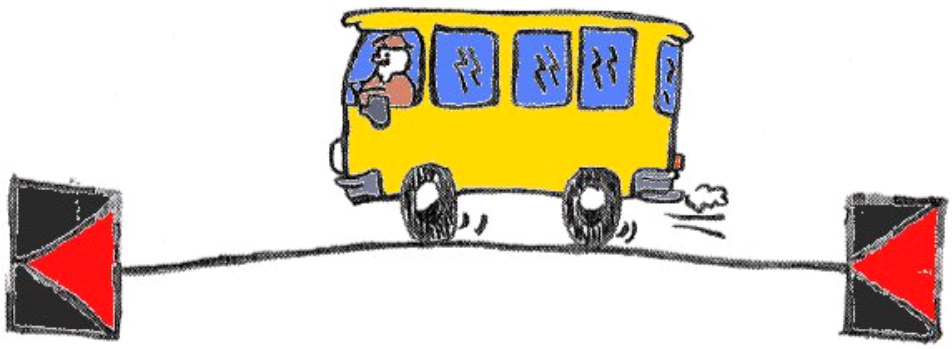
Class: FIN
Doc. no: TERMA/SDP/63/FINALREP/1001
Rev: 1.1
Date: 2014-04-15
Approved by: M. Alberti

TERMA

Development of new SystemC IP Models and OCP-IP Sockets Final Report



Prepared:

Antonio Bianco / Alberto Ferrazzi Date 2014-04-15

Checked:

Dan Søren Nielsen Date 2014-04-15
QA Manager

Approved:

Michela Alberti Date 2014-04-15
Department Manager

Checked:

Mariarosaria Cardone Date 2014-04-15
Project Manager

Michela Alberti Date 2014-04-15
Department Manager

This document is released for use only if signed by relevant staff or stamped "EDM Release Controlled".

CM:

Template no: 199997-FA, Rev. G

Record of Changes

ECR/ECO	Description	Rev	Date
	Released	1.0	2014-01-29
	List of rids from final review: • FR-6: explained the problems encountered on extending SoCRocket with BUS size configurability • FR-7: explained the problems encountered in the attempt of adding Retry and Split functionalities	1.1	2014-04-15

Contents

Figures

Tables

No table of figures entries found.

1 Introduction

1.1 Purpose

The purpose of this document is to serve as the final report of the work carried out in the *SystemC IP models development* project.

1.2 Scope

The scope of this document is the *SystemC IP models development* project.

2 Applicable and reference documents

2.1 Applicable documents

Ref.	Doc. No.	Title
[AD1]	2.0	AMBA™ Specification
[AD2]	JA32	OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL
[AD6]	ECSS-E-ST-50-12C	SpaceWire – Links, nodes, routers and networks
[AD7]	ECSS-E-ST-50-52C	SpaceWire - Remote memory access protocol
[AD8]	IDA-SCSV-UM-001	IP User Manual & Development Document
[AD9]	1.8	Quad Core LEON4 SPARC V8 Processor LEON4-NGMP-DRAFT Data Sheet and User's Manual
[AD10]	BSSC(2000)1 Issue 1	C and C++ Coding Standards

2.2 Reference documents

Ref.	Doc. No.	Title
[RD2]	IDA-SCSV-PD-001	SystemC IP Verification and Performance Document
[RD7]	1.2.2	GRLIB IP Core User's Manual
[RD8]	2.0.1	SystemC User's Guide
[RD9]	4.0	GreenReg User's Guide
[RD10]	4.2.0	GreenConfig User's Guide

3 Terms, definitions, and abbreviated terms

3.1 Terms

None.

3.2 Definitions

None.

3.3 Abbreviated terms

Abbreviation	Definition
TLM	Transaction-Level Modeling
TU	Technische Universität
DDR	Double Data Rate
DRAM	Dynamic Random Access Memory
SDRAM	Synchronous Dynamic Random Access Memory
MSDRAM	Multi-backend Synchronous Dynamic Random Access Memory
IOMMU	Input/Output Multi-Media Unit
IRQ	Interrupt ReQuest
IRQMP	Interrupt ReQuest controller with MultiProcessing
IRQ(A)MP	Interrupt ReQuest controller with Asymmetric MultiProcessing
NGMP	Next Generation MultiProcessor
AT	Approximately Timed
LT	Loosely Timed

4 Project overview

4.1 Scope of the activity

TU Braunschweig, in collaboration with ESTEC, has created and maintained SOCROCKET, a simulation framework based on SystemC and TLM technologies. The framework was conceived to simulate hardware transactions on AMBA AHB/APB buses between different intellectual property cores integrated in the hardware architecture. TU-Braunschweig has hence already developed many IP-Core models referencing LEON3 platform used by the agency.

The main aim of such a system was to enable early architectural exploration and validation. In fact, TLM provides two different coding standards for modeling bus transactions, AT (Approximately Timed) for which asynchronous functions model the different bus-phases (instruction and data), and LT (loosely timed) for which a synchronous function models each bus transaction. AT coding standard allows to model with a certain degree of accuracy the bus interaction between components, and thus is a good indicator for the architecture reliability once all the components are integrated and an application is executed on the emulated platform.

The main aspect of this project was the development of new IP-core models, extension of existing ones with new functionalities, and integration into the existing simulation infrastructure, in order to validate the overall NGMP architecture, currently under development.

The new components model the following IP-Cores:

- SDR/DDR2 Memory Controller
- Level 2 Cache
- GR-IOMMU
- GRSPW2 Spacewire Controller

An existing IRQMP has also been extended with Asymmetric multiprocessing capabilities.

Each IP-Core supplies a set of tests to verify the functionalities of the core in an environment isolated from the rest of the architectural elements.

A demonstrator has also been developed to validate the overall architecture, integrating the components developed in the scope of this project. The demonstrator connects most of the IP-Cores on the AMBA buses to replicate an emulated hardware platform with leon4-like architecture, capable of running SPARC programs and outputting execution time and components related statistics.

Finally, SPARC software capable to run on the virtual platform has been developed in order to demonstrate some functionality of the new components.

4.2 Programme of work

The project is composed of 4 major tasks, IP-Cores models definition, IP-Cores models implementation, IP-Cores models validation and verification, Virtual Platform integration and demonstration. The first three tasks are replicated for each IP-Core model being developed.

Preliminary data-sheet for NGMP [NGMP] multiprocessor, and GRLIB user's manual [GRIP] for Hardware IP-Cores designed and implemented by Aeroflex Gaisler AB served as guideline for models definition, implementation and validation. Generally, whenever a conflict between IP-Core descriptions contained in NGMP and GRLIP documents arose, GRLIB took precedence being this a finalized document for real working hardware. This rule has been applied to all IP-Cores with the exception of MSDRAM, being this IP-Core partially the synthesis of two existing cores plus a front-end on the AMBA-BUS, thus being this partially new hardware NGMP document was taking precedence in conflict resolution.

The work process used to accomplish each of the above tasks decomposes them further in subtasks as depicted in Figure 1.

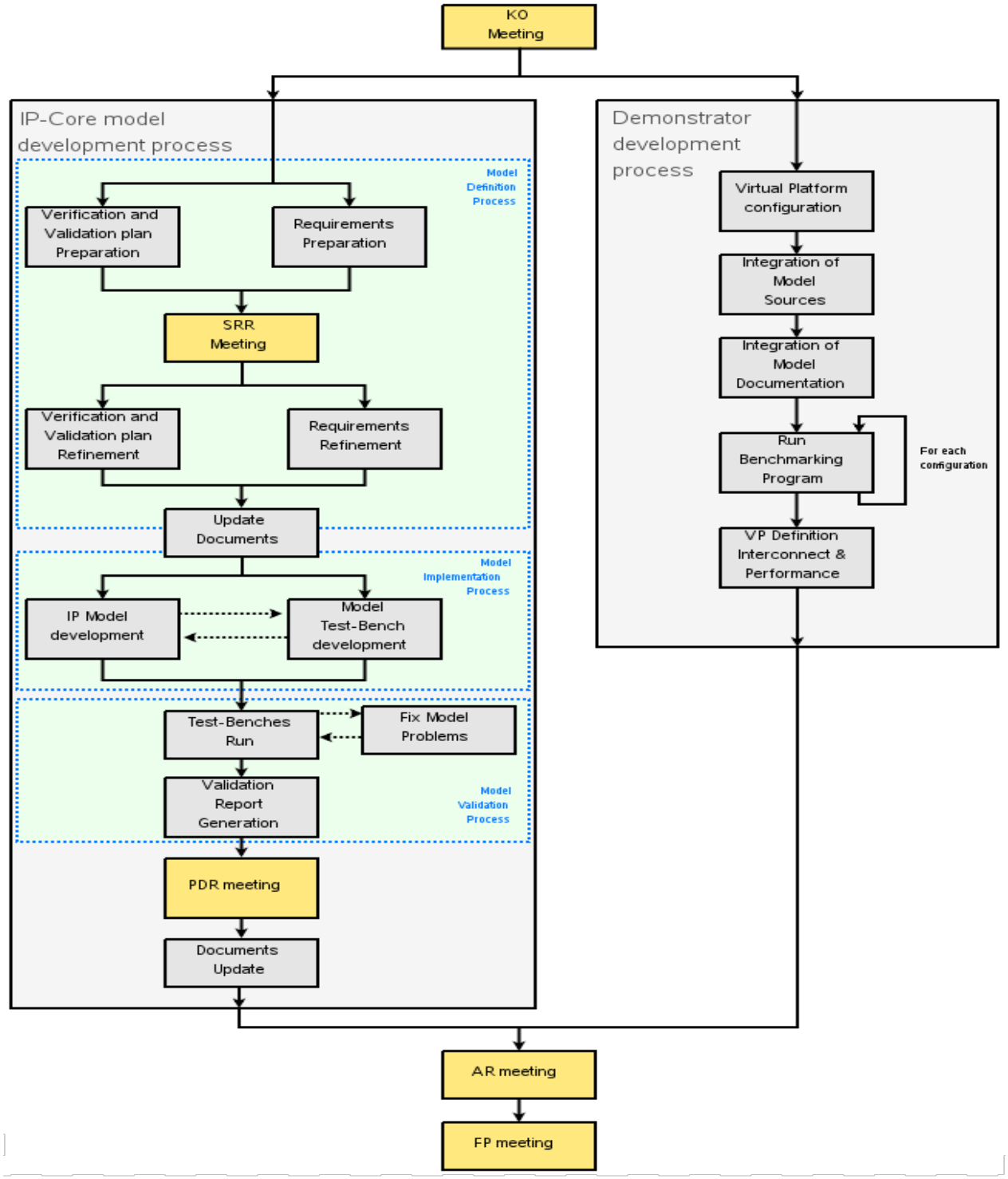


Figure 1: Work Program

5 Performed activities

5.1 Performed work

The workflow of the performed tasks and activities of the project has been described in Section 4.2. The task completion resulted in the delivery of updated documentation and new software source code for each item, with the exception of IRQ(A)MP, for which new source code has been integrated into the existing IRQMP IP-Core model.

The produced items are:

- MSDRAM memory controller IP-Core model
- GRIOMMU IP-Core model
- IRQ(A)MP IP-Core model
- Level 2 Cache IP-Core model
- GRSPW2 Spacewire IP-Core model
- Demonstrator

5.2 Delivered outputs

For each IP-Core being modeled and developed, the deliverable consisted in the update of the relevant chapter of document [AD1] that describes the functionalities offered by the IP-Core in terms of:

- configuration and instantiation parameters
- detailed description for defined control registers
- default values
- choices taken for undefined behavior
- compilation
- example IP-Core model instantiation

Also [AD2] chapter 6 has been updated with:

- user requirements definition for the IP-Core
- test description
- detailed test procedures
- test execution report.

In the course of the project development, in addition to documentation updates, source code for the model and for executed test-benches have been delivered in the form of a tag to a repository accessible to the ESTEC T.O.

5.2.1 MSDRAM IP-Core model and test-benches

In addition to the common set of document updates, MSDRAM IP-Core chapter 7 of document [AD1] also describes the additional functionalities offered by the MSDRAM in terms of:

- error detection and correction capabilities
- timings
- implementation behavior depending to the coding standard

5.2.2 GRIOMMU IP-Core model and test-benches

In addition to the common set of document updates, GRIOMMU IP-Core chapter 12 of document [AD1] also describes the additional functionalities offered by the GRIOMMU in terms of:

- Cache flushing options
- Diagnostic accessibility of the component
- Interrupts masking
- Prefetching limitations

5.2.3 IRQ(A)MP IP-Core model and test-benches

In addition to the common set of document updates, IRQ(A)MP IP-Core chapter 14 of document [AD1] also describes the additional functionalities offered by the IRQ(A)MP in terms of:

- Asymmetric Multiprocessing

5.2.4 L2C IP-Core model and test-benches

In addition to the common set of document updates, L2C IP-Core chapter 9 of document [AD1] also describes the additional functionalities offered by the L2C in terms of:

- Associativity and replacement policy
- Write policy
- Error detection and correction
- Flushing operations
- Diagnostic functionalities and interfaces
- Timings and latencies
- Cached Memory scrubbing
- Memory Type Range Registers cache behavior configurability
- Cache Way replacement Locking

5.2.5 GRSPW2 IP-Core model and test-benches

In addition to the common set of document updates, GRSPW2 IP-Core chapter 10 of document [AD1] also describes the additional functionalities offered by the GRSPW2 in terms of:

- Interface exposed on the Spacewire Link

- Interface exposed to the AMBA Bus
- Memory used for internal buffers
- Device Attributes for Plug and Play configuration

5.2.6 Virtual Platform

In order to demonstrate the usage of the developed components and test their functionality once integrated, a virtual platform with a leon4-like architecture has been implemented.

The overall architecture, with the involved components and their connections on the AHB/APB busses, is represented in Figure 1

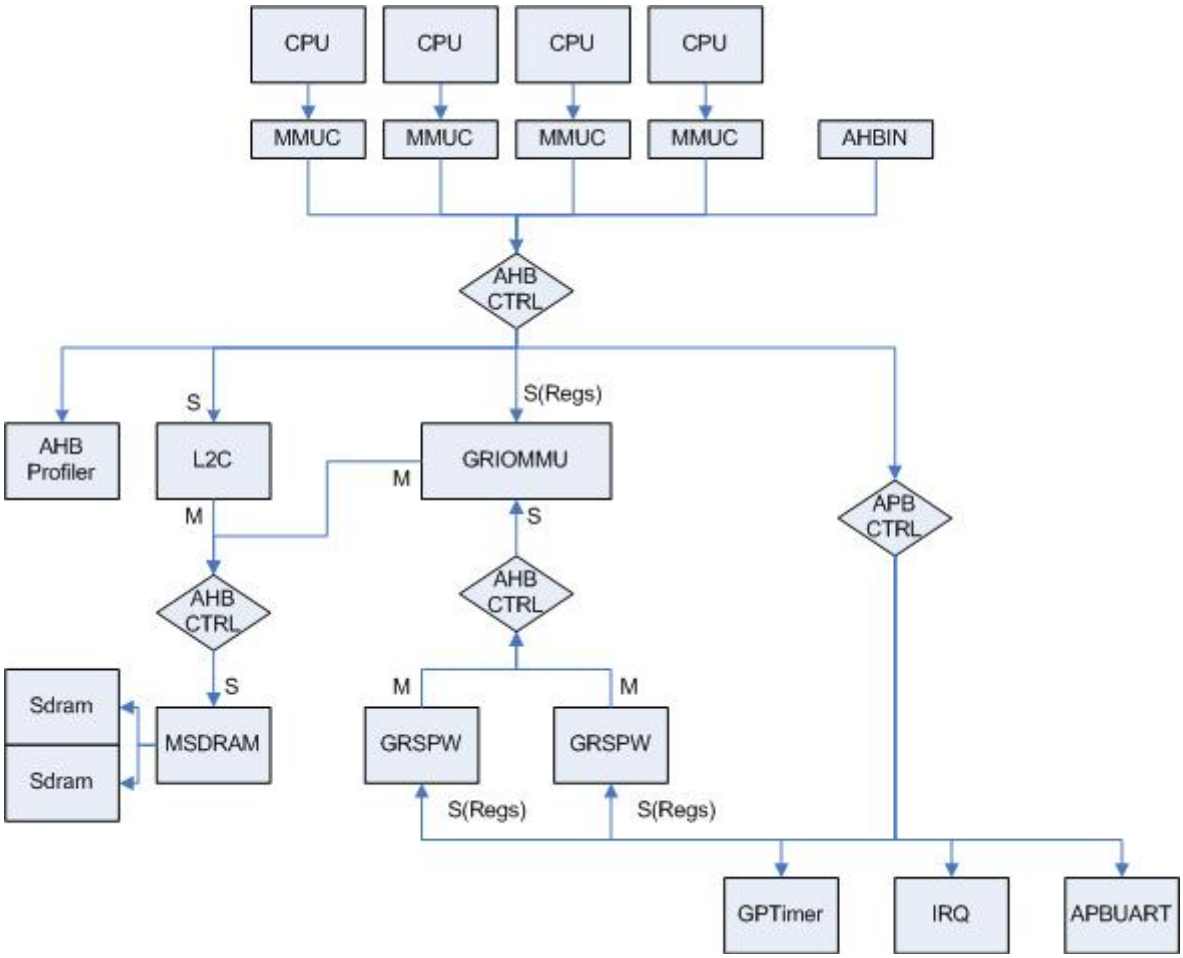


Figure 2: Virtual Platform Architecture

Wherever was possible, the memory mapping,AHB/APB interfaces ID and interrupt numbers used in the virtual platform matches the one stated by NGMP.

The GreenConfig framework has been used to supply constructor parameters for all the components allowing the platform to be highly customizable. Each component configuration can be tweaked through a JSON file that can be passed as an optional parameter when run-

ning the platform. In case that no file is supplied a default configuration is used. The SOCROCKET configuration generator wizard can be used to create the JSON configuration file as the xml template file for the generator has been implemented.

To run a program on the platform two SPARC programs, presented in ELF format, must be specified through parameters: the boot program, which performs initialization of some registers values required for further execution, and the program to be executed. Both programs are loaded into RAM to avoid the usage of an additional memory controller for PROM.

A set of programs focused on demonstrating the usage of the new components have been written and can be found under software/l4mpdemo folder.

Among those, griommudemo.sparc has been chosen to be the testbench program as it uses all the all the developed components (MSDRAM, L2C, GRIOMMU, GRSPW2, IRQMP).

The testbench configures the GRIOMMU to use IOMMU memory protection mode and sets an area with write protection and one where writing is allowed. Then it configures the two Spacewires to send packets from one to the other using interrupts to detect when packets have been sent or received. Tree packets are sent checking if they are written or not: the first falling into writeable area the second falling into the write protected area and the third into protected area but with GRIOMMU turned off. Program execution, GRIOMMU IOMMU table reading and SW descriptors and packets read/writes test MSDRAM and L2C implicitly.

6 Achieved results

6.1 IP-Core Models Implementation and Workarounds

The IP-Core models part of the project had to face some issues related to some minor inconsistencies in the hardware documentation still in draft status. The encountered inconsistencies were retrofitted to Aeroflex-Gaisler, and the documents promptly updated in the sequent release.

SocRocket BUS encapsulation architecture, however, represented the biggest issue to overcome during the scope of the project. For instance, AHBMaster and AHBSlave C++ classes hardcode the size of the bus being used. These classes are inherited by each component exposing either master or slave interfaces on the BUS. The modification of those classes to permit different bus sizes is still an open point. Both AHBMaster and AHBSlave inherit from classes passed as template parameter, and many solutions have been tried in order to overcome this limitation, such as the addition of another template parameter to specify the size of the bus being used. Unfortunately, such a solution would have implied the usage of methods template partial specialization, and the C++ standard only allows classes template partial specialization. In addition, partial specialization of the whole class was not a feasible solution, due to the extensive use of preprocessor macros defined by GreenRegs and SystemC. Facing this issue would imply a partial redesign of SocRocket infrastructure, and a refactoring of the existing IP-Core models. This issue has been worked around providing a template specialization for all the possible constructors for each allowed bus size (i.e. 32, 64 and 128 bits), nevertheless this leads to the suboptimal code due to the many time replication of the constructor in the modified classes, and of those using more than one size for the bus.

Another architectural limitation is that most of the components feature an access on AHB bus deriving from one of the two above stated classes. This prevents a multi-bus configuration extension of those IP-Core models. Such a model has not been applied to GRSPW2 component, in fact this limitation has been worked around quite neatly by a careful design of the model and by composing classes inheriting from AHBMaster and APBDevice for accessing the buses. A careful design of interfaces for classes accessing the BUS, also permitted an internal architecture of the GRSPW2 without the propagation of template parameters in each of the submodules.

Another very small technical limitation was represented by the GreenRegs library, since it does not allow direct read prevention of the defined registers.

6.2 Virtual Platform Performance Results

The following lines contain key performance values reported by the execution on the platform of grspw2demo and griommudemo programs:

6.2.1 Grspw2demo.sparc

Execution Time	
Real Time Execution	7 [s]
Simulation Time Execution	3009830 [ns]
CPU AHB bus	

Transactions on bus	42423
Bytes read on bus	151179
Bytes written on bus	8571
Memory AHB bus	
Transactions on bus	524
Bytes read on bus	16768
Bytes written on bus	0
Device AHB bus	
Transactions on bus	10
Bytes read on bus	55
Bytes written on bus	40
APB bus	
Transactions on CPU bus	90
Bytes read on CPU bus	4488
Bytes written on CPU bus	140
IRQS	
Total number of IRQ	2
L2C	
Number of accesses	41827
Cache hits	41601
Number of cache line loaded	524
GRSPW1	
Transmitted Packets	1
Transmitted Chars	16
Received Packets	0
Received Chars	0
GRSPW2	
Transmitted Packets	0

Transmitted Chars	0
Received Packets	1
Received Chars	16
MDRAM	
Bytes Read	16768
Bytes Write	61588
GRIOMMU	
Total accesses	10
Inhibited accesses	0
Cache hits	0

6.2.2 Grspw2demo.sparc

Execution Time	
Real Time Execution	200 [s]
Simulation Time Execution	123228840 [ns]
CPU AHB bus	
Transactions on bus	668884
Bytes read on bus	2082622
Bytes written on bus	1091292
Memory AHB bus	
Transactions on bus	35061
Bytes read on bus	1085728
Bytes written on bus	36224
Device AHB bus	
Transactions on bus	30
Bytes read on bus	165
Bytes written on bus	120
APB bus	
Transactions on bus	138

Bytes read on bus	4680
Bytes written on bus	308
IRQS	
Total number of IRQ	6
L2C	
Number of accesses	668203
Cache hits	768010
Number of cache line loaded	33929
Memory Reads	33929
Memory Writes	1132
GRSPW1	
Transmitted Packets	3
Transmitted Chars	48
Received Packets	0
Received Chars	0
GRSPW2	
Transmitted Packets	0
Transmitted Chars	0
Received Packets	3
Received Chars	48
MDRAM	
Bytes Read	1085728
Bytes Write	1166788
GRIOMMU	
Total accesses	30
Inhibited accesses	2
Cache hits	0

6.3 Lesson Learned

Modeling hardware can be a difficult task, and ensuring decent performance for the modeled hardware makes the task even more complex. This is the most likely rationale of a quite complex implementation for SocRocket founding classes, and for the used libraries.

6.3.1 SocRocket specific usage

SocRocket, as a framework, solves a set of quite challenging problems, such as linearization of the interleaved set of address/data phases in pipelined bus architecture while keeping performance at the same level of statically bound methods and allowing reusability of founding classes. The only way to achieve such a goal is to harvest from C++ template potential in order to ensure also compile time check correctness.

Nevertheless, the complexity added by this solution causes some problems when requested to scale to a more complex architecture design, or when required to be modified in complexity.

Another bug with no easy solution is that BUS pipelining was not working correctly, since an AT-mode 32-bit bus access to an idle 32-bit bus should return a delay of 1, but tests showed a delay of 8 as if the function was not returning until the request was completed. This somehow invalidates the conceptual separation between the AT and LT coding standard.

An early analysis of the capabilities of the founding classes would have probably spared us from many problems at a later stage when fixing them was harder. In particular, early testing for different bus sizes and for multi-bus capability of components would have highlighted framework limitations that we had to face later in the project.

6.3.2 Spacewire design

GRSPW2 was by far the most complex component we had to model. Luckily enough, this was also the last component modeled, thus we could count on the experience coming from modeling previous component with a smaller set of functionalities.

This allowed us to have a better design for the IP-Core model, with workarounds to overcome limitations defined at design phase. The most significant lesson here is that simpler component first is a good rule of thumb for testing framework functionalities, shortcomings, limitations and potential, in fact modeling a simpler model, but containing also unprecedented features, such as multi-bus connection, clearly showed what are the best strategies for implementing more complex ones.

Another lesson that we took into account in the design and the implementation of GRSPW2 component is that with a higher complexity of the Core modeled, it is usually worth to exchange some speed with a higher level of flexibility. In fact, we adopted a quite modular approach and this allowed the implementation and testing of smaller modules for the Spacewire and a later assembly of them in the complete component, at the cost of a couple more level of indirection in the function calls.

6.4 Proposed improvements and feedbacks

As a development framework for IP-Core models, SocRocket has a big potentiality, since it simplifies a lot the utilization of the AMBA Bus TLM model. Nevertheless, it is still not completely mature, and some of its components may undergo a rework activity to enhance flexibility, such as permitting to specify the BUS size, or to enhance usability, for example by providing a better interface structure for complex types making extensive use of class tem-

plates. Implementing those changes is for sure not effort free, but the benefit of doing that would enable the platform to model nearly everything that may exploit the AMBA architecture.

6.4.1 Bus size flexibility

One of the improvements we suggest for the framework is to enhance the configurability of the bus sizes used by each component.

The AHBMaster and AHBSlave classes in the initial SoCRocket version had the sockets BUS sizes hardcoded to 32 bits. In order to make the socket size configurable, we did a quick attempt adding a BUSSIZE template parameter, but we ran into the “partial specialization for methods” problem: in C++ it’s not possible to specialize only one of the method template parameters, while not specializing the others. Even if it was not leading to a nice solution, we attempted to solve this by specializing also the other parameter with predefined BUS sizes (32, 64,128). For each of the two classes we had to clone the constructor 2*3=6 times.

This attempt failed as well: it was now forbidden to pass template parameters from our model to the AHB classes because the latest could only handle some specific values of them.

Probably specializing also the model constructor with the possible bus size values could have solved the problem, but this is something we did not want to consider as it could lead to cases where you have to clone the constructor too many times. The L2C for example had to clone it 9 times.

We provided a work around that mimics the proposed improvement, nevertheless a thorough rework of AHBMaster, AHBSlave and APBDevices classes to permit flexible buffer would extend the already good potential of the framework.

6.4.2 Multi-BUS composition

A class that encapsulates the bus access and composes the classes accessing the single buses would make the usage of the framework extremely easier. It is recommended, then, to expand the founding classes portfolio with such a compositing class.

6.4.3 Transform SocRocket in an API

At current stage, SocRocket cannot be considered as a drop-in API for modeling AMBA components. It is recommended to evolve it in this direction, in order to have another layer on top of SystemC, TLM, GreenRegs nad GreenSocks, and that the modeling of IP-Cores models would be a clear access to a lower layer of abstraction. Despite it is our opinion that such an evolution is not effort-free and it also involves the rework of all the IP-Cores models in the database, the development of a new model currently leaves the user of the framework to deal with too many internal quirks, and a clearer cut would be preferable.

Currently each model implements its own set of functions for reading/writing from AMBA AHB BUS overriding the ones defined in the base classes. A possible solution that may be explored and that is likely to reduce the size of each component, is to define a reasonable set of default arguments and to expand the ahbread / ahbwrite functions of the base classes to encompass all the functionalities found to be missing by each model.

6.4.4 Read access prevention for GreenRegs

There is currently not a possible way to prevent reading from a defined set of registers defined using the GreenRegs library. This feature would be a nice addition to a library that already provides quite a large plethora of functionalities.

6.4.5 Burst modeling

Another possible improvement is to add functions to model burst accesses to the BUS, which is currently not possible. TLM defines a set of settings to model this kind of bus usage, but SocRocket does not currently exploit this possibility.

6.4.6 Split and Retry

The current version of the SoCRocket does not support Split nor Retry modelling. The AHBCtrl is not re-arbitrating when receiving a SPLIT signal and does not even consider the retry phase. As AHBMaster, AHBSlave and AHBCtrl effectively implement a protocol careful must be taken when modifying or adding something.

Moreover the framework does not provide a simple way to set phases. The only available way is to rewrite entirely the transport functions (which implement the protocol) making the AHBMaster and AHBSlave classes almost useless.

Therefore we think that implementing new phases in SocRocket is something that deserves a proper analysis.

6.5 SocRocket Bugs Fix

6.5.1 AHB Controller Fix

While performing many complete builds, it happened that sometime the tests executed for a complete build were not terminating. The investigation the problem led us to the AHB controller class, in the arbitrate procedure where the program was stuck in a never-ending cycle due to a strange value of data_bus_state member variable. At first, we thought it was a memory corruption problem. This bug was hard to investigate because, as it was reproducible only be executing a complete build, therefore it was time consuming and it was not possible to set watch on variables.

After further investigation, we found out that it was not a memory corruption problem: the data_bus_state was simply not initialized. We fixed this issue by adding the initialization of the state variable in the AHB controller constructor.