



Distributed Real-time Architecture for Mixed Criticality Systems

*Prototype implementation of simulation
framework for DREAMS architecture
D5.2.2*

Project Acronym	DREAMS	Grant Agreement Number	FP7-ICT-2013.3.4-610640	
Document Version	1.0	Date	2015-11-30	Deliverable No. 5.2.2
Contact Person	Mohammed Abuteir	Organisation	USIEGEN	
Phone	+49 271 740 2546	E-Mail	mohammed.abuteir@uni-siegen.de	

Contributors

Name	Partner
Mohammed Abuteir	USIEGEN
Zaher Owda	USIEGEN
Hamidreza Ahmadian	USIEGEN
Marcello COPPOLA	ST
Jörn Migge	RTAW
Tiziana Mastroti	RTAW
Lionel Havet	RTAW
Manuel Muñoz	FENTISS
Javier Coronel	FENTISS
Miltos Grammatikakis	TEI
Simon Barner	FORTISS

Table of Contents

Contributors	2
Executive Summary	6
1 Introduction.....	7
1.1 Relationship to other DREAMS Deliverables	7
1.2 Positioning of the Deliverable in the Project.....	7
1.3 Structure of the deliverable	7
2 System Model of the Virtual Platform.....	9
3 Implementation of Simulation Building Blocks for Virtual Platform	11
3.1 Implementation of the Application Core	11
3.2 Implementation of the Simulated LRS.....	11
3.3 Implementation of the GEM5 STNoC Instance	12
3.4 Implementation of the Simulated Memory Gateway	13
3.5 Implementation of the Simulated Gateway	14
3.5.1 Simulation Queue Elements	15
3.5.2 Bridge Layer	16
3.5.3 Serialization Layer.....	16
3.5.4 Local Controller	16
4 Implementation of Simulation Building Blocks for Execution Environment of Virtual Platform ..	18
4.1 Implementation of the Simulated Hardware Components.....	18
4.1.1 Initial Platform Configuration	19
4.1.2 Initial Execution Control	20
4.2 Simulation of the Software components: DRAL and Partitions.	20
5 Implementation of Coordination Components for Simulation Tools.....	21
5.1 Co-simulation of OPNET & GEM5.....	21
5.1.1 Co-simulation Architecture	21
5.1.2 Co-simulation Coordination	23
5.1.3 Co-simulation Coordination at Chip Level	25
5.1.4 Co-simulation Coordination at Cluster Level.....	27
5.2 Co-simulation of OVPSIM & GEM5.....	29
5.2.1 Co-simulation Coordination at Gem5 level	29
5.2.2 Co-simulation Coordination at OVPSim/Hypervisor level	30
5.2.3 GEM5 OVPSim Message definition.....	33
5.2.4 OVPSim Platform Validation.....	34
6 Implementation of Model-Driven Configuration of Simulation.....	35
6.1 Configuration example: on-chip/off-chip communication.....	35
6.2 Configuration Tools for the Cluster Level.....	39
6.2.1 Specification of configuration file formats with examples.....	39

6.2.2	DREAMS meta-model correspondences	43
6.3	Configuration Tools for the Chip Level	49
6.3.1	Specification of configuration file formats	49
6.3.2	DREAMS meta-model correspondences	56
6.4	Configuration Tools for the Execution Level	65
6.4.1	Xtratum.....	65
7	Analysis of Simulation Traces	66
7.1	Trace files.....	66
7.1.1	Off-chip network related events	66
7.1.2	On-chip network related events.....	68
7.2	Import of DREAMS System description	71
7.3	Import of virtual platform traces.....	74
7.4	Visualization of delays synthesized from the traces	76
7.4.1	Delay tables	76
7.4.2	Delay histograms	78
7.4.3	Delay graphs	80
7.4.4	Gantt charts.....	81
8	Formal Verification Framework.....	85
8.1	Formal Verification Methodology	85
8.2	Implementation of the Formal Verification Methodology.....	85
8.2.1	DS Monitor:	87
8.2.2	US Monitor:	87
8.2.3	AL scoreboard:.....	88
8.3	Result.....	91
8.3.1	Compute farm and formal tool licenses	91
8.3.2	Proof engines.....	91
8.3.3	Run times.....	91
8.3.4	Code coverage results	93
8.3.5	Conclusion	94
9	Bibliography.....	95

Table of Figures

Figure 1: Simulation Model of Virtual Platform	9
Figure 2 Simulated Network-on-Chip.....	12
Figure 3 Functional Architecture of the LRS.....	12
Figure 4: Gateway simulation Building Block	15
Figure 5: On/Off--chip Gateway Class Diagram.....	17
Figure 6: Simulated Platform Initial Configuration.....	19
Figure 7: Initial Configuration Execution Control Flow Diagram.....	20
Figure 8: On-chip/ off-chip Co-simulation framework	21
Figure 9: State Machine of the Local Controller.....	23
Figure 10: Local Controller Building Blocks in Gem5.....	25
Figure 11: Local Controller Building Blocks for OPNET.....	27
Figure 12: OVPSim/Gem5 co-simulation state machine	29
Figure 13: Integration OVPSim Platform with GEM5.	31
Figure 14: Modules involved in the integration of simulators.	32
Figure 15: Execution Control Flow Diagram.....	33
Figure 16: Inputs and outputs of the model to text configuration file generator	35
Figure 17: Schematic overview.....	36
Figure 18: Illustration of off-chip network related events to be traced	66
Figure 19: Illustration of on-chip network related events to be traced.....	69

Executive Summary

This deliverable describes the implementation of the DREAMS virtual platform, a simulation framework for the DREAMS architecture. The virtual platform consists of simulation building blocks for cluster- and chip level, implemented in accordance with the descriptions provided in the DREAMS deliverables D5.2.1 *Specification of simulation framework* and D1.2.1 *DREAMS architectural style*.

Configuration tools are implemented for the virtual platform allowing automatic creation of simulator configuration files from a given system description in order to avoid error-prone manual editing. Dedicated tools furthermore allow the visualization of simulation traces and delay statistics.

Furthermore, the development of a formal verification methodology is presented. The formal verification is used to validate the hardware components implemented WP2. This formal verification is focusing the STNoC and has turned out to be an effective approach for the verification of a given STNoC structure.

1 Introduction

This deliverable is dedicated to the implementation of the simulation building blocks of the DREAMS virtual platform. Different simulation building blocks are implemented to comply with the architectural style. The integration of these blocks induces the integration of different simulation tools to ensure the efficient transfer of data and control between co-running simulations.

In order to seamlessly integrate the virtual platform into the DREAMS tool chain, a model-driven configuration file generator is implemented. Its goal is the automatic creation of simulator configuration files from a given system description in order to avoid error-prone manual editing. Furthermore, to ease the analysis of simulation results, high-level properties such as end-to-end delays are synthesized out of simulation traces.

Furthermore, the implementation of a formal verification framework for protocol verification via a process algebra language and a modelling and a model checking tool for verification of temporal logic properties of the STNoC is presented.

1.1 Relationship to other DREAMS Deliverables

The architectural style document D1.2.1 and the specification of the simulation framework document D5.2.1 serve as primary source of input for the prototype implementation of the simulation framework for the DREAMS architecture.

The meta-model (Tasks T1.4 & T 1.6 of WP1) serves as main input for the model-driven configuration of the simulation building blocks.

The formal verification framework methodology is applied to verify the STNoC hardware components implemented within WP2.

1.2 Positioning of the Deliverable in the Project

The goal of task T5.2 - Simulation, verification and fault-injection framework - is to provide a framework for simulating and verifying the behavior of a mixed-criticality system based on the DREAMS architecture. To achieve this goal, the task T5.2 is divided into three deliverables: D5.2.1, D5.2.2 and D5.2.3.

- D5.2.1 Specification of simulation framework
- D5.2.2 Prototype implementation of simulation framework for DREAMS architecture
- D5.2.3 Fault injection framework

This document is the second deliverable of the WP5 task T5.2. The dissemination level of this deliverable is public (PU) i.e. once approved by the European Commission (EC), it will be freely available for download through the DREAMS project website (<http://dreams-project.eu>).

1.3 Structure of the deliverable

The remainder of this deliverable is structured into three main parts dedicated to the virtual platform (sections 2, 3, 4, 5), the configuration tools and trace (sections 6, 7) and the validation and formal verification framework (section 8)

The system model of the virtual platform is presented in section 2. The implementation details for the on chip simulation DREAMS chip as well as Dreams gateway are explained in section 3. Section 4 describes the implementation of the simulation environment of the hypervisor and section 5 details the integration of the different simulation tools.

Section 6 covers the detailed specification of the virtual platform configuration file generator as well as the mapping between the DREAMS meta-model entities and configuration file entities. Section 7 describes the functionalities implemented for the visualization of simulation traces and the delay statistics derived from the traces for verification purposes.

Finally, section 8 provides the description of a formal verification framework for DREAMS, where a new methodology to verify the robustness against SEUs of a DREAMS Chip instance is introduced.

2 System Model of the Virtual Platform

The simulation model of the virtual platform as depicted in Figure 1 consists of a mixed criticality multi-core Network on a chip (NoC), Time Triggered Ethernet (TTEthernet) end-systems and TTEthernet switches. The mixed criticality multi-core NoC is implemented using the Gem5 simulation tool, while the TTEthernet system is implemented on basis of the OPNET simulation tool.

Therefore, the integration of these simulation tools requires the presence of communication and coordination interfaces. Furthermore, the coordination of these simulation tools requires the synchronization of the simulation steps since the simulation step on one simulation tool may depend on the result of the other simulation tool.

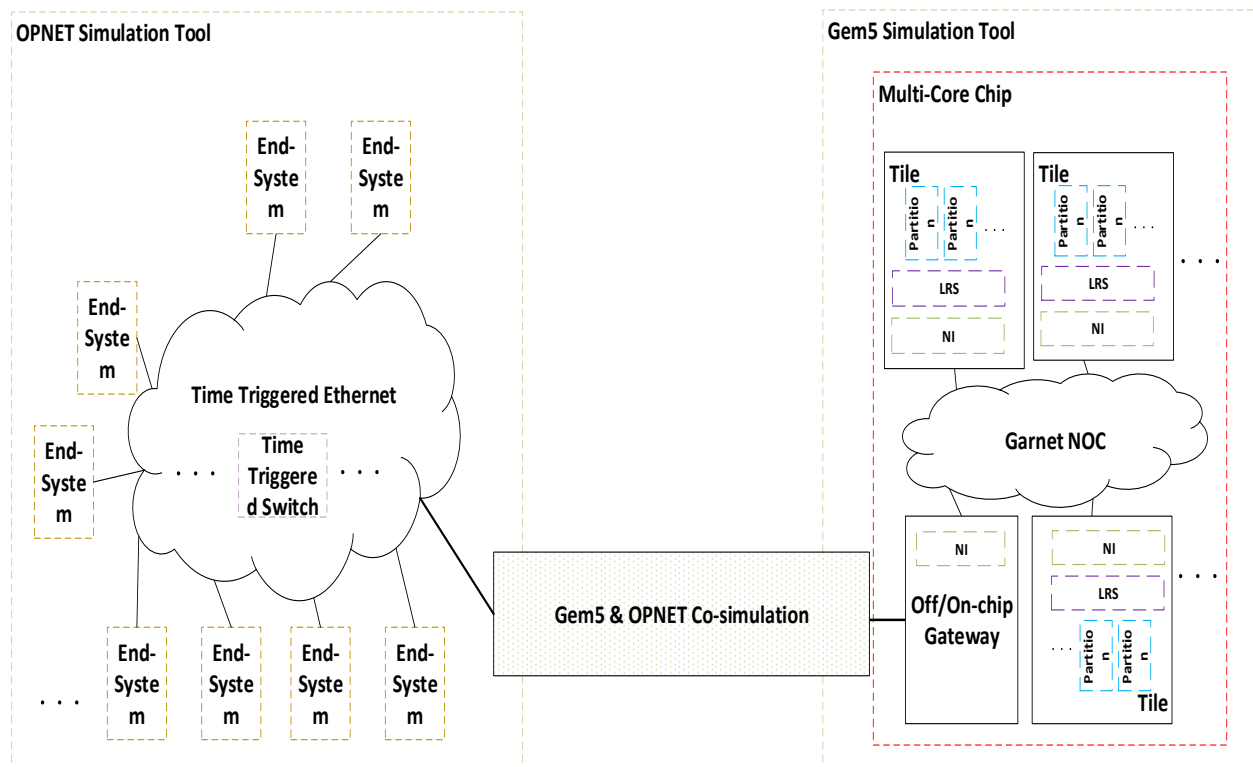


Figure 1: Simulation Model of Virtual Platform

In our system model, we distinguish three types of criticality according to traffic types:

- Time-triggered for periodic messages: They are temporally defined by a period and phase with respect to a global time base and they have highest priority.
- Rate-constrained for sporadic messages: Sporadic messages represent rate-constrained communication with minimum interarrival times between successive message instances.
- Best-effort for aperiodic messages: They have no timing constraints on successive message instances and no guarantees with respect to the delivery and the incurred delays and they have the lowest priority.

A description of the simulation environment for the mixed criticality multi-core NoC and TTEthernet system is provided in the following chapters, including a description of the integration of the Gem5 and OPNET simulation tools.

3 Implementation of Simulation Building Blocks for Virtual Platform

Each node is a multi-core chip containing tiles that are interconnected by a NoC. The tile consists of one or several partitions where each partition executes the application tasks, a Local Resource Scheduler, and a network interface. According to application and architecture requirements, the NoC has a corresponding topology for interconnecting the tiles and the on-chip routers (e.g., mesh, torus, folded torus, hypercube, octagon).

The LRS is responsible for providing the timing guarantees (e.g., bounded latency and jitter, guaranteed bandwidth) and the integrity of messages sent by other tiles using the time and space partitioning.

Each multi-core NoC has an on-chip/off-chip gateway. The on-chip/ off-chip gateways are used to establish the end-to-end communication over heterogeneous mixed-criticality networks. The connection between off-chip and on-chip networks is established through gateways. An on-chip/off-chip gateway is responsible for the redirection of messages between the NoC and the off-chip communication network. Additionally, the on-chip/off-chip gateway provides a solution for mixed-criticality systems with different timing models, fault isolation and real-time guarantees.

Both the LRS and gateway components are integrated into a NoC model, represented by the GEM5 STNoC model. The LRS is realized as a time-triggered extension layer on top of the on-chip network interface, while the gateway is realized as a tile connected to the others tiles through the NoC. The NoC simulation model implements a commercial cycle-approximate STNoC instance on top of the fixed pipeline GARNET interconnection network which allows a configurable network topology. GEM5 allows to configure different connected tiles (e.g., CPU, memory controller, L2 cache controller, etc.).

3.1 Implementation of the Application Core

The application running on the core is simulated using a single file emulating the message sources and sinks. This simulation module performs two main tasks:

- At the source core, it reads the parameters given by the trace file and injects the messages based on those parameters.
- At the destination core, it reads the messages arrived at the ports and logs the statistics (e.g., the arrival time, the message ID, etc.).

In case of co-simulation platforms, the application core can be substituted by the local controller which acts as an interface between two simulation environments.

3.2 Implementation of the Simulated LRS

The LRS is realized as a time-triggered extension layer for the on-chip network interface. The LRS is located between the application core and the NI of the simulated NoC and controls the incoming traffic of the injected messages by the application layer (see Figure 2). At the destination side, the LRS receives the messages from the on-chip NI and stores them to be retrieved by the cores.

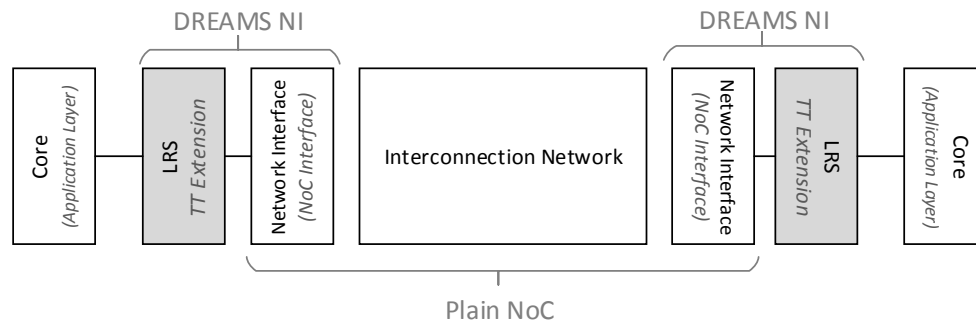


Figure 2 Simulated Network-on-Chip

Figure 3 represents the LRS functional architecture. The LRS serves as an egress/ingress point to/from the NoC that enhances services available at the NoC network interface by supporting offline scheduling, routing and traffic shaping of resource requests queued at the Priority Queues Unit (cf. DREAMS D2.1.2 and D2.1.3).

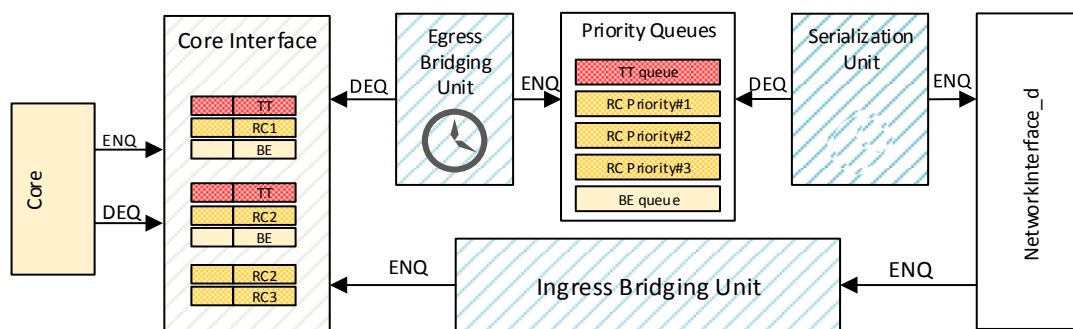


Figure 3 Functional Architecture of the LRS

As shown in Figure 3, the LRS is composed of the Core Interface, Egress Bridging Unit (EBU), the Priority Queues Unit, the Serialization Unit (SU) and the Ingress Bridging Unit (IBU).

The Priority Queues Unit provides an interface between the EBU and the SU, while the Core Interface establishes a management and configuration interface that handles message access control between the cores and the bridging units (i.e., EBU and IBU layers). The egress bridging unit (EBU) ensures timely dequeuing of TT ports, dequeuing of RC ports only when the pre-specified minimum inter-arrival time (MINT) is elapsed, and dequeuing of BE ports only if there is enough bandwidth left over by other two types of messages. Before enqueueing the TT and RC messages into the Priority Queues Unit, the EBU retrieves the type and destination of the message from the configuration parameters (e.g. period, phase, MINT) and embeds them into the message.

The LRS injects the messages into the NoC using the NIs. The connection between the LRS and the NI is done by a FIFO.

3.3 Implementation of the GEM5 STNoC Instance

The Gem5 STNoC backbone model is a wormhole network with credit-based link level flow control which represents an cycle-approximate instance of the commercial STNoC network-on-chip. The model extends the 5-stage Garnet Fixed Pipeline Model provided by the Gem5 simulator (see deliverables D5.2.1 and especially D2.1.1). In fact, most of the Garnet network

interface, router and link data structures have been modified, as described in detail in deliverable 2.1.1.

More specifically, in relation to the Garnet router, we have reduced pipeline depth to match an instance of the STNoC router architecture which processes a packet in two clock cycles (instances with cycle delays of one or zero cycles have not been modeled). In this instance, one cycle is spent for input buffering and route computation stages (Buffer Write and Route Compute, called IB+RC) and one more cycle is spent to perform all four other stages in Garnet: virtual channel arbitration, switch allocation, switch traversal and link traversal (VA+SA+ST+LT). More specifically, in our implementation we have encapsulated the Switch Allocator (stage 3 in Figure 6) within the VC Allocator (stage2). For this reason, changes have been made to the event scheduling infrastructure of all Gem5 router components (as outlined in sections 2.2 and 2.3 of Deliverable 2.1.1).

The Garnet Fixed Pipeline link has been modified to match the STNoC synchronous link which incurs zero cycles delay when “carrying” data or credit flits between two connected routers or between router and NI (or vice-versa).

The NI is an extension of Garnet Fixed Pipeline model that connects an IP to a router. This module

- checks the protocol buffer (CPU) for packets ready to be sent to the NoC. If the downstream output VC associated with this packet has credits left, then
 - it segments (called flitsize) any such outgoing packet into a number of flits,
 - places them into an output buffer, and
 - schedules the output link for the next cycle
- checks the input buffer for incoming flits to be sent to the attached IP and upon receiving a tail flit, the incoming packet is reassembled from the flits and inserted into the protocol buffer,
- updates credits sent by the downstream router; the credit round trip delay for the synchronous link is 5 cycles, whereas this delay refers to the round trip router-to-router delay for updating the credit of a departing packet.
- inserts rate control info for implementing NoC Firewall, and
- inserts QoS flags for supporting STNoC QoS technology (FBA) and memory interleaving.

3.4 Implementation of the Simulated Memory Gateway

The gem5 simulator is able to be executed in two modes, System Call emulation (SE) and Full System (FS). In SE mode, a statically compiled binary file (no dynamic linking to other executable files) is loaded to and the Gem5 simulator provides program execution services. In FS mode, Gem5 can simulate a real system, including operating system (e.g. Linux and Android) and hardware devices. System services, including shell commands, compiler and debugging tools, are available by connecting to a simulated console via a telnet application.

Gem5 simulator provides a clear representation and methodology that provides visibility for exploring the content of memory and register structures through full system simulation, making it easier to evaluate NoC QoS mechanisms, such as LRS, bandwidth regulation using MemGuard and Traffic Shaper developed in DREAMS WP2. It can furthermore be used to consider fault diagnosis, isolation and repair techniques. Thus, during the early design stages, i.e. far before the hardware is ready, Gem5 can be used as an alternative to support

preliminary (bare metal and linux) system driver/software development and verification in the controlled environment of virtual hardware.

Memory and interconnect structure in Gem5 contains two kinds of memory models: Classic and Ruby.

- The Classic memory system provides a fast and easily configurable Gem5 memory system, however, at the cost of accuracy and flexibility. All objects inherit functions from a common ancestor class MemObject. The classic memory model supports connections to memory objects through ports, for example, connecting CPUs to caches, caches to busses, and busses to devices and memories. Ports support three mechanisms for accessing data (functional, atomic, and timing) and an appropriate interface for configuring the topology and debugging. The classic memory model offers simulation efficiency and convenient configuration, but also limited cache coherence support and accuracy.
- Ruby memory system sacrifices simulation speed to provide a flexible Gem5 memory and interconnect infrastructure capable of accurately simulating a wide variety of memory systems. It uses ports to connect to devices and message buffers to internally connect to other Ruby objects. Its advantages are cache coherence support and model accuracy, at the expense of simulation speed and flexibility.

With the classic memory system, Gem5 can model an ARM uni-or multi-processor running Linux or Android in FS mode. In SE mode, statically linked (bare metal) applications are built with ARM compilers. ARM architecture models within Gem5 support an ARMv7-a profile of the ARM architecture with multi-processor extensions. This includes support for Thumb, Thumb-2, VFPv3 (32 double register variant) and NEON. Optional features of the architecture that are not currently supported are TrustZone, ThumbEE, Jazelle, Virtualization and Large Physical Address Extensions (LPAE). Moreover, as pointed out in Deliverable D2.1.1, there is limited, uniprocessor-only support of Ruby memory system, implying the use of co-simulation with instruction set simulators (ISS) for cycle-accurate simulation of ARM-based multicore systems.

We have used both memory models in simulations. For example, we have used Ruby memory controller to examine relative performance benefits when multiple DMA initiators access different memories and address interleaving is enabled. Ruby memory controller provides a detailed system configuration parameter space (number of banks, number of ranks, delays, tFAW, refresh time and other parameters), making it possible to explore in detail performance and power tradeoffs of different NoC QoS mechanisms, especially MemGuard, at the memory controller. In this context, TEI plans to focus on the relationship between existing STNoC QoS and proposed MemGuard access control policies implemented at the memory controller (see WP2, Deliverable D2.1.2).

3.5 Implementation of the Simulated Gateway

The structure of the simulation building block of the gateway is illustrated in Figure 4. The constituting layers of the gateway are the bridge, serialization, local controller as well as ingress, egress and VL queues.

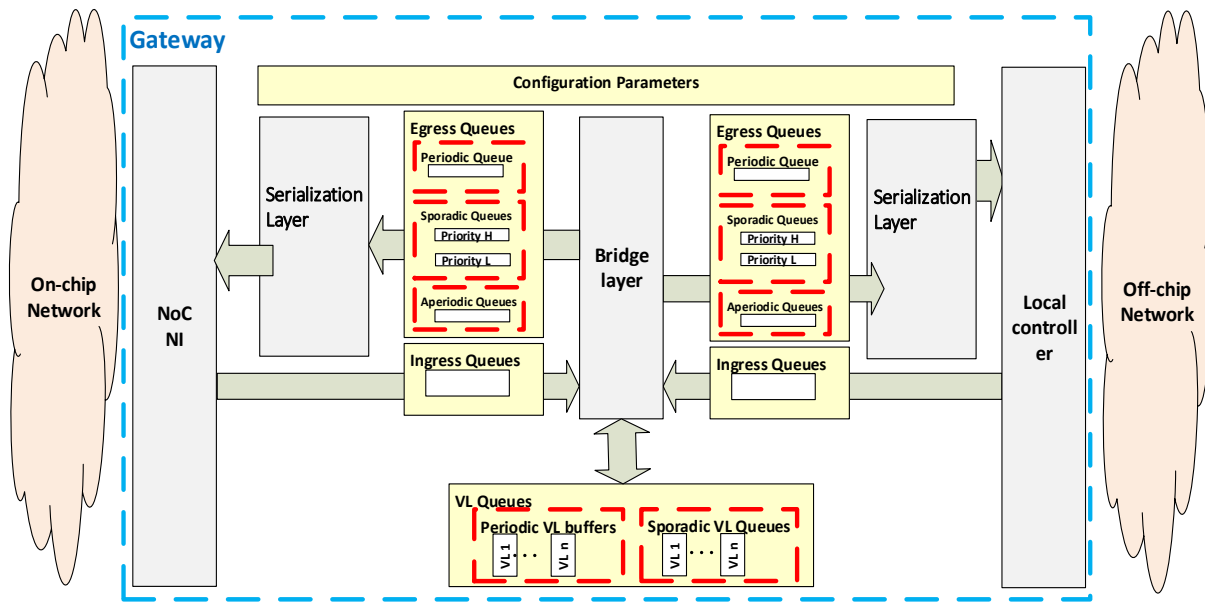


Figure 4: Gateway simulation Building Block

The cache coherence protocol “Network test” (1) was extended to allow the gateway to interact with the GARNET interconnection network. In what follows, a description of the simulation building block of the gateway is provided.

3.5.1 Simulation Queue Elements

The gateway has three types of queues, as follows:

- Ingress queue: It consists of one FIFO queue for each network. The ingress queue that connects the NoC interface with the bridge layer is implemented using a description language for specifying cache coherence protocols to establish the connection between the gateway and the GARNET interconnection network. The ingress queue that establishes the connection to the off chip network is implemented using the queue C++ class.
- Egress queues: They consist of one periodic egress queue, two sporadic queues and one aperiodic egress queue. Each sporadic queue has its own priority level. The egress queues are implemented using the queue C++ class.
- VL queues: There belong to two groups: one for the periodic messages and the other one for the sporadic messages.
 - Periodic VL buffers: Each periodic VL has one periodic VL buffer, which provides buffer space for exactly one message. In case this buffer is full and another message arrives with the same VLID, the newer message replaces the old one.
 - Sporadic VL queues: Each sporadic VL has one queue. It is possible to store several messages of the respective VL in this queue.

VL queues are implemented using the queue C++ class.

¹ http://www.m5sim.org/Network_test

3.5.2 Bridge Layer

The bridge layer classifies the incoming message based on the message types. The periodic and sporadic messages are stored in the corresponding VL queue. This is done by extracting the virtual link id from the message and comparing it with virtual link id stored in the look-up table. Aperiodic messages are stored in the aperiodic queue of the egress queues.

The bridge layer determines the point in time when the periodic message is relayed from the periodic VL buffers to the periodic queue in the egress queues based on the time parameters (i.e. period and phase). This ensures the deterministic communication behavior of the periodic messages.

On the other hand, the bridge layer guarantees the minimum interarrival time between two consecutive sporadic messages on the respective VL. The minimum interarrival time is part of the configuration parameters for each VL. According to this guarantees, the bridge layer controls the traffic shaping for the sporadic messages and redirects message from sporadic VL queues to one of the sporadic egress queues according to the direction and priority parameters.

The bridge layer is simulated as C++ objected which has multiple functions that ensure the described functionality of the bridge layer (see Figure 5).

3.5.3 Serialization Layer

The serialization layer forwards the messages from the egress queues to the network (off-chip or on-chip) according to the priority. The highest priority is assigned to periodic messages, whereas aperiodic messages have the lowest priority.

In addition, the serialization layer uses either shuffling or timely blocking to resolve contention between different traffic types. The timely block mechanism disables the sending of other messages in the egress queues during a guarding window prior to the transmission of a periodic message. For the shuffling mechanism, no guarding window is needed. In the worst-case, the gateway delays a periodic message for the duration of a sporadic or aperiodic message of maximum size.

The serialization layer is simulated as C++ objected which has multiple functions that ensure the described functionality of the sterilization layer (see Figure 5).

3.5.4 Local Controller

The local controller is responsible for interfacing with the other simulation tools in order to coordinate and communicate the transmission events between the different simulation tools. Additionally, the local controller communicates with other local controllers in the other simulation tools to synchronize the simulation steps since the simulation step on one simulation tool may depend on the result of the other simulation tool. More details are available in section 5.



Figure 5: On/Off--chip Gateway Class Diagram

4 Implementation of Simulation Building Blocks for Execution Environment of Virtual Platform

This section describes the strategy followed when simulating the Execution Environment. This is composed by a set of components such as Hardware components (processor core and memory), the DREAMS virtualization layer, and the set of DREAMS partitions.

A simulator that emulates the hardware components for simulating the execution environment is used as a basis. The software components of the execution environment (the DREAMS virtualization layer, and the set of DREAMS partitions) are then executed by the simulated hardware. This technique has been selected to reduce the efforts in development and maintenance.

Software components of the execution environment do not need any modification and the same software executed on the harmonized platform processor could be executed on the simulated processor. This avoids additional efforts in the adaptation of the software. Also due to this reason all the services provided by the hypervisor are available including extended services available for DREAMS architecture.

Regarding maintenance, this way of simulation provides isolation of the simulated hardware component and the executed software components. This entails that modifications in the software components of the execution environment do not trigger a maintenance operation in the simulator.

Additionally this isolates the application developer from the need of simulator internal knowledge.

4.1 Implementation of the Simulated Hardware Components.

In order to simulate the hardware components of the execution environment (processors and memory) OVPSim has been selected. OVPSim allows the creation of customized virtual hardware platforms that include processor cores, busses, memory and even peripherals. Additionally the simulator provides a huge interface that enables the access and control of the simulated elements. That allows controlling and monitoring the execution. In deliverable D5.2.1, sections 5.2 and 5.3 detail how a platform could be created.

OVPSim is now available on Windows and Linux, and is available for free only for non-commercial use. Commercial users can use the free download for evaluation etc., for commercial use, it is required to contact Imperas and obtain a license to a compatible commercial product. Current OVP models are provided as Open Source under Apache 2.0. A free unlimited license (academic) could be provided by Imperas for universities or academic use. Additionally discounted price unlimited license could be obtained for government or EU projects context.

As a first approximation, an initial version of the platform has been implemented. This version does not develop processor interaction with external elements. This version will allow us to validate the correct operation of the simulation of the hardware components. Additionally could be used to help during the development and validation of the DRAL (DREAMS Virtualization Layer), and the set of DREAMS partitions. The simulation provides a better control about the execution.

The following sections (4.1.1 and 4.1.2) described this simplified version. In a second approach the initial platform is modified to integrate the interaction with the simulator that supports the STNoC network. Such modifications are described in section 5.2.2.

4.1.1 Initial Platform Configuration

In a first approach PS (Processing System) is running with no interaction with PL (Programming Logic).

In that simulated platform the Hypervisor and Partitions could be tested. For this purpose a simplified platform has been developed.

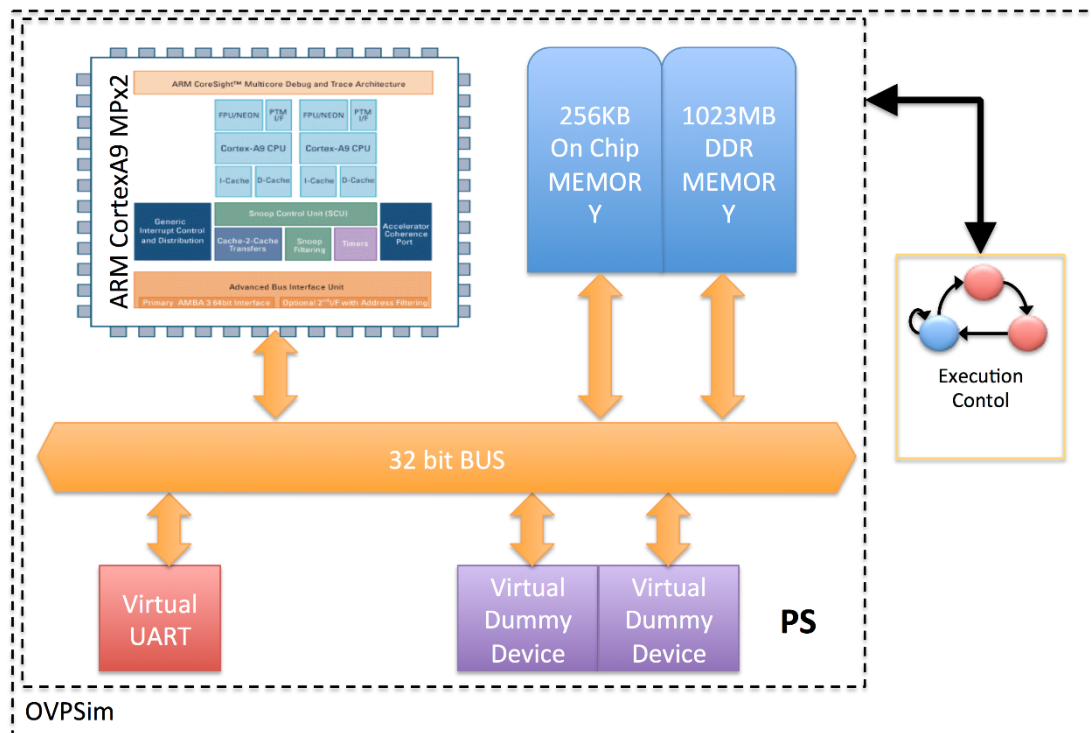


Figure 6: Simulated Platform Initial Configuration.

As shown in the Figure 6: Simulated Platform Initial Configuration., the elements contained in the platform are:

- ARM Cortex-A9-MPx2 Processor Model. [1]
- MMU
- SIMC
- NEON
- VFP
- Security extensions (Trust Zone)
- MPCore GIC
- Global Timer
- Private Timer / WatchDog.
- 32 bit Processor Bus.
- Memory Regions:
 - 256 Kb On Chip Memory mapped in 0x0000_0000-0x0003_FFFF.
 - 1023Mb DDR Memory mapped in 0x0010_0000-0x3FFF_FFFF.

- Virtual UART. Puts UART characters to stdout. Device mapped in 0xE000_1000, 0xE000_1FFF.
- Virtual Dummy devices. These are board devices not needed for simulation purposes. Must be linked to the bus to avoid errors on accesses.

4.1.2 Initial Execution Control.

On the first approach platform once the initialization process is done, the execution control performs a loop that executes, instruction by instruction, the program stored into memory. The platform performs the following actions in each iteration of the loop:

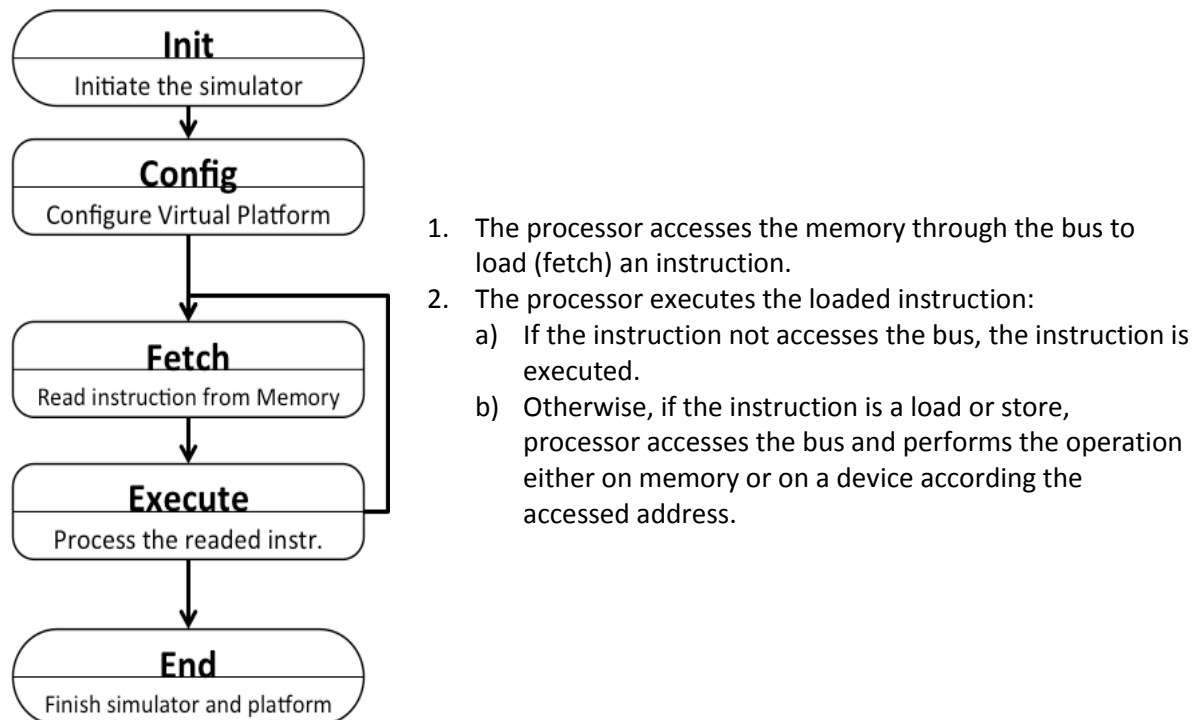


Figure 7: Initial Configuration Execution Control Flow Diagram.

4.2 Simulation of the Software components: DRAL and Partitions.

As is explained previously, software components are not really simulated. Instead, their execution is simulated through the virtual hardware platform.

The hypervisor including DRAL and the partitions are pieces of software. Once their sources are build properly, a runnable image of this software is obtained. This image is downloaded to the board and executed.

In the same way, the generated software could be downloaded to the virtual hardware memory for execution.

In the process of developing and validating these software components, the virtual platform could be very useful. It allows the developer to obtain controlled execution of the software, and debug facilities like traces, memory inspection, etc.

5 Implementation of Coordination Components for Simulation Tools

This chapter introduces the co-simulation approaches that have been developed in order to provide an integrated solution for the DREAMS virtual platform multiple simulators. Section 5.1 presents the co-simulation approach for the on-chip/off-chip simulators. The co-simulation approach that combines the on-chip multicore network-on-a-chip simulation with the Xtratum hypervisor is detailed in subsection 5.2.

5.1 Co-simulation of OPNET & GEM5

The proposed co-simulation framework (cf. Figure 8) consists of one off-chip simulation tool, one on-chip simulation tool and two local controllers that implement a socket-based communication between the simulation tools. The local controllers provide basic gateway functionalities, as described later in this section.

5.1.1 Co-simulation Architecture

The discrete event on-/off-chip simulation tools simulate the communication behavior of the on-/off-chip networks. Each simulation tool operates on a simulation model, which comprises a network of the simulated end-systems or CPUs. Moreover, the simulation tool executes its simulation based on its own local event calendar. The local event calendar contains the tool's local events as well as global events that are used to synchronize the co-simulation based on the co-simulation global calendar. The co-simulation global calendar represents the time model of the execution order of both simulation tools. It contains events that have causal relationship in the co-simulation. Moreover, managing and modifying the events of the local event calendar is mandatory in the selection of the simulation tools that will be executed based on the proposed framework.

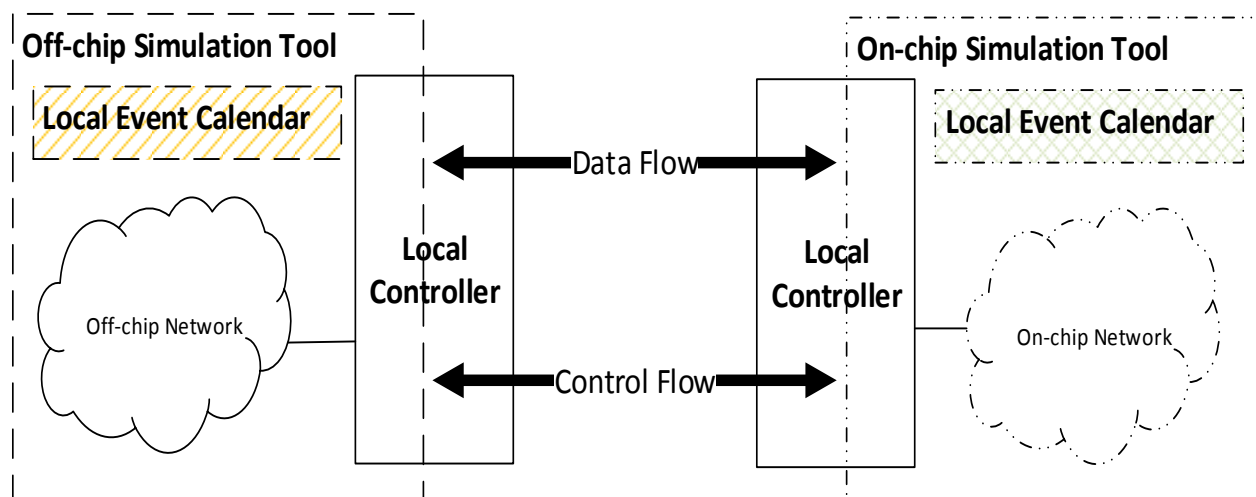


Figure 8: On-chip/ off-chip Co-simulation framework

The local controllers of the proposed co-simulation framework provide a communication interface from/to their networks. Furthermore, the local controllers introduce basic gateway

functionalities such as queuing and mapping of the incoming/outgoing messages. The gateway functionalities are responsible for converting the exchanged message formats between the simulation tools, and mapping the destination addresses of the incoming messages to the target application. These functionalities require queuing structures that handle the transmitted messages.

There are two types of communication messages between the simulation tools, data and control flow messages. The data flow contains the exchanged data message between the off-chip and on-chip networks. The control flow is used to manage the simulations' execution based on the co-simulation global calendar.

The co-simulation of different simulation tools would create inconsistency during the execution, in which a simulation step in one tool can depend on an incoming message from the other simulation tool. The proposed framework uses a socket-based interleaving approach to synchronize simulation steps and provide access for mutual modification of simulation tools calendars as well as the realization of the data exchange between the simulation tools. The socket-based interleaving approach means that simulation tools will run one at a time. In other words, one simulation tool will run for a specific time, and the other simulation tool shall suspend until it is time to run again. This decision is taken based on the simulation events of the calendars of both simulation tools. The socket-based communication of the co-simulation provides an abstraction from the simulation tools' location and their hosting operating system.

5.1.2 Co-simulation Coordination

In this section, the coordination of the local controllers that manage and control the data and control flows of the co-simulation is presented. This procedure is called co-simulation coordination. The co-simulation coordination is responsible for establishing the connection between the simulation tools. The coordination between the simulation tools is based on a client-server socket-based communication in order to establish and manage the communication between the simulation tools. At the co-simulation setup, the socket communication is initiated based on the configuration parameters of the co-simulation.

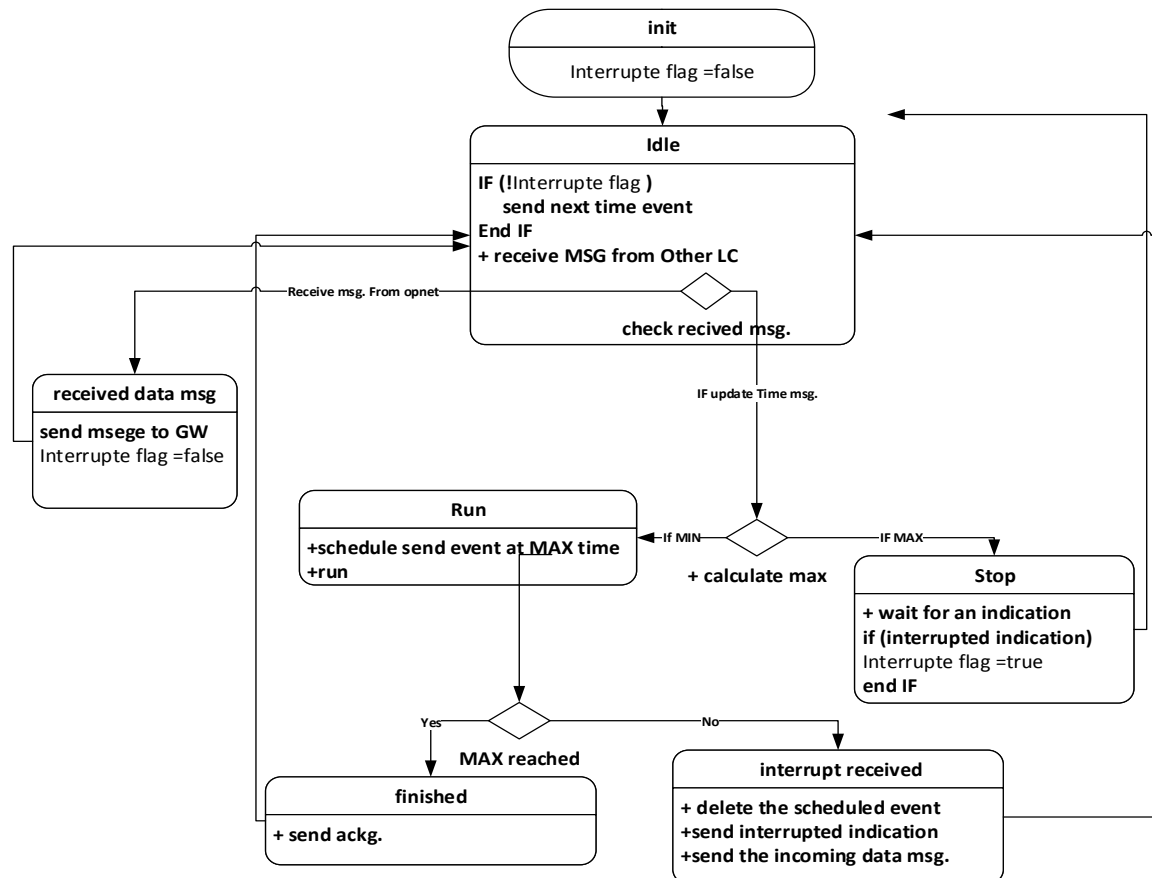


Figure 9: State Machine of the Local Controller

The co-simulation coordination defines a local controller for each of the simulation tools. The local controller serves as the interface between the simulation tool and the other local controllers. It is also responsible for sending data and control messages to the other local controllers. Control messages are used to modify the simulation calendar and its execution based on the overall co-simulation global time to determine which simulation tool should be executed. This decision is based on the calculation of the maximum time of next events of both simulation tools.

Most of multi-core platforms are based on shared memory without any messages for on-chip network simulations. In the proposed framework, message-based communication is assumed for both on-chip and off-chip simulation levels.

The following message types are used to communicate between the local controllers:

- Data message: it represents a message sent from the off-chip network to the on-chip network or vice versa. The data message structure contains the following fields:
 - Message Size: This parameter contains the message payload size.
 - Creation Time: The message creation time is used for statistical purposes and results analysis.
 - Message Sequence Number: The message sequence number is used for statistical purposes and results analysis.
 - Message Deadline: The time limit of a message for the communication from the sender to the receiver.
 - Sender ID: This contains the sender ID.
 - Destination ID: Declares the message destination ID.
- Control messages are defined as follows:
 - Time of next event: Contains the time of the next event according to the simulation tool's local event calendar.
 - Finish acknowledgment: This event is sent to announce the successful completion of a simulation step.
 - Interrupt indication: Simulation tools are given a time interval to execute based on the maximum next event times. This time interval may be interrupted before the end of this interval is reached. This case generates an interrupt indication with a time stamp of the interrupt event.

The behavior of the local controller is illustrated in the state machine in Figure 9. Let us discuss how the co-simulation coordination control works using an example where we have the local controllers LC_A and LC_B

Starting with the init state, the configuration parameters are initialized to establish the socket-based connection. Let us assume that LC_A has its next event time after the next event time of LC_B. In the idle state, each local controller sends the next event time based on its local event calendar. By receiving the control message with the next event time, each local controller compares the incoming next event time with its own next event time. According to our example, LC_A has the maximum next event time. Therefore, LC_A goes into the stop state and it suspends its execution until an acknowledgment message is received from LC_B. Meanwhile the simulation tool of LC_B runs. There are two cases where the simulation tool of LC_B is stopped. The first case occurs if the simulation reaches the next event time of the other simulation tool. In this case, LC_B sends a finish acknowledgement message to LC_A. When LC_A receives the finish acknowledgement, LC_A moves from the stop state to the idle state. Then the whole procedure is repeated.

The second case occurs, if LC_B receives an interrupt from its simulation that requires sending a data message to the other simulation tool. LC_B suspends its simulation tool and sends an interrupt indication to LC_A. Then, LC_B sends the message data. When LC_A receives the interrupt indication event message, LC_A moves from the stop state to the idle state to receive the message data. When the message data has arrived, LC_A adds a new event in the global event calendar of its simulation tool, and it injects the data message into the network simulation. Then the whole procedure is repeated.

5.1.3 Co-simulation Coordination at Chip Level

In this section, the implementation of the co-simulation coordination at Gem5 level and the interactions with the OPNET simulation tool in the co-simulation coordination are described. Moreover, the Gem5 local controller is implemented based on the state machine presented earlier, and it is integrated with the gateway of the DREAMS virtual platform, with means that it is taking advantages of the serialization, ingress and egress bridging layers of the gateway. The building blocks of the Gem5 local controller are illustrated in the following figure.

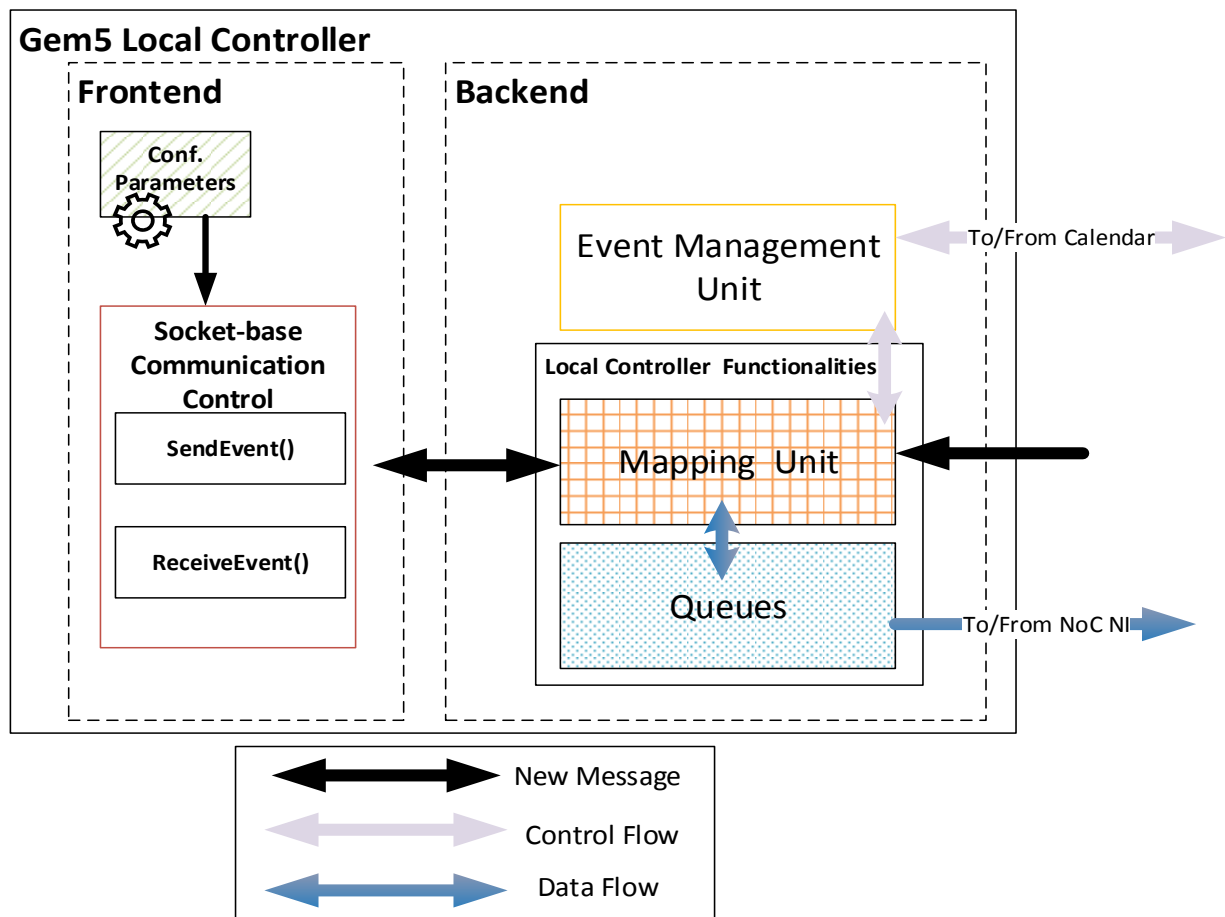


Figure 10: Local Controller Building Blocks in Gem5

It is required that the control of simulation tools execution is possible by modifying the simulation's local event calendar. In Gem5, it is possible to use Python and simulation checkpoints for this purpose, but this method has two disadvantages; In the first place, it would be required to define the simulation checkpoints before hand, which would make the co-simulation process unrealistic. Secondly, modifying the Gem5 local event calendar at run-time is not possible. Python and checkpoints do not allow access from the other simulation tools to the Gem5 simulation. The execution of the co-simulation in the proposed framework resolves these restrictions of Gem5 as illustrated in the following.

Based on the co-simulation model, the messages exchanged between the simulation tools are classified into the control messages that are responsible for guaranteeing the correct execution order of the simulation tools, and data messages that include the user information.

In Gem5, the simulation events are categorized into global events that are related to the overall simulation behavior (e.g., exit, run, checkpoint) and local events.

The Gem5 local event calendar contains two types of events queues; one main global events queue and object queues. Each object in Gem5 has its own local event queue for its local events. Control events of the co-simulation framework are implemented as Gem5 global events.

The Local Controller in Gem5 consists of a frontend that is responsible for the socket-based communication and a backend that is acting as an interface to the on-chip network. The local controller involves basic gateway functionalities and an event management unit.

The frontend part is similar in both co-simulation local controllers. The local controller in Gem5 is realized as a client and the local controller at the OPNET side acts as a server. The frontend uses the socket-based configuration parameters (i.e. simulation tool ID, IP address, port number) to setup the socket-based communication from the Gem5 side. The functions for sending and receiving messages are defined as part of the frontend. The blocking socket-based communication is controlled based on the execution steps. This blocking mechanism provides the capability to suspend and resume the Gem5 simulation controlled by the exchanged messages between the simulation tools of the co-simulation. In this way, the Gem5 simulation tool is regularly suspended and the modification of the simulation calendar is possible during the co-simulation.

The backend serves for the communication from one simulation tool to another one from an implementation point of view. It implements the functionalities for handling the data and control events using the local controller functionalities and the event management unit.

The Event Management Unit (EMU) of the local controller is responsible for managing existing and new simulation events of the Gem5 local event calendar. This unit handles the control events accordingly.

At the beginning of each iteration, the next execution order is determined locally. In case Gem5 is not next in the execution order, the EMU requests to suspend the Gem5 simulation by redirecting the control to the socket-based communication control, which will suspend the Gem5 execution using the socket-based blocking receive function. However, if Gem5 has the execution control, it schedules a new execution control event that will run the Gem5 simulation until the earlier determined time. Later it sends the finish acknowledgement event message to the Mapping Unit (MU) to declare the termination of this execution iteration. The MU forms the newly created control message and sends it to the socket-based communication control.

If the local controller receives an interrupt indication event from Gem5, the MU analyzes the interrupt event and queues the data message. Thereafter, the MU sends the control message to the EMU. Correspondingly the EMU deletes the latest scheduled execution control event, and then it forms the interrupt event including the interruption time. The MU dequeues the data message to form the event. Last, the formed event is sent using the socket-based communication control.

In case an interrupt is received from the other simulation tool, the MU analyzes the received message. The MU sends the control message to the EMU, and the data message is queued by the local controller functionalities. The EMU creates a new control event in the Gem5 local event calendar, which will allow Gem5 to execute until the newly received time-stamp from the other simulation tool. The queued data message is used at the resumption of the Gem5 simulation.

5.1.4 Co-simulation Coordination at Cluster Level

OPNET provides an API to interact with external systems such as other simulation tools, software programs, hardware devices or humans during simulation. The following figure shows the principle of co-simulation in OPNET using the local controller's building blocks.

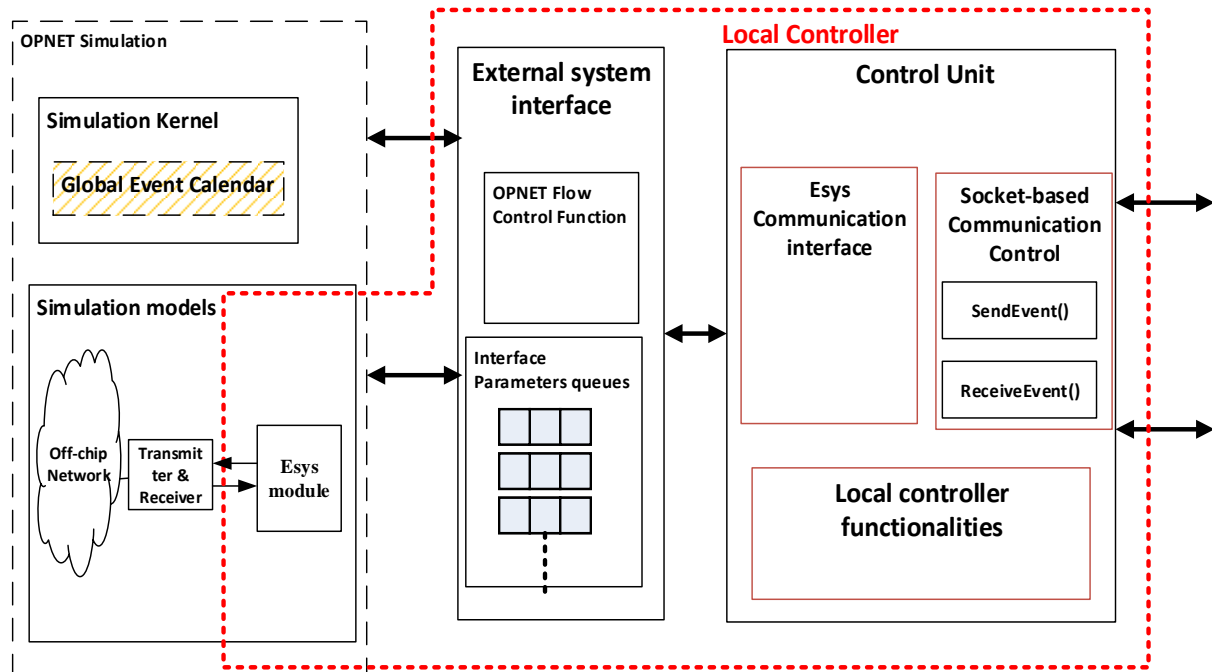


Figure 11: Local Controller Building Blocks for OPNET

This co-simulation consists of several components:

- **Simulator Description File (SDF):** OPNET uses the SDF configuration to define the platform, kernel, bit-size, the list of object files to bind and libraries that are required by the local controller during co-simulation.
- **External system:** In our case, it is the control unit, which represents the external code that is co-simulated with OPNET.
- **External System Definition (ESD):** It defines the interfaces that allow OPNET to communicate with an external system. These interfaces can be read or written by both OPNET and the control unit.
- **Esys module:** This module allows the user's simulation model to interface with the control unit through the external system interface.
- **External Simulation Access (ESA) API package:** The ESA API package contains functions that can be used as a OPNET flow control function, for interface access and input/output. The flow control function is used to setup the simulation time, process events and deal with the simulation time keeping. The interface access functions are used to communicate between internal end-systems and the co-simulation's control unit. The input/output functions deal with reading and writing the interfaces' values defined in the ESD model from process models during co-simulation.

The local controller in OPNET consists of the Esys module, the external system interface and the control unit. The first step in the implementation of the local controller is the definition

of the interface between the Esys module and the control unit using the ESD. This interface is realized by a message containing the follow information: application ID, message size, creation time and sequence number. In the external system interface, the ESD provides queues that can be read or written by the Esys model and the control unit.

The local controller processes the incoming message to the OPNET local controller. The local controller uses the Esys communication interface to OPNET to send either data messages that include the interface parameters or control messages that include the control events.

When the message contents are enqueued in the external system interface, the external system interface creates an event in the global event calendar at the injection time. This event interrupts the Esys module at the injected time. The Esys module reads the interface parameters from the queues and creates the message based on these parameters. Then, the Esys module sends the created message to the destination end-system.

Whenever the control message arrives to the external system interface, the flow control function analyses the message. The control message is either an execute until, stop, suspend or next time event request. The flow control function creates an event in the global event calendar based on the incoming message or sends the information to the control unit about the next event time.

When a message from the off-chip network arrives at the Esys module, the Esys module extracts the interface parameters from the message and puts them in the queues. The external system interface stops the simulation and informs the control unit that there is a new data message.

5.2 Co-simulation of OVPSIM & GEM5

The overall approach of the Gem5/OVPSim co-simulation is based on the interleaving tick-based simulation presented in D5.2.1. In this deliverable, the low-level building blocks description of the co-simulation is presented. The duration of the defined tick is defined by the OVPSim-MIPS execution since its execution cycles are longer than the once required by the Gem5. For this co-simulation setup, the Gem5 on-chip simulator will be the server of the blocking socket-based co-simulation. The initialization time of each of the simulators differ, therefore the simulators have to exchange their readiness for co-simulation as soon as the initial phase is done. The server (in our case, Gem5) is the one responsible for starting the co-simulation.

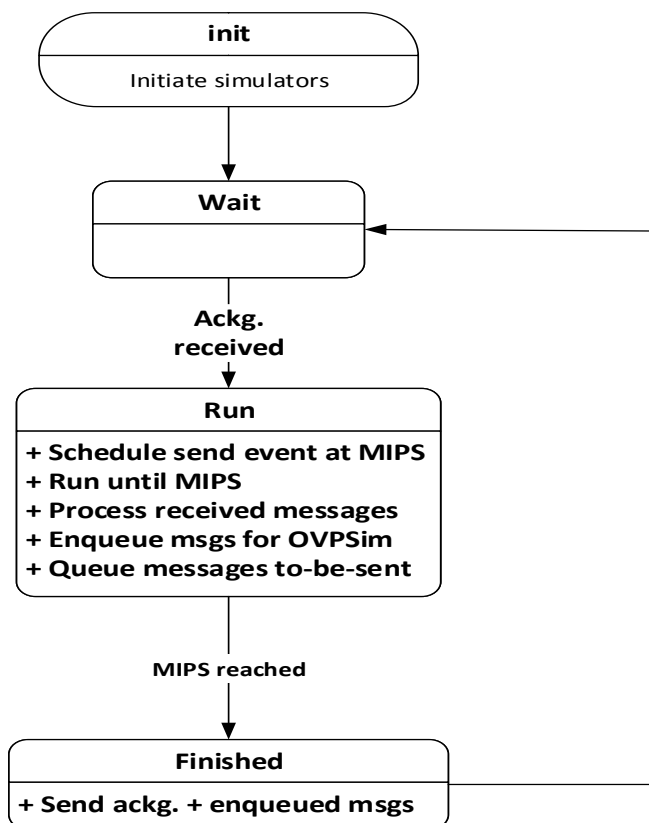


Figure 12: OVPSim/Gem5 co-simulation state machine

The concept of the co-simulation here is similar to the generic co-simulation architecture discussed earlier (cf. subsection 5.1.1), so one would expect two local controllers, one for each of the simulator tools as discussed in the next two subsections.

5.2.1 Co-simulation Coordination at Gem5 level

The local controller required for Gem5 in this co-simulation has the same structure as presented earlier. The only difference is the integration of the local controller directly to the network interface and it is talking directly to the LRS, which means that the OVPSim simulator is agnostic about the on-chip network and its virtual links. Therefore, a static configuration input is required to provide the ports mapping to the mapping unit of the local controller. The following configuration table is an example of such ports mapping configurations.

Table 1: Ports mapping configuration table

Port ID	Traffic Type	Memory	Size	Virtual Link ID
1	TT	0x0000000	64B	1
2	RC	0x0001000	2KB	1
..
N-1	RC	0x001D000	3KB	20
N	RC	0x0020000	1KB	20

There are two types of control messages used in this co-simulation:

- **Run:** This event is sent once by each of the simulators after the finishing of their initialization phase to indicate that they are ready for to start the co-simulation.
- **Finish acknowledgment:** This event is sent to announce the successful completion of a simulation step.

Data messages exchanged between the co-simulators are mainly the payload of the message, and the local controller of the Gem5 simulator is responsible of the correct mapping and addressing of the messages. This is achieved with the help of the LRS provided in the network interface (cf. Figure 13). The detailed content of the messages is explained in section 5.2.3.

Simulators will run one per time while the other simulator will be stopped and waiting for the finish acknowledgment of the running simulator in addition to the buffered messages. The execution flow from Gem5 point of view is shown in Figure 12. Starting with the initiation face, the Gem5 simulator will wait until it receives the acknowledgement control message and the enqueued data messages of the OPVSim previous execution. In case there are data messages that were addressed to the Gem5 on-chip simulation, Gem5 starts its run state with scheduling a send event at the next MIPS using the events management unit, and executes until this MIPS. Meanwhile, it starts processing the buffered data messages by sending them to the corresponding ports with the help of the local controller's mapping unit. Moreover, messages distanced to the hypervisor will be buffered and sent at the end of the execution tick of the Gem5 simulator.

5.2.2 Co-simulation Coordination at OVPSim/Hypervisor level

Based on the first approach described in section 4.1, where the hypervisor is running over the simulated PS without PL, a new platform is implemented. The extension allows the interaction with the PL where is placed the STNoC in the real HP. To perform the emulation of this interaction the OVPSim simulator and GEM5 need to be integrated. Some elements therefore need to be added to the initial proposal as shown in the Figure 13:

- Programming Logic Virtual Device
- OVPSim Local Controller
- Execution Control Module (Adaptation)

These elements and their internal modules are depicted in the Figure 14 and described in the following subsections.

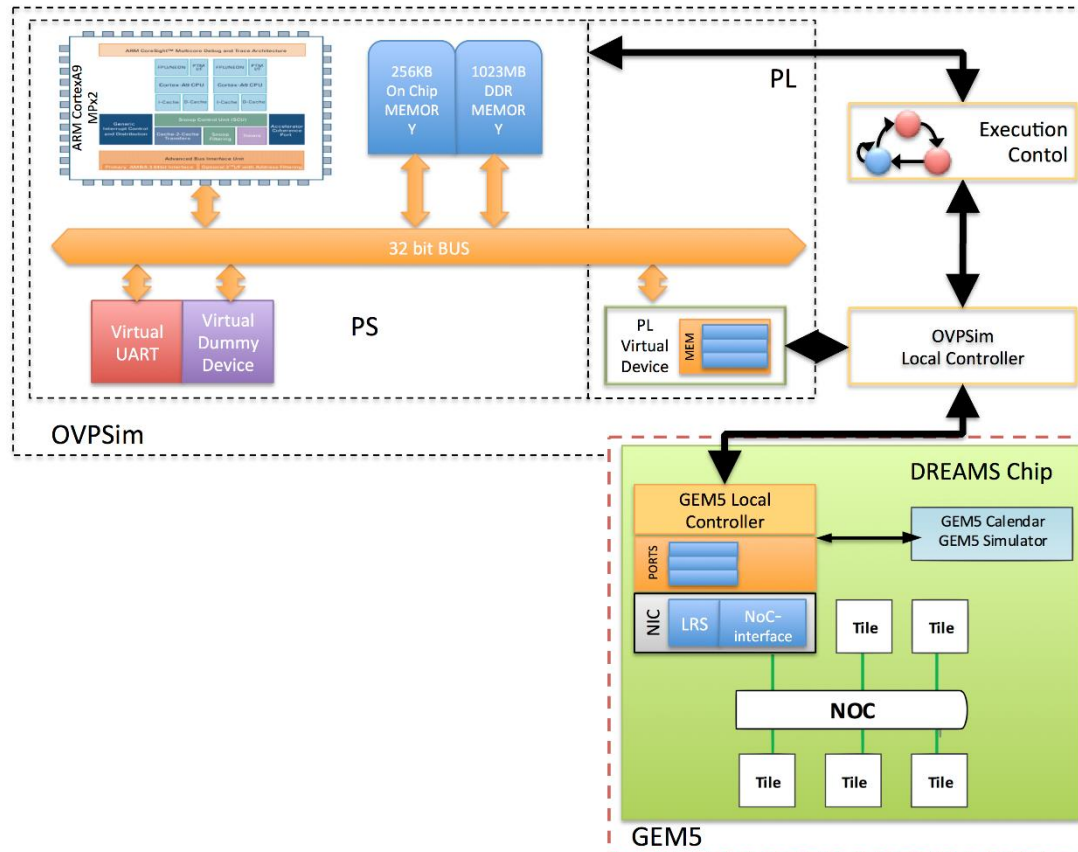


Figure 13: Integration OVPSim Platform with GEM5.

5.2.2.1 Programming Logic Virtual Device

The PLVD (Programming Logic Virtual Device) is a software module placed in the OVPSim platform and connected to the BUS in the memory range corresponding the PL [0x4000_0000-0xBFFF_FFFF]. Each time the processor accesses the bus in this address range, the device is accessed and take the simulation control.

The module contains 3 different components:

- MEMORY areas. These correspond to the memory interface available in the NIC (Network Interface Controller) of the LRS simulated by GEM5.
- MEMORY CONTROLLER is in charge to write or read the memory attending the bus interface. Moreover modifies the memory on DATA MANAGER request. And additionally alerts the DATA MANAGER when the processor modifies memory.
- DATA MANAGER is responsible to interact with the OLC (OVPSim Local Controller). The component controls the data modification between MEMORY CONTROLLER and OLC.

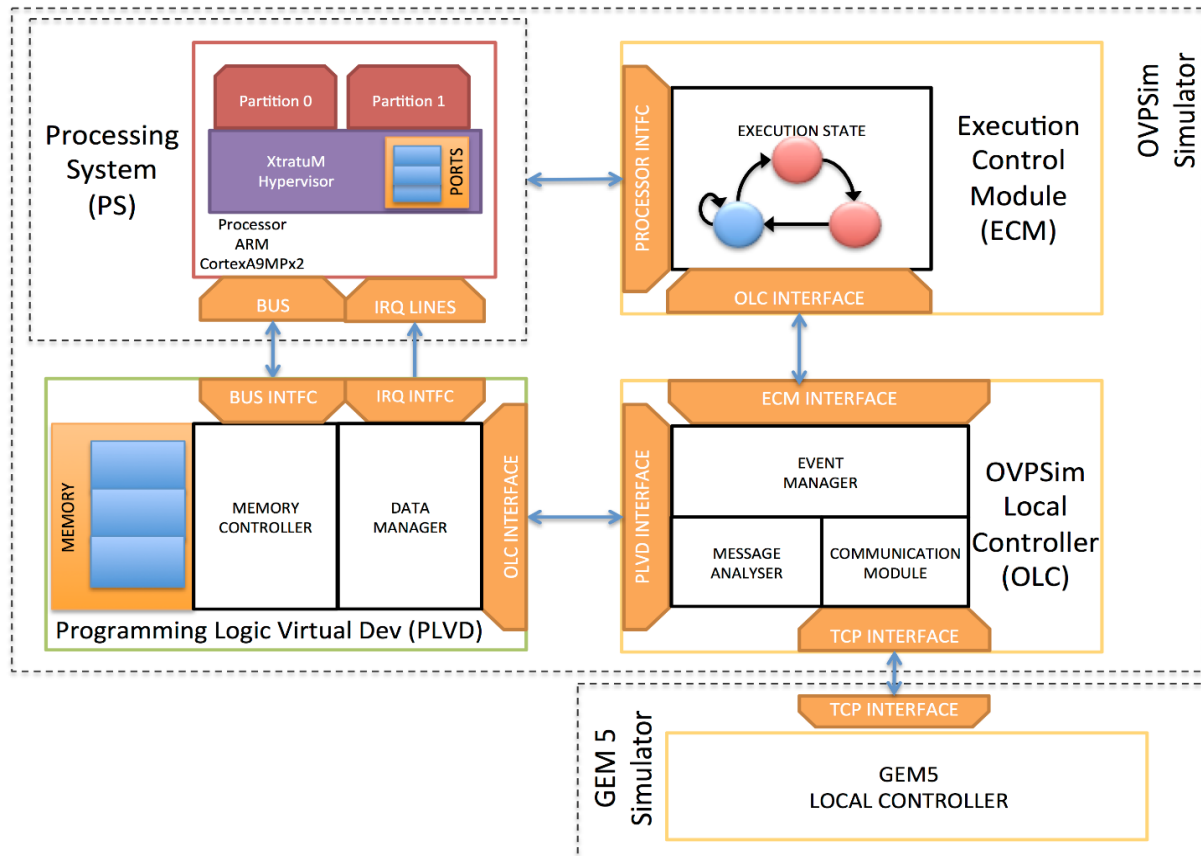


Figure 14: Modules involved in the integration of simulators.

5.2.2.2 OVPSim Local Controller

The OLC (OVPSim Local Controller) is a software module placed in the OVPSim platform. OLC is in charge to coordinate the simulators OVPSim and GEM5. As shown in Figure 14 the module has some internal components and interfaces that interact between the GEM5 simulator and the OVPSim modules.

- **COMMUNICATIONS MODULE** manages the TCP-IP Network Interface. Is in charge of configuring, write and read from the communication channels (TCP-IP sockets). The communication is based on string messages passing.
- **MESSAGE ANALYSER** receives and parses the messages providing from the COMMUNICATIONS MODULE. Depending on the type of the message: DATA or CTRL. Interacts with the PLVD (Programming Logic Virtual Device) or the EVENT MANAGER respectively. The detailed content of the messages is explained in section 5.2.3
- **EVENT MANAGER** Receives 3 different types of events:
 - New data event from the PLVD. In this case generate a DATA Message and queue in the COMMUNICATIONS MODULE.
 - Run event from the MESSAGE ANALYSER. The ECM (Execution Control Module (Adaptation)) must be requested to run a new instruction (or run for the equivalent amount of time).
 - Finish event from the ECM. The component proceeds to indicate to the COMMUNICATIONS MODULE that a CTRL message must be sent. If any DATA Message is queued both messages could be send together.

5.2.2.3 Execution Control Module (Adaptation)

The ECM (Execution Control Module) has been adapted to allow new synchronization features. A new interface with the OLC (OVPSim Local Controller) module is implemented. Through this interface the module receive commands to execute a new instruction in the processor (or run for equivalent time). When instruction is completely executed OLC is alerted through this interface to send the corresponding message to GEM5.

With these considerations, the execution flow loop explained in section 4.1.2 has been modified as follows:

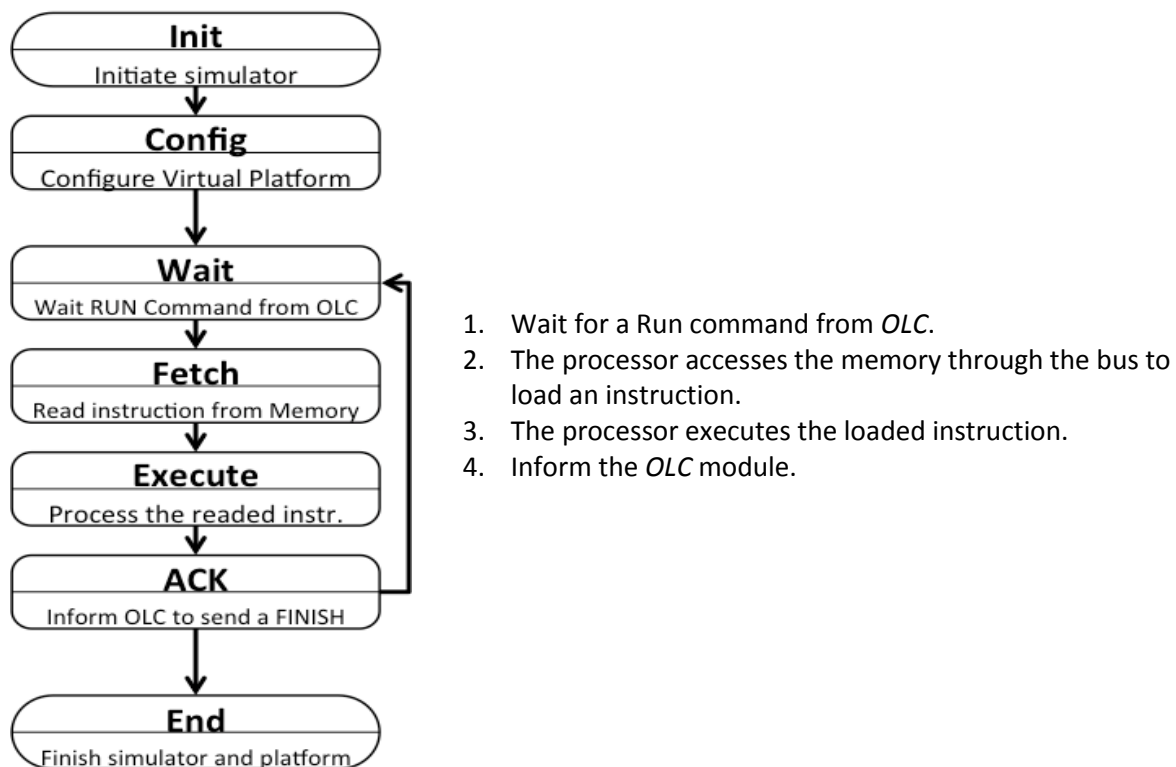


Figure 15: Execution Control Flow Diagram.

5.2.3 GEM5 OVPSim Message definition

The communication between both simulators is based on TCP-IP. Gem5 acts as server and is in charge of start the communication. Once the connection is stabilized, the communication is based on string message passing. The main characteristics of the messages are:

- Messages are Strings that are separated with “\$” in case of multiple fields.
- Messages start and end with “\$”.
- The first element in the message is the type:
 - DATA.
 - RUN (Control).
 - FINISH (Control).

Depending on the Type messages contain the following elements:

5.2.3.1 DATA Message

Data messages contains pairs of values separated by “|”. Each of the pairs contains an address and its value in hexadecimal separated by “:”. As example in the scenario where:

- The address 0x4000_0000 is written with the value 0x0000_0001.

- The address 0x4000_0004 is written with the value 0x0000_0010.

The corresponding message must be:

\$ DATA \$ 0x4000_0000 : 0x0000_0001 | 0x4000_0004 : 0x0000_0010 \$

5.2.3.2 RUN Message

Run messages represent the command to advance in the simulation. Each message contains the global simulation time in nanoseconds. This message is always sent from GEM5 to OVPSim. The time is included in order to maintain a global synchronised the global time.

The resulting message if GEM5 send a RUN command to OVPSIM where the end global time must be 1.254.132us:

\$ RUN \$ 1254132 \$

Additionally a run message could contain an embedded DATA message if required.

The resulting message assuming the previous scenario:

\$ RU \$ 125413 \$ DAT \$ 0x4000_000 : 0x0000_000 | 0x4000_000 : 0x0000_001 \$
N 2 A 0 1 4 0

5.2.3.3 Finish Message

Finish Messages are sent from OVPSim to GEM5 each time the first is ready to execute a new command.

The basic message only contains the type as follow:

\$ FINISH \$

Additionally a finish message could contain an embedded DATA message if required:

\$ FINISH \$ DATA \$ 0x4000_0000 : 0x0000_0001 \$

5.2.4 OVPSim Platform Validation

To validate the interaction between GEM5 and OVPSim a module that emulates the interface GEM5 Local Controller is developed. Also, an application that accesses areas corresponding to the memory of the ports is implemented in a partition.

The application writes a value in a memory address (addressX) then performs the reading in several related memory positions (addressX and addressX+Y).

The module follows the flow shown in Figure 15, generating the needed messages and parsing the received ones. When a message for the modification of a memory location (addressX) is received, the memory location (addressX + Y) is changed into this value by generating the corresponding message. Thus, when the application performs the reading in both memory locations, the read values should match.

6 Implementation of Model-Driven Configuration of Simulation

This section provides the detailed specification of the virtual platform configuration file generator. As recalled in Figure 16, its purpose is the automatic creation of the set of simulator configurations files that correspond to a given DREAMS system model, so as to avoid errors that would easily be introduced by a manual configuration and to increase this way the confidence in the outcomes of a simulation, which are used for validation purposes. The virtual platform configuration file generator is implemented with the help of the configuration framework provided by WP4 / T4.2.

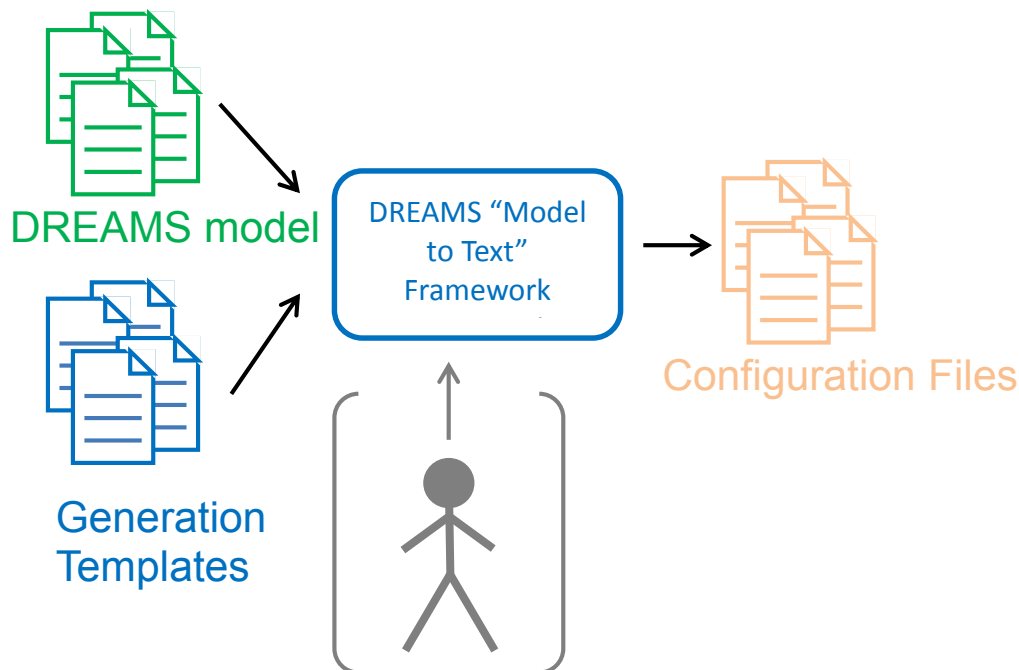


Figure 16: Inputs and outputs of the model to text configuration file generator

We start by describing in Section 6.1 an example system, which serves in the following section as basis for concrete configuration files examples. Then follows the detailed specification, which is divided into three domains: cluster level (Section 6.2), chip level (Section 6.3) and execution level (Section 6.3).

6.1 Configuration example: on-chip/off-chip communication

In this section, we describe an example scenario for the virtual-platform, which takes into account the following limitations of the virtual platform:

- At most one node may contain a NOC.
- No intra-tile communication.
- The simulated on-chip communication does not support multi-cast.
- The same txWindow is used for all ports through which a TT frame is emitted in a off-chip network router

The following figure gives an overview, where we suppose that each tile has exactly one processor core:

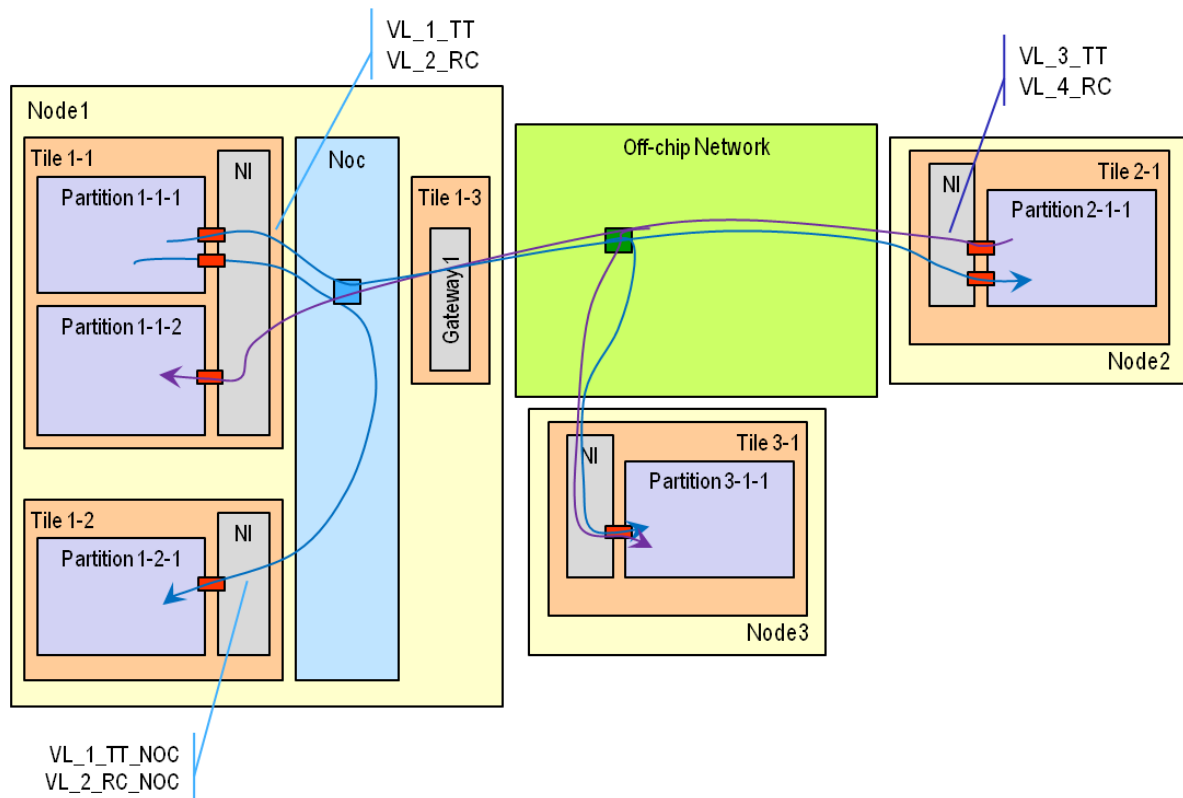


Figure 17: Schematic overview

Since partition ports can either receive or send exactly one virtual link and because the depicted arrows represent the paths of more than one message, the ports shown in Figure 17 correspond actually to several ports in the actual configuration.

The messages exchanged between applications are shown in the following table:

Messages		
Name	Sender	Receiver(s)
Msg_1_TT_NOC	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-3	Node1/Tile 1-2/Partition 1-2-1/Port 1-2-1
Msg_1_TT	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-1	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-1 Node3/Tile 3-1/Partition 3-1-1/Port 3-1-1
Msg_2_RC_NOC	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-4	Node1/Tile 1-2/Partition 1-2-1/Port 1-2-2
Msg_2_RC	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-2	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-2 Node3/Tile 3-1/Partition 3-1-1/Port 3-1-2
Msg_3_TT	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-3	Node3/Tile 3-1/Partition 3-1-1/Port 3-1-3 Node1/Tile 1-1/Partition 1-1-2/Port 1-1-5
Msg_4_RC	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-4	Node3/Tile 3-1/Partition 3-1-1/Port 3-1-4 Node1/Tile 1-1/Partition 1-1-2/Port 1-1-6

Each message is mapped to exactly one DREAMS Virtual Link, which is used to transport it:

DREAMS Virtual Links		
Name	Sender	Receiver(s)

VL_1_TT_NOC	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-3	Node1/Tile 1-2/Partition 1-2-1/Port 1-2-1
VL_1_TT	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-1	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-1 Node3/Tile 3-1/Partition 3-1-1/Port 3-1-1
VL_2_RC_NOC	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-4	Node1/Tile 1-2/Partition 1-2-1/Port 1-2-2
VL_2_RC	Node1/Tile 1-1/ Partition 1-1-1/Port 1-1-2	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-2 Node3/Tile 3-1/Partition 3-1-1/Port 3-1-3
VL_3_TT	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-3	Node3/Tile 3-1/Partition 3-1-1/Port 3-1-3 Node1/Tile 1-1/Partition 1-1-2/Port 1-1-5
VL_4_RC	Node2/Tile 2-1/Partition 2-1-1/Port 2-1-4	Node3/Tile 3-1/Partition 3-1-1/Port 3-1-4 Node1/Tile 1-1/Partition 1-1-2/Port 1-1-6

The characteristics of the VLS are shown in the following table:

VLS	Period / MINT	Size
VL_1_TT_NOC	10 ms	100 byte
VL_1_TT	10 ms	100 byte
VL_2_RC_NOC	15 ms	200 byte
VL_2_RC	15 ms	200 byte
VL_3_TT	20 ms	300 byte
VL_4_RC	30 ms	400 byte

The following table shows all DREAMS ids that are used in the CSV configuration files and the trace files, which are mostly based on numerical identifiers. These numerical identifiers must satisfy the following rules:

- message IDs must be unique in the considered system
- VL IDs must be unique in the considered system
- port IDs must be unique within the tile to which they belong
- tile IDs must be unique within the nodes to which they belong
- node IDs must be unique within the cluster to which they belong

Name	DREAMS IDs
Msg_1_TT	1
Msg_1_TT_NOC	11

Msg_2_RC	2
Msg_2_RC_NOC	12
Msg_3_TT	3
Msg_4_RC	4
VL_1_TT	1
VL_1_TT_NOC	11
VL_2_RC	2
VL_2_RC_NOC	12
VL_3_TT	3
VL_4_RC	4
Node1	1
Tile 1-1	1
Partition 1-1-1	1
Port 1-1-1	1
Port 1-1-2	2
Port 1-1-3	3
Port 1-1-4	4
Partition 1-1-2	2
Port 1-1-5	5
Port 1-1-6	6
Tile 1-2	2
Partition 1-2-1	1
Port 1-2-1	1
Port 1-2-2	2
Tile 1-3	3
Node2	2
Tile 2-1	1
Partition 2-1-1	1
Port 2-1-1	1
Port 2-1-2	2
Port 2-1-3	3
Port 2-1-4	4
Node3	3
Tile 3-1	1

Partition 3-1-1	1
Port 3-1-1	1
Port 3-1-2	2
Port 3-1-3	3
Port 3-1-4	4

6.2 Configuration Tools for the Cluster Level

6.2.1 Specification of configuration file formats with examples

The specification of the configuration parameters for the simulated off-chip network components is an updated version of the information provided in D5.2.1, Section 3.3. The shown examples files correspond to the example scenario presented in Section 6.1.

6.2.1.1 Simulated TTEthernet Switch

One configuration file is used for each off-chip switch. It contains parameters for time triggered and rate constrained Virtual Link (VL) that traverse the switch.

6.2.1.1.1 Definition of required configuration parameters

- Traffic type: allowed traffic type values are “TT” for time triggered and “RC” for rate constraint virtual links.
- Virtual link id: identifier of the VL. Range: 0 - 255 (integer)
- Period/MINT: Period in microseconds (double) for periodic messages or “minimum interval time” (MINT) in microseconds (double) for sporadic VLs
- Phase/ jitter: Phase in microseconds (double) for periodic message (Notice: same value for all Receiver ports) or Jitter in microseconds (double) for sporadic message.
- Sender port: numerical identifier of the switch port through which the VL is received. Range: 1-8 (integer)
- Receiver port: numerical identifier of a switch port through which the VL is send out. Range: 1-8 (integer)
- Size: Message size of Ethernet frame (payload + 28 bytes) in bytes (integer)
- Queue ID: numerical identifier of the queue used to store the VL (integer). Starts from 50.

6.2.1.1.2 CSV File Format

Each switch has its own configuration file:

- The semicolon “;” is used to separate fields.
- Each file must include the name of the switch in the second field of the first line.
- The second line should contain the column headers with the parameter names, in the order in which they are presented in previous Section. The actual contents of these header cells are however not parsed by the configuration tool of the simulator.
- Starting with the third line, there shall be exactly one line for each virtual link that crosses the switch. The cells of the line shall contain the values of the parameters in the order in which they are specified in the previous section.
- The first “Receiver port” cells shall be used as needed and the other shall contain -1.

6.2.1.1.3 Sample file

switch_R.csv:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	SW	R												
2	Traffic Type	VL	Period (us)	Phase(us)	Sender port	Receiver port	Receiver port	Receiver port	Receiver port	Receiver port	Receiver port	Receiver port	Size	Queue ID
3	TT	1	10000	1000	1	2	3	-1	-1	-1	-1	-1	100	50
4	RC	2	15000	500	1	2	3	-1	-1	-1	-1	-1	200	51
5	TT	3	10000	1000	2	1	3	-1	-1	-1	-1	-1	300	52
6	RC	4	15000	500	2	1	3	-1	-1	-1	-1	-1	400	53

6.2.1.2 Simulated Node with a Core

On configuration file is used for all simulated nodes that do not have a NOC. It contains parameters for each time triggered and each rate constraint Virtual Link (VL) sent by these nodes.

6.2.1.2.1 Definition of required configuration parameters

- Node Name: Name of the simulated node (String)
- Virtual link identifier: numerical identifier of the VL. Range: 0 - 255 (integer)
- Traffic Type: allowed traffic type values are “TT” for time triggered and “RC” for rate constraint virtual links.
- Period/MINT: Period in microseconds (double) for TT VLs or “minimum interval time” (MINT) in microseconds (double) for RC VLs.
- Start Time: Starting time in microseconds of the application. For TT VLs it is the phase parameter. For RC VLs, it is the first sending time, which is followed by random dates consistent with the MINT.
- Size: payload of the VL size, in bytes (integer)
- Queue ID: numerical identifier of the queue used to store the VL (integer). Each node starts the queue ID from 0 for the periodic and from 2 for the sporadic.

6.2.1.2.2 CSV File Format

All the nodes with one core share one configuration file.

- The semicolon “;” is used to separate fields.
- The first line of the configuration should contain the names of the parameters in the order in which they are presented in the previous section.
- Starting with the second line, there should be exactly one line for each sent virtual link.

6.2.1.2.3 Sample file

nodes-without-noc.csv:

	A	B	C	D	E	F	G
1	Node Name	VLID	Period/GAB	start Time	Traffic Type	Message size	Queue ID
2	Node3	3	20	0	TT	300	0
3	Node3	4	30	0	RC	400	1

6.2.1.3 Simulated on-chip/off-chip Gateway: time triggered virtual links

On configuration file is used for the parameters related to the sending and receiving of time triggered virtual links in the on-chip/off-chip gateway of the unique node with NOC.

6.2.1.3.1 Definition of configuration parameters

- VLID: numerical identifier of the VL. Range: 0 - 255 (integer)
- DIR: the message direction (IN=1/OUT=2)
- Period in microseconds (double)
- Phase in microseconds (double)
- Message size(in bytes) (integer)
- Queue ID: numerical identifier of the queue used to store the VL (integer). Values start from zero.

6.2.1.3.2 CSV File Format

All time triggered virtual links that transit through the on-chip/off-chip gateway of the unique node with NOC must be described in one configuration file:

- The comma “,” is used to separate fields.
- The first line of the configuration should contain the names of the parameters in the order in which they are presented in the previous section.
- Starting with the second line, there must be exactly one line for each time triggered virtual link.

6.2.1.3.3 Sample file

gateway-TT.csv:

	A	B	C	D	E	F
1	vl_id	period(us)	phase(us)	msg_size	direction	queue_id
2	1	10000	500	100	2	1
3	3	20000	1000	300	1	3

6.2.1.4 Simulated on-chip/off-chip Gateway: rate constraint virtual links

On configuration file is used for the parameters related to the sending and receiving of rate constraint virtual links in the on-chip/off-chip gateway of the unique node with NOC

6.2.1.4.1 Definition of configuration parameters

- VLID: numerical identifier of the VL. Range: 0 - 255 (integer))
- DIR: the message direction (IN=1/OUT=2)
- Minimum interval time (MINT) in microseconds (double)
- Maximum message size (in bytes) (integer)
- Priority: the priority is used only for sporadic messages (high =1 /low=0)
- Queue ID: numerical identifier of the queue used to store the VL (integer). Values start from zero.

6.2.1.4.2 CSV File Format

All rate constraint virtual links that transit through the on-chip/off-chip gateway of the unique node with NOC must be described in one configuration file:

- The comma “,” is used to separate fields.
- The first line of the configuration should contain the names of the parameters in the order in which they are presented in the previous section.
- Starting with the second line, there shall be exactly one line for each rate constraint virtual link.

6.2.1.4.3 Sample file

gateway-RC.csv:

	A	B	C	D	E	F
1	vl_id	bag(us)	priority	msg_size	direction	queue_id
2	2	15000	1	200	2	2
3	4	30000	1	400	1	4

6.2.2 DREAMS meta-model correspondences

In this section are specified the mappings between the DREAMS meta-model entities and the configuration entities. A separate subsection is present for each type of configuration file.

6.2.2.1 Simulated off-chip switch configuration file

Number of CSV files	As may files as there are switches in the off-chip network.
Scope of a CSV file	Exactly one switch.
File name rules	Switch_<name of the concerned switch>.csv
Cell separator	;
MM Package	eu.dreamsproject.psm.simulation.offchip.model
MM Namespace	http://www.dreamsproject.eu/psm/simulated/offchip

File Header		
MM EClass	SwitchConfiguration	
Field	DREAMS meta-model	Notes
SW	((INamedElement) SwitchConfiguration.modelElement).name	SwitchConfiguration.modelElement should point to a eu.dreamsproject.platform.model.offchipnetwork.OffChipNetworkRouter (in the physical PlatformArchitecture) that implements the INamedElement interface.

Line	
MM EClass	SwitchConfigurationItem (contained by SwitchConfiguration)

Column title	DREAMS meta-model	Conversion MM → CSV
Traffic Type	SwitchConfigurationItem.getTrafficType()	TrafficType.TIME_TRIGGERED → TT TrafficType.RATE_CONSTRAINT → RC
Virtual Link Id	SwitchConfigurationItem.getVirtualLinkId()	
Period /MINT	SwitchConfigurationItem.getPeriodMint_s()	* 1000000 (s to us)
Phase / Jitter	SwitchConfigurationItem.getPhaseJitter_s()	* 1000000 (s to us)
Sender port	SwitchConfigurationItem.getSenderPortId()	
Receiver port	SwitchConfigurationItem.getReceiverPortPortIdList().get(0)	
Receiver port	SwitchConfigurationItem.getReceiverPortPortIdList().get(1) or -1	Result of getReceiverPortPortIdList() is not padded to size 8
Receiver port	SwitchConfigurationItem.getReceiverPortPortIdList().get(2) or -1	See above
Receiver port	SwitchConfigurationItem.getReceiverPortPortIdList().get(3) or -1	See above
Receiver port	SwitchConfigurationItem.getReceiverPortPortIdList().get(4) or -1	See above
Receiver port	SwitchConfigurationItem.getReceiverPortPortIdList().get(5) or -1	See above
Receiver port	SwitchConfigurationItem.getReceiverPortPortIdList().get(6) or -1	See above
Size	SwitchConfigurationItem.getPayLoadSize_bytes()	+ 28 for Ethernet header

Queue ID	SwitchConfigurationItem.queueId	
----------	---------------------------------	--

Notes:

- The “Sender Port” is the port through which the frame arrives at the switch.
- A “Receiver Port” is a port through which the frame exits the switch.
- The Phase is valid for all ports of the switch through which the virtual link is sent out

6.2.2.1.1 Simulated Node with a Core configuration file

Number of CSV files	One per off-chip network
Scope of a CSV file	All virtual links send by nodes connected to the off-chip network but do not contain a on-chip network, i.e. that contain only one tile.
File name rules	NodesWithoutNoc.csv
Cell Separator	;
MM Package	eu.dreamsproject.psm.simulation.offchip.model
MM Namespace	http://www.dreamsproject.eu/psm/simulated/offchip

File Header
N/A

Line		
MM EClass	SingleCoreNodeConfigurationItem (contained by SingleCoreNodeConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV

Node Name	((INamedElement) SingleCoreNodeConfiguration.modelElement).name	
Virtual Link Id	SingleCoreNodeConfigurationItem.getVirtualLinkId()	
Traffic Type	SingleCoreNodeConfigurationItem.getTrafficType()	TrafficType.TIME_TRIGGERED → TT TrafficType.RATE_CONSTRAINT → RC
Period/MINT	SingleCoreNodeConfigurationItem.getPeriodMint_s()	* 1000000 (s to us)
Start Time	SingleCoreNodeConfigurationItem.getSendingTime_s()	* 1000000 (s to us)
Message Size	SingleCoreNodeConfigurationItem.getPayloadSize_bytes()	
Queue ID	SingleCoreNodeConfigurationItem.queueID	

Notes:

- SingleCoreNodeConfiguration.modelElement should point to a eu.dreamsproject.platform.model.node.Node (in the physical PlatformArchitecture) that implements the INamedElement interface.

6.2.2.2 Simulated on-/off-chip Gateway configuration files

6.2.2.2.1 Periodic frames configuration files

Number of CSV files	At most one, for the gateway of the unique node with Noc, if present.
Scope of a CSV file	The time triggered Virtual Links sent or received by the gateway of the unique node with Noc.
File name rules	gateway-TT.csv
Cell separator	,
MM Package	eu.dreamsproject.psm.simulation.offchip.model
MM Namespace	http://www.dreamsproject.eu/psm/simulated/offchip

File Header

N/A

Line		
MM EClass	PeriodicGatewayConfigurationItem (contained by GatewayConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV
VLID	PeriodicGatewayConfigurationItem.getVirtualLinkId()	
Period	PeriodicGatewayConfigurationItem.getPeriodMint_s()	* 1000000 (s to us)
Phase	PeriodicGatewayConfigurationItem.getPhase_s()	* 1000000 (s to us)
Message Size [bytes]	PeriodicGatewayConfigurationItem.getPayloadSize_bytes()	
Direction	PeriodicGatewayConfigurationItem.getTrafficDirection()	TrafficDirection.INPUT → 1 TrafficDirection.OUTPUT → 2
Queue ID	PeriodicGatewayConfigurationItem.queueID	

6.2.2.2.2 Sporadic frames configuration files

Number of CSV files	At most one, for the gateway of the unique node with Noc, if present.
Scope of a CSV file	The time triggered Virtual Links sent or received by the gateway of the unique node with Noc.
File name rules	gateway-RC.csv
Cell Separator	,
MM Package	eu.dreamsproject.psm.simulation.offchip.model
MM Namespace	http://www.dreamsproject.eu/psm/simulated/offchip

File Header
N/A

Line		
MM EClass	SporadicGatewayConfigurationItem (contained by GatewayConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV
VLID	SporadicGatewayConfigurationItem.getVirtualLinkId()	
MINT	SporadicGatewayConfigurationItem.getPeriodMint_s()	* 1000000 (s to us)
Priority	SporadicGatewayConfigurationItem.getPriority()	getPriority() > 0 → 1 getPriority() == 0 → 0
Maximum Message Size [bytes]	SporadicGatewayConfigurationItem.getPayloadSize_bytes()	
Direction	PeriodicGatewayConfigurationItem.getTrafficDirection()	TrafficDirection.INPUT → 1 TrafficDirection.OUTPUT → 2
Queue ID	PeriodicGatewayConfigurationItem.queueID	

6.3 Configuration Tools for the Chip Level

6.3.1 Specification of configuration file formats

The specification of the cluster level configuration file generator for the virtual platform is an updated version of the information provided in D5.2.1, Section 4.1.3.

Remember that the simulation model supports only one node with NOC. The perimeter of the configuration files is that of the unique node with NOC.

At on-chip level, the unit of time used in the gem5 based simulation is the “tick” and therefore all time values must be expressed as multiples of ticks. Since the tick is set to 1 nanosecond of real time, it is equivalent to say that time values are expressed in nanoseconds.

The shown examples files correspond to the example scenario presented in Section 6.1. For better readability of the examples, we have used the character “#” to indicate comments, which either recall the title of the columns, including units, or describe the following line.

6.3.1.1 Hardware configuration parameters

A configuration file is used to describe the hardware layout of the chip.

6.3.1.1.1 Definition of required configuration parameters

- Global period: The least common multiple of the periods of the periodic messages, expressed in ticks (integer)
- Number of tiles: The number of tiles on the chip (integer)
- Tile ID: numerical identifier of the tile (integer)
- Number of cores: The number of cores on each tile (integer)
- Number of partitions at the core : The number of the partitions at each core (integer)
- Number of ports at the tile: The number of ports on the tile (integer)

6.3.1.1.2 CSV File Format

The format of the CSV file is the following:

- First cell of first column: global period
- First cell of second column: number of tiles
- Starting with line 3, a line for each tile
 - First cell: numerical id of the tile
 - Second cell: number of core in the tile
 - Starting from cell 3: for each core, number of partition running on the core
 - Last cell: number of ports in the tile

Global period							
Number of tiles							
Tile_ID #1	numbers of Cores	Num. Partitions at Core#1	Num. Partitions at Core#2	Partitions at Core#3	...	Num. Partitions at Core#n	Number of ports
Tile_ID #2	numbers of Cores	Partitions at Core#1	Partitions at Core#2	Partitions at Core#3	...	Partitions at Core#n	Number of ports
...	numbers of Cores
Tile_ID #n	numbers of Cores	Partitions at Core#1	Partitions at Core#2	Partitions at Core#3	...	Partitions at Core#n	Number of ports

6.3.1.1.3 Sample file

HWConfig.csv:

	A	B	C	D
1	# Global period (tick=ns)			
2	10000000			
3	# Number of tiles			
4	2			
5	# Tile id	Number of cores	Num. Partitions at Core#1	Number of ports
6	# Tile 1-1			
7	1	1	2	6
8	# Tile 1-2			
9	2	1	1	2
10	# Tile 1-3			
11	3	1	1	1

6.3.1.2 Ports configuration parameters

One configuration file is used to specify which partition ports of the unique node with NOC send or receive which virtual link.

6.3.1.2.1 Definition of required configuration parameters

- Port ID: numerical identifier (integer) of a port
- Core ID: numerical identifier (integer) of the of the core, which has the right to write into/read from the port
- Partition ID: numerical identifier (integer) of the of the partition, which has the right to write into/read from the port

- Physical Address: The physical name² of the port: *Cluster.Node.Tile.Port*. (Integer. Integer. Integer. Integer)
- Logical Address: The Logical name of port: *Criticality.Subsystem.Component.Message* (Integer.Integer.Integer.Integer)
- Type : Each port can be used for sending or receiving of either of the following traffic types:
 - TT: for periodic messages (String, "TT")
 - RC: for sporadic messages (String, "RC")
 - BE: for aperiodic messages (String, "BE")
- VLID: In case of TT or RC ports, this parameter denotes the numerical identifier of the virtual link, to which the port corresponds to. This correspondence includes the temporal characteristics of the port. In case of BE ports, this value must be "-1" (integer)
- Direction: Defines if the port is used for outgoing message (OUT) or for the incoming ones (IN) (String, "IN " or "OUT")
- Semantics: Whether is an event port or an state port (String, "EVENT" or "STATE")

6.3.1.2.2 CSV File Format

A line of the CSV file corresponds to a sending or a reception of a virtual link at some port. The cells contain the values of the parameters in the order in which they are described in the previous section.

6.3.1.2.3 Sample file

HL_PortConfig.csv:

	A	B	C	D	E	F	G	H	I
1	# Port ID	Core ID	Partition ID	Phys. Addr.	Log. Addr.	Type	VL ID	Direction	Semantics
2	# Port 1-1-1	Core 1-1-1	Partition 1-1-1	Cluster.Node1.Tile 1-1.Port-1-1-1	Criticality.Subsystem.Component.Message		VL_1_TT		
3	1	1	1	1.1.1.1	1.1.1.1	TT		1 OUT	STATE
4	# Port 1-1-3	Core 1-1-1	Partition 1-1-1	Cluster.Node1.Tile 1-1.Port-1-1-3	Criticality.Subsystem.Component.Message		VL_1_TT_NOC		
5	3	1	1	1.1.1.3	1.1.1.11	TT		11 OUT	STATE
6	# Port 1-1-2	Core 1-1-1	Partition 1-1-1	Cluster.Node1.Tile 1-1.Port-1-1-2	Criticality.Subsystem.Component.Message		VL_2_RC		
7	2	1	1	1.1.1.2	1.1.1.2	RC		2 OUT	EVNENT
8	# Port 1-1-4	Core 1-1-1	Partition 1-1-1	Cluster.Node1.Tile 1-1.Port-1-1-4	Criticality.Subsystem.Component.Message		VL_2_RC_NOC		
9	4	1	1	1.1.1.4	1.1.1.12	RC		12 OUT	EVNENT
10	# Port 1-1-5	Core 1-1-1	Partition 1-1-2	Cluster.Node1.Tile 1-1.Port-1-1-5	Criticality.Subsystem.Component.Message		VL_3_TT		
11	5	1	2	1.1.1.5	1.1.1.3	TT		3 IN	STATE
12	# Port 1-1-6	Core 1-1-1	Partition 1-1-2	Cluster.Node1.Tile 1-1.Port-1-1-6	Criticality.Subsystem.Component.Message		VL_4_RC		
13	6	1	2	1.1.1.6	1.1.1.4	RC		3 IN	EVNENT

6.3.1.3 VLS configuration parameters

On configuration file is used to describe all virtual links exchanged in the modeled system.

6.3.1.3.1 Definition of required configuration parameters

- VLID: numerical identifier (integer) of a Virtual link.
- Type: VL type (String "TT" or "RC")
- Source: Physical name of the source port:
 - PhyName: *Cluster.Node.Tile.Port* (Integer. Integer. Integer. Integer)
- Destination: The current implementation of the NoC does not support multicast VLs. This field includes the physical name of the destination port.

² DREAMS architectural style (D1.1.1)

- PhyName_i: *Cluster.Node.Tile.Port* (Integer. Integer. Integer. Integer)
- Period/MINT: The value of the period for TT messages or the MINT for RC messages in ticks (integer)

6.3.1.3.2 CSV File Format

A line of the CSV file corresponds to one sending or reception of a virtual link at some port. The cells contain the values of the parameters in the order in which they are described in the previous section.

6.3.1.3.3 Sample file

VLConfig.csv:

	A	B	C	D	E
1	# VL ID	Type	Source	Destination	Period/MINT (tick=ns)
2	# VL_1_TT		Cluster.Node1.Tile 1-1.Port-1-1-1	Cluster.Node2.Tile 2-1.Port-2-1-1	
3		1 TT	1.1.1.1	1.2.1.1	10000000
4	# VL_1_TT_NOC		Cluster.Node1.Tile 1-1.Port-1-1-3	Cluster.Node1.Tile 1-2.Port-1-2-1	
5		11 TT	1.1.1.3	1.1.2.1	10000000
6	# VL_2_RC		Cluster.Node1.Tile 1-1.Port-1-1-2	Cluster.Node2.Tile 2-1.Port-2-1-2	
7		2 RC	1.1.1.2	1.2.1.2	15000000
8	# VL_2_RC_NOC		Cluster.Node1.Tile 1-1.Port-1-1-4	Cluster.Node1.Tile 1-2.Port-1-2-2	
9		12 RC	1.1.1.4	1.1.2.2	15000000
10	# VL_3_TT		Cluster.Node2.Tile 2-1.Port-2-1-3	Cluster.Node1.Tile 1-1.Port-1-1-5	
11		3 TT	1.2.1.3	1.1.1.5	20000000
12	# VL_4_RC		Cluster.Node2.Tile 2-1.Port-2-1-4	Cluster.Node1.Tile 1-1.Port-1-1-6	
13		4 RC	1.2.1.4	1.1.1.6	30000000

6.3.1.4 TT schedule for the EBU

One configuration file is used for the emission phase of time triggered virtual links in partition ports.

6.3.1.4.1 Definition of required configuration parameters

- Tile id : numerical identifier (integer) of a tile
- Phase: Time, with respect to the period, at which the message is dequeued from the port, expressed in ticks (integer)
- Port id: numerical identifier of the port which is dequeued (integer)

6.3.1.4.2 CSV File Format

A line of the CSV file corresponds to one sending of a TT virtual link at some port. The cells contain the values of the parameters in the order in which they are described in the previous section.

6.3.1.4.3 Sample file

TT_Schedule_EBU.csv

	A	B	C	D
1	# Tile ID	Phase (tick=ns)	Port Id	
2	# Tile 1-1		Port 1-1-1	VL_1_TT
3	1	10000	1	
4	# Tile 1-1		Port 1-1-3	VL_1_TT_NOC
5	1	10000	3	

6.3.1.5 TT schedule for the serialization unit

One configuration file is used for global parameters of the time triggered communication.

6.3.1.5.1 Definition of required configuration parameters

- Timely block/shuffling: activation of timely block in the entire chip. In case of non activation, i.e. activation of shuffling, the information given for guarding window will be ignored
(1 activated, 0 deactivated)
- Tile id : numerical identifier (integer) of a tile
- Period: period of the guarding window, expressed in ticks (integer)
- Opening phase: Starting phase of the guarding window, expressed in ticks (integer)
- Closing phase: Closing phase of the guarding window, expressed in ticks (integer)

6.3.1.5.2 CSV File Format

A line of the CSV file corresponds to a tile of the chip. The cells contain the values of the parameters in the order in which they are described in the previous section.

6.3.1.5.3 Sample file

TTSchedule_SU.csv:

	A	B	C	D	E
1	# Timely block / shuffling	Tile ID	Period (tick=ns)	Opening phase (tick=ns)	Closing phase (tick=ns)
2		# Tile 1-1			
3	1	1			
4		# Tile 1-2			
5	1	2			

6.3.1.6 Message configuration parameters

On configuration file is used to describe the messages send by applications.

6.3.1.6.1 Definition of required configuration parameters

- MsgId: numerical identifier (integer) of the message
- Type: type of the message (String, TT, RC or BE).
- VLID: identifier (integer) of a VL in case of TT and RC messages, and -1, otherwise, -1 (integer)
- Deadline: the message deadline, in ticks, which is used for statistic and result analysis. (integer).

- Max_MSG_Size: Maximum message size, in bytes (integer)

6.3.1.6.2 CSV File Format

A line of the CSV file corresponds to a message send by some application. The cells contain the values of the parameters in the order in which they are described in the previous section.

6.3.1.6.3 Sample file

Msg.csv:

	A	B	C	D
1	# Msg ID	Type	VL ID	Deadline (tick=ns)
2	# Msg_1_TT		VL_1_TT	
3		1 TT		1 30000000
4	# Msg_1_TT_NOC		VL_1_TT_NOC	
5		11 TT		11 10000000
6	# Msg_2_RC		VL_2_RC	
7		2 RC		2 30000000
8	# Msg_2_RC_NOC		VL_2_RC_NOC	
9		12 RC		12 10000000
10	# Msg_3_TT		VL_3_TT	
11		3 TT		3 50000000
12	# Msg_4_RC		VL_4_RC	
13		4 RC		4 40000000

6.3.1.7 Trace file

One configuration file is used to specify the times when messages instances are injected into the partition ports.

6.3.1.7.1 Definition of required configuration parameters

- Tile ID: numerical identifier (integer) of the tile from which the message is injected in the network. (integer)
- Tick: the time, in ticks, of the message injection (integer)
- Msg id: numerical identifier (integer) of the messages
- Port Id: numerical identifier (integer) of the port, in which the message is enqueued
- Logical address of Destination: in case the message is enqueued into a best-effort port, the logical address of the destination can be declared explicitly through this field. In case of TT or RC ports, -1 must be given (integer.integer.integer.integer OR -1)
- Payload: contains the payload of the message (integer.integer..., max 1024)

6.3.1.7.2 CSV File Format

A line of the CSV file corresponds to the sending of an instance of a message at some specified time through the foreseen port. The cells contain the values of the parameters in the order in which they are described in the previous section.

6.3.1.7.3 Sample file

MsgInjection.csv:

	A	B	C	D	E	F
1	# Tile ID	Tick (tick=ns)	Msg ID	Port ID	Logical Address	Payload
2	# Tile 1-1		Msg_TT	Port 1-1-1		
3	1	0	1	1	-1	0
4	# Tile 1-1		Msg_TT_Noc	Port 1-1-3		
5	1	10000	1	3	-1	0
6	# Tile 1-1		Msg_RC	Port 1-1-2		
7	1	100000	1	1	-1	0
8	# Tile 1-1		Msg_RC	Port 1-1-4		
9	1	110000	1	1	-1	0
10	# Tile 1-1		Msg_TT	Port 1-1-1		
11	1	10000000	1	1	-1	0
12	# Tile 1-1		Msg_TT_Noc	Port 1-1-3		
13	1	10010000	1	3	-1	0
14	# Tile 1-1		Msg_RC	Port 1-1-2		
15	1	15100000	1	1	-1	0
16	# Tile 1-1		Msg_RC	Port 1-1-4		
17	1	15110000	1	1	-1	0
18	# Tile 1-1		Msg_TT	Port 1-1-1		
19	1	20000000	1	1	-1	0
20	# Tile 1-1		Msg_TT_Noc	Port 1-1-3		
21	1	20010000	1	3	-1	0
22	# Tile 1-1		Msg_RC	Port 1-1-2		
23	1	23100000	1	1	-1	0
24	# Tile 1-1		Msg_RC	Port 1-1-4		
25	1	23110000	1	1	-1	0

6.3.2 DREAMS meta-model correspondences

In this section are specified the mappings between the DREAMS meta-model entities and the configuration entities. A separate subsection is present for each type of configuration file.

6.3.2.1 *Simulated on-chip communication*

6.3.2.1.1 *Hardware configuration parameters*

Number of CSV files	At most one, for unique node with Noc, if present.
Scope of a CSV file	The tiles of the unique node with Noc in the simulated system.
File name rules	HWConfig.csv
Cell separator	,
MM Package	eu.dreamsproject.psm.onchip.com.model
MM Namespace	http://www.dreamsproject.eu/psm/onchip/com

File Header		
MM EClass	OnChipNetworkConfiguration	
Field	DREAMS meta-model	Conversion MM → CSV
Global period	OnChipNetworkConfiguration.getHyperPeriod_s()	× OnChipNetworkConfiguration. simulationTicksPerSecond (from s to simulation ticks)
Number of tiles	OnChipNetworkConfiguration.getNumTiles()	

Line

MM EClass	OnChipNiLrsConfiguration (contained by OnChipNetworkConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV
Tile id	<code>OnChipNiLrsConfiguration.getTileId()</code>	
Number of Cores	<code>OnChipNiLrsConfiguration.getCores().size()</code>	
Number of partitions at the core	<code>OnChipNiLrsConfiguration.getPartitions(core).size()</code>	
Number of ports at the tile	<code>OnChipNiLrsConfiguration.getPorts().size()</code>	

Notes:

- There must be as many non commented lines as “Number of tiles” declared in the second field.
- There must be as many “Number of partitions at the core” cells in a line as the declared in the “Number of Cores” cell.

6.3.2.1.2 Ports configuration parameters

Number of CSV files	At most one, for unique node with Noc, if present.
Scope of a CSV file	The tiles of the unique node with Noc in the simulated system.
File name rules	PortsConfig.csv
Cell separator	,
MM Package	<code>eu.dreamsproject.psm.onchip.com.model</code>
MM Namespace	<code>http://www.dreamsproject.eu/psm/onchip/com</code>

File Header
N/A

Line		
MM EClass	PortConfigurationItem (contained by PortConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV
Port ID	PortConfigurationItem.getPortId()	
Core ID	PortConfigurationItem.getCoreId()	
Partition ID	PortConfigurationItem.getPartitionId()	
Physical Address	PortConfigurationItem.getPhysicalName()	
Logical Address	PortConfigurationItem.getLogicaName()	
Type	PortConfigurationItem.getTrafficType()	TrafficType.TIME_TRIGGERED → TT TrafficType.RATE_CONSTRAINT → RC TrafficType.BEST_EFFORT → BE
VLID	PortConfigurationItem.getVirtualLinkId()	
Direction	PortConfigurationItem.getDirection()	TrafficDirection.INPUT → IN TrafficDirection.OUTPUT → OUT
Semantics	PortConfigurationItem.getSemantics()	PortSemantics.EVENT → EVENT PortSemantics.STATE → STATE

6.3.2.1.3 VLS configuration parameters

Number of CSV files	At most one, for unique node with Noc, if present.
Scope of a CSV file	All DREAMS virtual links that are sent or received by a tile of the node with a Noc in the simulated system.
File name rules	VLsConfig.csv
Cell separator	,
MM Package	eu.dreamsproject.psm.onchip.com.model
MM Namespace	http://www.dreamsproject.eu/psm/onchip/com

File Header
N/A

Line		
MM EClass	VirtualLinkConfigurationItem (contained by VirtualLinkConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV
VLID	VirtualLinkConfigurationItem.getVirtualLinkId()	
Type	VirtualLinkConfigurationItem.getTrafficType()	TrafficType.TIME_TRIGGERED → TT TrafficType.RATE_CONSTRAINT → RC
Source	VirtualLinkConfigurationItem.getSourcePhysicalName()	
Destination	VirtualLinkConfigurationItem.getDestinationsPhysicalNameList().get(0)	
Period/MINT	VirtualLinkConfigurationItem.getPeridMint_s()	× OnChipNetworkConfiguration. simulationTicksPerSecond (from s to simulation ticks)

6.3.2.1.4 *TT schedule for the EBU*

Number of CSV files	At most one, for unique node with Noc, if present.
Scope of a CSV file	The unique node with a Noc in the simulated system.
File name rules	TTSchedule_EBU.csv
Cell separator	,
MM Package	eu.dreamsproject.psm.onchip.com.model
MM Namespace	http://www.dreamsproject.eu/psm/onchip/com

File Header
N/A

Line		
MM EClass	EgressBridgingUnitTimeTriggeredScheduleEntry (contained by EgressBridgingUnitTimeTriggeredSchedule)	
Column title	DREAMS meta-model	Conversion MM → CSV
Tile id	EgressBridgingUnitTimeTriggeredScheduleEntry.getTileId()	
Phase	EgressBridgingUnitTimeTriggeredScheduleEntry.getPhase_s()	× OnChipNetworkConfiguration. simulationTicksPerSecond (from s to simulation ticks)
Port id	EgressBridgingUnitTimeTriggeredScheduleEntry.getPortId()	

6.3.2.1.5 *TT schedule for the serialization unit*

Number of CSV files	At most one, for unique node with Noc, if present.
Scope of a CSV file	The unique node with Noc in the simulated system.

File name rules	TTSchedule_SU.csv
Cell separator	,
MM Package	eu.dreamsproject.psm.onchip.com.model
MM Namespace	http://www.dreamsproject.eu/psm/onchip/com

File Header
N/A

Line		
MM EClass	SerializationUnitTimeTriggeredScheduleEntry (contained by SerializationUnitSchedule)	
Column title	DREAMS meta-model	Conversion MM → CSV
Timely lock/shuffling	OnChipNiLrsConfiguration. mediaAccessConflictResolutionPolicy	MediaAccessConflictResolutionPolicy .TIMELY_BLOCK → 1 MediaAccessConflictResolutionPolicy .SHUFFLING → 0
Tile id	SerializationUnitTimeTriggeredScheduleEntry.getTile Id()	
Period	SerializationUnitTimeTriggeredScheduleEntry.getPeri od_s()	× OnChipNetworkConfiguration. simulationTicksPerSecond (from s to simulation ticks)
Opening phase	SerializationUnitTimeTriggeredScheduleEntry.getOpen ingPhase_s()	See above
Closing phase	SerializationUnitTimeTriggeredScheduleEntry.getClos ingPhase_s()	See above

Notes: Since the media access conflict resolution policy can be configured per NI LRS, the setting can be obtained in OnChipNiLrsConfiguration that references the corresponding Tile.

6.3.2.1.6 *Message configuration parameters*

Number of CSV files	Exactly one.
Scope of a CSV file	The simulated system.
File name rules	Msg.csv
Cell separator	,
MM Package	eu.dreamsproject.psm.simulation.model
MM Namespace	http://www.dreamsproject.eu/psm/simulation

File Header
N/A

Line		
MM EClass	MessageConfigurationItem (contained by MessageConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV
Message Id	MessageConfigurationItem.getMessageId()	
Type	MessageConfigurationItem.getTrafficType()	TrafficType.TIME_TRIGGERED → TT TrafficType.RATE_CONSTRAINT → RC TrafficType.BEST_EFFORT → BE
VLID	MessageConfigurationItem.getVirtualLinkId()	
Deadline	MessageConfigurationItem.deadline_s	× OnChipNetworkConfiguration. simulationTicksPerSecond (from s to simulation ticks)
MAX_Msg_Size	MessageConfigurationItem.getPayloadSize_bytes()	

6.3.2.1.7 Trace file

The purpose of this configuration file is to specify a scenario of the behavior of the applications with respect to the sending of messages. For this purpose it defines sequence of “message injection times”.

If the application is modeled in DREAMS as well as the scheduling of tasks, then the injection times of messages sent by periodic tasks can be generated for a given time horizon. Otherwise, the injection times need to be generated with a random number generator according to some pattern. But which patterns make sense also depend on the actual demonstrator applications and will therefore only be covered in D5.2.3 through a dedicated extension of the configuration file generator.

Number of CSV files	Exactly one.
Scope of a CSV file	The simulated system.
File name rules	Trace.csv
Cell separator	,
MM Package	eu.dreamsproject.psm.simulation.model
MM Namespace	http://www.dreamsproject.eu/psm/simulation

File Header
N/A

Line		
MM EClass	TraceConfigurationItem (contained by TraceConfiguration)	
Column title	DREAMS meta-model	Conversion MM → CSV
Tile ID	TraceConfigurationItem.getTileId()	
Tick	TraceConfigurationItem.getInjectionTime_s() or random value (see notes)	× OnChipNetworkConfiguration.simulationTicksPerSecond

MsgId	TraceConfigurationItem.getMessageId()	
Port Id	TraceConfigurationItem.getPortId()	
Logical address of Destination	TraceConfigurationItem. getDestinationsLogicalNameList().get(0)	
Payload	N/A	

Notes:

- The trace related extension of the configuration file generator (D5.2.3) will generate a line for each injection of an instance of a message, with appropriate values for “Tick” (injection time) and “Payload”.
- In case the “Tick” value should be derived from the meta-model, TraceConfigurationItem.resourceAllocation must point to a ResourceAllocation in an appropriate ResourceSchedule (e.g., a task schedule that contains the scheduling of the sender task of the message identified by TraceConfigurationItem.getMessageId()).

6.4 Configuration Tools for the Execution Level

6.4.1 Xtratum

Since Xtratum configuration files are the same for the physical and virtual platform the detailed specification of the configuration file generator is provided only in one deliverable, namely D4.2.1.

7 Analysis of Simulation Traces

In this section we describe the functionalities implemented for the analysis and visualization of communication traces produced by the DREAMS simulator. Further extensions, related to fault injection for instance, will be described in the next deliverable D5.2.3.

We provide in Section 7.1 a refined and extended specification of the communication trace file formats. Sections 7.2 and 7.3 respectively explain how to import the description of the simulated system and the trace files produced by the simulators. Section 7.4 finally describes how the traces and the derived statistics can be visualized.

7.1 Trace files

In this section we provide a refined and extend specification of the format of traces files related to off-chip-network communication and on-chip communication. It replaces the specification dedicated to these trace files in D5.2.1.

Furthermore, all trace files need to be generated in the same folder and their names must be conformant to the following schema:

Trace file	Schema
Off-chip network	off-chip-com_<off-chip network name>.txt
On-chip communication	on-chip-com_<chip name>.txt

7.1.1 Off-chip network related events

With respect to D5.2.1, we have added events at router level to be able to trace the different delays that occur during the traversal of the off-chip network. Furthermore, the FrameQueued event is added at gateway level to distinguish the time when a frame is queued, from the time where its transmission actually starts.

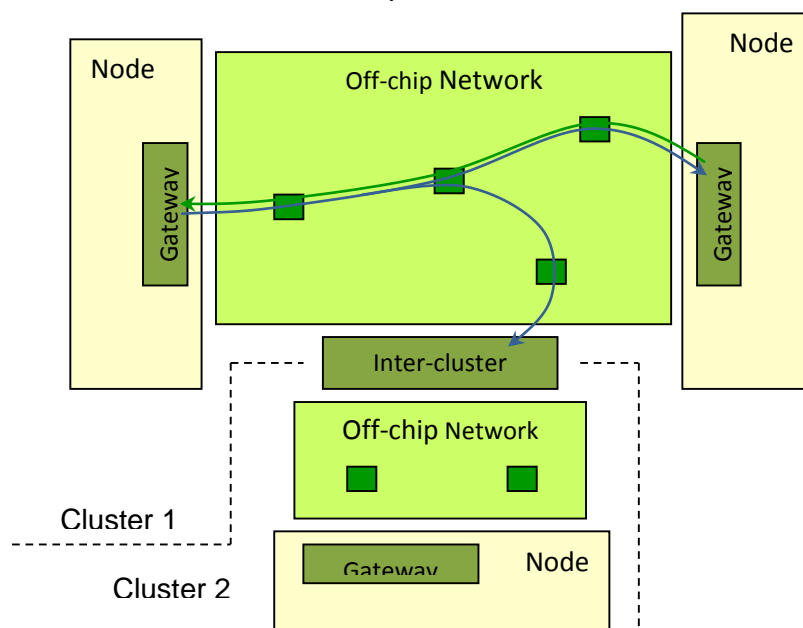


Figure 18: Illustration of off-chip network related events to be traced

Recall that for each off-chip network, the relevant events are written to a separate file. The events concern the emission and reception of frames between on-chip / off-chip gateways and inter-cluster gateways. The scope is illustrated in **Fehler! Verweisquelle konnte nicht gefunden werden.:**

Five different kinds of events are considered. Their trace file syntax is as follows:

```
<TIME> FrameQueued Gateway <GATEWAY_ID> <FRAME_ID> <FRAME_INSTANCE_ID>
```

```
<TIME> FrameTx Gateway <GATEWAY_ID> <FRAME_ID> <FRAME_INSTANCE_ID>
```

```
<TIME> FrameRx Gateway <GATEWAY_ID> <FRAME_ID> <FRAME_INSTANCE_ID>
```

```
<TIME> FrameRx Router <ROUTER_PORT_ID> <FRAME_ID> <FRAME_INSTANCE_ID>
```

```
<TIME> FrameTx Router <ROUTER_PORT_ID> <FRAME_ID> <FRAME_INSTANCE_ID>
```

With respect to the loss of frames due to insufficient memory in the routers, the following event is defined:

```
<TIME> FrameDiscarded <ROUTER_ID> <FRAME_ID> <FRAME_INSTANCE_ID>
```

The semantics and precise formats are defined in the following table:

Line item	Description	Format / Value
<TIME>	Time when the event occurred.	Time unit: 1 tick = 1 ns
FrameQueued Gateway	A frame instance is queued for transmission over the off-chip network, as soon as allowed by the protocol. Depending on the traffic class and priority, the frame is either immediately transmitted (time triggered) or may wait in a queue (rate constraint or best effort).	Verbatim.
FrameTx Gateway	Effective start of the transmission of a frame instance. It might have waited in the queue or not.	Verbatim.
FrameRx Gateway	Frame instance has arrived at a gateway over the off-chip network and the contained messages are ready for being forwarded inside the connected node or over the off-chip network of the connected cluster.	Verbatim.
FrameRx Router	Frame instance has arrived at a router port.	Verbatim
FrameTx Router	Effective start of the transmission of a frame instance. It might have waited in the queue or not.	Verbatim

<GATEWAY_ID>	Identifier of the concerned gateway.	Numerical ID of the node and the tile that hosts the gateway: <Node_ID>.<Tile_ID>
<ROUTER_PORT_ID>	Identifier of the concerned router port.	Name the of Router (i.e. not a numerical ID) and numerical ID of the port: <Router Name>.<Port_ID>
<FRAME_ID>	Identifier of the concerned frame.	Numerical ID of the DREAMS VL transported by the frame.
<FRAME_INSTANCE_ID>	Identifier of the frame instance. This identifier is incremented with each "FrameTx Gateway" event.	Sequence number of the off-chip frame.

7.1.2 On-chip network related events

With respect to D5.2.1, the MessageQueued event is added at partition port level to distinguish the time when a message is queued, from the one where its transmission actually starts.

Recall that for each on-chip network, the relevant events are written to a separate file. These events concern the emission and reception of messages between NI ports and the on-chip / off-chip gateways. The scope is illustrated in Figure 19:

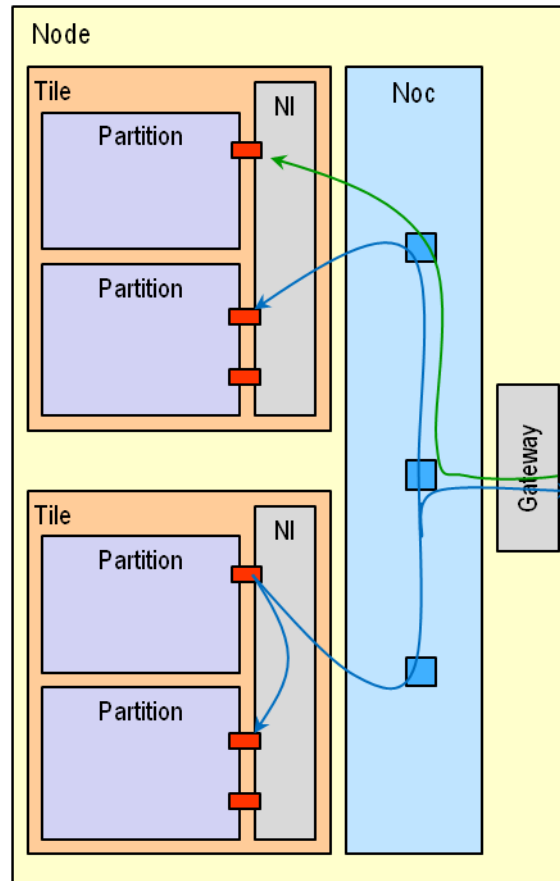


Figure 19: Illustration of on-chip network related events to be traced

Four different kinds of events can be identified. Their trace file syntax is as follows:

```
<TIME> MessageQueued OutPort <PORT_ID> <MESSAGE_ID> <MESSAGE_INSTANCE_ID>
<TIME> MessageTx OutPort <PORT_ID> <MESSAGE_ID> <MESSAGE_INSTANCE_ID>
<TIME> MessageRx InPort <PORT_ID> <MESSAGE_ID> <MESSAGE_INSTANCE_ID>
<TIME> MessageTx Gateway <GATEWAY_ID> <MESSAGE_ID> <MESSAGE_INSTANCE_ID>
<TIME> MessageRx Gateway <GATEWAY_ID> <MESSAGE_ID> <MESSAGE_INSTANCE_ID>
```

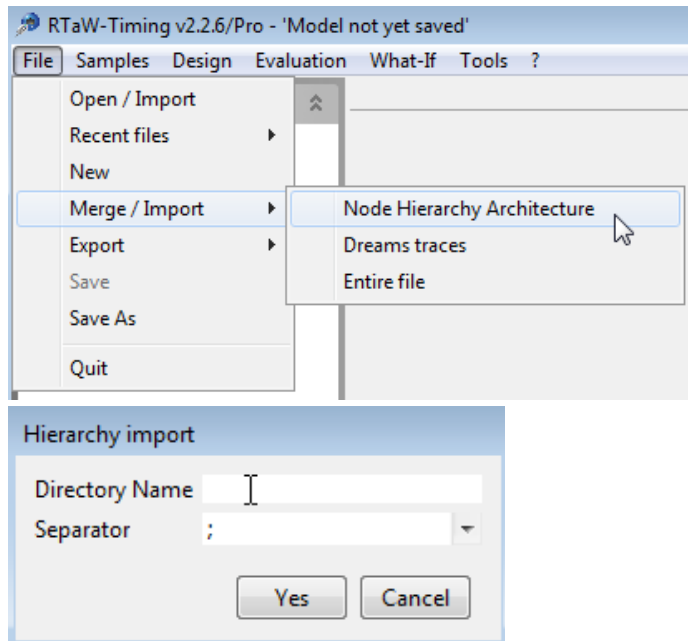
The semantics and precise formats are defined in the following table:

Line item	Description	Format / Value
<TIME>	Time when the event occurred.	Time unit: 1 tick = 1 ns
MessageQueued	A message instance has been written to the port, and is thus ready for being sent, as soon as the network interface foresees it (TT) or allows it (RC). It corresponds to the „tick“ column in trace configuration file of the virtual platform.	Verbatim.
MessageTx OutPort	A message instance, contained in the output port, is emitted. Depending on	Verbatim.

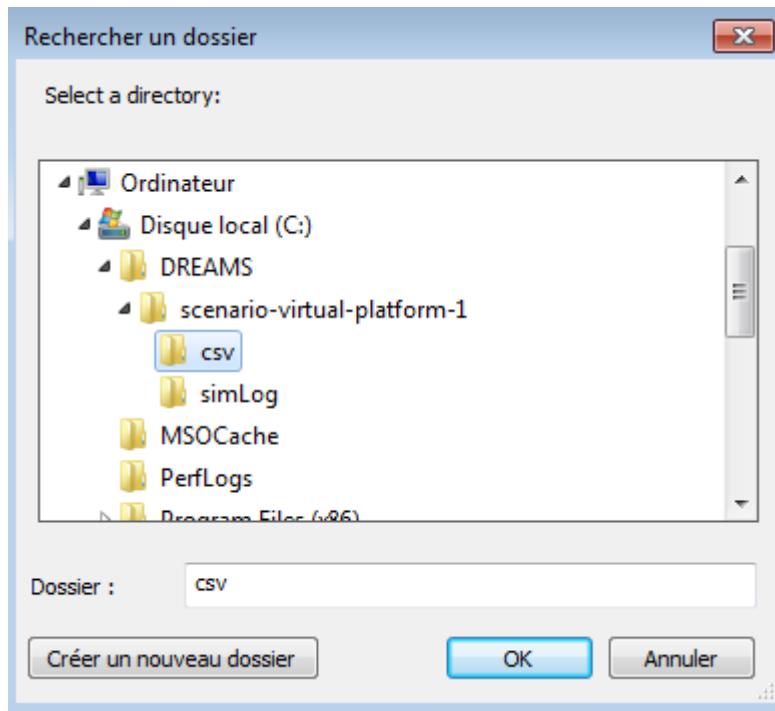
	<p>the location of the destination and the optional presence of a NOC, one or several of the following happen after the event has occurred:</p> <ul style="list-style-type: none"> • copying to destination ports on the same Tile • if NOC: reading by the bridging layer for further processing in order to sent the message over the NOC • if no NOC: reading by the on-chip / off-chip gateway for transmission over the off-chip network 	
MessageRx InPort	A message instance is ready in the input port for being read by tasks in the related partition.	Verbatim.
MessageRx Gateway	Message instance, coming from inside the chip (over the NOC or not) is ready for being put into a frame for sending over the off-chip network.	Verbatim.
MessageTx Gateway	<p>Message instance, coming from the off-chip network, is forwarded inside the chip:</p> <ul style="list-style-type: none"> • if NOC: reading by the bridging layer for further processing in order to sent the message over the NOC • if no NOC: copying to destination ports 	Verbatim.
<PORT_ID>	Hierarchical identifier of the concerned port, without Node and Cluster part, since the scope of the trace file is limited to a Node.	Numerical ID of the tile and the port: <Tile ID>.<Port ID>
<GATEWAY_ID>	Identifier of the Gateway.	Numerical ID of the tile that hosts the gateway.
<MESSAGE_ID>	Hierarchical identifier of the concerned message.	Numerical ID of the message exchanged between applications.
<MESSAGE_INSTANCE_ID>	Identifier of the message instance. This identifier is incremented with each "MessageTx OutPort" event.	Sequence number of the off-chip frame.

7.2 Import of DREAMS System description

In this section we describe how to import the description of a DREAMS model into RTaW-Timing. The screen-shots are based on the “Virtual-Platform scenario 1”, described in Section 6.1.



To be able to perform the import and analysis of DREAMS simulation traces, the description of the simulated system, needs first to be imported into RTaW-Timing. For this purpose, an “RTaW-Timing CSV file” exporter is developed in T4.4 as part of the tool-chain, see D4.4.1.

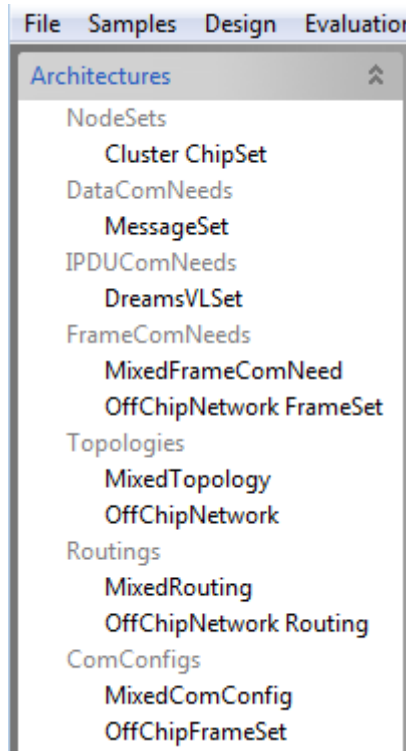


Before starting to import the system description, make sure to start with an empty model (use “New” from the files menu, if needed). In order to import the description of the

simulated system, select the “Node Hierarchy Architecture” entry from the “Merge / Import” sub-menu:

Leave “,” as separator and left-click in the “Directory Name” field to bring up the “folder selection dialog”:

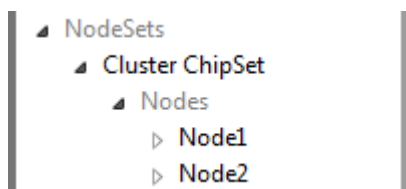
Then, navigate to the folder where the csv files have been produced by the “RTaW-Timing CSV exporter” and select that folder. Make sure that only the csv files produced by the “RTaW-Timing CSV exporter” are present in that folder. Then click “Ok” and “Yes” in order to execute the import:



Depending on your OS, you might have to manually open the nodes in the exportation tree, in order to see the created entities, which we will describe below.

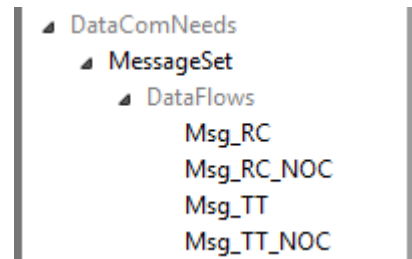
All entities whose name start by “Mixed”, are used in RTaW-Timing to describe the complete configuration that has been simulated.

The NodeSet “Cluster ChipSet” groups the chips that are present in the modeled system. In the shown example we have two chips called “Node1” and “Node2”:

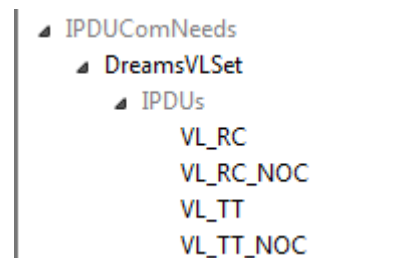


The DataComNeed “MessageSet” is the inventory of the messages that are exchanged between applications. In this example we have in principle one TT and one RC message, but

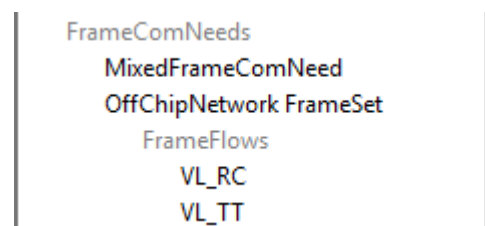
because the simulator does not support multicast at on-chip level, these message are duplicated so as to use “multi-unicast” instead.



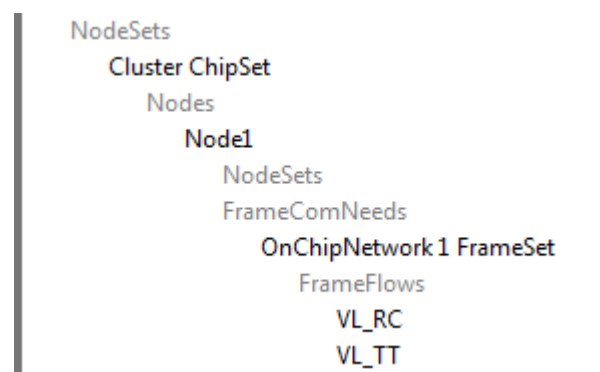
The IPDUComNeed “DreamsVLSet” is the inventory of the DREAMS Virtual Links used in the simulation to transport the messages.



The FrameComNeed “OffChipNetwork FrameSet” is the inventory of the frames used to transport the DREAMS Virtual Links over the off-chip network:



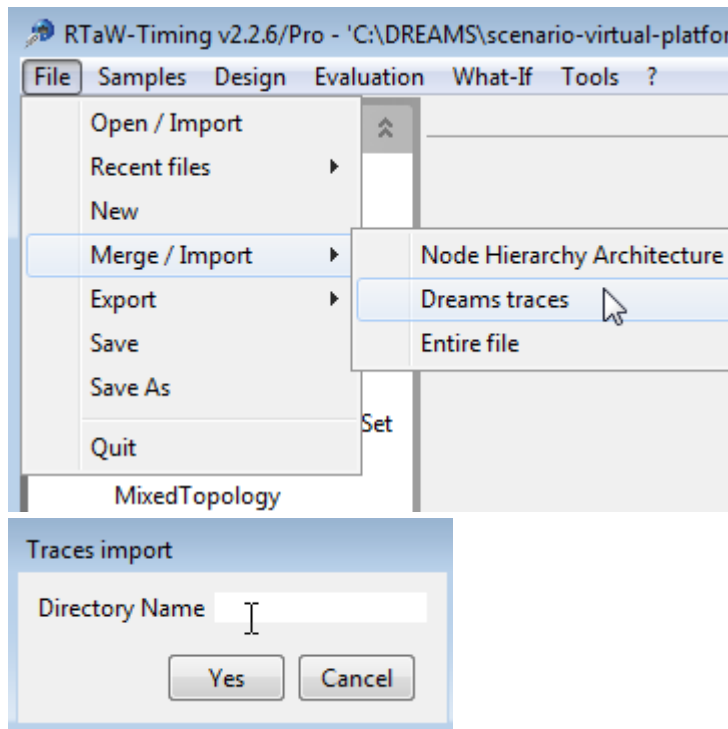
The FrameComNeed “OnChipNetwork 1 FrameSet” is the inventory of the frames used to model in RTaW-Timing the transportation of the DREAMS Virtual Links over the on-chip network:



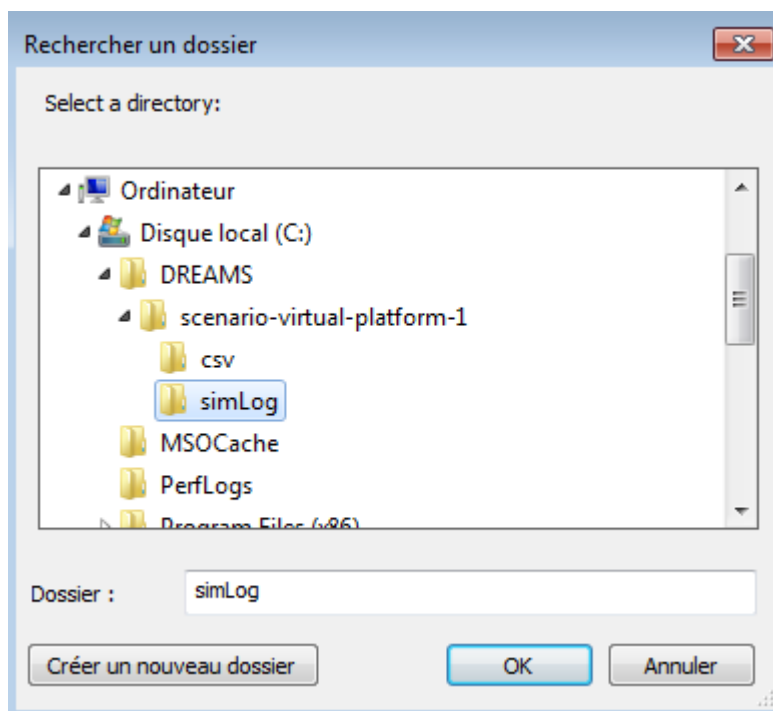
7.3 Import of virtual platform traces

Notice that before being able to import simulation traces, the description of the simulated system must be contained in the currently opened model (see Section 7.2).

In order to import simulation traces, select the “Dreams traces” entry from the “Merge/Import” sub-menu:

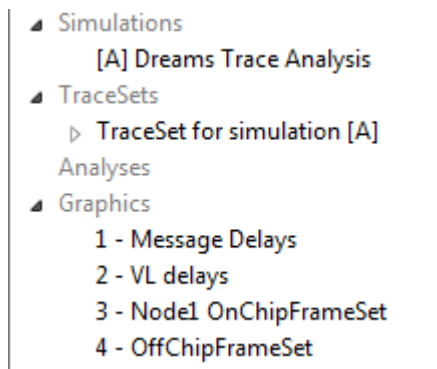


Left-click in the “Directory Name” field to bring up the “folder selection dialog”:



Navigate to the folder where the trace files are located and select that folder. Then click “Ok” and “Yes” in order to execute the import. The result is

- a “Simulation” entity that contains the derived statistics
- a “Trace” entity that contains composed Gantt charts, build from the traces
- several “Graphic” entities, that show delay under the form of graphs



In the next section is described how the synthesized statistics and Gantt charts can be visualized.

7.4 Visualization of delays synthesized from the traces

The derived statistics can be visualized in three ways: as tables, as histograms and as graphics. Furthermore, Gantt chart are available that show the traces from a particular point of view.

7.4.1 Delay tables

DREAMS Messages are modeled as DataFlows and therefore their transmission delay statistics are visible in the details pane of the DataComNeed entity “MessageSet” created during import. Double-click on the “MessageSet” node in the exploration tree on the left and select a simulation:

MessageSet ⓘ

Name* MessageSet

SRMap MessageSRMap

SystemLayout Implicite Node Hierarchy

Simulations [A] Dreams Trace Analysis

Sample Time 31ms 437µs

Name	Sender	Receiver	Min	Average	Q4	Q5	Q6	Max	Constraint
Msg_RC	Node1/Tile 1-1	Node2/Tile 2-1	2,928 ms	5,712 ms				7,742 ms	By 'Tile 2-1' in 10 ms
Msg_RC_NOC	Node1/Tile 1-1	Node1/Tile 1-2	1,114 ms	1,760 ms				2,680 ms	By 'Tile 1-2' in 5 ms
Msg_TT	Node1/Tile 1-1	Node2/Tile 2-1	8,137 ms	8,193 ms				8,234 ms	By 'Tile 2-1' in 10 ms
Msg_TT_NOC	Node1/Tile 1-1	Node1/Tile 1-2	3,581 ms	4,058 ms				4,362 ms	By 'Tile 1-2' in 5 ms

These message delays span from the time when a message is written to a partition port until the time when it is ready for reading in the destination partition port.

DREAMS Virtual Links are modeled as IPDUFlows and thus the transmission delay statistics are visible in the details pane of the IPDUComNeed entity “DREAMSVLSet” created during import. Double-click on the “DREAMSVLSet” node in the exploration tree on the left and select a simulation:

DreamsVLSet ⓘ

Name*

DreamsVLSet

SenderReceiverMaps

DreamsVLSRMap

SystemLayout

Implicite Node Hierarchy

Simulations

[A] Dreams Trace Analysis

Sample Time

31ms 437µs

Name	Payload	Min...	Period	Sender	Receiver	Min	Average	Max	Constraint	
VL_TT_NOC	100 byte		10 ms	Tile 1-1	Tile 1-2	0,271 ms	0,802 ms	1,214 ms	By 'Tile 1-2' in 5 ms	
VL_TT	100 byte		10 ms	Tile 1-1	Tile 2-1	4,828 ms	4,937 ms	5,060 ms	By 'Tile 2-1' in 10 ms	
VL_RC_NOC	200 byte	15 ms		Tile 1-1	Tile 1-2	0,477 ms	1,011 ms	1,745 ms	By 'Tile 1-2' in 5 ms	
VL_RC	200 byte	15 ms		Tile 1-1	Tile 2-1	2,626 ms	4,963 ms	6,807 ms	By 'Tile 2-1' in 10 ms	

These Virtual Link delays span from the time when the virtual link instance is created by the NI on emission port side, until the time when its contained message is ready for reading in the destination partition port.

Delays suffered by DREAMS Virtual links on the On-chip network, are associated to the corresponding on-chip frames and are therefore visible in the details pane of the FrameComNeed entity “OnChipNetwork 1 FrameSet” created during import under the node that has a Noc. Double-click on the “OnChipNetwork 1 FrameSet” node in the exploration tree on the left, below “Node1” (see end of Section 7.2) and select a simulation:

OnChipNetwork1 FrameSet ⓘ

Name* OnChipNetwork1 FrameSet NodeSet* TileSet SRMap Main Sender/Receiver M ▾

Partitionning Priorities ▾

Analysis ▾

Simulations [A]A OnChipNetwork1 FrameSet ▾ Sample Time 31ms 437µs ▾

End-To-End Delays Local Delays

Name	P...	MinD...	MaxSize	Sender	Receiver	Min	Average	Q4	..	Max	Bound	Constraint
VL_RC	2	15 ms	200 byte	Tile 1-1	Tile 1-2	0,477 ms	1,011 ms			1,745 ms		By 'Tile 1-2'
VL_RC	2	15 ms	200 byte	Tile 1-1	Tile 1-3	0,025 ms	0,492 ms			0,742 ms		By 'Tile 1-3'
VL_TT	1	10 ms	100 byte	Tile 1-1	Tile 1-2	0,271 ms	0,802 ms			1,214 ms		By 'Tile 1-2'
VL_TT	1	10 ms	100 byte	Tile 1-1	Tile 1-3	0,127 ms	0,295 ms			0,558 ms		By 'Tile 1-3'

These on-chip delays span from the time when the virtual link instance is handled over to the Noc on emission side, until the time when its contained message is ready for reading in the destination partition port.

Off-chip network frames are modeled as FrameFlows at root level and thus the transmission delay statistics are visible in the details pane of the FrameComNeed entity “OffChipNetwork FrameSet” created during import at root level. Double-click on the “OffChipNetwork FrameSet” node under the “FrameComNeed” node in the exploration tree on the left:

OffChipNetwork FrameSet ⓘ

Name* OffChipNetwork FrameSet NodeSet* Cluster ChipSet SRMap Main Sender/Receiver M ▾

Partitionning Priorities ▾

Analysis ▾

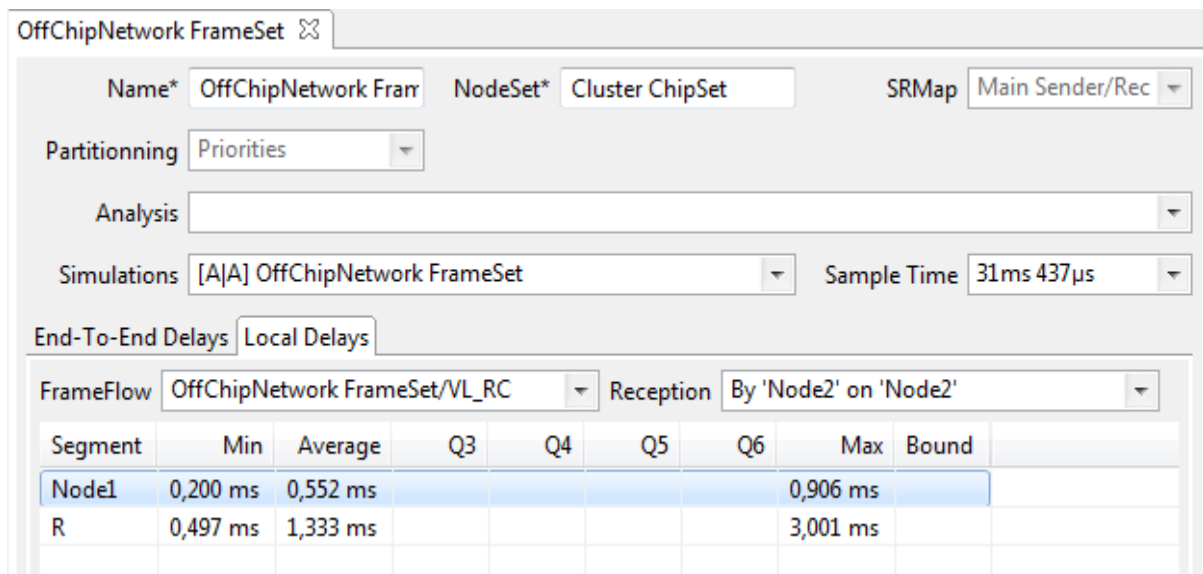
Simulations [A]A OffChipNetwork FrameSet ▾ Sample Time 31ms 437µs ▾

End-To-End Delays Local Delays

Name	Pri...	MinDis...	MaxSize	Sender	Recei...	Min	Average	..	Max	Bound	Constraint
VL_TT	1	10 ms	100 byte	Node1	Node2	0,876 ms	1,148 ms		1,567 ms		By 'Node2'
VL_RC	2	10 ms	100 byte	Node1	Node2	0,697 ms	1,885 ms		3,907 ms		By 'Node2'

The “End-To-End Delays” table shows statistics on network traversal times, which span from the time when the off-chip frame instance is created and ready for transmission, until the time when it is completely arrived in the destination node.

A simple-click on the “Local Delays” table brings to the front the table that shows statistics on the “network segment traversal times”:



OffChipNetwork FrameSet

Name* OffChipNetwork Fram NodeSet* Cluster ChipSet SRMap Main Sender/Rec

Partitioning Priorities

Analysis

Simulations [A] OffChipNetwork FrameSet Sample Time 31ms 437µs

End-To-End Delays Local Delays

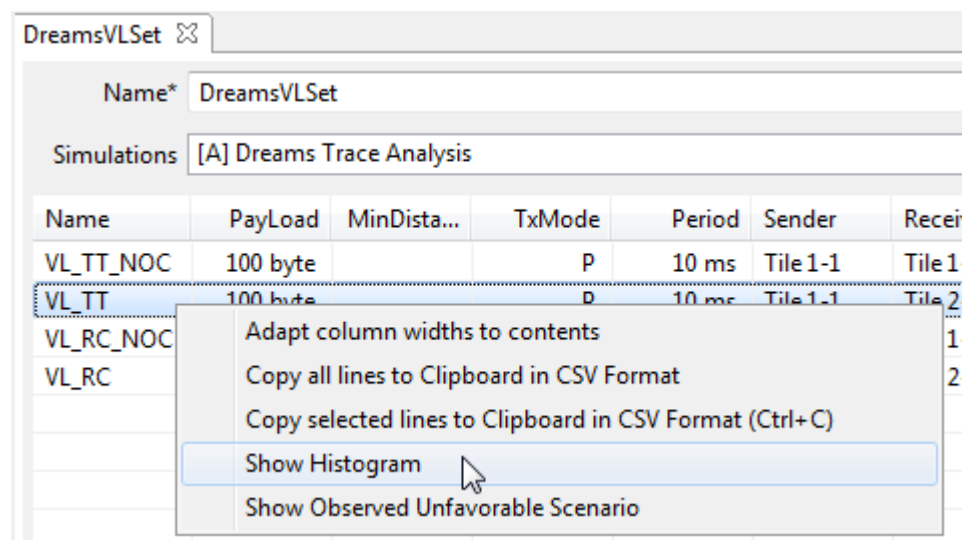
FrameFlow OffChipNetwork FrameSet/VL_RC Reception By 'Node2' on 'Node2'

Segment	Min	Average	Q3	Q4	Q5	Q6	Max	Bound
Node1	0,200 ms	0,552 ms					0,906 ms	
R	0,497 ms	1,333 ms					3,001 ms	

The “Segment” column shows the PortOwner (Node or Router) containing the port through which the frame is transmitted over a link to the next PortOwner in the routing path. The delay spans from the time when the frame has completely arrived in the indicated PortOwner until the time when it is complete arrived in the following PortOwner. This view allows indentifying which network segments induce the highest delays.

7.4.2 Delay histograms

By right-clicking on a line in a delays table (see previous section) one can bring up the following context menu:



DreamsVLSet

Name* DreamsVLSet

Simulations [A] Dreams Trace Analysis

Name	PayLoad	MinDista...	TxMode	Period	Sender	Recei
VL_TT_NOC	100 byte		P	10 ms	Tile 1-1	Tile 1
VL_TT	100 byte		P	10 ms	Tile 1-1	Tile 2
VL_RC_NOC						1
VL_RC						2

Adapt column widths to contents

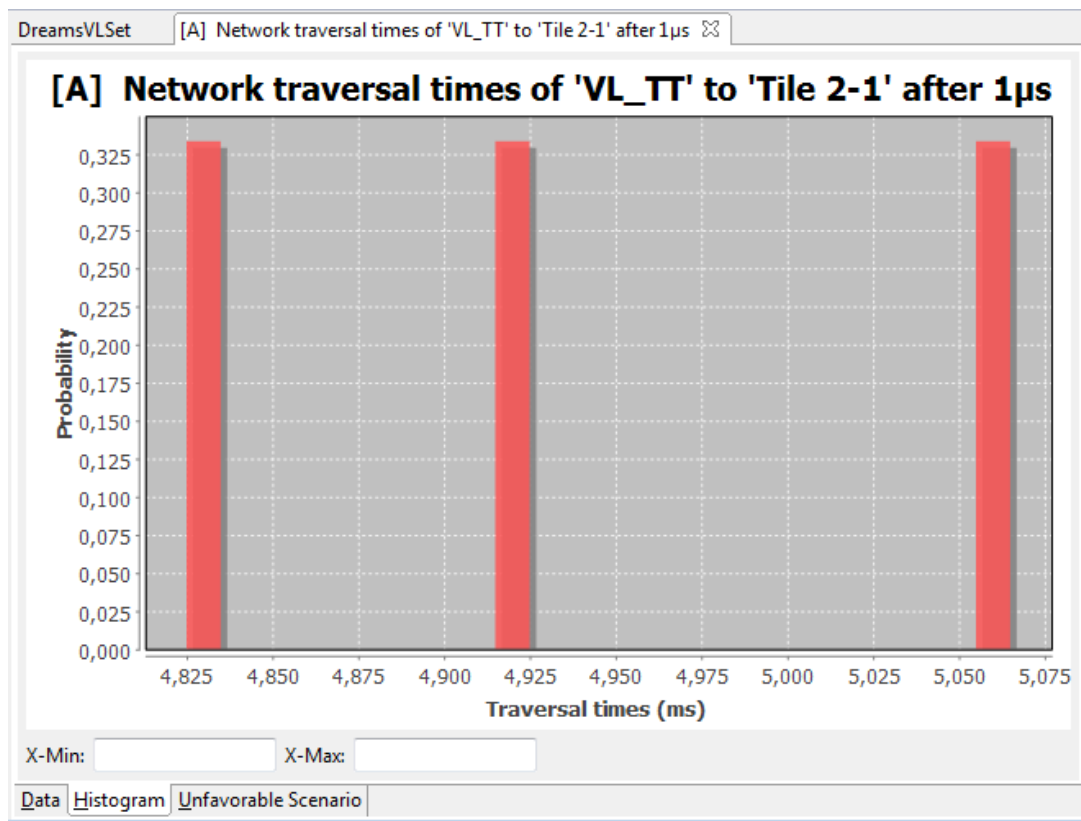
Copy all lines to Clipboard in CSV Format

Copy selected lines to Clipboard in CSV Format (Ctrl+C)

Show Histogram

Show Observed Unfavorable Scenario

Select the “Show Histogram” entry, in order to open the graphical view of the histogram:



As can be seen by clicking on the “Data” tab (lower left corner) one can see that in this small example there are only three values in the histogram:

DreamsVLSet [A] Network traversal times of 'VL_TT' to 'Tile 2-1' after 1 μ s

Reception* By 'Tile 2-1' on 'Tile 2-1'

Min* 4,828 ms Average* 4,937 ms

Q2 5,060 ms Q3

Q4 Q5

Q6 Max* 5,060 ms

SampleSize* 3

Entries	Value	Width	Count
	4830	10	1
	4920	10	1
	5060	10	1

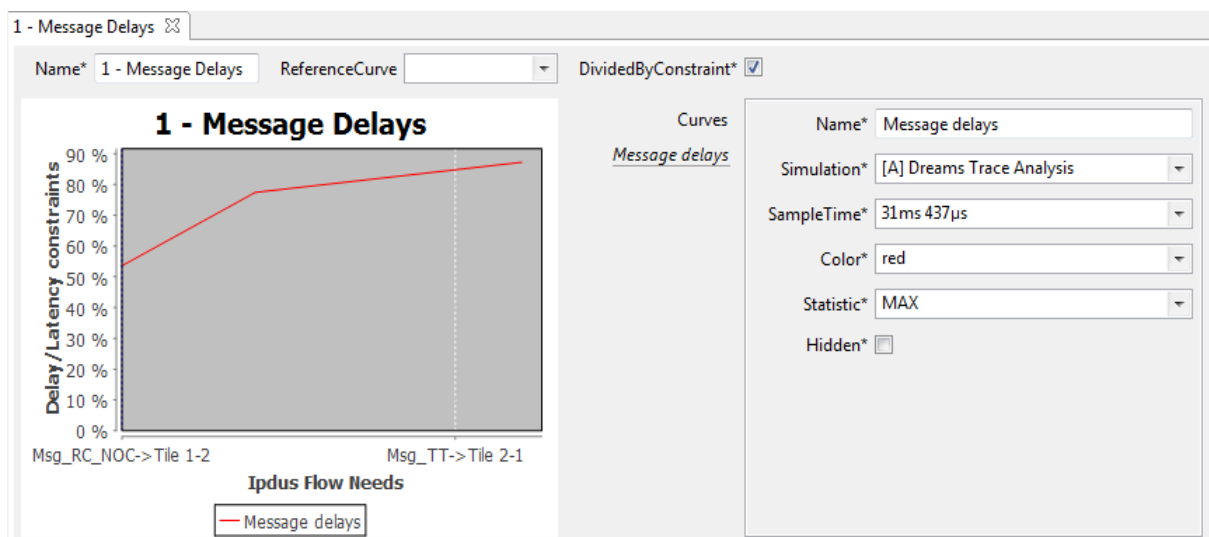
Data Histogram Unfavorable Scenario

7.4.3 Delay graphs

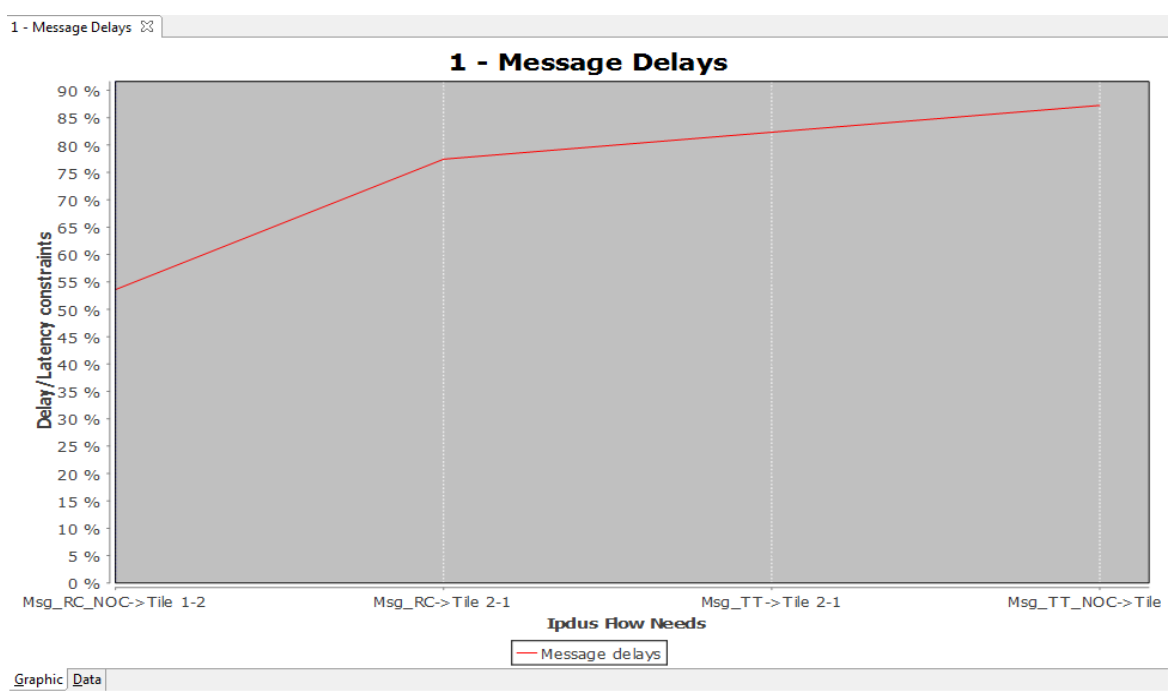
Recall that during the import of traces, some graphics have been created automatically:

- ▲ Graphics
 - 1 - Message Delays
 - 2 - VL delays
 - 3 - Node1 OnChipFrameSet
 - 4 - OffChipFrameSet

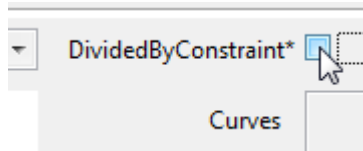
In order to visualize them, one must double-click on the corresponding node in the exploration tree on the left:



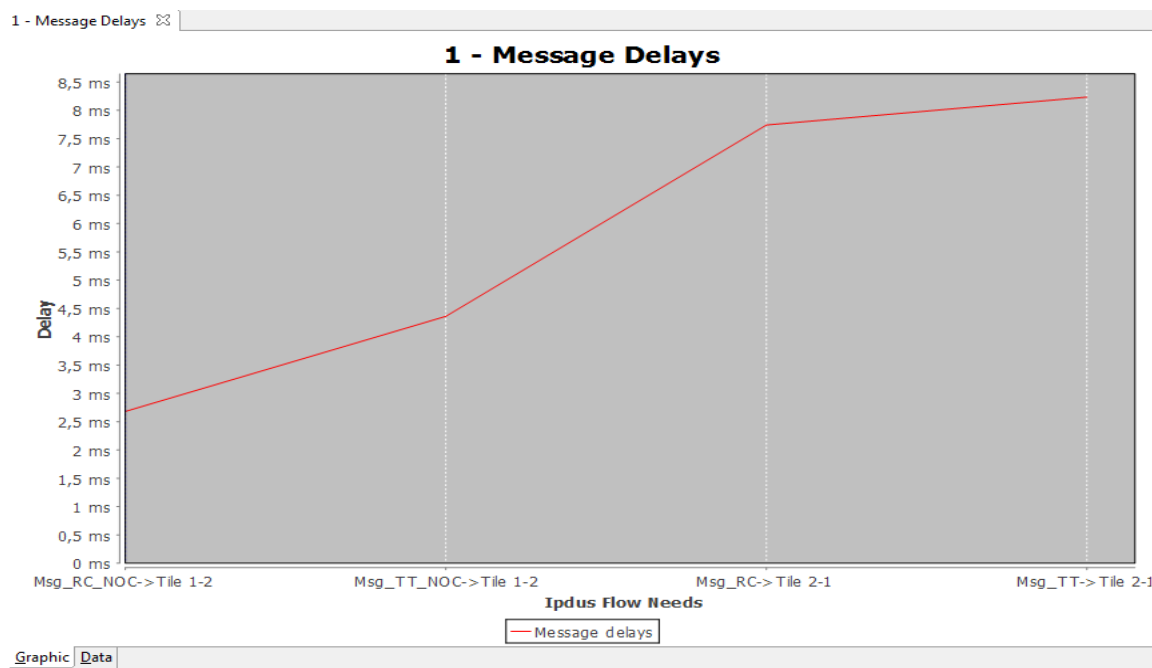
Notice that by default, the “Data” table is shown; click on the “Graphic” tab in the lower left corner to get the graphic in full size:



Since “DividedByConstraint” is checked, the y-axis shows the percentage of the delay with respect to the latency constraint. A value below 100% means that the constraint is satisfied. If you uncheck “DividedByConstraint”,



then the delays are displayed in ms:

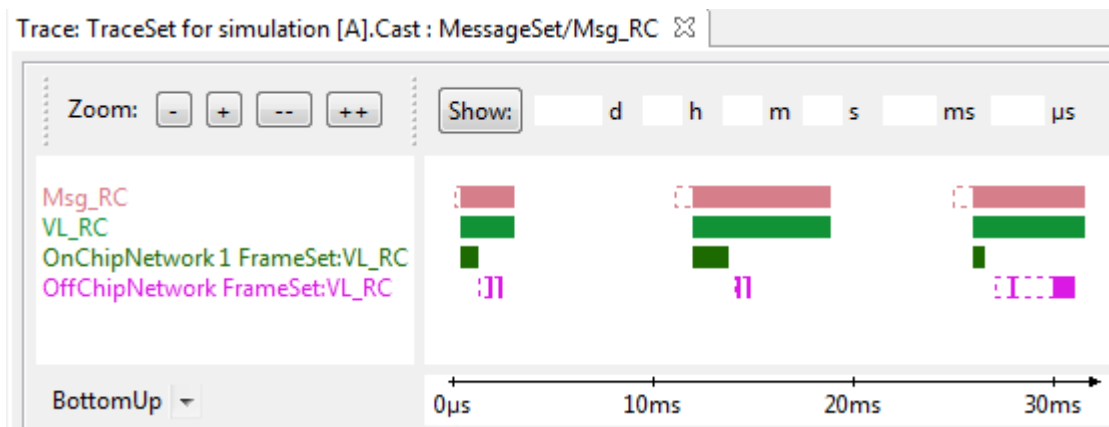


7.4.4 Gantt charts

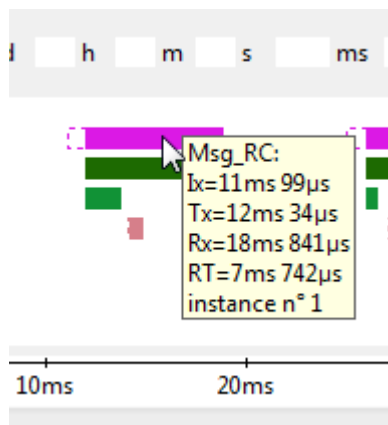
Three kinds of Gantt charts are created by the import under the form of traces:

- ▲ TraceSets
 - ▲ TraceSet for simulation [A]
 - ▲ Traces
 - Trace: Cast : MessageSet/Msg_RC
 - Trace: Cast : MessageSet/Msg_RC_NOC
 - Trace: Cast : MessageSet/Msg_TT
 - Trace: Cast : MessageSet/Msg_TT_NOC
 - Trace: MessageVLPacking
 - Trace: VLFramePacking

For each message a Gantt chart is created, which shows not only each message instance, but also the VL and frame instances used to transport it over the on-chip and off-chip network:



The screen shot shows the Gantt chart for “Msg_RC”, where three instances are visible. The sequence number (instance n° ...) is shown in the tool tip of the instance. To bring up a tool tip keep the pointer over the rectangle:



The information provided in the tool-tip have the following meaning:

Information	Meaning
Name	The name of the message or frame.
lx	Time when the message or frame is queued for transmission.
Tx	Time when the transmission starts.
Rx	Time when the message or frame has been completely received

The rectangles in the Gantt charts have to presentation styles, with the following meaning:

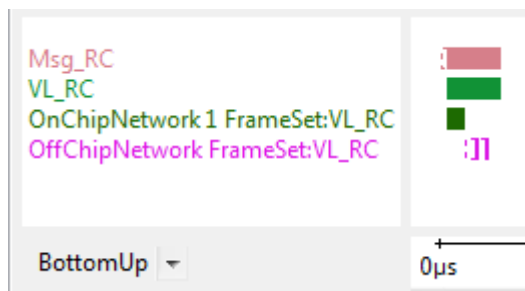
Presentation style	Meaning
Dotted line / empty rectangle	Waiting in a queue at the entrance point of the communication layer or in some intermediate element such as a router port.
Solid line / filled rectangle	Being transmitted.

Note: the waiting at the entrance point might not be known from the information available in a trace; in that case the “Solid line / filled rectangle” style is used for the entire communication time, as for example for the NOC.

By communication layer, is meant the following, depending on the entity:

Entity	Ingress point / waiting	Egress point
DREAMS Message	Source NI port	Destination NI port
DREAMS Virtual Link	Outgoing NI queue	Incoming NI queue
Off-chip frame	Source TTEthernet interface	Destination TTEthernet interface
	Emitting node or router port	Following receiving router or node port on the routing path
On-chip frame	Source NOC interface	Destination NOC interface

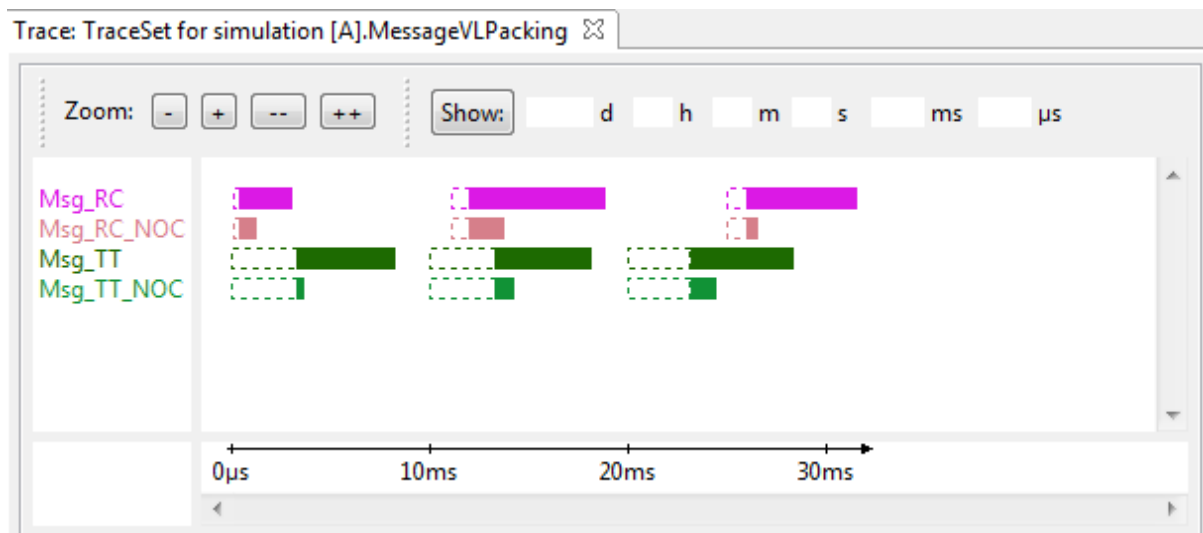
In the following Gantt chart extract we see the first instance of the message Msg_RC, together with the VL and frames used to transport it:



The following things can be observed:

- “Msg_RC”: the waiting in the “Source NI port” is known from the trace and thus a small “dotted line / empty rectangle” is shown. As soon as the NI takes the instance of Msg_RC into account, the rectangle is drawn filled until the arrival at the “Destination NI port”.
- “VL_RC”: There is no “dotted line” part, because the NI only creates VL instance, when their communication is allowed / foreseen.
- “OnChipFrameSet 1: VL_RC”: there is no “dotted line / empty rectangle” part, because no details are known from the trace about over the NOC transmission.
- “OffChipFrameSet: VL_RC”: two empty and two filled rectangles are shown. They correspond to the waiting and transmission times in the two network segments:
 - “Node1 → “R”
 - “R” → “Node2”

Another of the generated Gantt chart provides an overview of the communication of all messages:



8 Formal Verification Framework

8.1 Formal Verification Methodology

There are two distinct tasks involved in the verification of STNoC components that have to be performed at two different points in time. When designing a component, it must be verified that any configuration of this component that may be generated using the iNoC platform will be free of functional bugs. When using a component to create a NoC that meets the needs of a particular SoC, it must be verified that the configuration that has been generated is free of functional bugs in its utilization context.

The second task does not present uncommon challenges, but the first one definitely does due to the extremely large number of potential configurations of the components. Finding a bug in a component when generating a network for a SoC project could have an adverse impact. Correcting the component design could turn out to be a significant effort and a substantial re-verification effort could be required if interactions with other components are impacted by changes. As a result, it could be impossible to avoid delaying some milestones of the SoC project. Therefore, it is critical to verify as many configurations as possible during the design phase of the components, in the amount of time available before the platform gets used in SoC projects.

Using simulation to verify a component means developing a new testbench and a new set of tests for each type of configuration. Constrained random tests have to be used, which implies that adequate means to collect coverage metrics must be developed. Tests generation followed by coverage analysis loops are then required to gain sufficient confidence.

We estimated that formal verification would be better suited for the task for several reasons:

- The behavior of a component in its different types of configurations could be expressed with parameterized properties. The parameters used in the iNoC platform to generate a particular component configuration could also be used to configure the properties. A verification environment could then be readily available every time a new component configuration would be generated, with no setup time and effort.
- There would be no need to develop new tests to verify a new configuration. Only machine time would be required, thus saving precious verification resources.
- Exhaustive coverage of the configurations tested could be achieved, which is generally impossible with simulation given the available time and resources.

8.2 Implementation of the Formal Verification Methodology

The core of the formal verification methodology is based on the specification of a set of SVA properties for the STNOC, in particular to the communication part of the STNoC.

As in any network device, the communication in the STNoC is defined by a set of rules (STNoC protocol) that enable two or more STNoC components to communicate between them: transmit and receive information

There are essentially three parts in the STNoC protocol:

- The composition rules that specify the structure of a correct packet. The first flits of a packet convey packet routing and QoS/BE service information, and the following flits are either payload flits or ARM AMBA interconnect request/response flits.
- The sideband signals that are associated to flits. For example, a 3-bit signal indicates whether the flit that is present on the physical link is the first flit of a packet, an intermediate flit, or the last flit of a packet.
- The credit based control flow that ensures that an upstream component (router, AL, NI) only sends flits to a downstream component when it is ready to accept them, i.e. when it has space available for them in its input buffer.

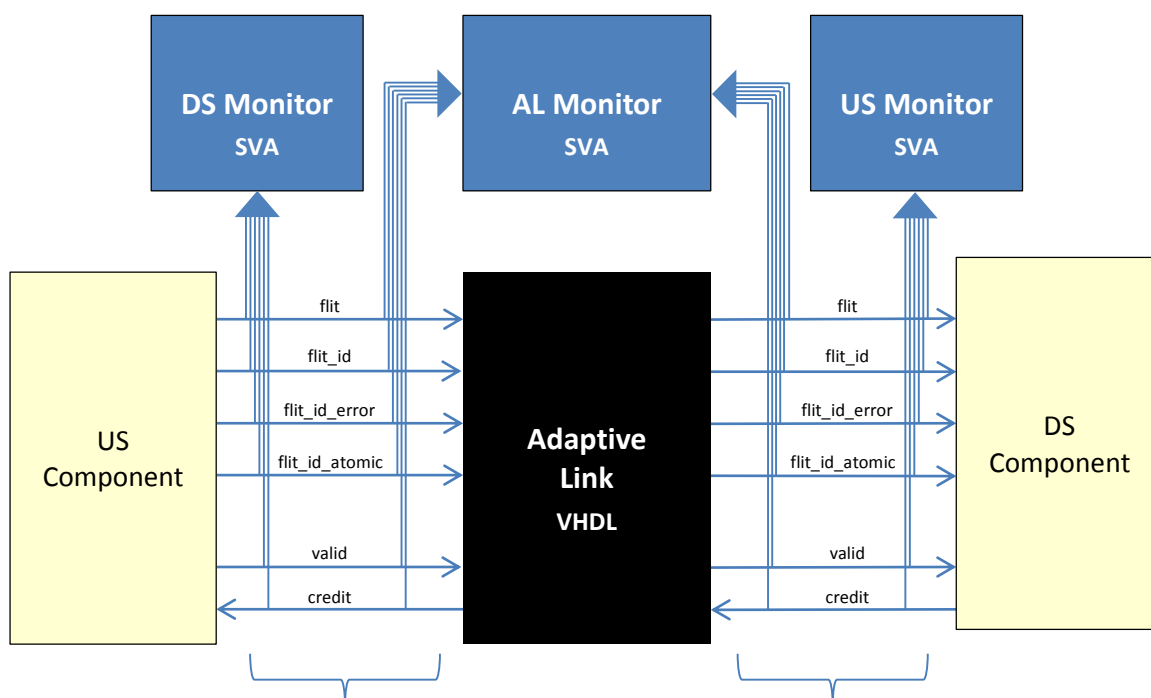
A set of SVA properties has been developed to model the communication protocol between an US component and a DS component over the physical link that connects them.

These properties are parameterized with the iNoC platform parameters that control the generation of STNoC components. Examples of parameters include size of the flit, length of the header, number of byte enable bits within a payload flit, presence/absence of optional sideband signals, etc. Because properties use the same parameters as the iNoC platform, a protocol checker is readily available every time STNoC components are generated.

RTL code has been written to keep track of the number of flits that the US component sends to the DS component and of the number of credits that the DS component sends to the US component. Properties make use of this RTL code to check that no overflow of the input buffer of the DS component can occur and that the valid/credit round trip delay is always within range.

In order to give a better idea what has been implemented, hereafter we provided a detailed description of the methodology by the Adaptive Link (AL) example. However considering the modularity of the STNoC technology the methodology applied to other components reflects the same approach.

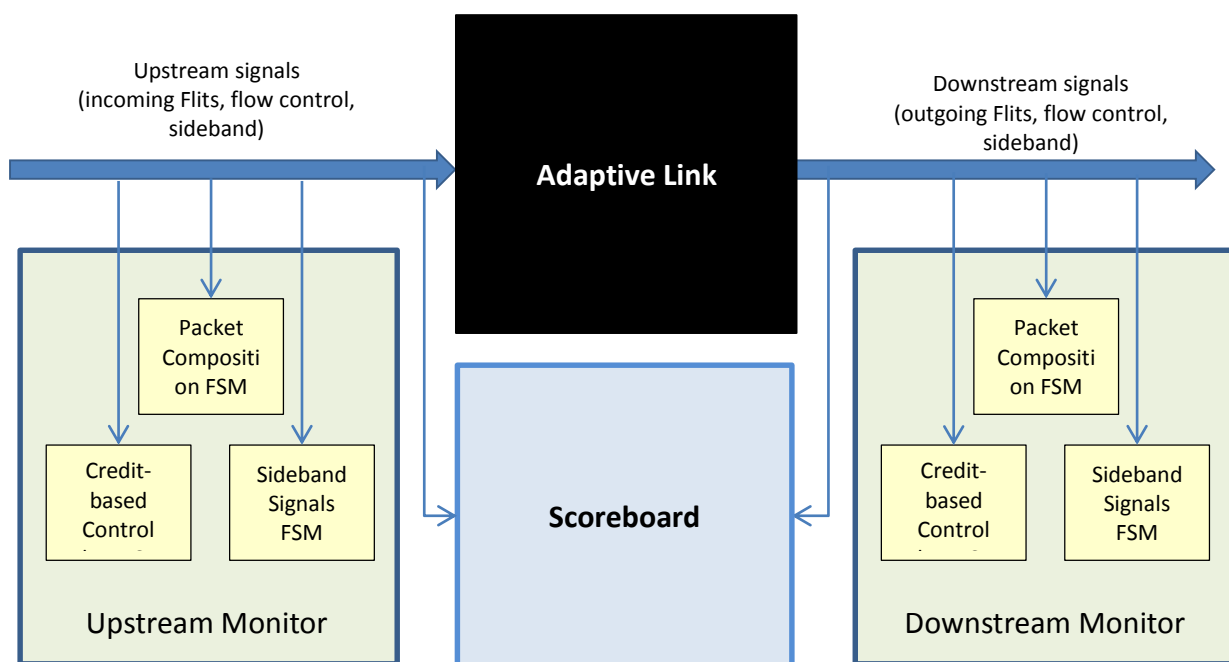
The Figure below illustrates the environment that has been developed to formally verify the AL.



The environment is a mix of VHDL and SystemVerilog elements. The RTL model of the AL is written in VHDL. The SVA properties used for formal verification are encapsulated in SystemVerilog modules that are attached to the AL entity using bind statements. The SystemVerilog modules only read input/output signals of the AL, so the approach is non-intrusive.

The AL is treated as a black-box for formal verification with no assumption made about its implementation. All properties are expressed in terms of input/output signals of the AL and no internal signal has been used. Properties have been developed using the functional specification of the AL only, which is the best possible approach to verification.

The following figure provides a more detailed view of the verification environment.



8.2.1 DS Monitor:

The role of the DS Monitor is to verify that the DS interface of the AL behaves properly when it communicates with the DS interface of a bug-free US component.

The DS Monitor module encapsulates the properties that model the communication protocol between a US component and a DS component over the physical link that connects them. All the properties that model the behavior of the DS interface of the US component are configured as constraints (assume properties) for formal verification and all the properties that model the behavior of the DS interface of the AL are configured as assertions (assert properties).

8.2.2 US Monitor:

The role of the US Monitor is to verify that the US interface of the AL behaves properly when it communicates with the US interface of a bug-free DS component.

Like the DS monitor, the US Monitor encapsulates the properties that model the communication protocol between a US component and a DS component. However, they are configured differently.

All the properties that model the behavior of the US interface of the DS component are configured as constraints (assume properties) for formal verification and all the properties that model the behavior of the US interface of the AL are configured as assertions (assert properties).

8.2.3 AL scoreboard:

The flow of flits through the AL may be suspended temporarily. This happens when the input buffer of the AL or the input buffer of the DS router gets full. If the input buffer of the AL gets full, the AL is unable to accept new flits from the US router until some space frees up. If the input buffer of the DS router gets full, the AL must stop sending flits to the DS router until it signals that it has space to accept them.

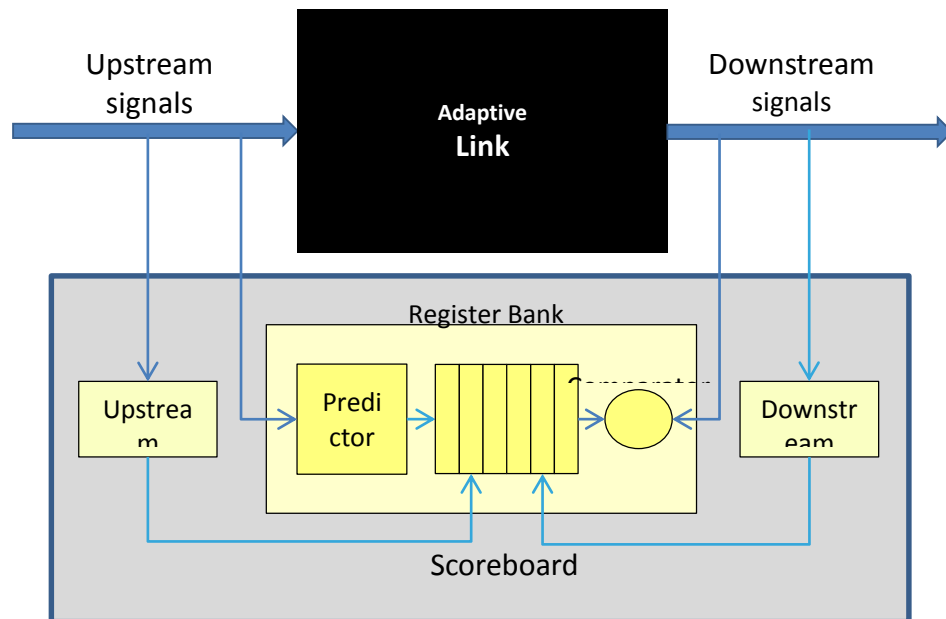
Therefore, we can have three types of scenarios:

- The AL does not accept flits from the US router but it sends flits to the DS router. The flow of flits is suspended on the US side of the AL only.
- The AL accepts flits from the US router but does not send flits to the DS router. The flow of flits is suspended on the DS side of the AL only.
- The AL does not accept flits from the US router and does not send flits to the DS router. The flow of flits is suspended on both sides of the AL.

In a downsizing configuration, a flit entering the US interface of the AL is split into several smaller flits that are sent one by one to the DS interface. In an upsizing configuration, several flits entering the US interface are stored within the AL and combined together to create a larger flit that is sent to the DS interface once it is complete.

In a frequency adaptation configuration, the US interface clock and the DS interface clock have different frequencies, so flits enter and exit the AL at a different pace. The DS clock and US clock may be asynchronous, with the same frequency or different frequencies.

In a packet store&forward configuration, the AL stores the incoming flits and does not forward them to the DS interface until a complete packet has been received. The scoreboard has been designed to handle several scenarios, from the simplest ones to the most complex ones. Its architecture is illustrated below



The flow of data through the scoreboard is as follows:

- Flits entering the US interface of the AL go through the predictor. The role of this module is to generate the reference flits, i.e. the flits that will eventually appear at the DS interface if the AL behaves correctly.
- The reference flits that get generated by the predictor are stored in the register bank. The AL introduces latency, so a flit that comes out of the predictor cannot be compared at once with the flit that is present at the DS interface.
- Every time a flit appears at the DS interface, a comparator checks that it is the same as its corresponding reference flit that has been previously stored in the register bank.

The US pointer module calculates the addresses in the register bank where reference flits that come out of the predictor have to be written. It is based on a counter that keeps track of the flits entering the AL through its US interface.

The DS pointer module calculates the addresses in the register bank where reference flits have to be read for comparison to the flits that appear at the DS interface. It is based on a counter that keeps track of the flits exiting the AL through its DS interface.

The comparator module checks that outgoing flits that appear at the DS interface are identical to their corresponding reference flits that were stored previously in the register bank.

The US side (predictor, US pointer, register bank write operations) and the DS side (DS pointer, comparator, register bank read operations) of the scoreboard are in two different clock domains that are not linked together in any way. No read/write races on the register bank can ever occur because the AL always introduces latency. Configurations of the AL with asynchronous US/DS clocks can be handled as well as configurations with synchronous US/DS clocks.

For the sake of clarity, we only mentioned flit signals in our description of the scoreboard. In reality, the scoreboard also takes care of the sideband signals using a similar mechanism as for the flits

As described in the previous deliverable, the verification environment has been written in SystemVerilog for the RTL part of it and in SVA for the properties. The RTL model of the AL is in VHDL, so mixed-language binding mechanisms of SystemVerilog have been used.

Table 2 shows the number of SVA properties that have been written and how they are configured in the different modules of the verification environment.

SystemVerilog module	Number of SVA properties	Property configuration
US packet composition FSM	22	Assume
US credit-based control flow FSM	8	4 Assume, 4 Assert
US sideband signal checker	18	Assume
DS packet composition FSM	28	Assert
DS credit-based control Flow FSM	8	4 Assume, 4 Assert
DS sideband signal checker	16	Assert
Scoreboard	4	Assert

Table 2: SVA properties and their configuration

The role of assume properties is to enforce constraints on the behavior of the output signals of the verification environment that drive the AL. Assert properties are applied to the AL and model the behavior it must exhibit to be exempt of functional bugs.

For example, the verification environment must not send a flit to the US interface of the AL if its input buffer is full. This behavior is captured using assume properties because this is what a US router does (a bug-free one). The US interface of the AL must send a credit to the US router every time a slot frees up in its input buffer. This behavior is captured as an assert property because this is what we expect the AL to do.

Note that assertions that apply to flits are bus-wide, i.e. they apply to the value of the entire bus.

Also note that, thanks to the architecture of the scoreboard, all properties are triggered by only one clock.

8.3 Result

Formal proof was run on large numbers of representative of all potential configurations. Special attention was paid to the most complex configurations that are more likely to be impacted by functional bugs. We also made sure that every configuration parameter got exercised.

Proof run times generally depend on the number of licenses of the formal tool and the compute farm that are being used. In the environment that was at our disposal, the average value of the proof per each STNOC completed for simplest configurations. Components that include downsizing configurations typically took a day or two to complete. While components including some upsizing configurations were still running after 5 days and the jobs were terminated leaving some assertions unproved. Many of the Store&Forward configurations that we tested did not finish in 5 days, which was clearly due to the presence of large FIFOs.

8.3.1 Compute farm and formal tool licenses

An obvious factor that influences the proof run times is the compute farm you have access to:

- Number of processors and RAM size of machines
- Number of jobs that can be run in parallel

In our experience with the AL, increasing the number of processors beyond 4 and the RAM size beyond 256Gbytes did not bring further improvements.

The number of jobs run in parallel has a massive impact on proof run times. However, running more jobs in parallel requires more licenses of the formal proof tool, so cost is a constraint.

8.3.2 Proof engines

The formal proof tool that we used has 18 proof engines that execute different types of algorithms. Depending on some characteristics of the design to be verified and its associated properties, some proof engines perform better than others. The tool uses heuristics to automatically select proof engines.

The tool we used can display graphs that show the progress of the different proof engines over time as the proof goes on. We used this feature to thoroughly analyze the performance of the various types of proof engines for the different families of configurations of the AL (upsizing, downsizing, packet store&forward, etc). We found out that for some configurations of the AL, the proof engines automatically selected by the tool were making very slow progress or even hardly noticeable progress. In these cases, we experimented with the other engines and analyzed their relative performance. Through a trial and error process we were able to identify the best proof engines for all families of AL configurations, which led to tremendous improvements.

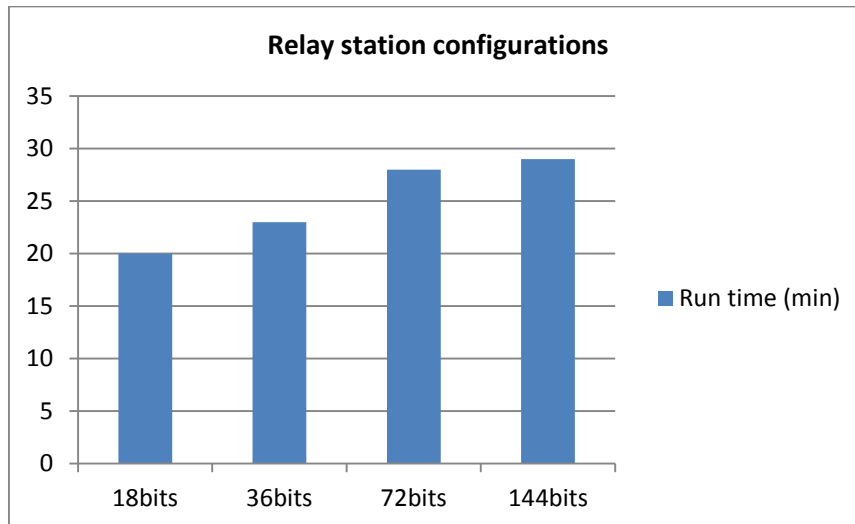
8.3.3 Run times

The run times that are given hereafter were obtained with up to 30 licenses of the formal proof tool, up to 50 jobs running in parallel on the compute farm, and 4CPU machines with 256GBytes of memory.

Relay station configurations

The relay station is the simplest of all the AL configurations. The AL delays incoming flits by one clock cycle and forwards them as is to the DS router. Relay stations are mainly used to break long wires that are incompatible with physical design constraints.

A complete proof of this type of configurations could be obtained for all the possible flit sizes. Run times are shown in the diagram below.



As it can be seen on the diagram, the run times do not double with the flit size. This is because the formal proof tool is able to vectorize the assertions applied to flits, i.e. it handles a flit as a single entity rather than a collection of individual bits.

Downsizing/upsizing configurations

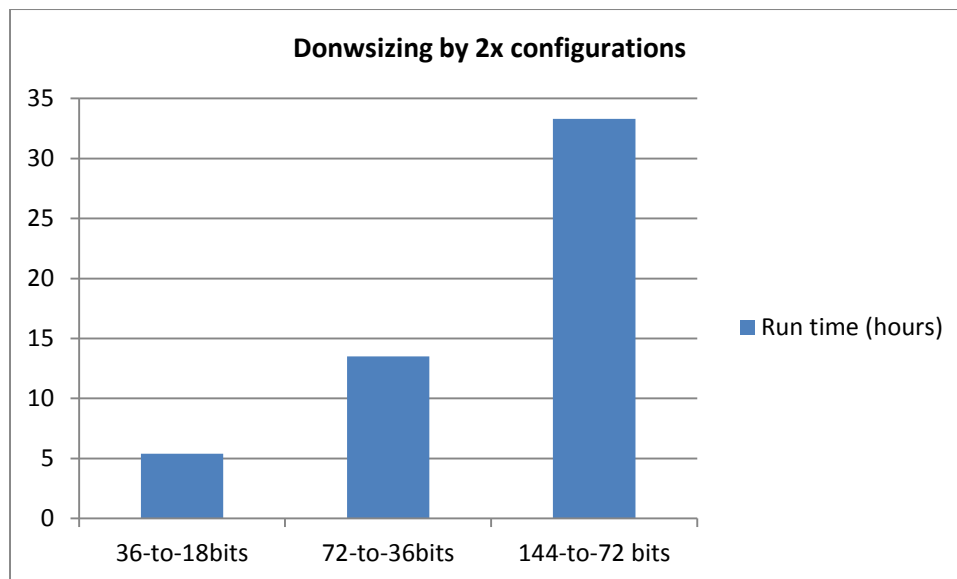
The proof run times for downsizing/upsizing AL configurations mainly depend on the input flit size to output flit size ratio, and on the flit sizes.

Unlike in the relay station configuration, it is difficult for the formal proof tool to vectorize the flits.

In the case of a downsizer, the AL splits the input flits into several output flits but the output flits are not a mere partition of the input flits. As an example, a 72-to-18bits downsizing AL receives 72bits flits from the US router on its US interface and sends 18bits flits to the DS router through its DS interface (x4 downsizing ratio). Each 72bits input flit includes 64bits of payload and 8 byte-enable bits, one for each payload byte. The AL reassigns the byte-enable bits to the output payload bits, each output flit containing 2 bytes of payload and 2 byte-enable bits.

A downsizing AL may also drop some output header flits because they don't convey any network layer header or transport layer header information. Using the same 72-to-18bits example, assume that the header occupies 100bits. At the US interface, two flits are required to transmit the header. At the DS interface, only three flits are sufficient to transmit the header, so the divide x4 ratio does not apply to the header.

The diagram below shows run times for x2 downsizing configurations.



Upsizing configurations present similar challenges as downsizing configurations, and proof run times are similar.

Packet store&forward configurations

AL configurations that implement the packet store&forward functionality use a FIFO to store incoming flits until a complete packet has been received.

The presence of this FIFO has a dramatic impact on proof run times. As the depth of the FIFO increases, the run times increase sharply.

We have not been able to get complete proofs in 3 days with the maximum number of licenses we had at our disposal for ALs with a FIFO depth greater than 14 flits, the maximum depth being 32 flits.

The EDA vendor of the formal proof tool that we used provides 'proof accelerators' that are designed to address the specific challenges posed by FIFOs. These accelerators are blackbox modules that encapsulate some 'secret sauce' and that the user has to instantiate in the verification environment. We experimented with one of these proof accelerators and got excellent results for AL configurations with the same input/output flit sizes. We were able to prove a 144-bits AL with a 32 flits deep FIFO in less than 4 hours.

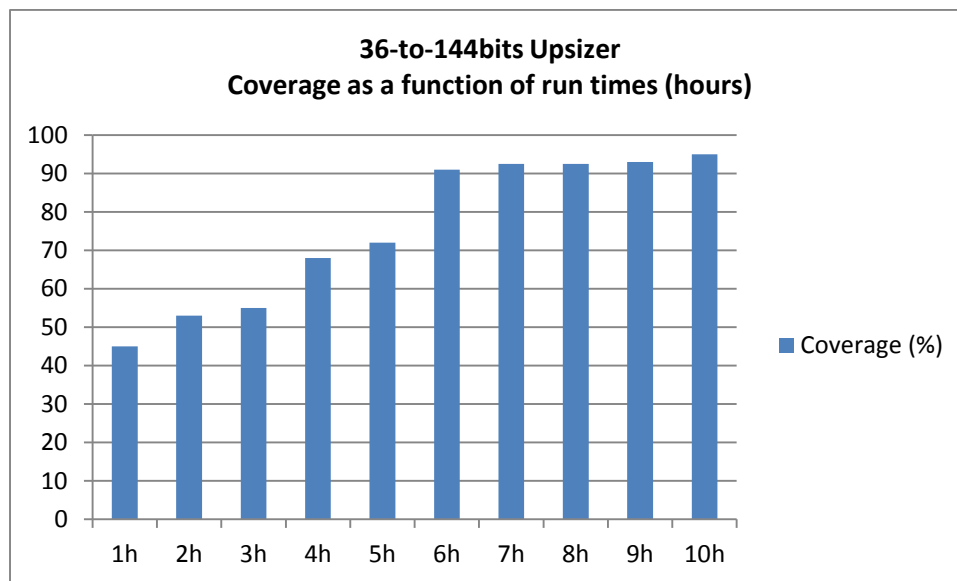
Unfortunately, due to packet header handling issues, we could not apply the proof accelerator to packet store&forward configurations that also perform flit downsizing and upsizing.

8.3.4 Code coverage results

In order to measure code coverage, the formal proof tool generates cover properties that it then tries to prove.

A cover property is associated to a line in the RTL code of the AL. If the tool can prove the property, then the line is covered. Some other properties are associated to 'if-else' branches and aim at checking whether both the 'if' branch and the 'else' branch are covered.

The diagram below shows the coverage achieved as a function of run times for a 36-to-144bits upsizing configuration. Coverage increases rather rapidly during the first 6 hours, and then tends to stagnate during the remaining 4 hours until all the cover properties get proven, yielding 95% coverage.



The code coverage figures that we obtained ranged from 74% to 95%, depending on AL configurations.

Packet store&forward configurations had the lowest coverage of all the configurations that we verified. As described above, we were not able to get complete proofs for FIFO depths greater than 14 flits.

For most of the other configurations, coverage numbers were within a 90% to 95% range. For each of them, coverage results pointed to some dead code in the RTL model of the AL, which can be explained as follows.

The RTL model of the AL has been designed to implement all configurations with the same code. VHDL generic's (parameters) are used to select a given configuration. As a result, each configuration leaves unused some code pieces that are only useful to other configurations. The first step in the formal proof process consists in synthesizing the design, so most of the logic that does not contribute to the selected configuration gets eliminated during that step (tied nets, redundant logic, etc). However, some of that logic cannot be removed by synthesis.

8.3.5 Conclusion

Formal verification has proved a very effective approach to verifying the STNoC.

9 Bibliography

- [1] Imperas Software Limited, "Model Specific Information for variant ARM_Cortex-A9MPx2", http://www.ovpworld.org/modeldocs/OVP_Model_Specific_Information_arm_Cortex-A9MPx2.pdf