國立中山大學資訊工程學系

碩士論文

Department of Computer Science and Engineering

National Sun Yat-sen University

Master Thesis

以QEMU與SystemC為基礎之虛擬平臺上高效率OSCI TLM2.0介面設計與實作

On the Design and Implementation of an Efficient OSCI TLM-2.0 Interface for QEMU and SystemC Based Virtual Platform

研究生：陳智笙

Chi-Sheng Chen

指導教授：江明朝 博士

Dr. Ming-Chao Chiang

中華民國100年7月

July 2011

陳智笙

中山 eThesys 學位論文

ⓒ中華民國100年7月

陳智笙

All Rights Reserved

國立中山大學研究生學位論文審定書

本校資訊工程學系碩士班

研究生陳智笙（學號：M963040055）所提論文

以QEMU 與SystemC 為基礎之虛擬平臺上高效率OSCI TLM-2.0 介面設計與實作

On the Design and Implementation of an Efficient OSCI TLM-2.0 Interface for QEMU and SystemC Based Virtual Platform

於中華民國 100 年 7 月 19 日經本委員會審查並舉行口試，符合碩士學位論文標準。

學位考試委員簽章：

召集人 李宗南	李宗南	委　員 江明朝	[illegible]
委　員 鄺獻榮	鄺獻榮	委　員	
委　員		委　員	

指導教授(江明朝) [illegible] (簽名)

學年度：	99
學期：	2
校院：	國立中山大學
系所：	資訊工程學系
論文名稱(中)：	以QEMU與SystemC為基礎之虛擬平臺上高效率OSCI TLM2.0介面設計與實作
論文名稱(英)：	On the Design and Implementation of an Efficient OSCI TLM-2.0 Interface for QEMU and SystemC Based Virtual Platform
學位類別：	碩士
語文別：	英文
學號：	M963040055
提要開放使用：	是
頁數：	59
研究生(中) 姓：	陳
研究生(中) 名：	智笙
研究生(英) 姓：	Chen
研究生(英) 名：	Chi-Sheng
指導教授(中) 姓名：	江明朝
指導教授(英) 姓名：	Ming-Chao Chiang
關鍵字(中)：	QEMU-SystemC, QEMU, SystemC, QSC, OSCI TLM-2.0
關鍵字(英)：	QEMU-SystemC, QEMU, SystemC, QSC, OSCI TLM-2.0

摘要

為了改善之前所提出稱為QSC，以QEMU與SystemC為基礎之虛擬平臺上的模擬效能及使用OSCI TLM-2.0標準的便利性，本篇論文提出一套新的方式整合OSCI TLM-2.0與QSC。透過將OSCI TLM-2.0的匯流排移至匯流排功能模組(BFM)之外，所提出的方法不只加速模擬速度也使得新增OSCI TLM-2.0元件(IPs)至QSC上更為方便使用。實驗結果顯示所提出的方法與之前的作比較，其模擬速度的加速比率平均值為2.8到3.255倍。

關鍵詞： QEMU-SystemC, QEMU, SystemC, QSC, OSCI TLM-2.0

ABSTRACT

In order to improve the performance of simulation and the convenience of use with OSCI TLM-2.0 Standard on QEMU and SystemC based virtual platform we proposed previously called QSC, this thesis presents a novel approach for integrating OSCI TLM-2.0 with QSC. By moving the OSCI TLM-2.0 interconnect bus outside of the Bus Function Model (BFM), the proposed approach can not only accelerate the simulation speed but also make it easy to add OSCI TLM-2.0 based Intellectual Properties (IPs) to the QSC. Experimental results show that the proposed approach can speed up all the simulations by a factor from 2.8 up to 3.255 on average when compared with the previous approach.

Keyword: QEMU-SystemC, QEMU, SystemC, QSC, OSCI TLM-2.0

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

For a higher level and early stage design for system, Transaction Level Modeling [1] (TLM) is being used in the industry for a peroid of time. The TLM hides the lower level of the implementation details and allows the design system at the higher level of abstraction. The Open SystemC Initiative [2] (OSCI) publishes the OSCI TLM-2.0 Standard [3] in 2008. It provides a standardized approach for creating models and TLM simulations. This standard enables communicating between of models with same well-defined protocol interface.

QEMU-SystemC [4], the QEMU and SystemC-base framework, was proposed for software/hardware system co-design in 2007. It is a virtual platform which connects the QEMU and SystemC for System on Chip (SOC) development. After a series of patches by the QSC [5, 6, 7, 8, 9] for the enhancements of performance and usibilty, including collecting the system information (instructions, bus signals, cycle counts, interrupts), the QSC platform is still in the development for OSCI TLM-2.0 Standard. Although the QSC is able to provide the interface on the QSC platform by inserting OSCI TLM-2.0 objects directly into SystemC hardware modules, the performance and the convenience is not good enough yet. Most important, the performance of inserted OSCI TLM-2.0 objects strongly depends on SystemC modules and SystemC simulation kernel. By this approach, it loses the meaning of the efficency of OSCI TLM-2.0 and increase the difficulty of the hardware modeling for integration. In this thesis, by seperating the bus out of the SystemC-based models and isolating the abtiter model as an Intellectual Property (IP). Becuase the proposed bus is modeled with OSCI TLM-2.0 Standard, it helps to improve the performance in each arbitration (Fixed Priority or Round-robin), and to increase the convenience for any numbers of IPs to connect with the bus. Furthermore, the development programmers can very easily develop OSCI TLM-2.0 virtual hardware or extend system architecture on the bus to communicate with QEMU or other SystemC/OSCI TLM-2.0 virtual hardware models on the QSC virtual platform.

1.1 Motivation

The QSC virtual platform is implemented as two POSIX threads and communicating with each other by a system FIFO. One side is QEMU and the other side is SystemC-based models. The prototype of OSCI TLM-2.0 communicating modules is placed on the SystemC side and deeply hided in the Bus Funtional module (BFM). They run with SystemC models and process the internal signals very closely. Because the overall performance bottlenecks of the QSC platform is found on the SystemC side, SystemC-based models expand the instructions from QEMU and cannot process the queue by the same rate. To build the OSCI TLM-2.0 communicating modules inside SystemC-based modules will increase the load of SystemC side models and worsen the performance of the QSC.

1.2 Contributions of the Thesis

In this thesis, the goal is to improve the performance and enhance the convenience of use for OSCI TLM-2.0 on the QSC virtual platform, to keep the QSC architecture, and to implement the integration for OSCI TLM-2.0 virtual hardware IPs. By the efficency and convenience of OSCI TLM-2.0, the QSC will be more flexible for the simulations.

1.3 Organization of the Thesis

The organization of this thesis is described as following: The related works about the QSC framework, prototype of OSCI TLM-2.0 model on the QSC, and other related issue are discussed in Chapter 2. Chapter 3 describes the detail of modeling the transmision of TLM signals, and discuss the difference between the prototype and the proposed feature. In Chapter 4, the discussion and the experience results of proposed simulation group are present. Chapter 5 gives the conclusion and the future work, and also provides the summary of all the issues in this thesis.

Chapter 2

Related Work

In this chapter, the brief introduction to the background knowledge will be described. The previous works on several frameworks built with QEMU and SystemC-based virtual hardwares, and other hardware modeling with OSCI TLM-2.0 standards will be discussed.

2.1 Background

2.1.1 QEMU-SystemC

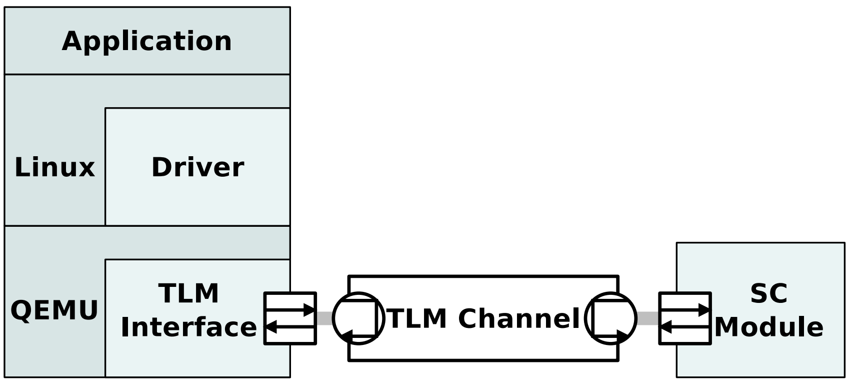


Figure 2.1: QEMU-SystemC [4]

QEMU-SystemC [4] is proposed in 2007. It is a framework for SOC development, the framework connects with QEMU [10] and virtual hardware model written in SystemC [2]. It allows designer to develop and test hardware modules for system level application on specific operating system in early stages of development. The virtual Advanced Microcontroller Bus

Architecture (AMBA) bus written in SystemC shown in Figure 2.1, connecting to the patched QEMU, is registered as a PCI device in QEMU, and the AMBA bus communicates with this PCI hardware in the QEMU through the TCP/IP protocol. Every instruction has sent from QEMU to PCI hardware which will be forwarded to virtual AMBA bus via the TCP/IP socket operations.

The achievement of this project is to build up a platform which allow virtual hardware linking to the QEMU emulator and directly running the real operating system on it. It builds an emulation framework allowing designers to connect to QEMU with the hardware model designed in SystemC. Through monitoring the virtual hardware, the engineers can do more appropriately test and debugging. With regard to this method, it is able to model the hardware to the system level. This framework can make it possible to get a short time-to-market and improve the engineer's productivity.

2.1.1.1 QSC

the QSC is proposed for adding several enhancements facilities for the QEMU-SystemC virtual platform, for gathering the information from target processor, counting numbers of different kinds of cycle counts or speedup the simulation performance. The proposed fast cycle-accurate instruction set simulator (CA-ISS) enables the SystemC hardware model to access the virtual hardware in QEMU through an interface at CA level, and make performance estimate possible for the target system. The interface between two parts of elements in this virtual platform shown as Figure 2.2, QEMU and BFM written in SystemC, is built of shared memory mechanism as a unidirectional FIFO instead of socket which is the original design of QEMU-SystemC. Each end of the interface, QEMU and BFM, is written in a thread and both in synchronization.

Much improvement has been made for the performance for the QEMU-SystemC co-designs in the QSC. For the purpose of acceleration for the simulation speed, cycle-count-accurate (CCA) has also been proposed for fast simulation. Besides at the cycle-accurate (CA), the correctness is also taken into consideration at CCA level simulation. The method of the QSC supporting the CCA level simulation is that the simulation at CCA level will only ensure the correctness

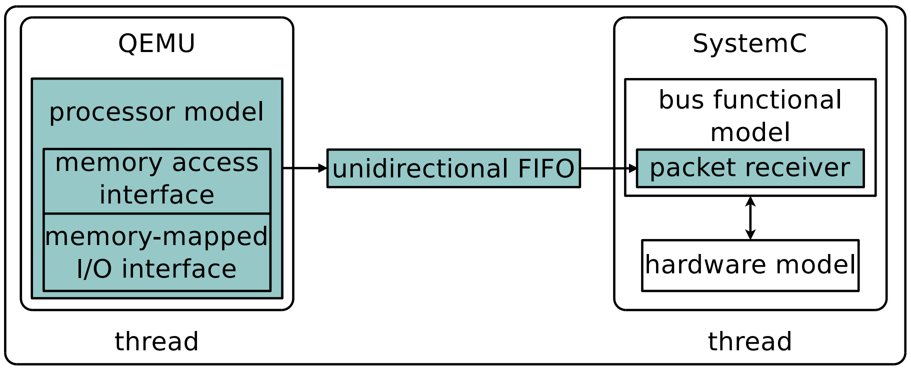


Figure 2.2: The inter-process communication (IPC) mechanism used by the ISS described herein [5]

of the total counts of the loading and storing instructions decoded from QEMU. Because the instruction fetch stage is ahead of the instruction decoding, the rest stags will be canceled after the summation for the cycle counts. Thus, sufficient accuracy remains and the time of simulation is reduced significantly.

With the modulization of the virtual hardware on QEMU, the development of arbitration has also been done with two simple arbitraion policies, Fixed Priority (FP) and Round-robin (RR). Recently the OSCI TLM-2.0 feature is added into the QSC virtual platform, but it is still a prototype architecture with a simple OSCI TLM-2.0 channel in the SystemC model. But there is a problem with the QSC, the time of the simulation varies from machine to machine. It depends on unknown factors not discovered yet, and not including the setting of machine equipments, Linux kernel configs or Linux distributions, but its simulation time will rise at a rate proportionate to the complexity of the source codes of the virtual hardwares.

2.1.1.2 OSCI SystemC

SystemC [11] is a Hardware Description Language (HDL), and it is a set of library routines and macros implemented in C++. SystemC is able to simulate from system level down to gate level, It can be easily that effectively create a cycle-accurate model of software algorithms, hardware architectures, and interfaces of the SOC or system level designs. Based on C++, SystemC contains the class library for describing the hardwares and system functions. It can make the communication between the software and hardware possible; moreover, SystemC can describe different level design. When the hardware specification is changed, the time of modeling, simulation and verification will be shorten, furthermore, it helps to analysis performance for choose the better hardware architecture.

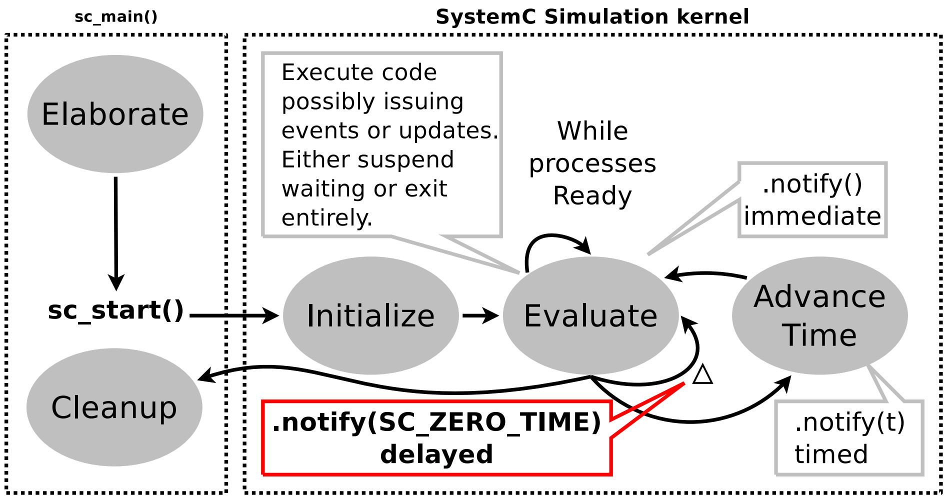
The source files of SystemC will also compiled by C++ compiler, linker, and loader. The external SystemC library will support the compilation process in compile time and link time, finally generates executable files. While the executable file running, it can input some files that s designed to do in advance or using *sc_trace* function in the source file to generate the wave form outputs.

SystemC defines many objects to support modeling the hardware of system. The module is the block diagram; typically, it can contain the channels, ports and processes used to implement their required behavior. The module inherits from the base class, *sc_module*, defined by SystemC library.

Ports and interfaces defined in SystemC is used to communicate between modules. Modules can use ports to input or output the data from with other modules. The interface on the port can determinate the behavior of the port. In the view of C++, the interface is the pure virtual function template, it can be formed as many features of datatype by assigning the type specifier with datatype for polymorphism that can decided which datatype of function return type for the interface. As well as port, interface is the type-specifier to indicate what kinds of datatype operations on the port is allowed. To simplify the design, SystemC provides numerous predefined ports, which are macros of some well defined templates with assigned type specifies already.

Processes are the part of SystemC that describe system operations. They contain sequential statements and work with other processes concurrently. The way to synchronize with one another by sending the *sc_event* objects. SystemC design the *sc_event* to handle the concurrency issue, a process can wait an event arriving or not going anywhere but waiting. Also, process can use the sensitivity list to wait the event sequentially.

Except the method of using event to handle the concurrency issue, SystemC provides an exhaustive range of predefined channels for generic uses, including the semaphore channel, it can be used to control the race condition very easily. The channels are SystemC's communication medium, it contains the value and be accessed by the interface method on the port.



sc_main()
SystemC Simulation kernel
Elaborate
sc_start()
Cleanup
Execute code possibly issuing events or updates. Either suspend waiting or exit entirely.
While processes Ready
.notify() immediate
Initialize
Evaluate
Advance Time
△
.notify(SC_ZERO_TIME) delayed
.notify(t) timed

Figure 2.3: SystemC Simulation Kernel [12]

The simulation kernel is the most important part of SystemC. It has two major phases shown as Figure 2.3, the elaboration and the evaluation. All variables and data structures will be initialized in the elaboration phase. The simulation behavior will be shown in the evaluation phase. The ready process will be sent to the ready queue waiting for the event notification, and the process will stay in waiting queue whenever it encounters a *wait()* delay. When the specific event arrived or the delay time is reached, processes will be moved to the ready queue again. This mechanism can be used to control the emulation of the time delay situation or control the concurrency by the event notification between SystemC modules. Every evaluate-update runs will synchronize the value speratedly on each component. The QSC uses un-timed concurrency policy and synchronizes the values in the delta cycle. Every signal value on the QSC virtual platform will follow the numbers of calls to *wait(SC_ZERO_TIME)* to manage the synchronization.

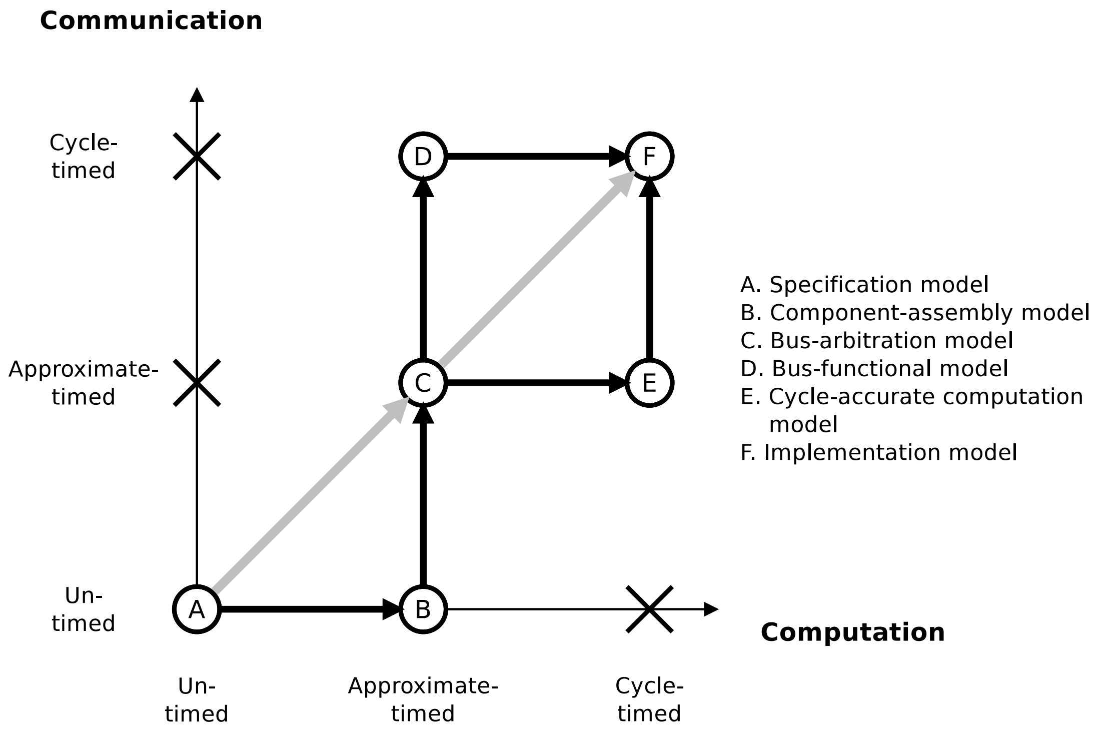


Figure 2.4: TLM modeling graph [1]

2.1.2 Transaction Level-modeling and OSCI TLM-2.0

2.1.2.1 Transation Level-modeling

TLMs have been widely used in system-level design. In [1], Cai *et al.* describe how the TLMs are used in the general design domains. The TLMs can be used in top-down approaches, starting design by high-level abstract models at begining, then adding the communication detail and completing the computational functionality shown as Figure 2.4, and the further implementation is decided on its need.

2.1.2.2 OSCI TLM-2.0

It is well-known that the TLM is a very important technique in System Level Design. The latest version of OSCI TLM-2.0 was released in 2009 as version 2.0.1. For the reason of different use of model in different transaction-level modeling, the timing accuracy and the contentions details for the SystemC system modeling need to be standardized. OSCI defines a set of interfaces in TLM-1 for transporting data accroding to its transport type (blocking/non-blocking), direction (unidirectional/bidirectional) as bellow:

1. Core TLM Interface classes

   - *tlm_transport_<REQ, RSP>*
   - *tlm_block_put_if*,
     tlm_nonblocking_put_if, *tlm_put_if*
   - *tlm_block_get_if*,
     tlm_nonblocking_get_if, *tlm_get_if*
   - *tlm_blocking_peek_if*,
     tlm_nonblocking_peek, *tlm_peek_if*

2. Extended TLM Interfaces classes

   - *tlm_blocking_get_peek_if*,
     tlm_nonblocking_get_peek_if, *tlm_get_peek_if*

3. Master/Slave Interfaces

   - *tlm_master_if<REQ, RSP>*,
     tlm_slave_if<REQ, RSP>
   - FIFO Specific Interfaces classes

4. *tlm_fifo_debug_if*,
   tlm_fifo_config_size_if

5. *tlm_fifo_put_if*

But there are 3 shortcomings of TLM-1 for modeling:

1. There is no standard transaction class for modeling.

2. There is no timing annotation.

3. The data transporting is required by value or const reference.

OSCI publishes the OSCI TLM-2.0 standard in 2008. Because TLM-1.0 lacks of model interoperability, OSCI TLM-2.0 defines a solid API and suggested data structures that, when used as proposed, enable model interoperability. It provides a standardized approach for creating models and TLM simulations. The standard enables the exchange of models and a common ground for interfacing. The overview of the OSCI TLM-2.0 classes are shown as Figure 2.5.

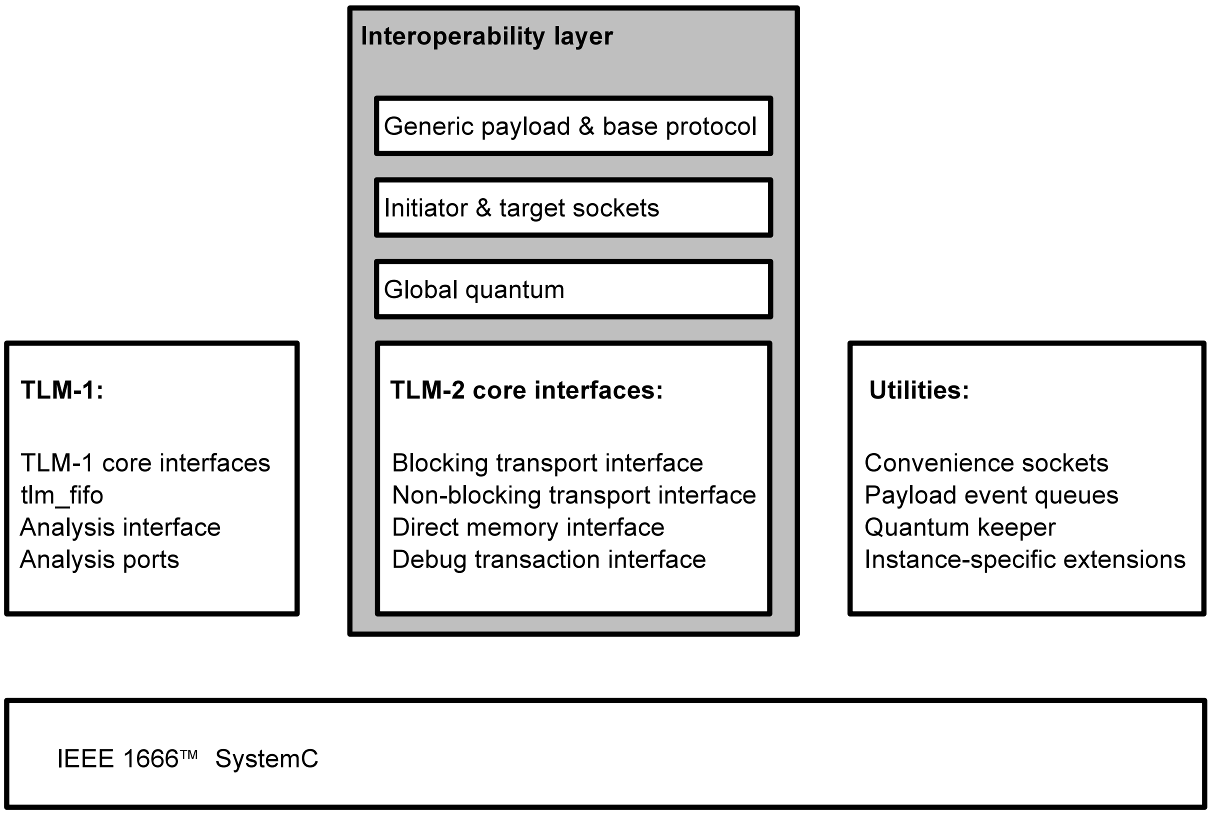


Figure 2.5: OSCI TLM-2.0 Classes [2]

2.1.3 AMBA 2.0

With the growing complexity of SOC modeling, the IP reusing has become very important and the design of the bus is the key factor. Base on these points, several companies publish their own bus specification. Among these specifications, AMBA is wildly used by many IP design companies and SOC system integrators, even becoming a popular structure.

AMBA specification includes three parts: Advanced High Performance bus (AHB), Advanced System Bus (ASB) and Advanced Peripheral Bus (APB), they use the traditional design of

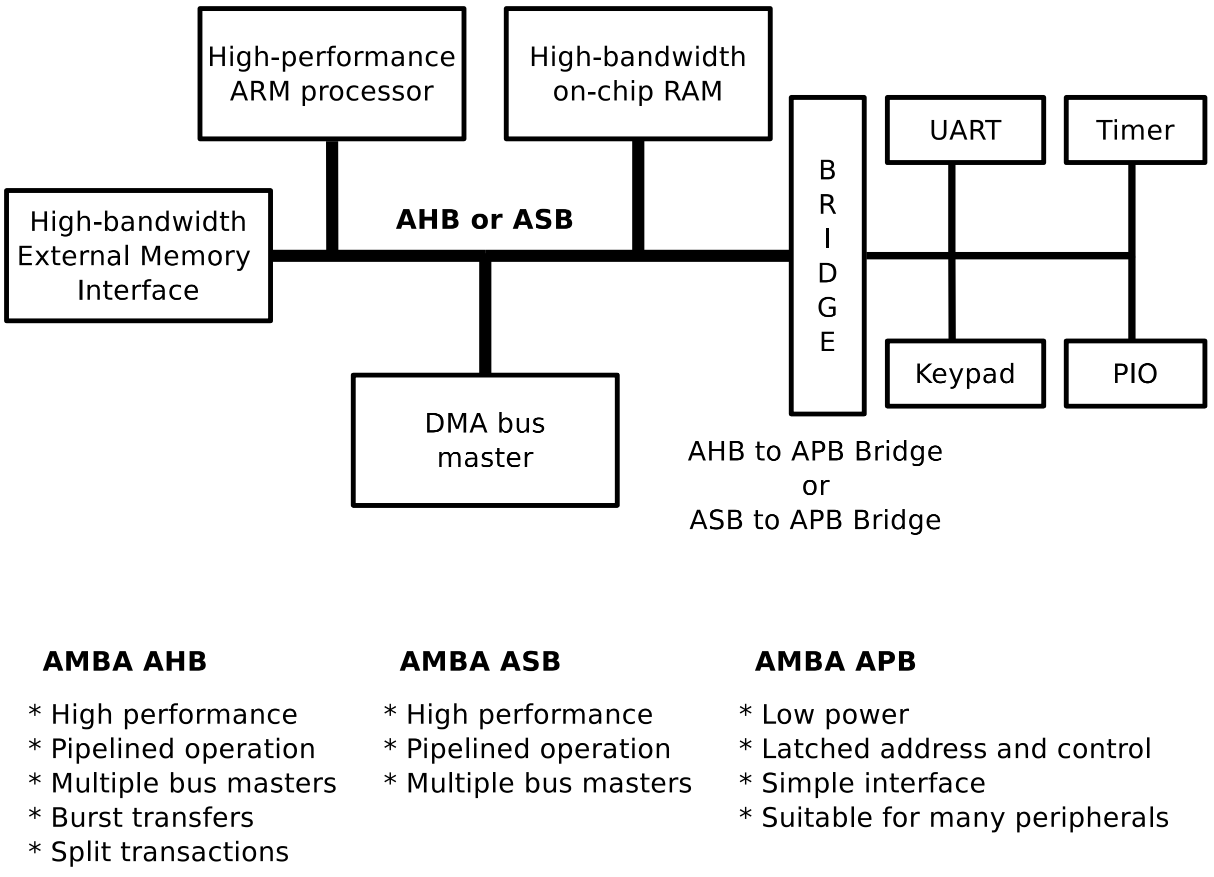


Figure 2.6: A typical AMBA system [13].

share bus, containing the master and slave structure, and providing the arbiter and decoder to do arbitration and decoding the address. Due to the complexity of the QEMU internal instruction and for the reason of simplying implementation detail, such as instruction pipeline, the emulation of the bus behavior for the BFM on the QSC is only made of AHB.

2.2 The Other Simulation Platforms

The hardware simulation with SystemC is quite common in this domain. Boukhechem *et al.* [14] bring up the STARSoC co-simulation environment. The platform aim to explore at higher levels of abstractions on a multiprocessor system on chip architectures shown as Figure 2.7. It uses the OpenRISC CPU architecture to model the environment, and instruction set simulators (ISSs) wrapped under SystemC. Also, the basic peripherals is within the SystemC simulation framework. Its purpose is to develop a complete design space exploration tool. In detail, it uses the OpenRISC CPU to process the instructions. The model of its environment is that OpenRISC CPUs is outside of SystemC scheduler environment, and communicate with SystemC by Inter-process Communication (IPC). It could not completely simulate the system behavior as it running a real operating system instead of just synthesis of hardware model and verification.

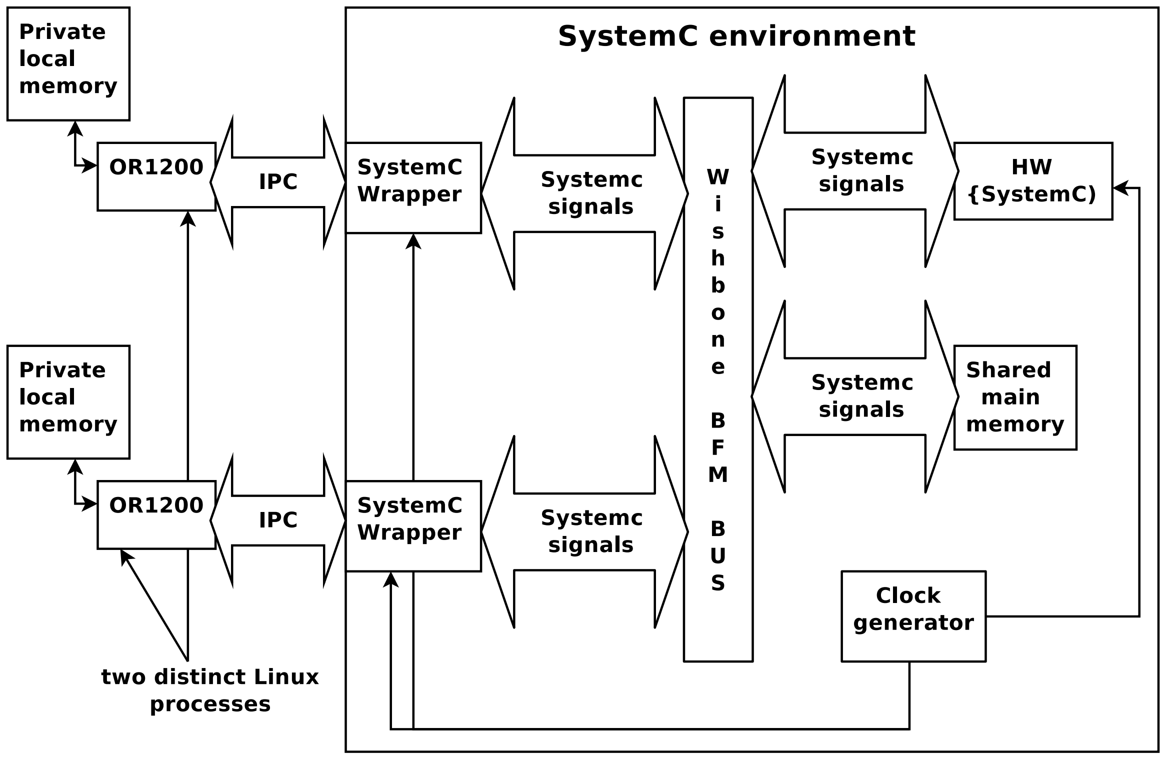


Figure 2.7: STARSoC Platform [14]

The main profit of the simulation running on QEMU with virtual hardwares is that QEMU is a free and open-source software, it is able to run a real operating system (OS), performs the instructions and hardware interacetion as if it runs on a real hardware machine. Most important, QEMU only contains the necessary abstract hardware behaviors so that the simulations speed [9] on QSC are fast as well. Most researches run the simulation on commercial softwares (such as CoWare) are also able to collect the information of hardware system generated by them. In [15, 16], they use QEMU and CoWare connecting together by a socket interface. The server-client architecture will slow down the simulation and it had been replaced by a unidirectional FIFO on QSC. Monton [17] *et al.* also use QEMU-SystemC virtual framwork with a slow socket interface. By the refinement for OSCI TLM-2.0-based virtual hardware [18], Becker *et* *al.* proposed a methodology, starting from an un-timed TLM model to accurate model of target platform. But it is use a simple RTOS which is not realistic enough for a full system simulation.

While designing the bus, choosing what the level to be implemented is very important. The SystemC language can do the modeling from Register Transfer Level (RTL) to system level. It is known to all that the lower level modeling cause the worse performance. There is an interest issue of how its performance will be while concatenating different level of IPs together.

The TLM [1] is a way to decrease the presure of time to market in design process. It becomes a usual practices for simplifying the system-level design and earlier building up the architecture. Before inserting the RTL IPs into the TLM design environments, it must ensure its effectiveness by using transactor to the platform. About the effectiveness of the Transactor-Based Verification (TBV) compared to a more traditional RTL verification strategy [19], Bombieri et al. rewrite the assertions and test-benches at RTL to test the quality of each TBV. This paper use the theoretically-based methodology to evaluate the quality. In that theoretically-base result, assertion via TBV is been proved the effectiveness, and high level fault is not covered by the assertions/properties.

Bombieri *et al.* [20] integrate the RTL IPs into TLM design using the automatic transactor generation. They use the transactor to concatenate the different level model to the same design environment. It can be used to communicate between TLM and RTL with their transactor generated automatically. The main idea is that it has defined several corresponding function library, with the input of arguments, it can generate the correct transactor to match the couple of RTL and TLM IPs.

With the growing complexity of the SOC chip design, the OSCI publish the OSCI TLM-2.0 specification trying to build up the standard for TLM model design principle. Chen *et al.* [21] use a top-down methodology for generation of RTL tests from the SystemC TLM specifications. It is a trend that, while using the OSCI TLM specification to model TLM IPs with SystemC, it publishes the method of TLM test into RTL tests using a set of transformation rules. The validation result of the SystemC TLM design can be reused at the RTL validation, in the same time, it reduces the time for RTL validation effort. In some researches, the implementation for the TLM designs into RTL is achieved by an automatic synthesis techique. By breaking down the TLM into RTL with the extended finite state machine, the OSCI TLM-2.0-based hardware model can show the AMBA emulation singals. In [22], Bombieri *et al.* also proposed mutation model to analysis the possible situation of OSCI TLM-2.0-based virtual harewares. Some researches [23, 24] Moll *et al.*, use CoWare with OSCI TLM-2.0-based hardwares run the simulation with the OSCI TLM-2.0 packets which wrap the RTL signals generated by CoWare.

It is difficult to debug the hardware model written in SystemC programming language because SystemC is built of templates and macros in the C++ programming language. The source codes of the hardware model can be treated as a normal C++ programs, but the compiling messages doesn't match the sense of hardware models, that it will not present the corresponding messages of hardware feature. Debug with GUI front end [25], ViSyC and RTLVision can be the choice. ViSyC is a statical analysis tool for SystemC, it can compute the intermediate representation of the behavior of SystemC, then send it to RTLVision as an input for generating the graphical view of the hardware models. These tools can help to figure out the source code of a SystemC model.

Chapter 3

OSCI TLM-2.0 Interconnect Interface

3.1 Design of the OSCI TLM-2.0 Interconnect

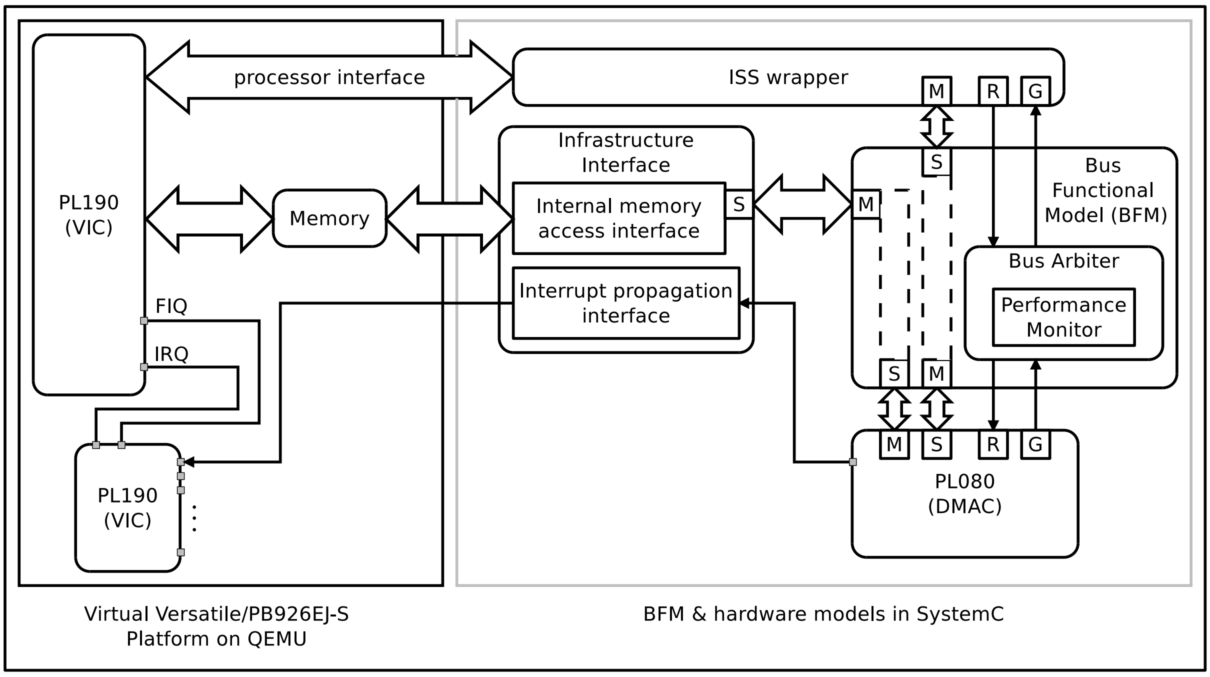


Figure 3.1: QSC-CCA overview

Due to using the QSC-CCA virtual platform, there are some pin-level AMBA signals remaining in the BFM (shown as Figure 3.1). Consequently, when the connection is made between the OSCI TLM-2.0 IPs on the QSC platform, the signal adapter has to provide the signal transferring between different level of models. In addition to the rearrangement of the integration, the arbiter will be moved out of BFM, and isolated as an OSCI TLM-2.0 IPs on the OSCI TLM-2.0

interconnect. As a result of the rearrangement, the virtual hardwares written by OSCI TLM-2.0 Standard are able to communicate with the QSC by this OSCI TLM-2.0 interface.

3.1.1 Functional Module Design

For the purpose of transferring the necessary information from the QSC-CCA to the OSCI TLM-2.0 interconnect feature, the adapter can be used for gathering the QSC-CCA information for each IP attempt to connect. The OSCI TLM-2.0 initiator will be used as the transferring adapter. There will be *sc_port*s on one side of initiator and on the other side is a TLM-2.0 initiator socket. By a *sc_thread* process, the AMBA bus signals will be stored in a unified data structure, and transfered by the transport function on the initiator socket to the interconnect feature of bus. The designed adapter is shown as Figure 3.2.

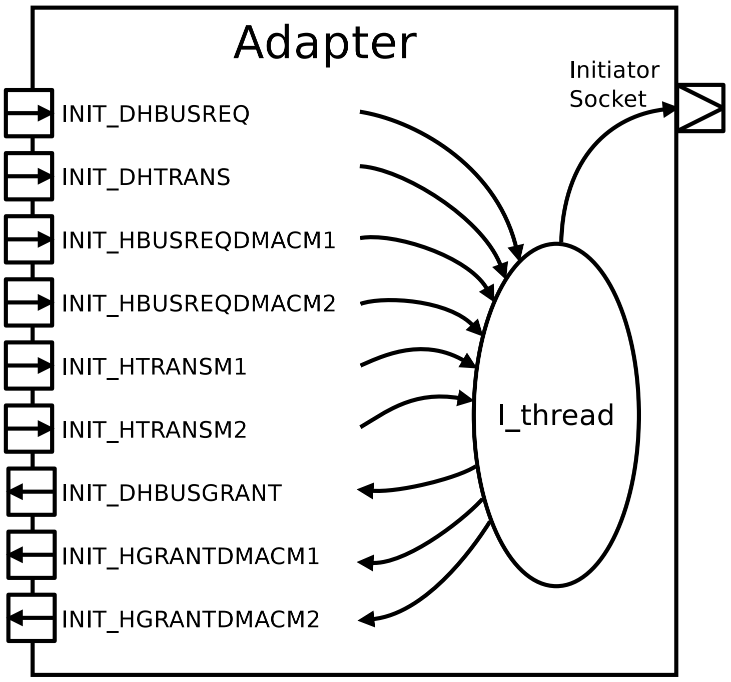


Figure 3.2: Adapter

The interconnect is the concreate feature of the bus. Each IP attempt to connect the QSC platform will use the TLM-2.0 sockets to communicate with this interconnect. For the reason of simplicity, the *multi_passthrough* sockets are used by the interconnect for outgoing and incoming sockets connected by the multiple external IPs. It allows multiple hardware modules to connect directly with this single socket, and identified by an arugment in the transport function. The *sc_thread* process in the interconnect is able to take this argument to activate the corresponding behaviors. The interconnect is shown as Figure 3.3.

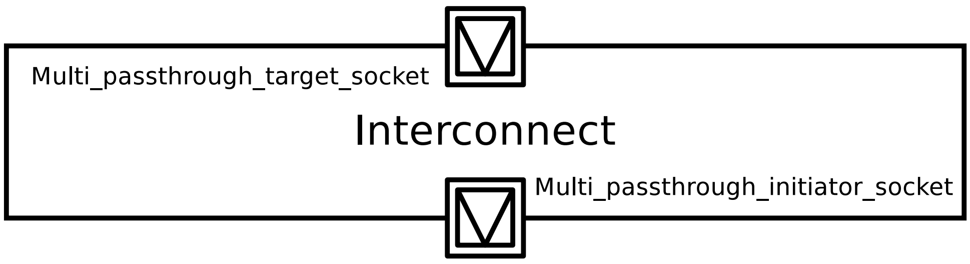

Multi_passthrough_target_socket
Interconnect
Multi_passthrough_initiator_socket

Figure 3.3: Interconnect with *multi_passthrough* sockets

The arbitration plays an important role in CCA level of the QSC. The arbiter in the previous design is deep hidden in the BFM, the bus communication will pass through many *sc_port*s and *sc_module*s. In order to increase the performance and reduce the complexity of the design, the arbiter is seperated out of the original BFM which is made of SystemC, and built as a slave IP. As a result, the modulization of the arbiter IP makes the external IPs can observe the arbitration conveniently on the OSCI TLM-2.0 interconnect feature without another adapter to do the signals transferring in SystemC. The bundled arbiter IP is shown as Figure 3.4.

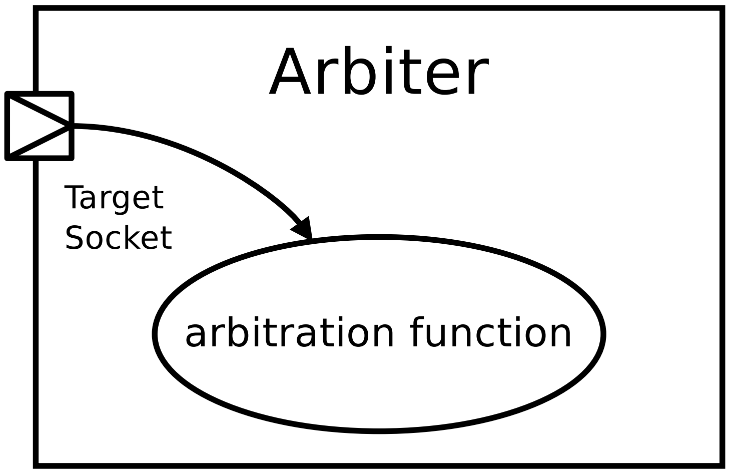


Figure 3.4: Arbiter

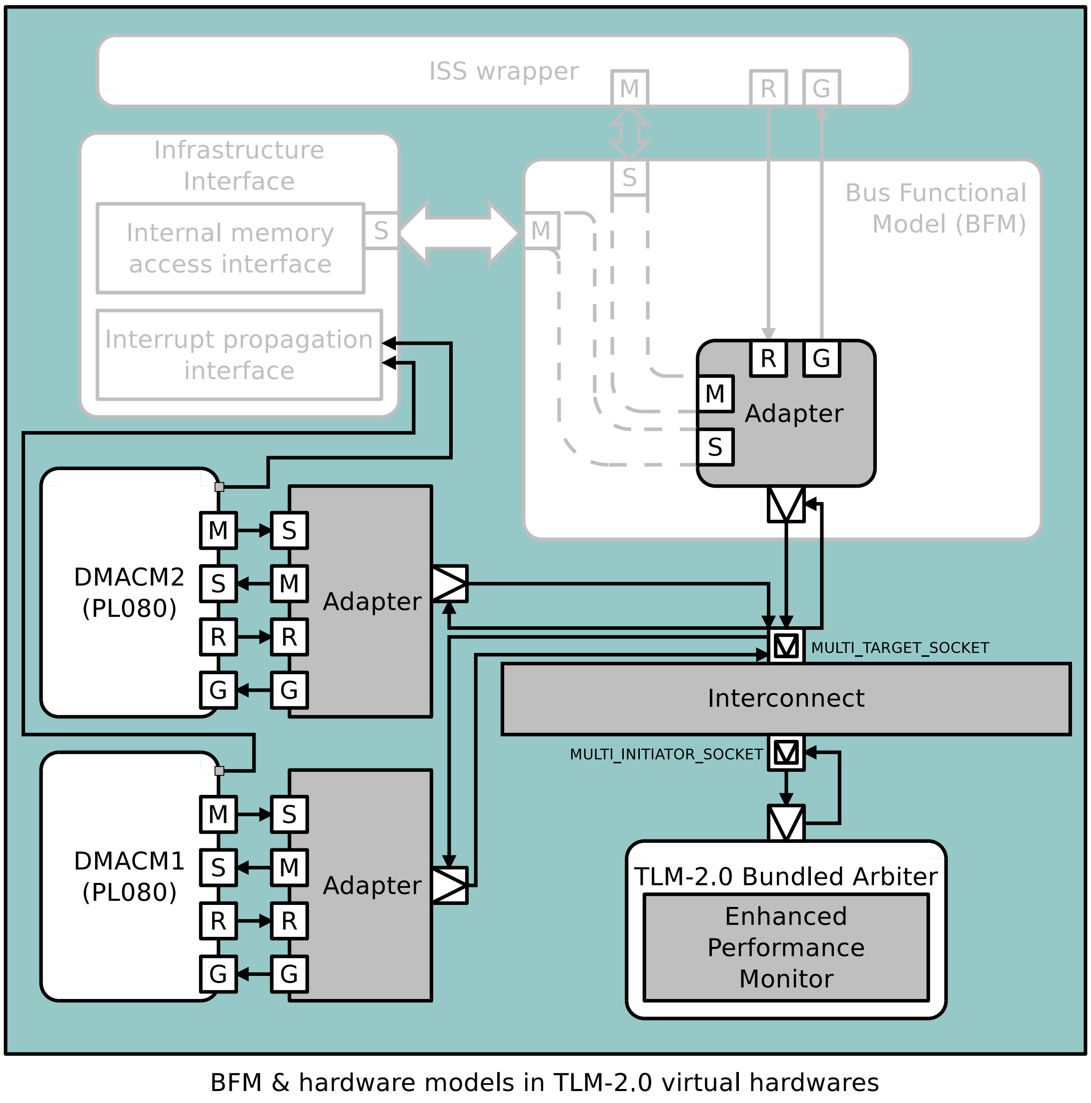


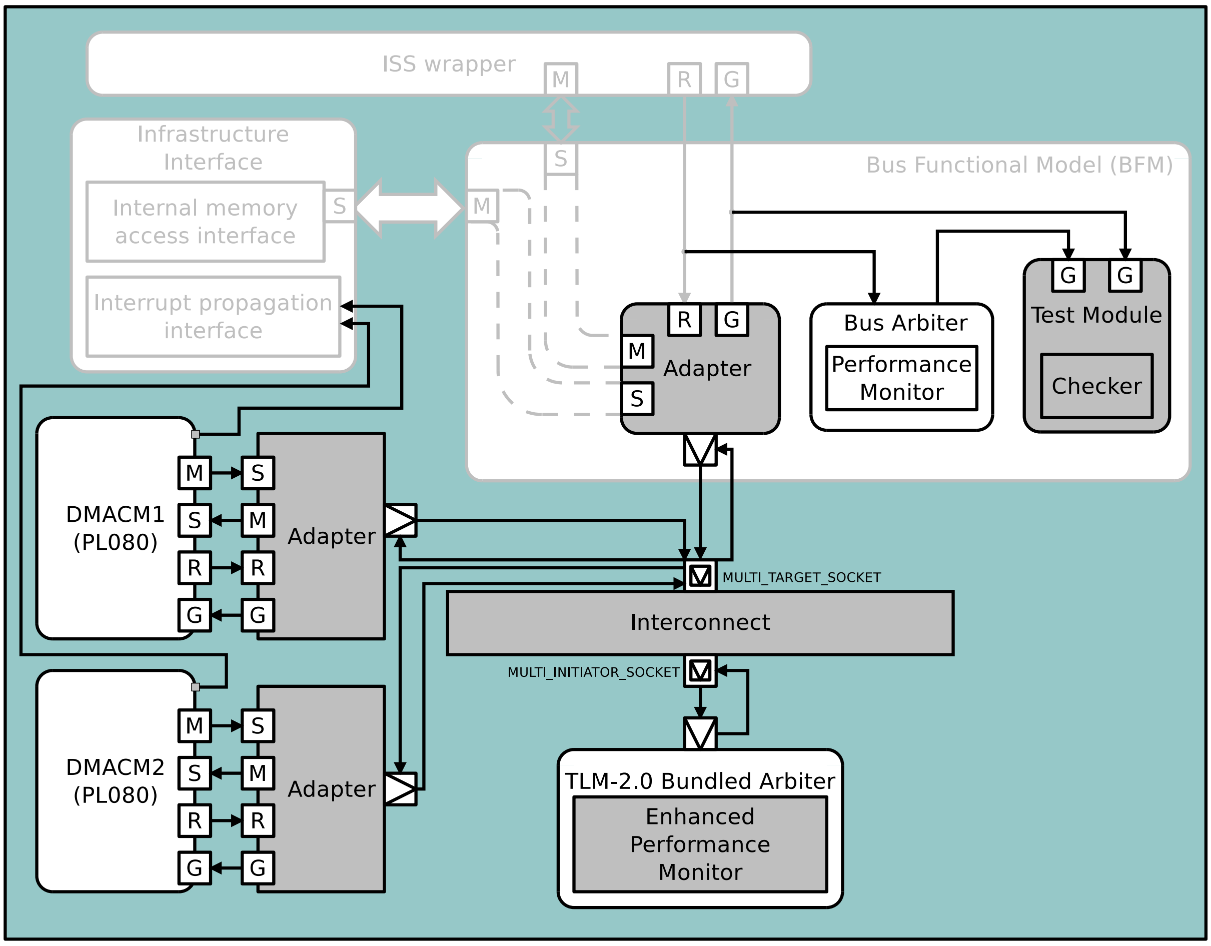
Figure 3.5: Overview for the OSCI TLM-2.0 interconnect environment

3.1.2 Communacation Deisgn

Because the QSC-CCA uses un-timed mechenism, and synchornizes the SystemC modules with same timing point of *wait(SC_ZERO_TIME)*, there is no time value to evaluate time phase. OSCI TLM-2.0 supports two specific named coding styles; *loosely-timed* (LT) and *approximately-timed* (AT). Due to the sequential bus arbitration of the QSC-CCA, the coding style of temporal decoupling is not suitable for the QSC-CCA virtual platform.

The transport interfaces, blocking transport and non-blocking transport for the coding style corresponding to *loosely-timed* and *approximately-timed*, will be implemented. The LT modeled by blocking transport interface, there will be only two timing points for the start and the end of transaction without *TLM_PHASE*. And the AT performs the early completion by returning the variable of TLM-2.0 phase, *TLM_COMPLETED*, to skip response phase of non-blocking transport interface on the interconnect feature.

OSCI TLM-2.0 provides a series of convenience sockets, they can be found by using namespace *tlm_utils*. In order to design communication of the adapter and arbiter, the *simple_initiator_socket* is chosen to be set on the adapter and *simple_target_socket* on the bundled arbiter. In the interconnect, the *multi_passthrough_initiator_socket* and *multi_passthrough_target_socket* are added for the multiple IPs to connect with same socket. The overview of the system architecture is shown as Figure 3.5.



BFM & hardware models in TLM-2.0 virtual hardwares and SystemC test Models

Figure 3.6: Parallel Processing Environment

3.1.3 Parallel Processing Model

Because the level of the QSC-CCA is built at CCA level of hardware model, intuitively it is able to connect with the IPs written with SystemC in the level of CCA. With the OSCI TLM-2.0 interconnect interface, it is able to connect the OSCI TLM-2.0 IPs, therefore, the combination of the different level of simulation is possible. The IPs described at CCA level and the IPs written in OSCI TLM-2.0 at TLM level are able to connect, respectively, original BFM and the OSCI TLM-2.0 interconnect interface. The parallel simulation model could be made and the system architecture is shown as Figure 3.6. To ensure the accuracy of the values generated and passed under the parallel simulation, the singal correctness have to be guaranteed carefully. The test module inserted in the BFM takes this job to watch every single singals passed between these two different levels of simulation. The test module is shown as Figure 3.7.

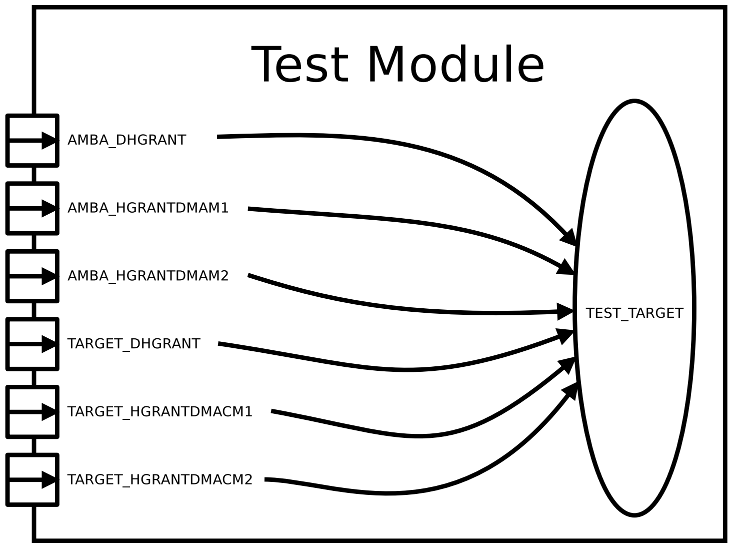


Figure 3.7: Test module

3.2 Proposed Implementation

3.2.1 Coding Style and Core Interface

Different use models of TLM level simulations require different levels of coding styles in OSCI TLM-2.0. The coding sytles the OSCI TLM-2.0 Standard provides are LT and AT. In general,

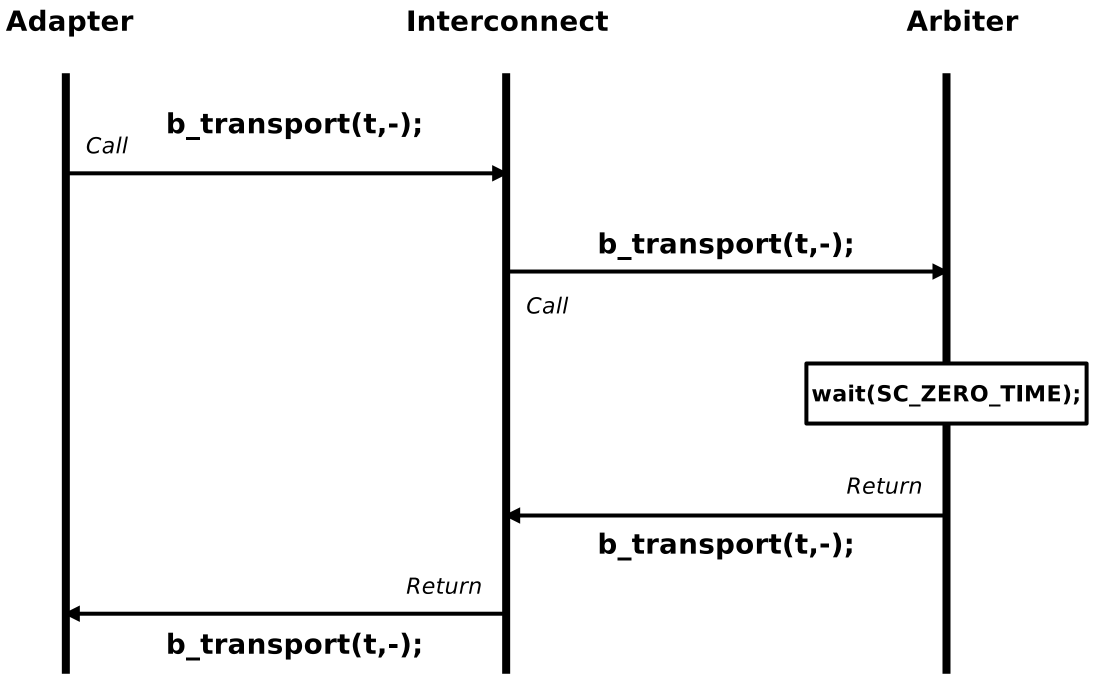


Figure 3.8: The message sequence chart of LT with blocking transport interface

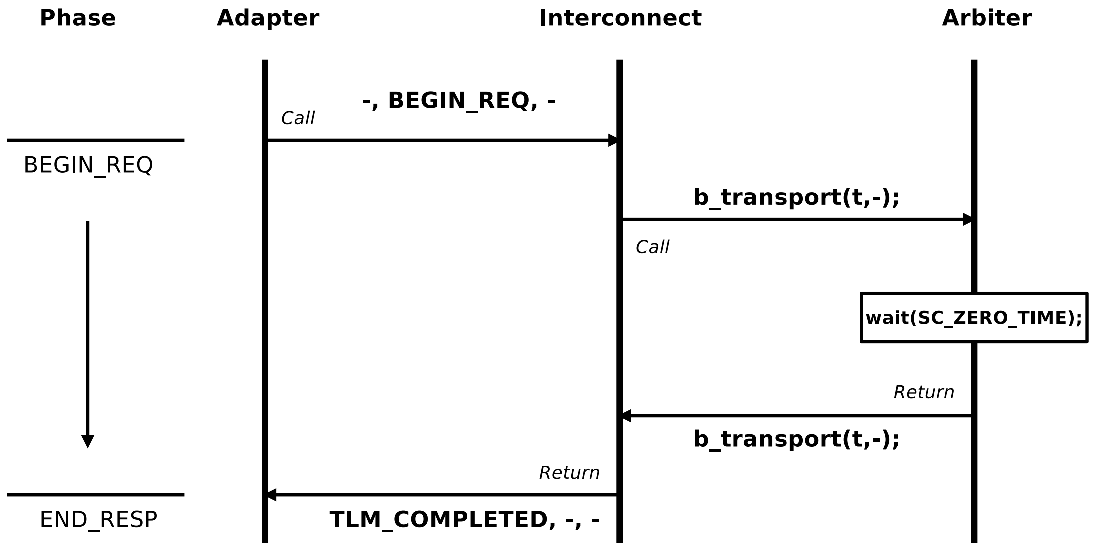


Figure 3.9: The message sequence chart of AT with non-blocking transport interface

LT is more suitable for higher level design and AT is on the other hand. Considering the architecture of the QSC-CCA and behavior of QEMU internal instructions present on the QSC, the LT and blocking transport *b_transport()* is choosen to implement the signal transfer between the QSC and OSCI TLM-2.0 IPs; and non-blocking transport *nb_transport()* for AT will be implemented by calling LT transport function *b_transport()* and returning *TLM_COMPLETED* value. The message sequence charts of LT and AT are shown as Figures 3.8 and 3.9.

Because the QSC is built of SystemC communicating with QEMU, the QSC-CCA information which come from the QSC-CCA is able to provide cycle counts and abitration information by SystemC communication mechenism. In order to integrate the OSCI TLM-2.0 system and the QSC, the necessary AMBA signals and arbitration information on the QSC is required and transformed into the form of a data type suited for OSCI TLM-2.0.

3.2.2 Sockets and Payloads

Listing 3.1: pldata(mypldata.h)

```
typedef class mypldata {
    public:
        bool DHBUSREQ;
        sc_uint<2> DHTRANS;
        bool HBUSREQDMACM1;
        sc_uint<2> HTRANSM1;
        bool HBUSREQDMACM2;
        sc_uint<2> HTRANSM2;
        bool sim_term;
        bool DHGRANT;
        bool HGRANTDMACM1;
        bool HGRANTDMACM2;

        // constructor, initial data valuse to 'boolvar'
        mypldata();
        mypldata(
                bool ildhbusreq ,
                sc_uint<2> ildhtrans ,
                bool ilhbusreqdmac1 ,
                sc_uint<2> ilhtransm1 ,
                bool ilhbusreqdmacm2 ,
                sc_uint<2> ilhtransm2 ,
                bool ilsim_term ,
                bool ildhgrant ,
                bool ilhgrantdmacm1 ,
                bool ilhgrantdmacm2
```

```
                    );
            ~mypldata();
            unsigned int arm_ctcnt, dmam1_ctcnt, dmam2_ctcnt;
            unsigned int arm_wncnt, arm_wscnt, dmam1_wncnt, dmam1_wscnt,
            dmam2_wncnt, dmam2_wscnt;
            time_t starttime;
            time_t endtime;
    } mypldata;
```

The socket mechenism is an important part of OSCI TLM-2.0, it decides which type of core interfaces to use and how payload to pass through initiators, interconnects and targets. The reason of using *multi_passthrough* is that it allows multiple bindings on a single socket without creating multiple sockets with each connection handling only one channel. Also, for the purpose of simple design, the *tlm_generic_payload* is used for the communicating with the proposed data type. Because of the QSC-CCA, the default attributes of *tlm_generic_payload* does not suit very well. As the result, a data type is created, *mypldata*, which carrys the full required information on the bus, such as arbitration information, cycle counts, AMBA request and grant signals. The instance of *mypldata* will be initialized and assigned as an attribute for *tlm_generic_payload*, called by the name of *set_data_ptr*. The pointer of *mypldata* is used in the socket transport function as an argument, passing through initiators, interconnect and targets. So that the value can be got whenever the function call arrives by dereferencing this pointer.

3.2.3 Adapter

The Arbiter IPs (shown as Listings 3.2, 3.3, and 3.2) is built of OSCI TLM-2.0 initiator. In order to make the QSC-CCA information useful on OSCI TLM-2.0 interconnect, the *i_thread* will collect all information on the QSC-CCA into the *mypldata* data structure. Next, the address of mypldata instance is assigned into a pointer which is used by *b_transport* on *multi_passthrough* socket opposite the adapter. Surely, the initiator socket on the adapter provides the *nb_transport_bw* for the backware path communicating. Finally, *i_thread* writes the value back through *sc_port*s to the QSC-CCA as the end of a transaction.

Most important, as a result of the socket on the interconnect connecting with the socket of adaper is a *multi_passthrough* socket, the socket id have to be indicated while the data transfer between adapters and interconnect, it is shown as Listing 3.3.

Listing 3.2: adapter(myinitiator.h)

```
struct myinitiator: sc_module
{
    // socket
    tlm_utils::simple_initiator_socket<
        myinitiator,
        32,
        tlm::tlm_base_protocol_types
        > socket;
    // payload
    tlm::tlm_generic_payload* impl;
    // phase and delay time
    tlm::tlm_phase* ttt;
    sc_time delay;


    sc_in<bool>         INIT_DHBUSREQ;
    sc_out<bool>        INIT_DHGRANT;
    sc_in<sc_uint<2> >  INIT_DHTRANS;
    sc_in<bool>         INIT_HBUSREQDMACM1;
    sc_in<sc_uint<2> >  INIT_HTRANSM1;
    sc_out<bool>        INIT_HGRANTDMACM1;
    sc_in<bool>         INIT_HBUSREQDMACM2;
    sc_in<sc_uint<2> >  INIT_HTRANSM2;
    sc_out<bool>        INIT_HGRANTDMACM2;
    sc_in<bool>         INIT_sim_term;

    mypldata carrydata;

    SC_HAS_PROCESS(myinitiator);
    myinitiator(sc_module_name nm);

    virtual tlm::tlm_sync_enum nb_transport_bw(
            tlm::tlm_generic_payload& trans,
            tlm::tlm_phase& phase,
                sc_time& delay);

    ~myinitiator();
    void i_thread();
    void suminfo();
};
```

Listing 3.3: adapter b_transport(myinitiator.cpp)

```
void myinitiator::i_thread()
{
    while (1){
        //setting the carrydata from signal input port
        carrydata.DHBUSREQ      = INIT_DHBUSREQ->read();
        carrydata.DHTRANS       = INIT_DHTRANS->read();
        carrydata.HBUSREQDMACM1 = INIT_HBUSREQDMACM1->read();
        carrydata.HTRANSM1      = INIT_HTRANSM1->read();
        carrydata.HBUSREQDMACM2 = INIT_HBUSREQDMACM2->read();
        carrydata.HTRANSM2      = INIT_HTRANSM2->read();
        carrydata.sim_term      = INIT_sim_term->read();

        if(carrydata.sim_term) {
                cout << carrydata.sim_term << endl;
            suminfo();
        }

        // goes with a wait(SC_ZERO_TIME)
        // indicate 'id = 0'
        socket[0]->b_transport(*impl,delay);

        // transfer done. write the port
        INIT_DHGRANT->write(carrydata.DHGRANT);
        INIT_HGRANTDMACM1->write(carrydata.HGRANTDMACM1);
        INIT_HGRANTDMACM2->write(carrydata.HGRANTDMACM2);
    }
}
```

3.2.4 Interconnect

The interconnect (shown as Listings 3.4, 3.5 and 3.3) is a concrete feature of a bus, the OSCI TLM-2.0 IPs is able to connect on it and communicates with the QSC-CCA virtual platform. There are 2 *multi_passthrough* sockets on it, the number of initiators or targets are parameterized by last field of template argument. Any number of OSCI TLM-2.0 IPs on this interconnect are able to connect with a singal socket name. Generally, the IPs are able to connect to *i_socket* as a master because they may change the values on the interconnect and ask for some information in return; relatively those connecting to *t_socket* as slaves, they receieve the changes and make relevent responses.

Listing 3.4: interconnect(myinterconnect.h)

```
struct myinterconnect
:sc_module
{
    tlm::tlm_generic_payload* hispl;

    // multi_passthrough,
    // initiator side socket: i_socket
    // target side socket: t_socket
    tlm_utils::multi_passthrough_target_socket<
        myinterconnect,32,
        tlm::tlm_base_protocol_types,
        1> i_socket;

    tlm_utils::multi_passthrough_initiator_socket<
        myinterconnect,
        32,
        tlm::tlm_base_protocol_types,
        1> t_socket;

    mypldata* inter_carrydata;
    sc_time delay;

    SC_HAS_PROCESS(myinterconnect);
    myinterconnect(sc_module_name nm);

    // i_socket
    virtual void b_transport(
            int id,
            tlm::tlm_generic_payload& trans,
            sc_time& delay);
    virtual tlm::tlm_sync_enum nb_transport_fw(
            int id,
            tlm::tlm_generic_payload& trans,
            tlm::tlm_phase& phase,
            sc_time& delay
            );

    // t_socket
    virtual tlm::tlm_sync_enum nb_transport_bw(
            int id,
            tlm::tlm_generic_payload& trans,
            tlm::tlm_phase& phase,
            sc_time& delay);

    void inter_thread();
    ~myinterconnect();
};
```

Listing 3.5: interconnect(myinterconnect.cpp)

```
void myinterconnect::b_transport(
        int id,
        tlm::tlm_generic_payload& trans,
        sc_time& delay)
{
    if (id < i_socket.size())
        switch (id) {
            case 0:  // from adaptor in BFM
                //1. got the payload from initiator, do something
                inter_carrydata = reinterpret_cast<mypldata *>(trans.
        get_data_ptr());
                //2. b_transport to target
                t_socket->b_transport(trans,delay);
                //3. return from target to interconnect, do something rest

                break;
            case 1:  // other TLM2.0 IPs
            case 2:
            case 3:
            default:
                //1. got the payload from initiator, do something
                inter_carrydata = reinterpret_cast<mypldata *>(trans.
        get_data_ptr());
                //2. b_transport to target
                t_socket->b_transport(trans,delay);
                //3. return from target to interconnect, do something rest
                break;
        }
}
```

3.2.5 Bundled Arbitration Model

The bundled arbitration model is a OSCI TLM-2.0 target module (shown as Listings 3.6, 3.4). The *t_socket* receieves the QSC-CCA information by dereference the attribute of data pointer in the generic payload. Then the function within the arbiter will decide which master will be granted to process the bus.

Listing 3.6: arbiter(mytarget.h)

```
struct mytarget:
    sc_module
{
    tlm_utils::simple_target_socket<mytarget,32,tlm::
        tlm_base_protocol_types> socket;

    mypldata*   target_carrydata;

    sc_time     delay;
    sc_event    ta;

    //SC_HAS_PROCESS(mytarget);
    mytarget(sc_module_name nm);

    virtual void b_transport(
            tlm::tlm_generic_payload& trans,
            sc_time& delay);
    virtual tlm::tlm_sync_enum nb_transport_fw(
            tlm::tlm_generic_payload& trans,
            tlm::tlm_phase& phase,
            sc_time::sc_time& delay);

    ~mytarget();
    int arbitraion_p;
};
```

3.2.6 Test Module

To ensure the accuracy for the result of the parallel simulation, the most important part of the bus feature is the arbitration model. Because the arbiter has been pulled out from QSC2-CCA BFM on the OSCI TLM-2.0 interconnect as a OSCI TLM-2.0 IP, it needs to connect the *sc_out* port on the arbiter of original QSC2-CCA to the *sc_channel* together with the output of bundled arbiter model as proposed. Next, the verification module, test module (shown as Listing 3.7), is built to ensure every signal from each modules will be the same.

Listing 3.7: test module(test_init.h)

```
#ifndef __TEST_INIT_H__
#define __TEST_INIT_H__

#include <iostream>
#include "systemc.h"

SC_MODULE(test_init)
{
    sc_in<bool> AMBA_DHGRANT;
    sc_in<bool> AMBA_HGRANTDMACM1;
    sc_in<bool> AMBA_HGRANTDMACM2;

    sc_in<bool> INIT_DHGRANT;
    sc_in<bool> INIT_HGRANTDMACM1;
    sc_in<bool> INIT_HGRANTDMACM2;

    //SC_HAS_PROCESS(test_in);
    SC_CTOR(test_init);
    //test_in(sc_module_name nm);

    void p1();
};
#endif
```

Chapter 4

Experimental Results

In this chapter, The experimental result of each simulation will be presented. With 2 policies for arbitration: Fixed Priority (FP) and Round-robin (RR), to show the platform is able to change the arbitration method. The simulation uses the proposed interconnect as the bus of virtual Versatile/PB926EJ-S platform and as the interconnect of ARM PrimeCell PL080 DMAC modeled in SystemC. For all the experiments given in this chapter, a 2.00GHz Intel® Core2 T7200 processor machine with 2GB of memory is used as the host, and the target OS is built by using the BuildRoot package. The Linux distribution is Gentoo, and the kernel is Linux version 2.6.37. QEMU version 0.11.0-c1, SystemC version 2.2.0 (includeing the reference simulator provided by OSCI), and OSCI TLM-2.0 (release on July 15, 2009) are all compiled by GGC version 4.4.4.

4.1 Simulation Models

In order to gather the statistics, the initial shell script is modified to enable the option of executing the DMAC test bench and rebooting the virtual machine automatically as soon as the booting sequence is completed. Furthermore, the predefined *no-reboot* option of QEMU will catch the reboot signal once the OS completed the DMAC test bench and started the reboot command to shut the QEMU down. Thus, the test bench can easily estimate the co-simulation time of QEMU and SystemC at the OS level.

Table 4.1: Full Simulation Groups

Simulation	Arbitraion Policy
The Original QSC-CCA	FP
without the OSCI TLM-2.0 interconnect Interface	RR
The OSCI TLM-2.0 Interconnect Interface	FP
(blocking-transport)	RR
The OSCI TLM-2.0 Interconnect Interface	FP
(non-blocking-transport)	RR
The OSCI TLM-2.0 Interconnect Interface	FP
with testing module(blocking-transport)	RR
The OSCI TLM-2.0 Interconnect Interface	FP
with testing module(non-blocking-transport)	RR

The five methods of the simulation will be run for each of group list below. The first simulation is the original QSC-CCA version without any OSCI TLM-2.0 feature, this simulation will be the control group and the following simulations are experimental groups. In the second and third groups, the OSCI TLM-2.0 interconnect interface will be used for the simulation, the difference between two groups is the transport interface of *blocking* and *non-blocking*. The forth and the fifth groups will simulate with the parallel model of TLM-2.0 and SystemC, the difference between is still the transport interface. The two policies of arbitration: FP and RR will be used for each simulation. The full simulation group is shown as Table 4.1.

4.2 Simulation Results

4.2.1 The Original QSC-CCA Model

The version of origianl QSC-CCA without any OSCI TLM-2.0 interconnect interface. Each simulation result of different arbitration policies, FP and RR, are shown as Tables 4.2 and 4.3.

Table 4.2: The Simulation Result of Original the QSC-CCA without any OSCI TLM-2.0 Interconnect Interface (using FP)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	978.579s	488263264	149433939	66133023
max	1042.436s	503274154	154867917	69698019
μ	1012.467s	495824852.450	152005933.817	68023396.817

Table 4.3: The Simulation Result of the Original QSC-CCA without any OSCI TLM-2.0 Interconnect Interface (using RR)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1114.304s	492797962	151168529	67123547
max	1176.606s	505570181	155772236	70411748
μ	1144.039s	498855842.517	153569906.517	68590861.317

4.2.2 The Proposed Model: The OSCI TLM-2.0 Interconnect Interface Model

The model of the proposed OSCI TLM-2.0 interconnect interface with the bundle arbiter as a OSCI TLM-2.0 IP. There are two simulation groups with different OSCI TLM-2.0 transport interface, blocking and non-blocking. Although the simulation time of non-blocking is still larger than the other, the difference of two groups is only the transport interface, and non-blocking transport interface is a warpping function of blocking transport, the simulation time is almost the same.

Surely, the QSC-CCA supports two different arbitration policies, the arbitration IP (arbiter) is still able to do these two kinds of policies, fixed priority and round-robin. Obviously, the simulation result shows that RR priority needs more time to accomplish the job. Each simulation result of different arbitration policies, FP and RR, and different OSCI TLM-2.0 transport interfaces, blocking and non-blocking, are shown as Tables 4.4, 4.5, 4.6 and 4.7.

Table 4.4: The Simulation Result of the OSCI TLM-2.0 Interconnect Interface with blocking transport interface (using FP)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1185.089s	493486088	151510266	67281481
max	1239.794s	504822255	155812203	70230706
μ	1215.310s	498892386.600	153670182.75	68824443.417

Table 4.5: The Simulation Result of the OSCI TLM-2.0 Interconnect Interface with non-blocking transport interface (using FP)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1176.763s	493382487	151652896	67144330
max	1261.099s	506990974	156563305	70725616
μ	1217.634s	500301180.650	153982523.517	68727210.733

4.2.3 The Parallel Processing Model: The OSCI TLM-2.0 Interconnect Interface with SystemC BFM model

The version of the proposed OSCI TLM-2.0 interconnect interface with the bundle arbiter as a OSCI TLM-2.0 IP. This model use two model to do the simulation, the OSCI TLM-2.0 interconnect interface and the QSC-CCA BFM. This combination allows the QSC-CCA run the parallel co-simulation with different level of hardware models at the same time. It makes it possible to connect the OSCI TLM-2.0 interconnect interface with original SystemC BFM to the QSC-CCA concurrently.

With respect to the single simulation group of the OSCI TLM-2.0 interconnect interface, there are additional SystemC virtual hardwares than the former group, the increment of simulation time is considerable. In this case, the accuracy checked by the test model in the BFM block, which is connecting with the same *sc_channel* to verify the signals correctness, also increase the overhead of the simulation time. Each simulation result of different arbitration policies, FP and RR, and different OSCI TLM-2.0 transport interfaces, blocking and non-blocking, are shown as Table 4.8, Table 4.9, Table 4.10 and Table 4.11.

Table 4.6: The Simulation Result of the OSCI TLM-2.0 Interconnect Interface with blocking transport interface (using RR)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1371.241s	496116276	152825684	67779702
max	1418.934s	509437319	157013754	70937901
μ	1396.338s	502856417.750	154800770.633	69476382.167

Table 4.7: The Simulation Result of the OSCI TLM-2.0 Interconnect Interface with non-blocking transport interface (using RR)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1370.391s	496787363	152838239	67887701
max	1450.806s	512705110	158750205	71957359
μ	1412.533s	505326610.433	155984246.150	69990719.0

4.3 Summary

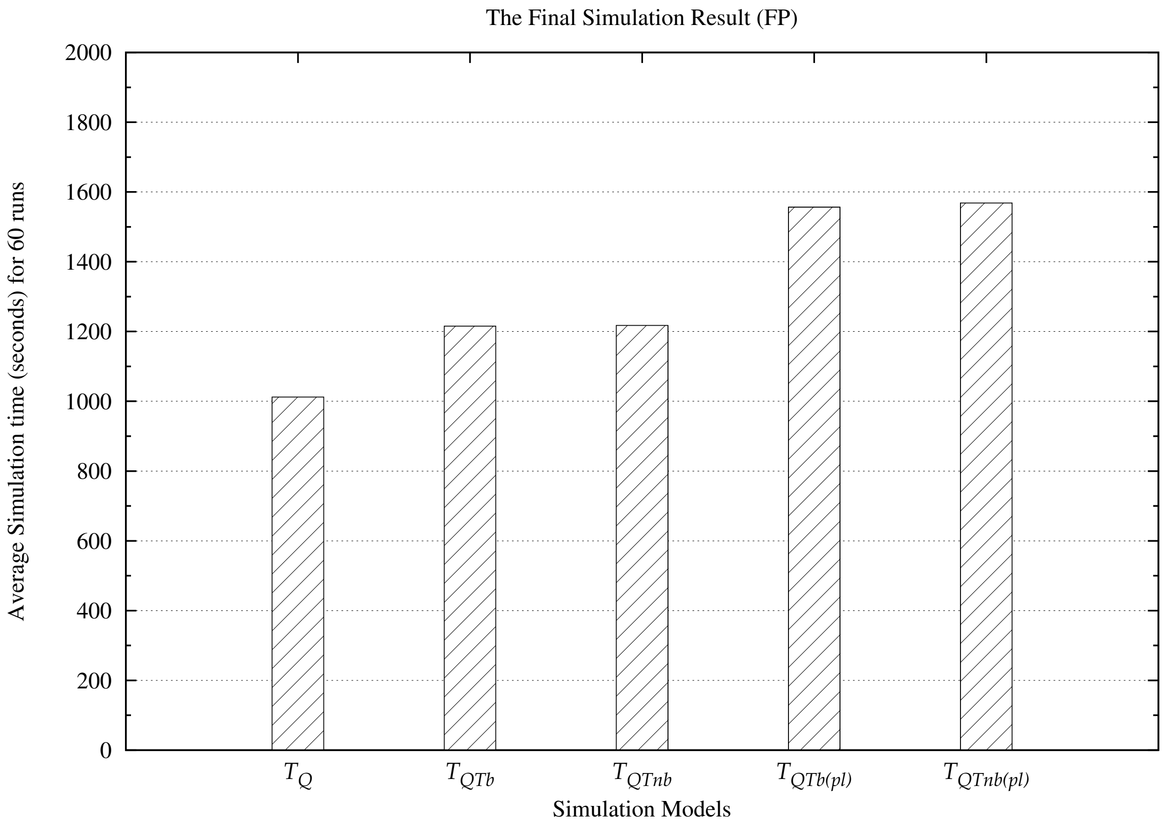


Figure 4.1: Full Simulation Result using FP policy

Table 4.8: The Simulation Result of Parallel model of the OSCI TLM-2.0 Interconnect Interface and SystemC BFM with blocking transport interface (using FP)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1515.008s	516248941	161086601	73226230
max	1595.631s	532524738	166881788	77088748
μ	1556.495s	524835662.933	163857543.167	75148258.817

Table 4.9: The Simulation Result of Parallel model of the OSCI TLM-2.0 Interconnect Interface and SystemC BFM with non-blocking transport interface (using FP)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1507.398s	516660676	161588729	73287838
max	1634.717s	538246515	169111528	78663447
μ	1568.431s	527888044.567	165286512.517	75723283.767

Obviously, adding the OSCI TLM-2.0 interconnect interface to the QSC-CCA increases the overhead of the simulation time, running parallel co-simulation group takes more time than any other groups. The simulation with proposed OSCI TLM-2.0 takes 20.326% overhead by FP and 38.250% by RR with the blocking trasnport interface, and almost the same simulation time interval by non-blocking transport interface. By the same situation, the parallel simulation group with blocking transport takes 54.106% by FP and 83.530% by RR. The results are shown as Figures 4.1 and 4.2.

Because the simulation time on the QSC-CCA virtual platform rises at a rate proportionate to the complexity of the source code, the result can be brought into comparison. The simulation time of the original QSC-CCA and the QSC-CCA with OSCI TLM-2.0 on the different machines are taken as the results, which are allowed in comparison in percentage with each overhead.

Alternatively, the simulation time of the co-simulation model of the QSC-CCA with the prototype of the OSCI TLM-2.0 [26] is 42m05.767s in average, but the simulation of its original base model is only takes 10m44.956s. The overhead of the co-simulation model of the QSC-CCA with its prototype of the OSCI TLM-2.0 is extreamly huge by the value of 291.618% in

Table 4.10: The Simulation Result of Parallel model of the OSCI TLM-2.0 Interconnect Interface and SystemC BFM with blocking transport interface (using RR)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1805.915s	521584226	162838807	74141489
max	1906.123s	539894432	169884653	78868330
μ	1853.673s	530278079.767	166658543.283	76571658.017

Table 4.11: The Simulation Result of Parallel model of the OSCI TLM-2.0 Interconnect Interface and SystemC BFM with non-blocking transport interface (using RR)

Statistics	Time	N_{TI}	N_{LD}	N_{ST}
min	1791.650s	520539236	162591560	73902863
max	1967.391s	549512461	173078706	80891471
μ	1881.795s	535139436.183	168582898.367	77458143.367

average (shown as Table 4.12, and the overhead defined by (4.1)), which is compared to [5].

Clearly, either FP or RR, the overhead of simulation time for the proposed OSCI TLM-2.0 interconnect interface has been reduced significantly (shown as Figures 4.3 and 4.4, Tables 4.13 and 4.14). Finally, the speedup by using the each simulation model of the OSCI TLM-2.0 interconnect interface is shown in the column of "speedup" in Table 4.16 defined by (4.3). Compared to Q'pT model, the best speedup result is QTb by a factor of 3.255.

The percentages given in Tables 4.12, 4.13 and 4.14 are defined by

Table 4.12: The Simulation Result of the QSC-CCA with the Prototype of the OSCI TLM-2.0 Interconnect Interface [5, 26] and the Overhead Compared to the QSC-CCA

Statistics	$T_{\text{Q'}}$	$T_{\text{Q'pT}}$
min	08m47.359s	40m50.756s
max	12m48.265s	44m20.429s
μ	10m44.956s	42m05.767s
overhead	n/a	291.618%

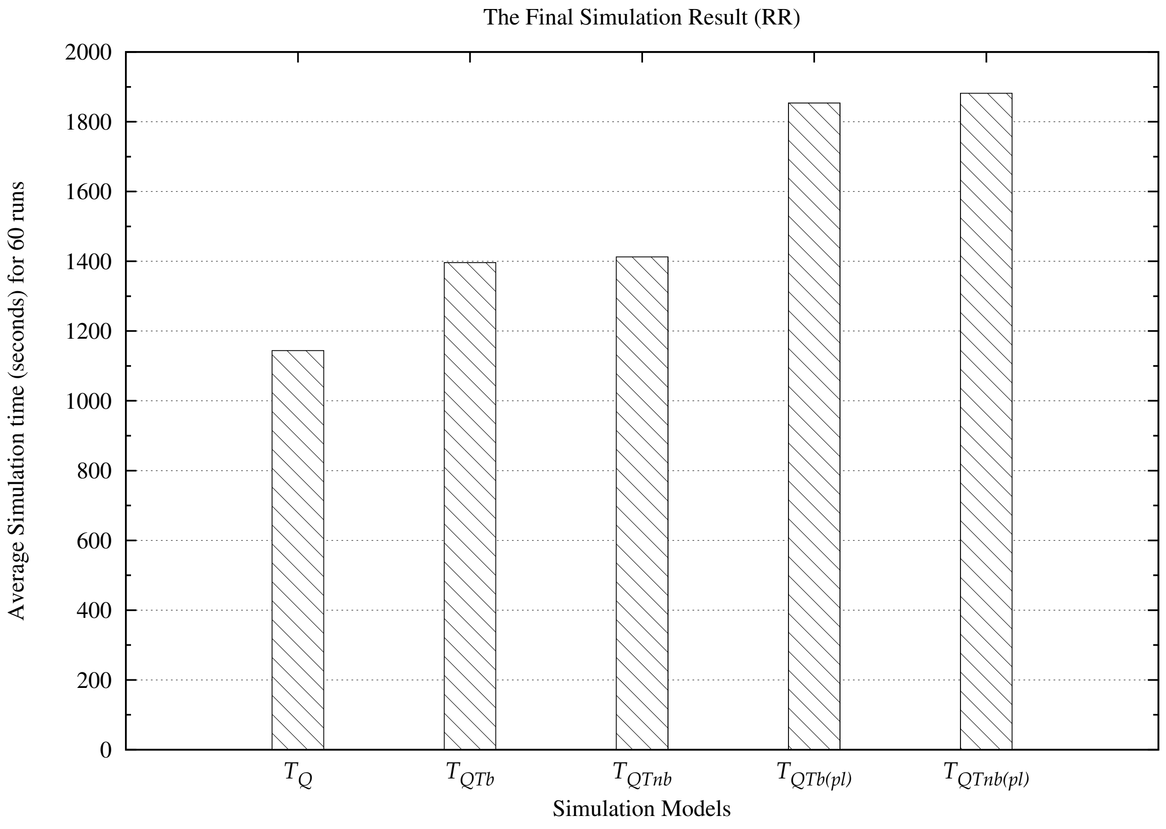


Figure 4.2: Full Simulation Result using the RR policy

Table 4.13: The Simulation Result of each Simulation Group using FP Arbitraion Policy, and the Overheads Compared to Original QSC-CCA

Statistics	T_{Q}	T_{QTb}	T_{QTnb}	$T_{\text{QTb(pl)}}$	$T_{\text{QTnb(pl)}}$
min	977.663s	1176.164s	1182.383s	1504.878s	1513.957s
max	1042.848s	1261.099s	1241.003s	1636.159s	1595.926s
μ	1010.013s	1215.310s	1217.633s	1556.495s	1568.431s
overhead	n/a	20.326%	20.556%	54.106%	55.288%

$$overhead = \frac{T_\alpha - T_Q}{T_Q} \times 100\% \tag{4.1}$$

each T_α represent its mean value, μ, where the subscript α is either QTb, QTnb, QTb(pl), QTnb(pl) or Q'pT. For instance, the percentage given in the column labeled "T_{QTb}" of the row labeled "overhead" of Table 4.13 is computed as

$$\frac{1215.310 - 1010.013}{1010.013} \times 100\% = 20.326\% \tag{4.2}$$

Table 4.14: The Simulation Result of each Simulation Group using RR Arbitraion Policy, and the Overheads Compared to Original QSC-CCA

Statistics	T_{Q}	T_{QTb}	T_{QTnb}	$T_{\text{QTb(pl)}}$	$T_{\text{QTnb(pl)}}$
min	977.663s	1371.241s	1370.391s	1805.915s	1791.650s
max	1042.848s	1418.934s	1450.806s	1906.123s	1967.391s
μ	1010.013s	1396.338s	1412.533s	1853.673s	1881.795s
overhead	n/a	38.250%	39.853%	83.530%	86.314%

Table 4.15: Notations used in figures and tables of this chapter

min	The best-case co-simulation time
max	The worst-case co-simulator time
μ	The mean of co-simulation time
N_{TI}	The number of target instructions simulated
N_{LD}	The number of load operations of the virtual processor
N_{ST}	The number of store operations of the virtual processor
Q'pT	The co-simulation model of the QSC-CCA with the prototype of the OSCI TLM-2.0 interconnect interface [26]
QTb	The co-simulation model of the QSC-CCA with the improved OSCI TLM-2.0 interconnect interface (using *blocking_transport*)
QTnb	The co-simulation model of the QSC-CCA with the improved OSCI TLM-2.0 interconnect interface (using *non-blocking_transport*)
QTb(pl)	The co-simulation model of the QSC-CCA with the parallel model of the improved OSCI TLM-2.0 interconnect interface (using *blocking_transport*)

Table 4.15 (cont.)

	and the QSC-CCA BFM
QTnb(pl)	The co-simulation model of the QSC-CCA with the parallel model of the improved OSCI TLM-2.0 interconnect interface (using *non-blocking_transport*) and the QSC-CCA BFM
$T_{\mathrm{Q'}}$	The co-simulation time of original QSC-CCA model without the OSCI TLM-2.0 interconnect interface recorded in [5]
T_{Q}	The co-simulation time of original QSC-CCA model without the OSCI TLM-2.0 interconnect interface
$T_{\mathrm{Q'pT}}$	The co-simulation time of Q'pT recorded in [26]
T_{QTb}	The co-simulation time of QTb
T_{QTnb}	The co-simulation time of QTnb
$T_{\mathrm{QTb(pl)}}$	The co-simulation time of QTb(pl)
$T_{\mathrm{QTnb(pl)}}$	The co-simulation time of QTnb(pl)
$O_{\mathrm{Q'pT}}$	The overhead of simulation time as compared to Q'pT (%)
O_{QTb}	The overhead of simulation time as compared to QTb (%)
O_{QTnb}	The overhead of simulation time as compared to QTnb (%)
$O_{\mathrm{QTb(pl)}}$	The overhead of simulation time as compared to QTb(pl) (%)
$O_{\mathrm{QTnb(pl)}}$	The overhead of simulation time as compared to QTnb(pl) (%)
overhead	The overhead of addtional simulation time as compared to each simulation model (%)

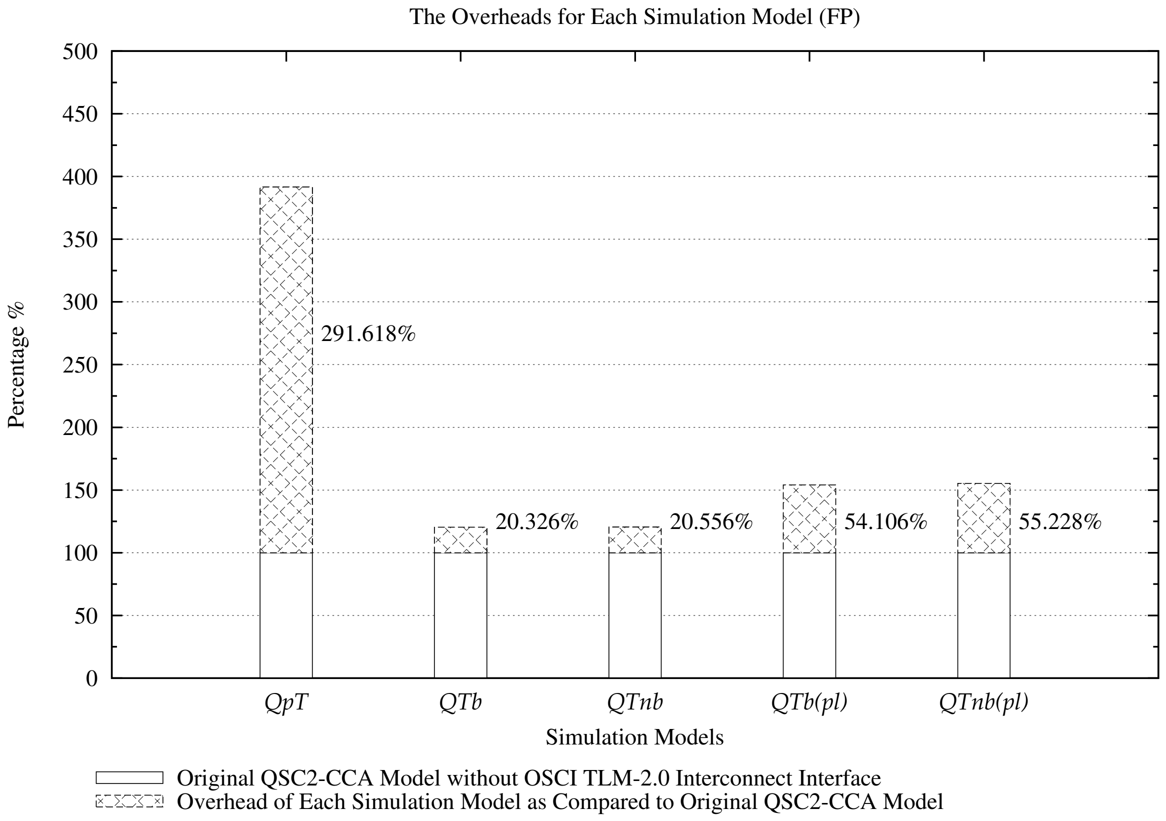


Figure 4.3: The Comparison Result (Fixed Priority)

Table 4.16: Comparison of the simulation overhead

Statistics	original	average case	speedup
QSC-CCA-FP with OSCI TLM-2.0 (blocking)	291.618%	20.326%	3.255
QSC-CCA-FP with OSCI TLM-2.0 (non-blocking)	291.618%	20.556%	3.248
QSC-CCA-RR with OSCI TLM-2.0 (blocking)	291.618%	38.250%	2.832
QSC-CCA-RR with OSCI TLM-2.0 (non-blocking)	291.618%	39.853%	2.800

The value of Table 4.16 in the column labeled “speedup”are defined by

$$speedup = \frac{Time_{\text{(Original)}}}{Time_{\text{(Improved)}}} = \frac{1 + O_{Q'pT}}{1 + O_{\beta}} \tag{4.3}$$

each O_β represent its mean value, μ, where the subscript β is either $O_{\text{Q'pT}}$, O_{QTb}, O_{QTnb}, $O_{\text{QTb(pl)}}$ or $O_{\text{QTnb(pl)}}$. For instance, the percentage given in the column labeled “speedup” of the first row

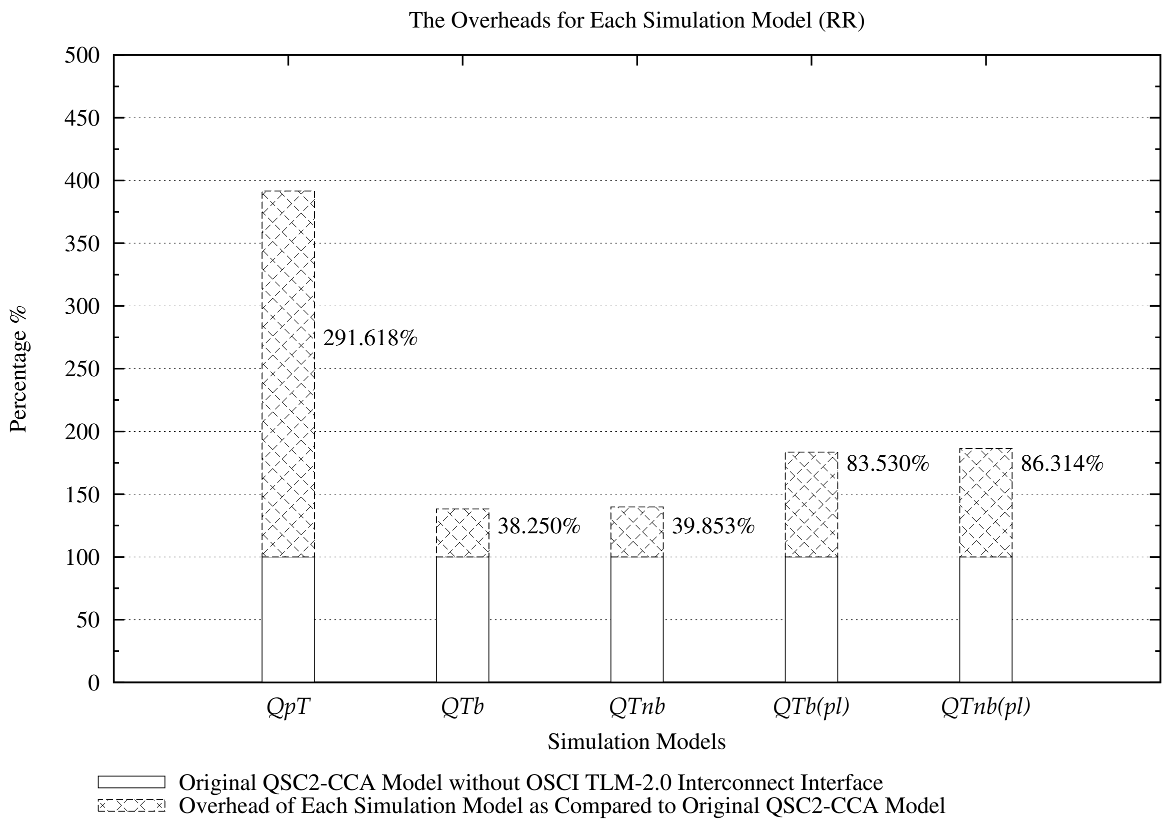


Figure 4.4: The Comparison Result (Round Robin)

of Table 4.16 is computed as

$$\frac{1 + 291.618\%}{1 + 20.326\%} = 3.255 \tag{4.4}$$

Chapter 5

Conclusion and Future Works

5.1 Conclusion

The QSC-CCA is a very convenient and efficient framework for simulation. It can directly do the simulation of running an operating system on the QSC framework. Using the SystemC component to run with QEMU, the simulation can work with each other directly. By connecting with the QSC-CCA, the necessary BUS signals at CA level and most information about the cycle counts can be easily collected.

With the release of OSCI TLM-2.0 Standard, the combination of the proposed the OSCI TLM-2.0 interconnect interface and the QSC now provides the highly efficient interface of bus connections for OSCI TLM-2.0 virtual hardwares. The simulation speed is faster than origianl QSC-CCA and easily be used for any numbers of TLM-2.0 IPs.

5.2 Future Works

Although it is a very efficient method to support the TLM-2.0 IPs, and very convenient to use, the bottlenecks of the full QSC-CCA SystemC is on the SystemC side. This thesis gives a hint about the loading of the OSCI TLM-2.0 feature does not cost very heavily on the QSC simulation.

Meansuring the trade-off between convenient and efficiency, some rearrangements of the necessary parts of the QSC virtual platform to TLM-2.0 can be taken into consideration. SystemC is a Hardware Description Language (HDL), in the user's point of view, they wish to build the model without coding. Moreover, the OSCI TLM-2.0 can be used to model the hardware and possible to connect to this framework. It can be realized by a GUI front-end, and it interacts with the TLM-2.0 interconnect as back-end. The final feature of the QSC will be a very useful tool that building hardware model without coding and directly simulating with other external virtual hardwares.

Bibliography

[1] L. Cai and D. Gajski, “Transaction Level Modeling: An Overview,” in *Proc. First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 19–24, 1–3 Oct. 2003.

[2] “Open SystemC Initiative (OSCI).” [Online]. Available: http://www.systemc.org/.

[3] D. John Aynsley, *OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL*, 2.0.1 ed., July 2009. Document version: JA32.

[4] M. Montón, A. Portero, M. Moreno, B. Martínez, and J. Carrabina, “Mixed SW/SystemC SoC Emulation Framework,” in *Proceedings of IEEE International Symposium on Industrial Electronics*, pp. 2338–2341, June 2007.

[5] T.-C. Yeh and M.-C. Chiang, “On the interface between QEMU and SystemC for hardware modeling,” in *Proc. Int Next-Generation Electronics (ISNE) Symp*, pp. 73–76, 2010.

[6] T.-C. Yeh, G.-F. Tseng, and M.-C. Chiang, “A fast cycle-accurate instruction set simulator based on QEMU and SystemC for SoC development,” in *Proc. MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conf*, pp. 1033–1038, 2010.

[7] T.-C. Yeh and M.-C. Chiang, “Bus performance exploration at CCA and CA levels on QEMU and SystemC-based virtual platform,” in *Proc. Int. SoC Design Conf. (ISOCC)*, pp. 376–379, 2010.

[8] T.-C. Yeh, Z.-Y. Lin, and M.-C. Chiang, “Optimizing the Simulation Speed of QEMU and SystemC-Based Virtual Platform,” in *Proc. 2nd Int Information Engineering and Computer Science (ICIECS) Conf*, pp. 1–4, 2010.

[9] M.-C. Chiang, T.-C. Yeh, and G.-F. Tseng, “A QEMU and SystemC-Based Cycle-Accurate ISS for Performance Estimation on SoC Development,” vol. 30, no. 4, pp. 593–606, 2011.

[10] F. Bellard, “QEMU.” [Online]. Available: http://bellard.org/qemu/index.html.

[11] Design Automation Standards Committee, *IEEE Standard System C Language Reference Manual*, 2005.

[12] D. C. Black and J. Donovan, *SystemC: From The Ground Up*. Springer Science+Business Media, 2004.

[13] Inc. ARM Components, “Amba specification (rev. 2).” May 1999.

[14] S. Boukhechem, E.-B. Bourennane, and H. Samahi, “Co-simulation Platform Based on SystemC for Multiprocessor System on Chip Architecture Exploration,” in *Proc. Internatonal Conference on Microelectronics ICM 2007*, pp. 105–110, 29–31 Dec. 2007.

[15] C.-C. Wang, R.-P. Wong, J.-W. Lin, and C.-H. Chen, “System-level development and verification framework for high-performance system accelerator,” in *Proc. Int. Symp. VLSI Design, Automation and Test VLSI-DAT ’09*, pp. 359–362, 2009.

[16] J.-W. Lin, C.-C. Wang, C.-Y. Chang, C.-H. Chen, K.-J. Lee, Y.-H. Chu, J.-C. Yeh, and Y.-C. Hsiao, “Full system simulation and verification framework,” in *Proc. Fifth Int. Conf. Information Assurance and Security IAS ’09*, vol. 1, pp. 165–168, 2009.

[17] M. Monton, J. Carrabina, and M. Burton, “Mixed simulation kernels for high performance virtual platforms,” in *Proc. Forum Specification & Design Languages FDL 2009*, pp. 1–6, 2009.

[18] M. Becker, G. Di Guglielmo, F. Fummi, W. Mueller, G. Pravadelli, and T. Xie, “Rtos-aware refinement for tlm2.0-based hw/sw designs,” in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 1053–1058, 2010.

[19] N. Bombieri, F. Fummi, and G. Pravadelli, “On the Evaluation of Transactor-based Verification for Reusing TLM Assertions and Testbenches at RTL,” in *Proc. Design, Automation and Test in Europe DATE ’06*, vol. 1, pp. 1–6, 6–10 March 2006.

[20] N. Bombieri, N. Deganello, and F. Fummi, “Integrating RTL IPs into TLM Designs Through Automatic Transactor Generation,” in *Proc. Design, Automation and Test in Europe DATE ’08*, pp. 15–20, 10–14 March 2008.

[21] M. Chen, P. Mishra, and D. Kalita, “Towards RTL Test Generation from SystemC TLM Specifications,” in *Proc. IEEE International High Level Design Validation and Test Workshop HLVDT 2007*, pp. 91–96, 7–9 Nov. 2007.

[22] N. Bombieri, F. Fummi, and G. Pravadelli, “On the mutation analysis of systemc tlm-2.0 standard,” in *Proc. 10th Int Microprocessor Test and Verification (MTV) Workshop*, pp. 32–37, 2009.

[23] H. van Moll, “Cycle-accurate and protocol specific modeling methodology using tlm 2.0,” Master’s thesis, The Faculty of Electrical Engineering of the Eindhoven University of Technology, 2008.

[24] H. W. M. van Moll, H. Corporaal, V. Reyes, and M. Boonen, “Fast and accurate protocol specific bus modeling using tlm 2.0,” in *Proc. DATE ’09. Design, Automation & Test in Europe Conf. & Exhibition*, pp. 316–319, 2009.

[25] C. Genz, R. Drechsler, G. Angst, and L. Linhard, “Visualization of SystemC Designs,” in *Proc. IEEE International Symposium on Circuits and Systems ISCAS 2007*, pp. 413–416, 2007.

[26] T.-C. Yeh, Z.-Y. Lin, and M.-C. Chiang, “Enabling tlm-2.0 interface on qemu and systemc-based virtual platform,” in *Proc. IEEE Int IC Design & Technology (ICICDT) Conf*, pp. 1–4, 2011.