1: A basic verification plan

The architectural design is defined as the first step of every digital ASIC development flow. These are developed in algorithm level using Matlab/C++ tools. Before jumping to the RTL modelling, a transaction level model (TLM) defined in SystemC is developed to gain better hardware accuracy and save time in handover to the verification team. The verification plan of the TLM model and RTL model will be similar except in terms of abstraction level of the hardware model and tools used.

The first step of verification plan is to view the design from a top level and extract its SoC level functionality or features. At this stage, a thorough understanding of the specification is required because misunderstanding often lead to bugs and wastage of time. The next step is developing the verification environment with a verification methodology. The selection of methodology should be based on re-usability and compatibility to the legacy verification environment. A part of the verification environment development, standard verification components such as drivers, monitors and reference models have to be developed. For processor based ASICs, a processor is used to configure the stimulus. Whereas for standard protocols, third party Verification Intellectual Property (VIP) can be used for speeding up the verification process.

In the verification and debugging step, different test scenarios are developed respective to the ASIC features, these testcases can be reused to speedup verification of similar ASIC designs that share the same features. Debugging is a time consuming process, proper understanding of the detailed architecture of the design blocks will help reducing the debugging time. The functional and code coverage closure is one of the major step for the successful tape out of ASICs. Proper analysis and review will help in closing to 100% coverage. In the code coverage for the uncovered code, a proper justification as to either it is redundant RTL code or that code will be never be exercised or it is not covered during functional verification should be given. Then those types of code have to be added to the exclusion list after the designers confirmation.

In the final functional verification closure, all the criteria have to be fulfilled that has been defined in the verification sign-off list such as 100% functional and code coverage, consecutive defined number of clean regression and all bugs closed or fixed. Functional and code coverage provides confidence on the maturity of the design.

1.2 Design Under Test

The hardware design or IP used for the verification task is a configurable multi rate Filter Engine Asymmetric FIR filter (FE_AFIR) with real valued coefficients able to process complex data. With FE_AFIR, it is possible to cascade complex asymmetric FIR filters to build larger setups. These filters inside the FE_AFIR are called the Cascadable Multiplexed Asymmetric FIR (CMAFIR).

1.2.1 Hardware Architecture

The Hardware architecture of the FE_AFIR is given in Figure 1.2.

The Hardware architecture shows that there are multiple CMAFIR inside a FE_AFIR which are cascaded to build larger filters. Depending on the the no of CMAFIR blocks there will be corresponding standard interface block to these filters in FE_AFIR. Considering the data path in FE_AFIR, the input data could be given from input interface to the CMAFIR block and goes out through the output interface, or it could be transferred from one CMAFIR block to another in case of cascaded filters. Whereas, considering the control path, there will be control signals to handle these two different flow of data. The control delay block adds a configurable delay to the control signal from the input interface block to the output interface taking in account of the data processing time in the CMAFIR block.

1.2.2 Functional Specification

FE_AFIR acts as a filter engine facilitating the option of cascading multiple filters to build large systems. So, functional specification of FE_AFIR will be mainly related to CMAFIR block which actually performs the filter operation. CMAFIR is a parametric block that shall perform certain properties:

- It should process on real or complex data with real coefficients
- It should have a asymmetric impulse response.

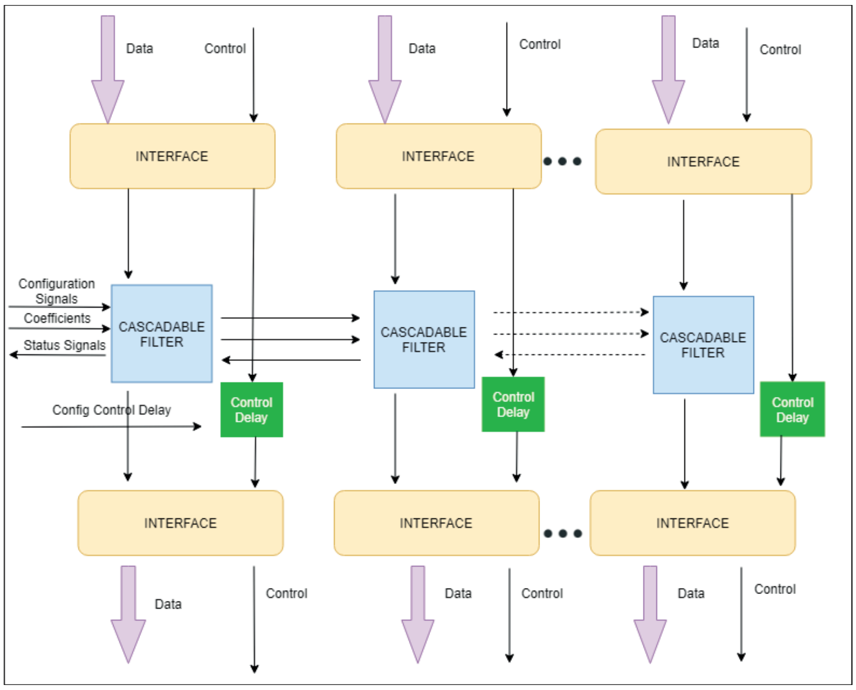


Figure 1.2: A representation of the FE_AFIR

- One section of a CMFAIR should contain one Multiplier Accumulator and a delay line. These should be parametric sections with parametric number of coefficient for one multiplier.
  - The number of active coefficients per section should be configurable.
  - It should be able to support Clock to Sample Ratio (CTSR) of 1.
- FE_AFIR could use single or double coefficient bank.

1.2.3 Design Parameters

As discussed above, most of the Hardware Design parameters are configurable. So, there are options to chose the number of filter branches in FE_AFIR, sections in CMAFIR, the coefficients per section in CMAFIR. Even the word length of the the input, interface and output signals are configurable.

1.2.4 Signal Interface

A DUT interacts with the testbench and the external world through the interface signals. These signals are the basic information needed for the Bus Functional Block (BFM) in verification. These signals could be individual or buses, they are categorized accordingly in Table 1.1.

S.No	Signal Interface	HW/SW interface
1	Clk	Signals of read and write type
2	Reset	Status signal of read type
3	Standard interface inputs, Standard interface outputs	Configuration signals of command type
4	Configuration and Status ports for the filters	Interrupt signals
5	Configuration and Status ports for selecting/changing coefficient banks	Memory Interface Signals
6	Overflow Interrupts	
7	Dynamic Power Save	
8	Memory Interface for filter Coefficient Configuration	

Table 1.1: List of DUT interfaces

1.2.5 HW/SW Interface

These are the configuration and the status signal interfaces of the FE_AFIR. They are grouped according to different register type they are associated with as seen in Table 1.1.

1.3 SystemVerilog UVM

The need for more advanced ways of testing a digital hardware ASIC's has come in place because of their complex designs. It has moved out of the traditional HDL testbench to SystemVerilog testbench which provides additional tools and then to a more accepted standard called System Verilog UVM [1].

1.3.1 Traditional Testbench

In the traditional testbenches using VHDL/Verilog, the tests are implemented as a group of procedures or a module containing these procedural blocks. There is little or no active verification [2]. A block diagram is given in Figure 1.3.

The Stimulus generation is limited to the designers imagination. There is no proof of which part of the DUT has been tested. Sometimes this has changed to monitor some checking in the testbench but they are also very specific to the test and are not reusable. There is no standard approach in designing these testbenches [3].

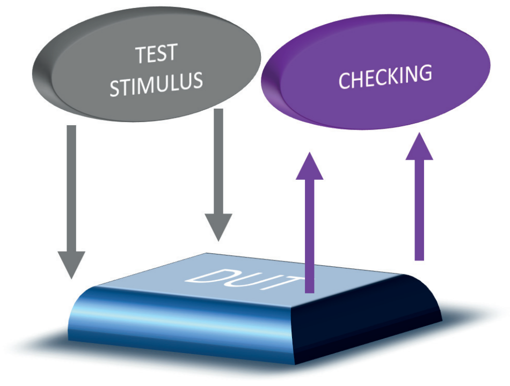
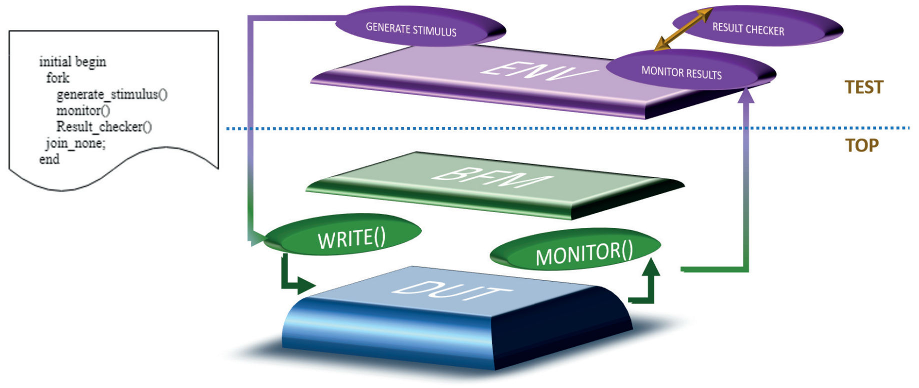


Figure 1.3: Traditional testbench Architecture

1.3.2 System Verilog Testbench

In moving to System Verilog testbenches as shown in Fig 1.4, verification jobs are arranged into tasks. Tasks with the help of functions help in breaking large procedures into simpler steps that are easy to read, understand, debug, maintain and reuse. System Verilog introduces Bus Functional Module(BFM) interfaces which are like modules that contains all the I/O signals. These interfaces can be enhanced with tasks and functions to modify the interface variable and DUT ports. The main advantage of the BFM interfaces are that it helps encapsulating bus protocols inside the interfaces and not the test. The generate_stimulus task will randomize the input data and will give to the DUT via the BFM whereas the monitor will catch the output from the DUT also through the BFM and then it is transferred to the result_checker to compare with a reference model. All the three tasks are run simultaneously [4].



```
initial begin
 fork
   generate_stimulus()
   monitor()
   Result_checker()
 join_none;
end
```

Figure 1.4: System Verilog Testbench Architecture

SystemVerilog also replaces process statements by join_none statements, where any number of process can be spawned simultaneously without any impact on the flow of main process. To have communication between these process, SV uses the Mailboxes which are FIFO queuing mechanism between threads [5].

1.3.3 Universal Verification Methodology

It is a verification methodology that can verify from system level to RTL. UVM was created by Accellera based on the OVM (Open Verification Methodology) version 2.1.1. The roots of these methodology lie in the application of the languages IEEE 1800^{TM} SystemVerilog, IEEE 1666^{TM} SystemC, and IEEE 1647^{TM} e [6]. It provides an environment that makes use of the advanced verification techniques such as functional coverage, constrained random stimulus generation, assertions etc. It also provides classes for observing information about the DUT activity, reuse stimulus generation and mirroring registers in hardware. The main directive of the UVM is to reduce the time to built a verification environment. It must be able to reuse all the work as much as possible.

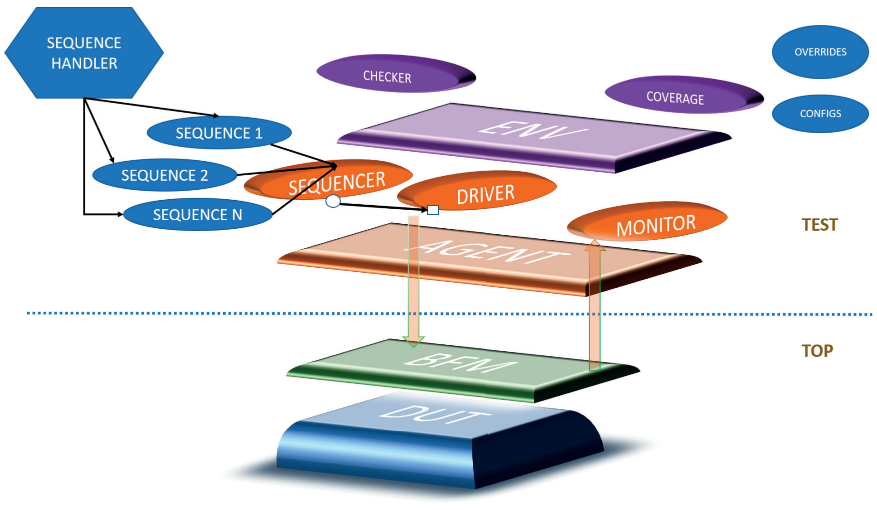


Figure 1.5: UVM System Verilog Testbench Architecture

A UVM standard testbench will encapsulate verification components called Universal Verification Components (UVC's) which are defined in a class based approach (Object Oriented Programming). Classes helps to encapsulate both the data and its behaviour in one component. The architecture of a UVM SystemVerilog testbench is given in Figure 1.5. The UVM *test* is the top UVC and it performs mainly three functions: instantiates the environment UVC, configures it via overrides and invokes the UVM sequences through the sequence handler. The *env* UVC groups together other interrelated components targeting the DUT such as Agents, Checker, Coverage etc. The UVM *Agent* UVC will target a specific DUT interface and group together related components such as *Sequencer* to control the stimulus flow, *Driver* to apply stimulus to the DUT and *Monitor* to monitor the DUT interface. The *Checker* component is responsible for comparing with a reference model and the *Coverage* is responsible for ensuring the functional coverage is achieved based on the human generated metrics derived from the test plan.

1.4 Previous Work on SystemC

Currently, SystemC is used to describe digital systems at different levels of abstraction, starting from functional or algorithmic description to synthesizable RTL. Due to the general capabilities of the C++ language, it can be used for system-level modeling, architectural exploration, performance modeling, software development, functional verification, and also high-level synthesis.

Several academic works have done case studies on using SystemC for System on Chip (SoC) modelling and verification. For instance, Fernando A. Escobar et al. present results of using a SytemC/TLM2.0 to closely represent or model a hardware behaviour and also for faster evaluation of multiple test scenarios. It is seen how the whole router functionality is explained as SystemC methods and how it invokes other processes. The authors claim improvement in simulation performance by using SystemC TLM model compared to HDL ones but concludes the lack of detailed information [8]. Another interesting work from Ali Sayinta and et al. shows how a SystemC testbench can be used for SystemC model validation, implementation and prototype tests of an OFDM based Wireless LAN SoC. This is referred to as "Vertical Testbech Reuse". It also shows that a valid SystemC model can be used as a golden reference for later stages of the hardware design. A foreign language RTL description cannot interface with SystemC model directly for the co-simulation purpose without having interface wrappers. The wrappers are responsible for two tasks: First, it refines the abstract data types of SystemC to more basic ones. Second, it implements missing communications, protocols etc. which are abstracted at the system level. The authors describe the shortcoming of the tools to provide support for interfacing the different abstraction levels [9].

Additional work by Jeongwood Park and et al. shows that SystemC TLM can be co-simulated with an RTL HDL to a surveillance camera system in order to reduce the effort to develop and simulate the whole design in RTL HDL. SystemC TLM provides a higher abstraction therefore components of the system can be implemented by using it. For verifying the co-simulation, the SystemC components are operated at PC and the RTL blocks are downloaded to FPGA to support emulation. The authors conclude that no additional time was needed to change the architecture for co-simulation to the FPGA platform [10]. Whereas in a paper by Stuart Swan from Cadence Desing Systems, Inc. describes how SystemC transaction level models are being reused for RTL verification. There are two keys enabling SystemC TLM models to be leveraged for RTL HDL verification: First, the simulator tool should support co-simulation and co-debug of mixed SystemC and HDL models. Second, proper transactors should be developed to interface with the SystemC TLM model which is dependent on the level of abstraction of the TLM testbench [14].

A close work related to this thesis topic was done by George Kalo for the ISS Group, Stockholm. The work describes on how to write a testbench in SystemC to verify an existing VHDL design. It is quite interesting to see that the interface between SystemC and VHDL is tool vendor specific. The author used a stub module, a foreign module declaration using the *scgenmod* command in Modelsim. The testbench is not created under any verified methodology such as UVM which could help verifying with large and complex systems. Furthermore, the author does not compare the simulation performance of the SystemC testbench with the VHDL testbench which is also mentioned in the future works [11]. Another close work by Myoung-Keun You and et al. on using SystemC for functional Verification of RTL can be seen where the infrastructure of the non UVM verification system uses intermediate user-defined channels as communications interface between SystemC and RTL HDL. It is implemented using the Verilog Procedural Interface which cannot work with Verilog source code but only with simulation data structure. But the author does not discuss the results on the simulation performance of the the testbench in this work [12].

Frank Poppen and et al. have worked in connecting an industry's verification methodology to standard concepts of UVM. They had a Integrated Functional Verification Script Environment (IFS) verification methodology implemented in SystemC. They have instantiated UVM components in their SystemC test environment to verify design specified in VHDL, Verilog or any other language. Since Multi - Language UVM approach is vendor tool specific, they have tried with Mentor Graphics Questasim and Cadence Incisive and claim that their work is not dependent on proprietary technology but applicable to any SystemC based environment [13].

Chapter 2

Objectives, Challenges and Methodology

In this chapter the main objectives of this thesis work will be presented. And once the objectives are described, the challenges they present and the proposed methodology for dealing them are explained under the same section.

2.1 Main Objectives of this Work

The objectives are divided into 3 main tasks: Implementation of the UVM SystemC testbench, finding possible interfaces between UVM SystemC testbench and the Verilog RTL and finally evaluating the simulation performance between UVM testbenches in System Verilog and SystemC.

2.1.1 UVM SystemC Testbench

This task is to develop an UVM SystemC testbench based on the Accellera standard in βeta version. The motivation is to study if a unified verification methodology completely based on SystemC/C++ is possible or not. By developing a UVM SystemC testbench, we are enabling the possibility to create more advanced system-level testbenches and developing reusable verification components for both system-level and block-level verification.

Challenge: The existing System Verilog testbench is an optimized version with best known simulation performance. The optimizations are implemented by executing concurrent parallel processes (Forked process) in the transactors such as drivers and monitors of the testbench. As these processes determine the simulation time of the testbench, implementing them in SystemC and making sure their functionality are not compromised is a huge challenge.

Methodology: Multithreading is an important feature that helps in optimizing the simulation performance of the testbench during run time. Therefore understanding their functionality and implementation of these features in UVM SystemC are crucial in terms of simulation performance. Most of these are used in the run phase of the instantiated UVM components.

In the UVM SystemVerilog testbench, Multithreading is implemented through 3 different types of forked process as shown in Figure 2.1

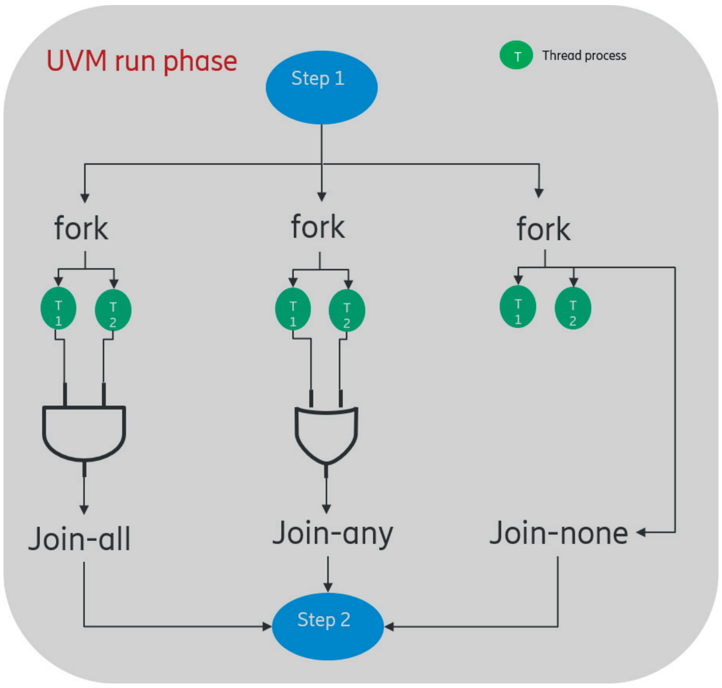


Figure 2.1: Different implementation of the forked process

In the fork-Join-all feature, the parent process *(mostly the run phase)* executing the fork statement is blocked until the processes spawned by the fork are terminated. Whereas, in the Fork-Join-any feature, the parent process does not start its execution until one of the processes it spawned has terminated. In Fork-Join-none, the parent process continues to execute concurrently with the spawned processes but they do not start executing until the parent process comes across a blocking statement. Apart from Fork Join all, there are no pre-defined macros to use the other two features. Therefore, it will be challenging to implement Fork-join-none and Fork-join-any in SystemC to give the same functionality as in Verilog.

The architecture of the UVM SystemC testbench that needs to be developed is given in Figure 2.2. The top level i.e sc_main has the test(s) and the DUT. There are no virtual interface in SystemC as compared to SystemVerilog, hence the interconnection among the UVCs in the top module should be stored in the configuration database from where it can be used by the UVC. The UVM configuration database is a convenience interface used for configuring UVM_component instances. In this testbench there are 5 UVCs and hence there should be corresponding number of agents and subscribers in the scoreboard to record all the signals for verifying with the golden C++ reference model. Subscribers helps to design analysis modules in determining the coverage of the tests.

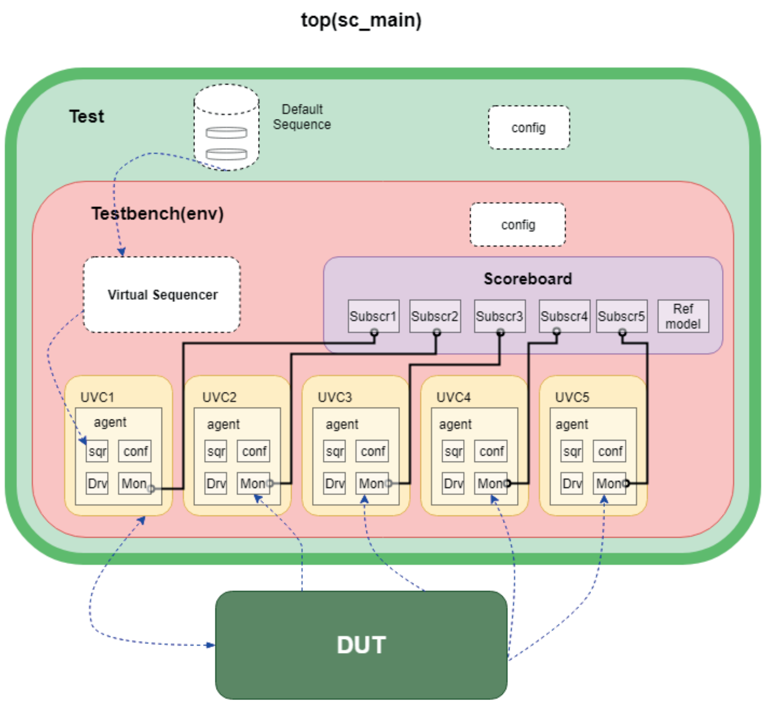


Figure 2.2: SystemC UVM testbench architecture

Implementation of constrained randomization, assertions and functional coverage as in UVM SystemVerilog are not needed as they are available only in later release of the official version of the UVM SystemC library.

Also it is important to know that all the components of the UVM testbench are kept at the Transaction Level Modelling (TLM) abstraction layer as seen in Figure 2.3, whereas the driver and monitor also known as transactors are dragged down to signal-oriented register transfer level of abstraction in order to connect to the DUT in cycle accurate level. TLM is a high level description of a hardware model where the emphasis is on the functionality of the data transfer among modules *(type of data)* and less on the their actual implementation. TLM can be described in System Verilog as well as SystemC. Whereas cycle accurate models like RTL have more finer details of the implementation of the modules.

2.1.2 Testbench - RTL Interface

Since the testbench is in UVM SystemC and the DUT is in System Verilog, there is a need for Multi Language (ML) interface between them as seen in Figure 2.2. The ML interface should have two main features:

- The interface should be able to use relative names to identify the ports in both the language frameworks.
- They also must be able to synchronize the UVM phases for both the frameworks.

UVM phases are synchronizing mechanism for the UVM environment. The major functional steps when a simulation is run through are:

- *Build phases*: Steps where the testbench is constructed, connected and configured.
- *Run phases*: Stimulus generation, time consuming processes and optimizations on them occur here.
- *Clean up phases*: Here the test results are collected and reported.

Challenge: Explore all the interfaces that satisfies the ML features and select the optimal one based on the implementation simplicity and better simulation performance.

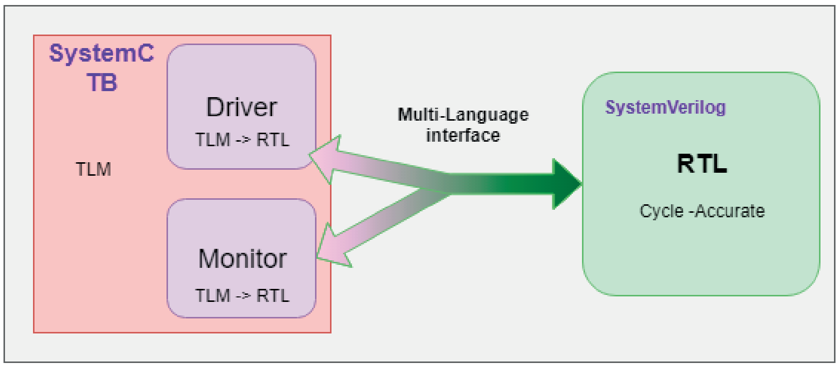


Figure 2.3: Multi-Language Interface

2.1.3 Evaluation of the Simulation Performance

The objective in evaluating simulation performance is based on using UVM SystemC instead of UVM SystemVerilog. SystemVerilog describes hardware models in a cycle-accurate abstraction level and therefore all the modules are active and sensitive to an event trigger whereas in SystemC, hardware models can be described in various timed accurate models as seen in figure 2.4 [15]

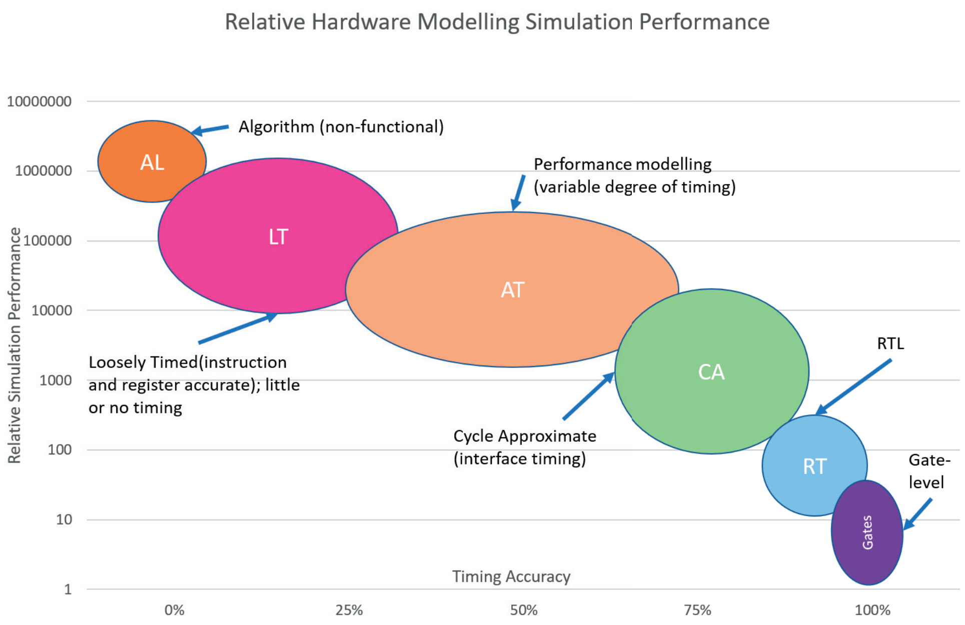


Figure 2.4: Relative hardware modelling simulation performance with timing Accuracy

Challenge: SystemC allows describing hardware from Algorithm to RTL models. As we move towards the RTL description, the timing accuracy increases compromising on the simulation performance. This flexibility in SystemC to be more time independent in TLM abstraction layer favors to conduct a simulation performance comparison between the UVM SystemC and UVM SystemVerilog testbenches. Also finding a hybrid solution that balances accuracy and performance is a challenge here.

Methodology: Hybrid Solution

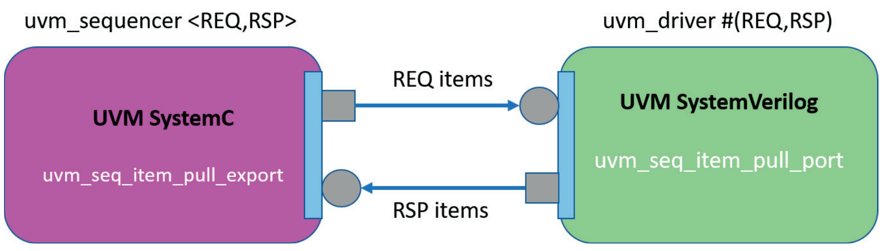


Figure 2.5: Multi Language TLM ports

The transactors communicate with other UVCs through TLM ports both in UVM SystemVerilog and UVM SystemC. Through the multi-language TLM ports available in the UVM-ML Open Architecture (OA) library of Cadence simulator, TLM transactions can communicate across languages as shown in Fig 2.5.

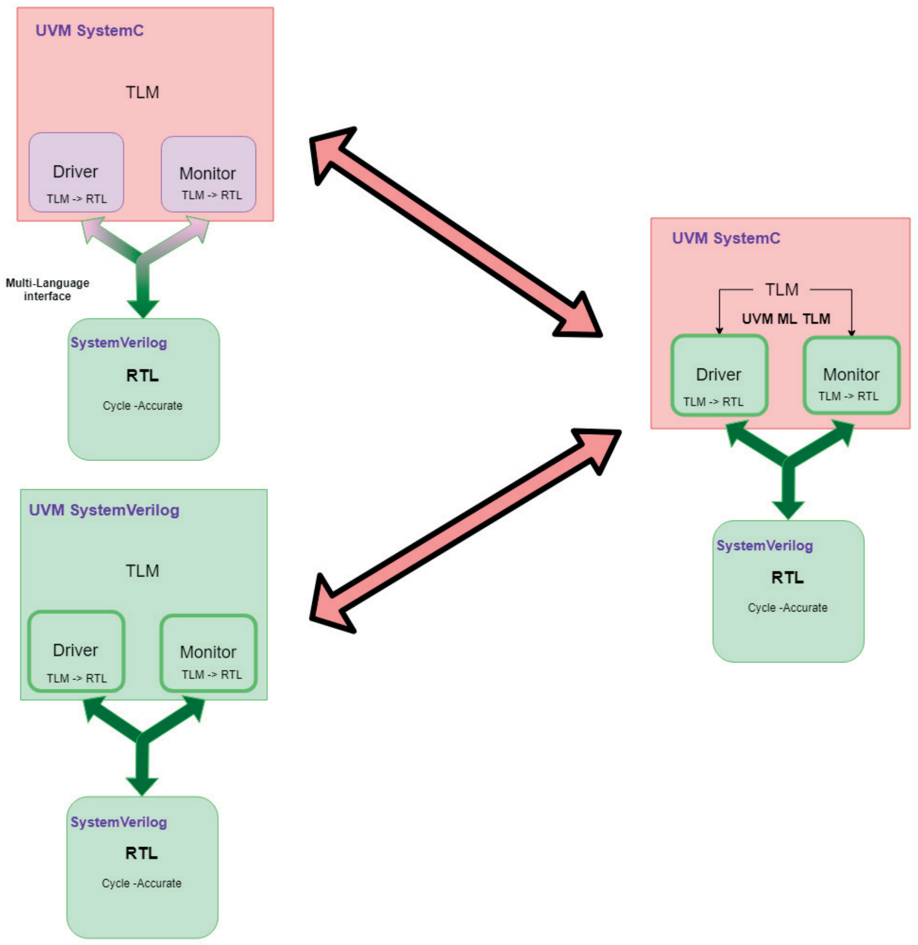


Figure 2.6: Architecture of the Hybrid solution

The requirements for enabling TLM communication in a multi-language environment are as follows:

- It has to be ensured that the data types, type mapping - *translates data type between languages* and serialization - *transfer of classes between language* should be matched for both the languages.
- Register TLM ports or sockets for communication. The ports are registers to its respective language framework adapter. As a result of this registration, the adapters create an internal channel for passing the transactions among multiple language frameworks.
- Binding the TLM ports to the framework, which can be done in any one language framework or even another framework whose ports are not registered.

There are a few limitations to use the UVM ML TLM communications such as : Disabling or killing of blocking the multi-language threads are not supported. The interface class type arguments are passed by copy and therefore the changes made to argument valued are not immediately visible across language boundaries. They are only updated at the end of the interface call.

The proposed architecture ensures the simulator optimization for SystemVerilog transactors are obtained whereas the UVM SystemC top module enables the reuse of the test bench for RTL as well as block level verification.

Chapter 3

Implementation and Results

In this chapter the details of implementing the UVM SystemC testbench are explained. The main differences while implementing UVM in SystemC from System Verilog are highlighted in the first section. In the second section, the details of exploring three different Multi-Language interfaces between the testbench and RTL are described. Also, the analysis of the simulation performance between the viable interfaces is discussed in the same section. In the final section of the chapter, the simulation performance of multiple Testbench - RTL configurations are discussed.

3.1 Key Differences in UVM SystemC from UVM System Verilog

The top differences in UVM SystemC are mainly with forward declaration of UVM objects, dynamic array size allocation, segmentation fault debugger, SystemC wire separators and forked process implementation.

3.1.1 Forward Declaration

In the UVM System Verilog testbench, forward declaration of UVM objects are used in order to implement the concept of re-usability of UVM components. Forward declaration means to declare an object before its complete definition has been given. In Verilog, the construct *typedef* is used to forward reference a class declaration. This will indicate the compiler not to be concerned with the position of the class definition.

In SystemC, there is also the option to forward declare a object by using same constructs as that of C++. But when these objects before they are defined are used as members of intermediate classes, they have to be declared as pointers and should never be initialized as they are forward declared. This is not an issue in System Verilog as they are taken care during elaboration of the hierarchy.

3.1.2 Dynamic Array Size Allocation

In System Verilog, when an array of UVM objects are declared dynamically as a member of UVM class, they can be defined with construct *class_name ar-*

ray_ name[ ]. This is different in SystemC as arrays should be declared dynamically as pointers *class_ name *array_ name*. The pointers to the arrays are initialized by using *new* where the memory in heap is allocated or *malloc* where the memory in free-store is allocated. Inappropriate initialization will lead to segmentation fault. All the memory should be released with *free* or *delete[ ]* before the next allocation to avoid wastage of dynamic memory.

3.1.3 Segmentation Fault Debugger

In UVM SystemC, it is more likely to fall into issues like segmentation fault than System Verilog. It is mainly because *'one tries to access memory that does not belong to the them'*. In complex testbenches that follows UVM methodology, debugging these issues and locating them manually will be tiresome and inefficient. Hence, we need to use a tool or debugger for such issues.

Since the testbench is developed on top of the C++ kernel, using one of the most popular debugger like GDB (GNU Debugger). The benefits of GDB are:

- The testbench can be started irrespective of the type of run time error affecting its behaviour.
- The testbench can be stopped at specific conditions or breakpoints.
- Helps examine the stack trace, with appropriate memory address.
- Also helps isolating multiple errors and re-execute the testbench solving one error at a time.

If the testbench development is done with built-in SystemC in Cadence EDA. The GDB tool can be used to debug the errors by just enabling a switch in Xcelium/Incisive.

3.1.4 SystemC Bit Reduction and Range Selection

In SystemC modules, bit reductions and range selection cannot be performed on input/output ports or signals. Whereas, in System Verilog both the features are possible.

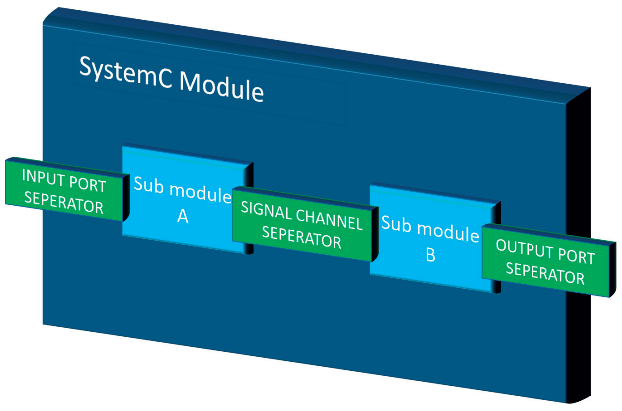


Figure 3.1: Separator modules at Input, Output and signal channels

In order to perform range selection on SystemC ports or signals for the UVM SystemC testbench, a generic separator module has been designed with the width *(W)* of the channel as the generic parameter. This separator will separate the ports and signal from bus format to individual lines. This separator is implemented as input, output or signal separator as shown in the figure 3.1.

3.1.5 Forked Process Implementation

In order to understand the implementation of forked process in SystemC, it is relevant to explain concepts such as SystemC spawn, SystemC bind, SystemC event list and SystemC terminated event function.

SystemC Spawn

One important element of SystemC is concurrency and this is achieved through processes similar to Verilog/VHDL. SystemC has two types of processes:

- **SC_THREAD** - a process that will start automatically at the start of simulation run and suspend itself for the remaining of the simulation. They are only run once.
- **SC_METHOD** - a process similar to SC_THREAD but this process can be run multiple times and cannot have any wait() statements to be halted.

SC_THREAD and SC_METHOD are static processes, that means they are created before the execution of the simulation. But for multi-threading, we need to create processes dynamically and this is possible through a function called as SC_SPAWN. Statement 3.1 gives an example, how sc_spawn with the help of an additional function sc_bind helps to create a dynamic process. Function sc_bind() helps to bind the input parameters to its respective function.

$$sc\_spawn(sc\_bind(\&class :: function, this, (function\_parameters))); \tag{3.1}$$

SystemC Event List

An event list is used to catch multiple events during mulithreading of processes. There are two different kind of SystemC event list : *sc_ event_ or_ list*, this event list is triggered only if one of the events the list catches is triggered and *sc_ event_ and_ list*, this event list is triggered only if all the events the list catches are triggered. If other processes needs to be triggered based on these lists, they can be used inside a wait() function.

SystemC Terminated_event() Function

The sc_spawn() function returns the process handle or the memory address of the dynamic process it has created. In order to determine if the dynamic process has finished executed, the sc_terminated_event function is used. The function is a member of sc_process_handle class and returns an event if the respective process is terminated. Similarly, if other processes are triggered based on this function return, they can be used inside a wait() function.

Implementation

For the fork join all implementation, the processes are spawned with sc_spawn function inside the SystemC SC_FORK/SC_JOIN macro. This macro blocks the execution of parent process until the spawned processes are terminated. Whereas for the fork join none implementation, the processes are simply spawned using sc_spawn function without the macro. This will ensure the execution of parent process is not blocked irrespective of the completion of the spawned child processes. And for the fork join any implementation, the processes are spawned and their process handles are stored in a vector list. The termination of these processes are checked through the *terminated_ event()* function. This will ensure that as soon as one of the spawned process in the vectors list is terminated, the parent process can continue execution.

3.2 Testbench to RTL Interface

In order to have a ML interface between the testbench in UVM SystemC and DUT in RTL, 3 different interfaces have been explored. In this section, explanation on the implementation of these interfaces and the performance comparison among the most viable interfaces are given.

3.2.1 UVM Multi-Language(ML)

UVM ML is a open architecture binded with Cadence simulators, where it uniquely integrates e, SystemVerilog, SystemC, C/C++ and other languages into a cohesive verification hierarchy. Given its extensive features, the foreign module instantiation API helps to have a ML interface in the simulator. In this API, a SystemC module shell is created around the SystemVerilog RTL where the DUT is instantiated as a foreign language(SystemC) child component.

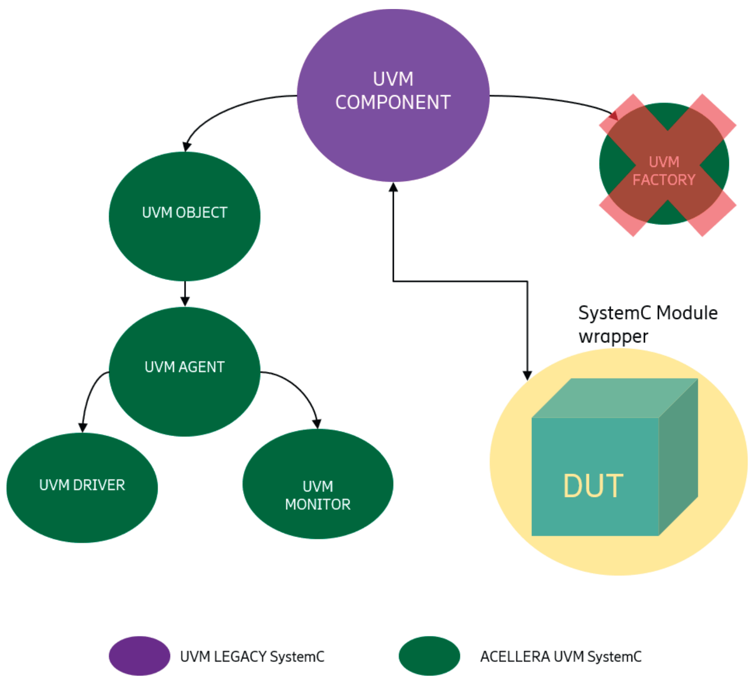


Figure 3.2: UVM ML interface using the Cadence Legacy UVM hierarchical components

In Figure 3.2, it can be seen how this method has been approached. The DUT is wrapped with SystemC module shell and interfaced with the UVM ML API. Cadence simulator has a built-in UVM SystemC which we will also be referred as the Legacy version henceforth. The Legacy version has only basic UVM component and factory classes compared to Accellera UVM SystemC. But, to adapt the fully functional UVM SystemVerilog testbench, we need additional classes in the hierarchical tree such as UVM object, UVM agent, UVM driver, UVM monitor etc. which are available in βeta version from Accellera.

Inheritance is a feature of Object Oriented Programming (OOP) and since UVM SystemC is build on C++, constructing a hierarchical structure where the additional β UVM SystemC classes inherits from the UVM component in Legacy UVM brings simplicity in the interface. This approach proved to be successful until the integration of the βeta UVM SystemC configuration database to Legacy UVM SystemC factory could not be done. This integration is an added feature to the βeta UVM SystemC and hence causes to be a showstopper for this approach.

3.2.2 βeta over Legacy(βoL)

With the incapacitation of UVM ML method, the next simplest approach would be to build the whole βeta UVM SystemC library from Accellera based on the Cadence SystemC architecture.

The Legacy UVM SystemC is binded with the Cadence SystemC in the simulator and unbinding them is not an option for the users. Therefore in this method, a switch has been implemented between the βeta and Legacy UVM systemC so that once switched to βeta UVM SystemC, there will be no more dependencies to the UVM components of the Legacy UVM SystemC as seen in Fig 3.3.

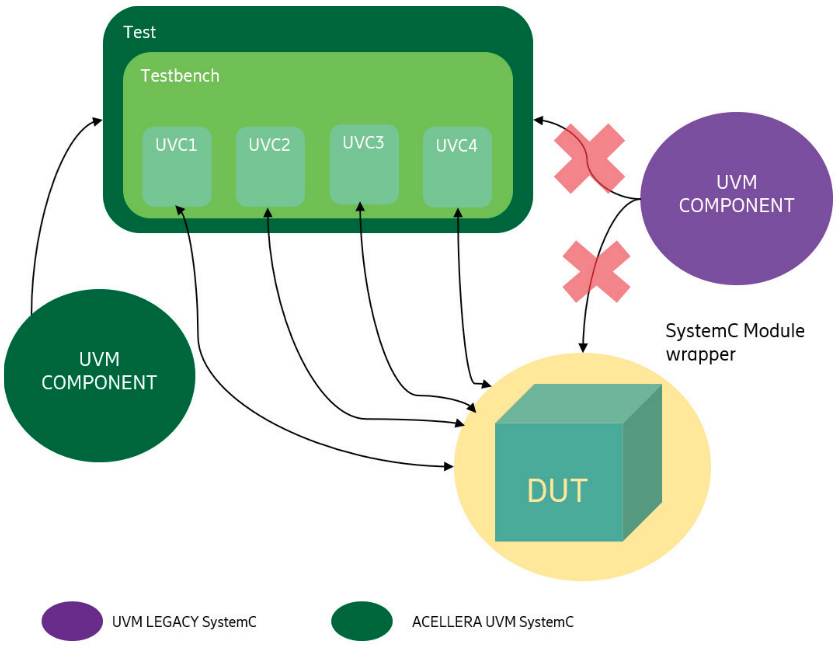


Figure 3.3: βoL interface implemented with a switch

The βeta UVM SystemC comes with a build script targeted for the Accellera SystemC. Cadence SystemC is different to OSCI SystemC as it is pre-implemented to have a Master-Slave configuration. This is to enable the reuse of the SystemC components in a UVM SystemVerilog testbench and hence SystemC in Cadence acts as a slave to SystemVerilog. Failing to understand this architecture in the initial stages has resulted in a bug where any SystemC wait statement *(sc_ wait)* in a run phase of a UVM component will terminate the whole simulation. Resolving the run phase issue later has made this method a viable option for interfacing.

3.2.3 Standard Co- Emulation Modelling Interface (SCE-MI)

In order to explore all the known possible methods in interfacing the UVM SystemC testbench to RTL, the SCE-MI is implemented with the Accellera UVM SystemC. In this method, the testbench runs in a Cadence Xcelium simulator whereas the RTL (DUT) runs in the Cadence Palladium XP Emulator.

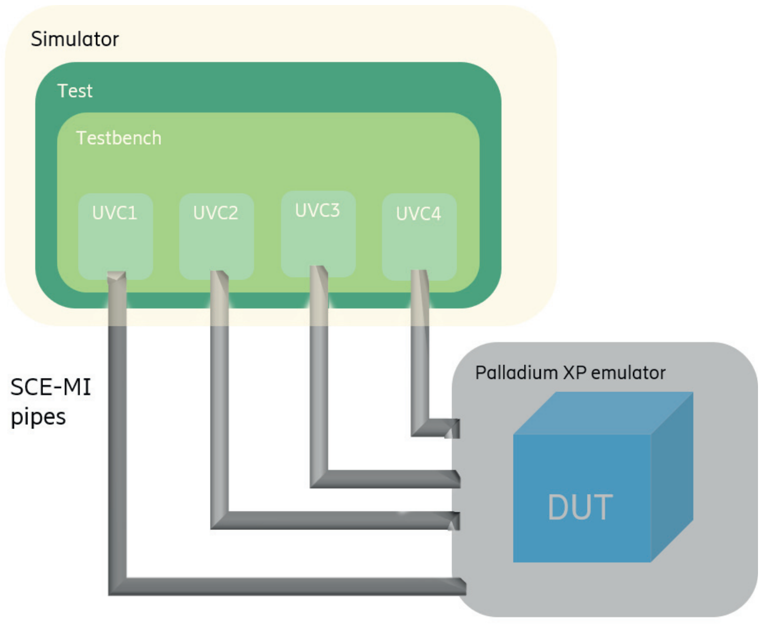


Figure 3.4: SCE-MI interface between simulator and emulator

As seen in Fig 3.4, the SCE-MI pipes helps in streaming of transaction between the testbench partition(UVM SystemC side) and hardware partion(HDL-verilog). SCE-MI pipes are unidirectional, which means that the transactions flow only in one direction. This ensures that the transactions are received by the receiver through function calls in the same order as they are sent. There are two types of SCE-MI pipes: ***Input pipes*** and ***output pipes***, where the input pipes pass the transactions from testbench to RTL and the output pipes transfer in the reverse direction. These pipes can also be clocked where they are used for complex models that requires multiple elements simultaneously.

Also in SystemC, C++ datatype assignments are sequential and blocking therefore SystemC provides sc_signal datatype to imitate non-blocking signal assignment in VHDL/Verilog. sc_signal uses the evaluate update cycle to model non-blocking assignments. To avoid race conditions in the SCE-MI interface between the testbench and DUT, non blocking assignments should be used to generate clocks in DUT and blocking assignments should be used to generate clocks in testbench.

It is important to know that SCE-MI pipes have two different operation modes: ***Deferred Visibility*** and ***Immediate visibility*** modes. In the deferred visibility mode, there is a certain lag between elements written to the pipe by the producer side and when they are visible or available for consumption by the consumer side. Whereas, in immediate visibility mode, any element produced by the producer is immediately consumed by the consumer. For cycle accurate HDL interface as it is here, it is necessary to use the immediate visibility mode.

3.3 Simulation Time Comparison between SCE-MI and βoL Interfaces

The viable solutions to interface the UVM SystemC testbench and the Verilog RTL is through βoL and SCEMI. In Fig 3.5, the results of comparing the simulation performance of the same testbench but different interfaces to the DUT are given.

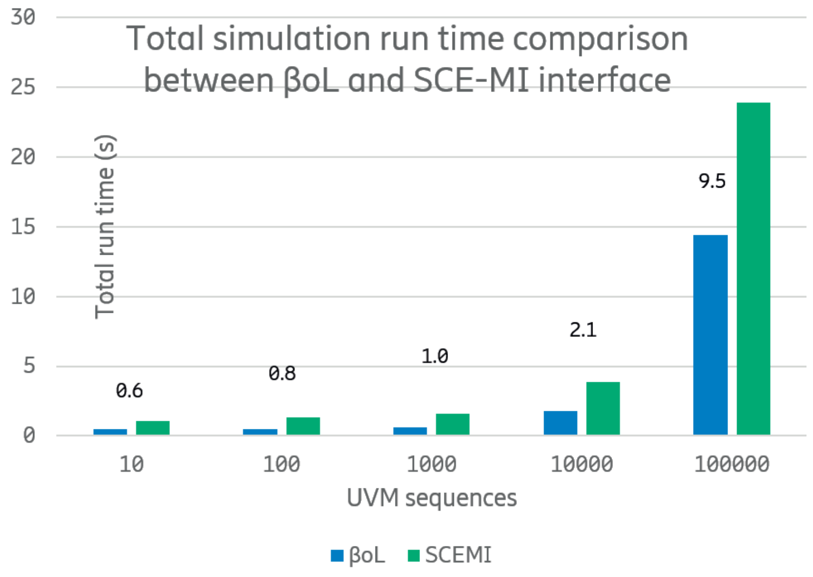


Figure 3.5: Performance comparison between SCE-MI and βoL interface

For the comparison, both the testbenches with different methods of interfacing to the RTL are run with 10 to 10^4 UVM sequences. For the analysis, the built-in profiler of Cadence simulators are used to measure the simulation time. The profiler gives the following important details:

- **Real time** : This is the clock time from the start to the finish. This includes the time slices taken by all the processes in the system and also the time spent in blocking state waiting for the I/O to complete.
- **User time** : The amount of CPU time spent outside the kernel in the user mode for executing the process. All other system processes and time spent in blocking state is not calculated towards this.
- **System time** : Amount of CPU time spent inside the kernel for system calls as opposed to library code which is calculated in user time.

In total run time includes the sum of User and System time from simulation profiler.

It is seen that the βoL interface takes much lesser time to simulate than the SCE-MI interface. This is because in the βoL interface, both the UVM SystemC testbench and the RTL are run in the simulator whereas in SCE-MI, the RTL is run in the emulator. The SCE-MI pipes appears to bring a lot of overhead to the simulation run time. The difference in performance also increases with increasing number of UVM sequences, therefore in regression mode of verification βoL promises to be more faster than SCE-MI.

3.4 Performance Evaluation between Multiple Testbench - RTL Configurations

In this section, the performance evaluation of multiple Testbench - RTL configurations based on SystemVerilog and SystemC are analyzed. The different cases are shown in Fig 3.6

3.4.1 Configurations

Case A: UVM SystemC Testbench + SystemVerilog RTL

In this case, the testbench is developed based on Accellera UVM SystemC and the RTL will be in SystemVerilog. The βOL interface between the testbench and RTL is through the driver and monitor UVCs of the testbench also known as transactors. The transactors are responsible to convert the stimulus and response data from TLM to cycle accurate hardware abstraction level.

Case B: UVM System Verilog Testbench + SystemVerilog RTL

In this case, the existing testbench is developed in UVM SystemVerilog. Apart from the golden reference model in C++, all the other UVCs are in SystemVerilog. The UVCs communicate with each other through TLM ports and the TLM transactions are translated by the transactors to cycle accurate data. There is no need of a multi language interface in this case since the RTL is also in SystemVerilog.

Case C: UVM SystemC Testbench + SystemC RTL model

In this case, the same UVM SystemC testbench is used whereas SystemVerilog RTL is replaced with its equivalent SystemC cycle accurate model. Also in this case, there would be no requirement of a multi language interface because both the testbench and RTL are in SystemC.

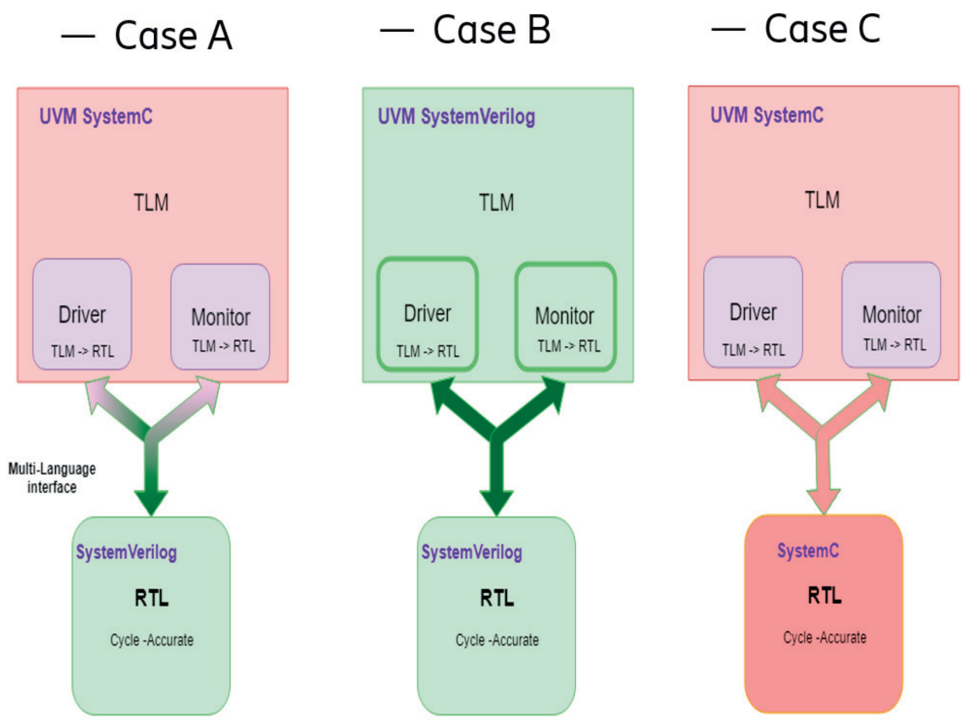


Figure 3.6: Multiple Testbench RTL configuration

All these configurations are chosen based on exploring the capability of two different hardware modelling languages : SystemVerliog and SystemC to perform roles in design as well as verification. Also, the different layers of design abstraction (TLM to RTL) are seen in these configurations.

3.4.2 Performance Analysis

All the above configurations are simulated with the same environment and increasing number of UVM sequences. In Fig 3.7, the analysis of the simulation time of all configurations are given. The performance is profiled with UVM sequence ranging from 10 to 10^4 and the total run time is the sum of user time and system time.

The results show that the **Case B**: **UVM SystemVerilog testbench** + **SystemVerilog RTL** appears to run faster in simulation time compared to Case A and Case C. Whereas, **Case A** and **Case C** have comparable simulation performance. The reason in improved performance for **Case B** lies in the concept of context switching. Context switching is a procedure that the CPU follows to change from one process to another while ensuring that the processes does not conflict.

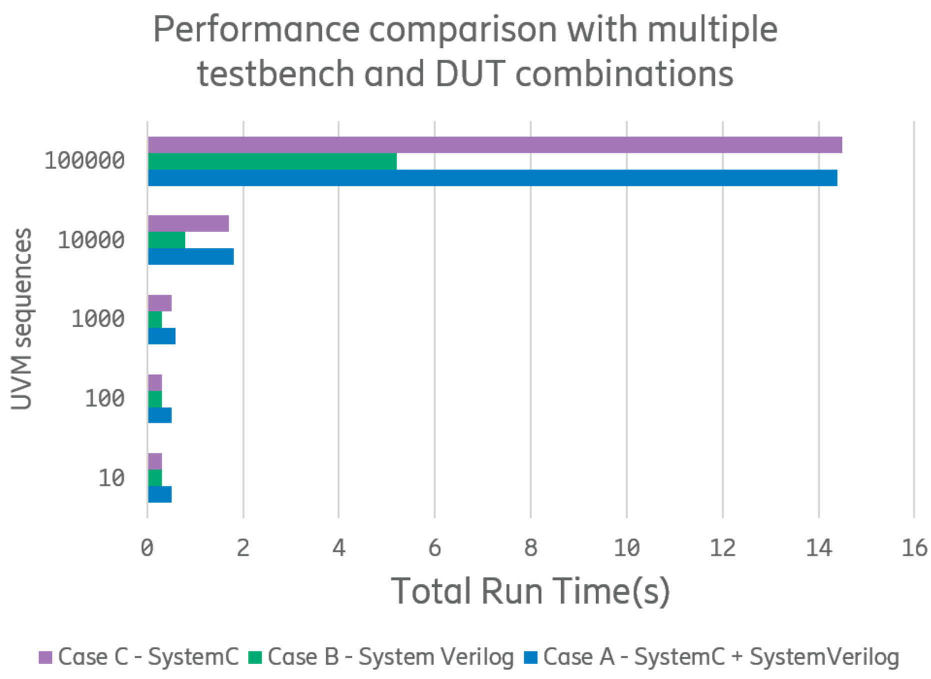


Figure 3.7: Analysis of multiple Testbench RTL configurations

RTL or cycle accurate model has a lot of processes sensitive to a clock edge to model flip-flops(registers). A non-optimized simulator will perform many context switches among these processes to maintain cycle accuracy whereas optimized simulators will combine all of these processes into a single process to save simulation time. It has been investigated that Cadence EDAs performs such optimizations in SystemVerilog simulators due to its popularity in verification and design industries. But, unfortunately these tools do not perform such optimzations for SystemC. This is verified in the simulation profiling report that the bottleneck in any configuration that involves SystemC are the transactors, as they are the verification components that bring the hardware modelling abstraction from TLM to RTL.

Since the bottleneck in Case A and Case C Testbench RTL configuration is determined to be the transactor components : driver and monitor, implementing these UVCs in SystemVerilog whereas all the other components and top module in UVM SystemC will be an optimum direction for simulation performance improvement. This concept is described as hybrid solution in Chapter 2 and Fig 2.6.

3.5 Scope of results

3.5.1 High Level Synthesis Verification

The High Level Synthesis (HLS) model allows to describe the functional intent of the hardware at a more productive abstraction level. The Cadence platform Stratus HLS allows to synthesize optimized hardware from these models. Since these models use SystemC to be described, re-using the UVM SystemC testbench can save verification time and effort. The UVM SystemC testbench will guarantee same standard of verification for the HLS model compared to the RTL model. Moreover, improvement in simulation performance can be predicted since majority of the functional verification is focused on algorithmic or TLM level. There would also be minimum RTL simulation and formal equivalence checking to verify the RTL.

3.5.2 Hardware Prototyping Platforms (Validation)

The same UVM SystemC testbench can be used across verification/simulation as well as validation/hardware prototype platforms. They require languages like C++/C for running the tests on the prototypes and measurement equipment. Since SystemC is build on top of C++ framework, compatibility is not an issue. This testbench can also enable Hardware in the Loop (HiL) simulation and Rapid Control Prototyping(RCP) strategies.

Chapter 4

Conclusion and Future Work

In this chapter, a summary of this thesis project will be provided, giving an overview of the objectives and challenges defined in the beginning, major steps taken for implementing theses goals and comments on the discussion of the results obtained from implementation. In the final section, the future works that can be derived from this thesis will also be presented.

4.1 Conclusions

This Master thesis project, titled *Case study on Universal Verification Methodology (UVM) SystemC for RTL verification* had three main objectives : One, implement an UVM SystemC testbench inspired from the optimizations in existing UVM SystemVerilog testbench. Two, find possible interfaces between UVM SystemC testbench and Verilog RTL. Third, evaluate the simulation performances between UVM testbenches in SystemC and SystemVerilog.

The implementation of the UVM SystemC testbench has led to the development of multithreading processes such as fork join none and fork join any in SystemC. These processes are critical in determining the simulation performance of the testbenches. Also, three different ways of interfacing the UVM SystemC testbench and the Verilog RTL have been explored based on viability. The three different interfaces are: *UVM-ML, βOL and SCE-MI.* The simulation performance comparison between the most viable interfaces such as *βOL and SCE-MI* has led to discovery of benefits in using βoL for better simulation time and less overhead.

In general, to understand performance gain between UVM SystemC and UVM SystemVerilog, multiple Testbench - RTL configurations have been evaluated and hence results obtained. The different configurations are : *UVM SystemC testbench + Verilog RTL, UVM SystemVerilog testbench + RTL and UVM SystemC testbench + SystemC RTL.* The improved performance in UVM SystemVerilog Testbench - RTL configuration is due to the optimizations by the simulator tool in process context switching for cycle accurate processes. Though these optimizations are not present for UVM SystemC, implementing a Mutlti-Language TLM port for the transactors in the UVM SystemC testbench will open the scope for improvement.

For the author, this project has been a huge aid in understanding the UVM methodology used for verifying digital ASICs. Application of this methodology in SystemC for RTL verification has demanded a deep understanding of the Cadence Xcelium/Incisive simulators in terms of testbench -RTL interfaces and simulation performance. Finally, the opportunity to develop this thesis with the ASIC verification team in Ericsson has given the author hands on experience to the latest verification tools and libraries.

4.2 Future Work

The future work this Master thesis spawns are as given below:

- The most immediate analysis that can be done from this thesis is to study the simulation performance from the UVM SystemC testbench implemented with UVM-ML TLM ports in the transactors. This can be done once the library dependency issues are solved by the Cadence EDA vendor.
- Another study that time did not permit in this work is to evaluate a testbench-RTL configuration where the testbench would be in UVM SystemC whereas the DUT will be a SystemC TLM model. This would help in determining SystemC RTL to SystemC TLM simulation gain.
- Constrained randomization in UVM helps to explore the design state space and combinations in the DUT to the maximum. This feature is not part of the β version of Accellera UVM SystemC but will be included in the future release version. Randomization will help in bringing UVM SystemC closer to the existing UVM SystemVerilog verison.
- Also, coverage and assertions are other features that are not part of the βeta version but once included in the official release will also enhance the features of UVM SystemC for RTL verification.

References

[1] N. Khan and Y. Kasha, *From Spec to Verification closure: A case study of applying UVM-MS for first pass success to complex MS-SoC design*, Design and Verification Conference (DVCon), 2012.

[2] J. Bergeron, *Writing testbenches: functional verification of HDL models*, in , Springer Science & Business Media, 2012.

[3] C. Pixley, A. Chittor, F. Meyer, S. McMaster, D. Benua *Functional verification 2003: Technology tools and methodology.*

[4] Willamette HDL, A. Chittor, MentorGraphics *Introduction to SystemVerilog and to the Universal Verification Methodology for designers*, Version 1.1, 2014.

[5] IEEE, 1800-2009,*IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification*, 2009.

[6] UVM 1.2 User Guide standardized by Accelera.

[7] Fernando Escobar, Mauricio Guerrero Hurtado, Lorena Garcıa Posada, Antonio Garcıa Rozo, et al. *Performance evaluation of a network on a chip router using systemc and tlm 2.0.* In Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on, pages 1–4. IEEE, 2011.

[8] Fernando Escobar, Mauricio Guerrero Hurtado, Lorena Garcıa Posada, Antonio Garcıa Rozo, et al. *Performance evaluation of a network on a chip router using systemc and tlm 2.0.* In Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on, pages 1–4. IEEE, 2011.

[9] A.Sayinta, G.Canverdi, M.Pauwels, A.Alshawa, W.Dehaene, *A mixed abstracton level co-simulation case study using SystemC for system on chip verification* Design, Automation and Test in Europe Conference and Exhibition, 2003, pp 95-100.

[10] J. Park, B. Lee, K. Lim, J. Kim, S. Kim, K.H. Baek *Co-simulation of SystemC TLM with RTL HDL for surveillance camera system verification*, IEEE Int'l Conf. Electronics, Circuits and Systems, pp.474-477, August 2008

[11] George Kalo, 2006 *Functional Verification with SystemC* (Master Thesis) , KTH Syd, Campus Telge, Stockholm .

[12] M.K. You, Y. Oh, G. Song, *Implementation of a Hardware Functional Verification System Using SystemC Infrastructure*, Proc. of TENCON 2009.

[13] F. Poppen, M.Trunzer, J.H.Oetjens *Connecting a Company's Verification Methodology to Standard Concepts of UVM.*

[14] S. Swan, *SystemC transaction level models and RTL verification,* in Proceedings of Design Automation Conference, 2006, pp. 90–92.

[15] Charles Wilson, *System Modelling and SystemC,* Xtreme EDA.

Printed by Tryckeriet i E-huset, Lund 2019

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2019-696
http://www.eit.lth.se