July, 2010

SCoPE

(System Co-Simulation and Performance Estimation)

User Manual

Version 1.1.5

Bugs and Suggestions

Please report bugs and suggestions about this document to
http://www.teisa.unican.es/scope

The SCoPE infrastructure has been created by the following individuals:

Main developer:
Héctor Posadas

Assistant developers:
Juan Castillo
David Quijano

Additional contributors:
Luis Díaz
Sara Real
Daniel Calvo
Pablo Gonzalez
Patricia Botella
Gerardo Caballero

This document was authorized by:
Prof. Eugenio Villar

(All contributors from the University of Cantabria)

Contents

1.- OVERVIEW

SCoPE is a simulation infrastructure oriented to fast simulation of MPSoC HW/SW systems. It combines easy creation of platform models with fast techniques for SW execution modeling. SCoPE provides a simulation technique alternative to instruction set simulators (ISS) obtaining speed ups in factors about x100 in exchange of performance estimation errors of about 10%. To do so, SCoPE applies the latest techniques on annotated native co-simulation with high-level TLM HW platform models.

SCoPE is a tool specially oriented to the fist steps of development in the design process. I fact, it is a good technical solution for two main purposes:

- Explore the best system configurations. SCoPE allows fast modeling of your SW and HW components in a complete platform model. Thus, you can analyze the performance results of your possible configurations, exploring the effects of different components, such as processors, or the effects of different architectures. SCoPE provides facilities for automatic system exploration.
- Put at disposal of your SW designers a virtual platform where the interaction with HW components, and in general, the effects of the planned system can be considered. Thus, the development of the SW can start without requiring a HW prototype, reducing the time to market of your products. Additionally, as the native simulation speed is close to the obtained running the SW in your PC, SW development time is highly reduced compared with current ISS-based simulation techniques.

From a technical point of view, SCoPE is a SystemC extension for system modeling based on approximately (loosely) timed descriptions of the system components. In more detail, SCoPE is a C++ library that extends, without modifying, the standard language SystemC to perform the co-simulation. On one side, it simulates C/C++ software code over High-level Operating System models. Currently three different operating system interfaces are supported (POSIX, win32 and uC/OS) but modeling other OSs is also possible. On the other hand, it co-simulates these pieces of code with hardware described on SystemC language.

From that simulation the user obtain a timed simulation of the system, which allows to check timing constraints, delays, timeouts and optimize the occupation of the processing resources. Additionally, the SW timing modeling allows analyzing the effects of interrupts and HW/SW communications in the system operation.

SCoPE simulation infrastructure is composed of:

- A core with a set of central services provided as open-source code, and directly accessible from the web-site.
- A set of plug-ins for specific purposes can be added, some of them available from the web and others requiring specific requests to the authors or to scope@teisa.unican.es

The following sections describes the entire structure of plugins and the SCoPE open-source simulation core in more detail.

2.- Global SCoPE infrastructure

SCoPE simulation infrastructure provides a complete set of facilities for fast modeling of complex embedded systems. SCoPE allows modeling from simple systems composed of a processor a bus and a memory to MpSoC with NoC systems. Designers can use that simulations for functional purposes, developing and checking the correct behavior of the system components, or for performance evaluation, obtaining information about delays, resource use, power consumption, cache efficiency, bus or network traffic, temperature increase, etc.

The open-source components of SCoPE are:

- SCoPE simulation core
- M3-Plugin for Design Space Exploration

The non open-source components that can be added to extend the basic SCoPE facilities are the following:

- Optimized SW estimation plug-in
- Cache modeling plug-in (including optimized instruction caches, data caches and L2 caches)
- IP-XACT platform description plug-in
- Win32 API plug-in
- Thermal floor-plan analysis plug-in
- Sicosys network modeling plug-in
- TCP/IP plug-in
- Real-Time Linux plug-in

2.1 Open-source components

2.1.1 SCoPE simulation core

The SCoPE simulation core, or in general SCoPE, contains the basic simulation infrastructure, and it is required for all the other components to work. It is delivered in open-source format under GPL and LGPL licenses. The main features are the following:

- SW modeling
  - Approximate estimation and modeling of SW simulation times
  - Approximate instruction cache modeling
- Operating system modeling
  - SMP Operating system core model, based on the POSIX standard
  - POSIX api
  - uCos2 api
  - HAL infrastructure, following the Linux functions for character devices

        - Network driver for the generic HW network interface
- HW components
    - Generic HW component models for bus, memory, dma, mesh network and network interface.

An engineer using SCoPE can simulate a specific software over a custom platform and obtain estimations of:

- Number of context switches
    - Thread and process
- Total execution time
- Processor running time
- Use of processor (%)
- Executed instructions
- Instruction Cache misses
- Processor energy and power consumption

Additional detail about the tool will be presented in the following chapters.

2.1.2 M3Plugin

M3P, is an open source plug-in developed to integrate SCoPE within the Design Space Exploration (DSE) generic flow defined in the european Multicube project. The plug-in extends the modeling infrastructure to enable the simulation tool to communicate with a design space exploring tool, such as modeFrontier(http://*www.esteco.com*) or M3Explorer (http://m3explorer.sourceforge.net).

It can be downloaded from http://www.teisa.unican.es/gim/en/scope/multicube.html or from the SCoPE official web site. Extended information about the plug-in can be found in the user-guide at disposal at the web site.

The main features of the plugin are the following:

- **Dual use of M3P-SCoPE as a tool or a library**

  The extended version provides an example-independent SCoPE tool executable, avoiding manual creation of SystemC standard “sc_main” functions. This capability reduces compilation times and providing a more flexible and user-friendly solution. Furthermore, the possibility of using the code as a library is still maintained.

- **Automatic system model generation**

  The open source tool provides a XML input capable of creating the system models from XML descriptions. This capability allows dynamically creating the system models, and thus, dynamically supporting any system modification that want to be explored by the Design Space Exploration (DSE) tools. As a consequence, DSE process can be completely performed without compiling the simulation model, and thus reducing the exploration time.

- **Dual input for system descriptions**

  The refined tool has been prepared to accept system descriptions in two formats: as a SystemC code or in XML files. This extends the capabilities of the initial prototype, where only XML description was valid. To allow exploration of SystemC code based descriptions, the new feature of system configuration through global variables has to be used. In the Multicube project the SystemC code is used to model the power-line use case, while the XML files are used in the multimedia use case.

- **Automatic system configuration**

  The tool provides a second XML input to easily modify the system configuration in order to perform the DSE process.

- **Exploration through global variables**

  The basic solution for exploring systems is defining unfixed parameters in the XML system description file. Nevertheless, the tool allows also exploring internal details of the system component models. These internal details are characteristics that are described in the model code, but without visibility in the XML system description file. To allow these exploration, the parameters must be declared as global variables in the code files and in the XML files. Furthermore, this technique allows exploring system described completely in SystemC code, not in XML files.

- **Memory space separation for system component models**

  The extended tool is prepared to isolate the memory space of each component model. As the tool is developed as a host process running the SystemC kernel, only one memory space is provided for the entire simulation. Thus, duplicating names of global elements result in simulation errors. The isolation provided avoids conflicts when using the same names for functions or global variables in several component models. Furthermore, it allows instantiating the same component several times.

- **SW/HW communication through direct register memory addresses**

  In embedded systems HW/SW communication is usually performed letting the SW to access the memory address where the peripheral registers are mapped. These addresses are handled as pointers where data are read or stored. Previous versions of the tool, and in general SystemC-based solutions does not support this solutions, as it implies trying to access the peripherals of the host machine. This refined version provides the features required to support direct pointer access to HW peripherals from SW component models.

- **Time and power parameters in generic HW integration wrappers**

  Integration of HW components, both generic and user defined ones, is done in SCoPE by using generic Integration Wrappers. The use of these generic wrappers allows the automatic system model creation. Now, the wrappers also provide facilities to include time and power information for each component in an standard way. Thus, components performance effects can be easily integrated in the XML system description file. This increases the accuracy of time and power metrics obtained with Multicube-SCoPE.

- **Integration of IMEC ARP library for reuse of timing information**

  The Atomium Record/Playback library of IMEC has been integrated in Multicube- SCoPE. This new feature allows storing data from a reference simulation in order to be re-used in other simulations. During the exploration process multiple simulations of the same code are done, considering different HW configurations. As the code executed is the same, performance information from one simulation can be reused in other simulations with compatible configurations. Furthermore, performance information obtained with other simulators at different abstraction levels can be used during the exploration process, increasing the accuracy of the process.

- **Automatic metric reporting**
  The tool is capable of automatically reporting a list of system metrics, following Multicube XML interchange formats. The metrics the user wants to be reported must be indicated through a XML file at the beginning of the simulation. When a simulation finishes, a new XML file with the results is automatically created.

- **Portability**
  The tool is portable across a wide range of systems. This goal is achieved by not sacrificing the efficiency of the overall exploration engine. The standard ANSI C++ programming language is used for developing the open source framework. The Standard Template Library as well as other open source libraries are used during the development process.

- **Multiple parallel executions**
  The tool has been adapted to allow multiple concurrent executions in different directories with complete independence among them. This feature allows the exploration tool to run several instances of the simulation to speed up the exploration process.

2.2 Non open-source components

2.2.1 Optimized SW estimation plug-in

SCoPE basic package provides preliminary technology for estimating time and power of SW components. This plug-in offers a more accurate and simple estimation technique, even considering cross-compiler details and optimizations.

2.2.2 Cache modeling plug-in

SCoPE basic package provides an initial technique for instruction cache modeling, but no data cache or L2 cache modeling is possible. This plug-in provides a complete modeling solution obtaining much more accurate results with a minimal simulation overhead.

2.2.3 IP-XACT platform description plug-in

This plugin allows describing the HW platform using IP-XACT descriptions, instead of SystemC codes.

2.2.4 Win32 API plug-in

The plug-in provides an additional API extending the operating system model provided with SCoPE. This plug-in models a win32 api.

2.2.5 Thermal floor-plan analysis plug-in

Considering SCoPE power estimations, the designer can define different floor-plans for the implementation of the system under development and evaluate the resulting temperature values for the running SW.

2.2.6 Sicosys network modeling plug-in

SICOSYS is a general-purpose interconnection network simulator for multi-procesor systems that allows the modeling a wide variety of message routers in a precise way. Results are very close to those obtained by using hardware simulators but at lower computational cost.

It is provided by the University of Cantabria at "http://www.atc.unican.es/SICOSYS/".

An extension of the plug-in has been developed to connect the network modeling tool with SCoPE for accurate modeling of MpSoC + NoC systems. It allows increasing the accuracy of the simple mesh network model provided with the SCoPE open-source infrastructure.

2.2.7 TCP/IP plug-in

In order to provide TCP/IP capabilities to the OS model provided with SCoPE, an extension of the lwIP stack (http://www.sics.se/~adam/lwip/) adapted to execute within SCoPE simulations is provided. The plug-in allows modeling TCP connections among the nodes of the system simulated and connecting the simulation with applications running on the host computer and, thus, accessing the Internet.

2.2.8 Real-Time Linux plug-in

Provides an extension of the OS modeling containing the Real-Time Linux extensions developed in the Hyades project.

3.- SCoPE modeling capabilities

The SCoPE library provides a complete set of facilities for fast modeling of complex embedded systems. The goal of ScoPE is to give the designer the components required to easily create a virtual model of the embedded system in design. To do that, SCoPE provides element for modeling the both SW and the HW subsystems.

The SW subsystem is modeled in SCoPE by separating the application SW code and the SW infrastructure (operating systems and associated components). To modeling the application SW, the source code is first annotated, adding performance information, and then natively executed together with the rest of the simulation.

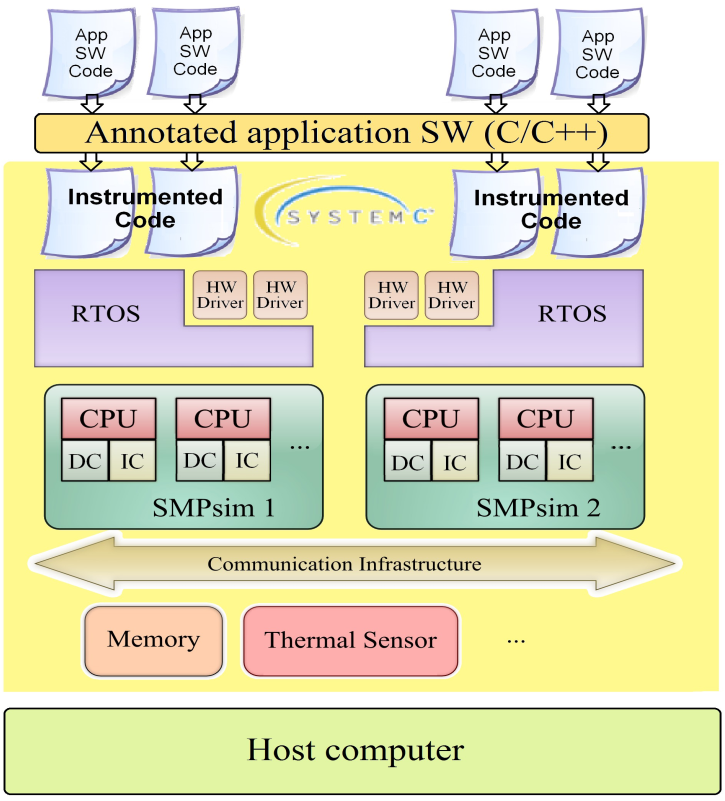

Figure 1: Structure of a complete SCoPE system model

The SW infrastructure is composed of a OS model, which is in charge of providing concurrency scheduling, communication and timing services to the application SW. To do so, a POSIX API is provided. Additionally, HW/SW support, based on the Linux functions for device drivers are included in the model.

Finally, generic models for several common components such as buses, memories, network interfaces or DMAs are provided, using SystemC/TLM2 standard interfaces. Combining these components, and adding user-defined HW components a wide range of platform architectures can be modeled.

In order to create the system model, indicating the HW and SW components and their interconnection, a sc_main function, following the SystemC rules is required from the user. Its creation will be explained the the following chapter. This file can be avoided providing all this information in XML format, through the M3P or the IP-XACT plug-ins.

3.1 Application SW performance annotation

For modeling the application SW performance, the source code is analyzed, in order to estimate the time and number of instructions of each code basic block. This information is annotated in the code, generated an instrumented code containing performance information. Then, the code is compiled to be natively simulated. The tool ***scope-g++*** is an extension of gcc/g++ that perform all the annotation process automatically generating the final binary file.

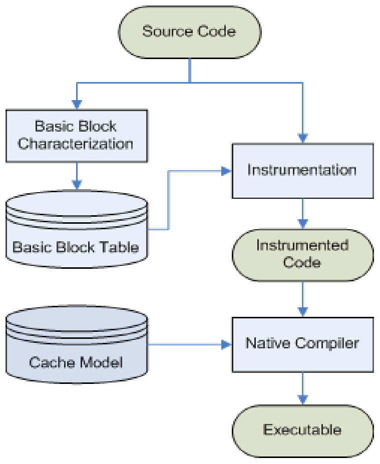

Figure 2. Estimation process

The code performance is analyzed identifying the operations included on each basic block (+,-,%, if, for, ...). The cost of each operator depends on the target processor where the SW is expected to run. The costs for each processor are stored in the “processors.xml” file included in the “config” folder of SCoPE sources. The costs are stored for an “ideal” frequency of 1GHz. Additionally, the xml information contains a default frequency, that can be modified in the sc_main function or dynamically during simulation. This frequency is used to make a correction on the annotated costs in order to model the performance of the processor at that frequency.

The compiler, also is in charge of redirect the OS function calls in order to call the SCoPE OS model, not the host OS.

scope-g++ receives all the gcc/g++ required flags for compiling the application SW. Additionally it accepts the following scope-specific flags,

--scope-cpu=*cpu_to_use (e.g. arm926t)*
--scope-method=*op-cost (other plugins enables other SW modeling mechanisms)*
--scope-verbose *(show the internal operations)*
--scope-preserve-files *(preserve the annotated files)*

--scope-language=*c,c++ (indicates if the file to be compiled is c or c++, to call the gcc or g++ native compiler. Otherwise the file extension is used).*

SCoPE compiler at disposal in the basic distribution performs 3 types of operations: Modify the OS function calls to access the OS model instead of the host OS, annotate execution times and annotate instruction cache accesses. There are additional flags to select which operations must be done.

--scope-notime *(disable the time annotation)*
--scope-noicache *(disable the instruction cache annotation)*

Other flags can be used when using the SCoPE plug-ins. Its usage can be shown in the corresponding plug-in user guides.

SCoPE annotation is made in a way that is hidden during debugging steps. Thus, SW development using gdb or other c/c++ debuggers is possible.

3.2 RTOS Modeling

This library models the detailed behavior of the RTOS including concurrency (among tasks in the same processor), parallelism (among tasks in different processors), scheduling and synchronization.

Although the SystemC kernel executes processes following a non-preemptive scheduling policy without priorities, SCoPE models preemption under different scheduling policies based on priorities.

3.2.1 Operating system interfaces

SCoPE integrates a POSIX based API that allows the execution of a large number of software applications that follows this standard.

POSIX is the main operating system interface nowadays, but it is not the unique. Thus, SCoPE has been improved to support extensions for other types of interfaces. An example is the integration with the uC/OS interface. This is a demonstration of the scalability of the tool, in terms of software support.

The following POSIX libraries has been included in the OS model.

- *include/:*

  mqueue.h pthread.h sched.h semaphore.h signal.h stdlib.h time.h unistd.h

- *include/sys:*

  ioctl.h select.h time.h types.h wait.h

Additionally, as SCoPE is executed on a Linux/UNIX host computer, some of the libraries of the host computer can be used, such as string.h, math.h, stdio.h, etc. In general, all functions from the native host OS that are non blocking, does not modify scheduling parameters, and does not depend on timing, can be used.

To use the functions for handling sockets and TCP/IP connections, the lwIP plug-in is required.

3.2.2 Device Drivers

In order to enable the integration of HW/SW communication, the main functions for developing charater device drivers and interrupt handling provided by Linux 2.6 kernels are included in the OS model. The main files are the following:

- *include/asm:*

  bitops.h dma.h io.h siginfo.h

- *include/linux:*

  fs.h if_tun.h interrupt.h ioport.h miscdevice.h poll.h

  if.h init.h io.h irqreturn.h netdevice.h smp.h

To load a driver, the function “insmod” must be called in the “sc_main” function.

3.3 HW Platform modeling

3.3.1 Hardware platform simulation

SystemC is the language used for the modeling of the hardware platform due to the easiness of implementation (C++ extension) and its simulation kernel. For the purpose of simulate different platforms SCoPE incorporates some generic hardware modules.

- Bus based on TLM2 used for the communication with peripherals and the transmission of hardware interruptions.
- DMA for coping large amount of data.
- Simple memory for the simulation of cache and DMA traffic.
- Hardware interface for an easy custom hardware connection.
- Network interface that work as a net card for the NoC.

3.3.2 System simulation

- Multi-computation: One of the advantages of this tool is the possibility of interconnection among independent nodes and simulate the interaction among them.
- Modular structure: Each RTOS component is an independent object that does not share any data with the others. Furthermore, each process is isolated from the rest of the system, thus, a process with global variables can be replicated in many nodes without data collision problems.

4.- INSTALLATION

Before you can use SCoPE, you must have some software installed.

4.1 Software Requirements

- GNU C/C++ compiler >= 4.0
- SystemC 2.2

4.2 Environment variables

Edit your .bash_profile or .bashrc file and add the next environment variables:

- CXX=g++
- SCOPE_HOME=/path/to/scope
- SYSTEMC=/path/to/systemc

Additionally, it is recommended to add the $(SCOPE_HOME)/bin folder to your PATH. Otherwise, the complete path to scope-g++ compiler will be required each time a SW task is

Example:

#.bashrc
export SYSTEMC=/home/SystemC/systemc-2.2.0 export SCOPE_HOME=/home/SCoPE/SCoPE-1.1.5 export PATH=$PATH:$SCOPE_HOME/bin

4.3 Internet resources

Once you have reached this point, you can get SCoPE and proceed with its installation. To download it you should go to:

- Download: http://www.teisa.unican.es/scope

Once there download SCoPE 1.1.5 version (July 2010).

4.4 Installation rules

For the tool compilation, once set the previous environment variables, it is only required to execute the make command.

Command	Explanation
make	Prints the Makefile help
make all	Compiles SCoPE sources
make test	Compiles all tests

5.- CREATING A SCoPE SIMULATION

In order to create a SCoPE simulation, the designer must follow the next steps:

- Compile the application SW, using scope-g++ tool instead of the standard gcc/g++
- Compile the user-defined HW components (using g++, not scope-g++)
- Generate and compile a sc_main.cpp file, including the sc_main function with the system description.
- Link the sc_main, SW object files, the HW files, the SCoPE library and the SystemC library to create the executable.

5.1 Compiling the application SW

To compile the application SW, it is only required to execute the same gcc/g++ instructions used to generate the object files in the host computer, but replacing gcc/g++ by scope-g++. Make sure the command contains the gcc flag “-c” in order to generate object files (“.o”) instead of an executable.

Additionaly, it is required to add to the compilation command:

- `--scope-cpu=`*`cpu_to_use (e.g. arm926t)`*
- *`-I$(SCOPE_HOME/include) -I$(SCOPE_HOME/scope)`*

If the main functions of the application SW are called “main”, it is recommended to change the name manually. Otherwise, scope-g++ will replace them by “uc_main”. (The “main” function is SystemC is implemented inside the SystemC library). If several applications will be simulated, this modification is mandatory, in order to identify each application in the sc_main file.

5.2 Create and Compile the user-defined HW

SCoPE HW components are interconnected using the standard TLM2 interfaces. Thus, any HW model developed in such way can be connected. Nevertheless, in order to make the creation and interconnection of HW models easier, SCoPE provides HW wrappers.

The wrappers are:

- uc_master_base: for master devices
- uc_master_slave_base: for master/slave devices, such as dma, bridges, etc
- uc_hw_if: for slave devices

Master interfaces can call the bus using the following functions:

- *`this->m_port->write(void *address, void *data, int size)`*
- *`this->m_port->read (void *address, void *data, int size)`*

- *this->m_port->b_transport(tlm_generic_payload &trans, sc_time &delay);*

Slave components can describe their interaction with the bus implementing the following functions:

- *write(void *address, void *data, int size)*
- *read (void *address, void *data, int size)*

Using these functions the TLM2 protocol is automatically handled.

5.3 Generating the sc_main function

The description of the system to be simulated is done in the sc_main function. In this function is described the HW platform, the SW infrastructure and the application SW. The structure of a common SCoPE sc_main function is the following:

```
#include “sc_scope.h”

int sc_main(int argc, char **argv){

        - Create the SW infrastructure

        - Load the application SW (and drivers)

        - Describe the HW platform

        sc_start(time);

        - Destroy the previously created components

        return 0;

}
```

5.3.1 Creating the SW infrastructure

To create the SW infrastructure, it is required to instantiate as OS models as required. For each model it is required to indicate the number of processors that will be controlled by the OS. Remember that the OS models are SMP systems, so they can control several processors at the same time. The creation of the OS models can be done with the following line:

```
UC_rtos_class  *rtos = new UC_rtos_class(num_processors);
```

If it is required to load any device driver, a call to “insmod” will be required.

```
insmod(rtos, driver_init_function);
```

5.3.2 Loading the application SW

To load the application SW it is required to load in the OS models, the name of the entry function of each application in the following way:

```
        (*rtos)[0]->new_process((int(*)(int, char**))function, (void*)arguments);
```

(*rtos)[0] indicate the processor where the task should start. However, as the OS model is SMP, the OS can decide to move the task to any free processor when required to balance the system load. To define the processors where a task can/cannot be moved, the “sched_setaffinity” function can be called in the application SW code (not in the sc_main).

5.3.3 Creating the HW platform description

To create the platform, the HW components must be created and connected. The following example create a node with a bus, a memory and a network interface:

```
// Creating the bus
UC_TLM_bus_class *bus = new UC_TLM_bus_class(sc_gen_unique_name("HAL"), freq);

// Processor binding
(*rtos)[0]->bind(bus); /* repeat for all the cpus ( (*rtos)[0], (*rtos)[1],
                         (*rtos)[2], ...)*/

// Creating and connecting the memory
UC_hw_memory  *mem  =  new  UC_hw_memory(sc_gen_unique_name("mem"),  RAM_START,
RAM_START + RAM_SIZE - 1, 100/*resp.time(ns)*/);
bus->bind(mem);

// Creating and connecting a network interface
UC_hw_NoC *eth1 = new UC_hw_NoC(sc_gen_unique_name("eth"), NET_START,  NET_START
+ 0x3F, NET_IRQ/*irq*/, 10/*delay*/, noc_simulator);
bus->bind(eth1);
```

Finally, it is required to indicate the address of the main memory, and create the memory map. The memory map is created with the address of all the components connected, and it will be used to deliver all the incoming transfers to the appropriate HW peripherals.

```
bus->set_memory_address(RAM_START);
bus->generate_memory_map();
```

To include a network, the following steps must be followed:

```
UC_NoC_Interface *noc_simulator = new UC_NoC_Interface("simulator", 1, false);
int addrs[x_size][y_size][z_size];
numbers[0][0][0] = eth1->get_node();
numbers[1][0][0] = eth2->get_node();
...
noc_simulator->set_structure((int ***)addrs,x_size,y_size,z_size);
```

5.3.4 Creating the executable

Once accomplished the previous steps, the simulation executabe can be obtained by linking the sc_main object file, SW object files, the HW files, the SCoPE library and the SystemC library to create the executable.

Remember to compile the sc_main file including the SCoPE and SystemC headers. This file is compiled using g++ (not scope-g++).

6.- Application software running in SCoPE

If you want to run your software application in SCoPE you have to be careful in some aspects:

6.1 Restrictions of SCoPE to the application SW

SCoPE restrictions are mainly due to problems of implementing the internal parser, although there are some other.

- *Use of SCoPE protected names*

  SCoPE uses some variables and functions for its internal operations. Their names can not be redefined in the application SW, to avoid name collisions. All SCoPE names start with "uc_", so it is highly recommended not to include in the application sw any name starting with that prefix.

- *Application Programming Interface API.*

  Nowadays only two APIs are implemented, which are POSIX and UCOS. If the embedded software application is made on another interface, will not work.
  - POSIX

    POSIX SCoPE's API was the first API implemented. The functions are the same but adding the prefix "uc_". For example, to create a task it will be uc_pthread_create() instead of pthread_create(). The parser is responsible for redirecting the original API calls to the API SCoPE calls. The prefix is automatically added by scope-g++ tool when compiling.

- *Grammatical constraints*
  - Enum

    Using SCoPE it is no possible to finish a enum declaration with a coma. scope-g++ uses a C++ grammar for analyzing the application SW so, grammatical rules not supported in C++ cannot be accepted.

Allowed	Not allowed
enum fruits ={apple, lemon, orange};	enum fruits ={apple, lemon, orange,};

6.2 General restriction on the code for the native simulation

- *Use of libraries*

  All you want to simulate must be source code. If we have a binary code (object code or library) compiled for the target CPU, we will not can simulate it. This has its drawbacks, namely that there are many companies whose code is closed and all you have are the libraries compiled and a series of headers. We have to give the sources in C / C + +.

- *Pragmas, advanced directives and compiler attributes*

  Many codes use pragmas and directives that are specific to each compiler. As advice to avoid any problem related to this, you can request that both codes (the one at the target CPU and the one at the workstation) respect the ANSI standard C. It is highly recommended when you are working in real-time critical applications where reliability must be maximum.

7.- Bugs and suggestions

Please report bugs and suggestions about this document or about the SCoPE tool to: scope@teisa.unican.es

8.- Glossary

Application Programing Interface (API): Set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications.

Approximately timed: Modeling style for which there exists a one-to-one mapping between the externally observable states of the model and the states of some corresponding detailed reference model such that the mapping preserves the sequence of state transitions but not their precise timing.

Computing group: Group of computing elements, usually processors, organized to work cooperatively, emulating a more powerful single computing unit. The elements of the computing group work controlled by a single operating simple and usually in a symmetric way.

Design Space Exploration (DSE): Process that explores all the system design possibilities in order to obtain an optimal design.

Extensible Markup Language (XML): General-purpose specification for creating custom markup languages. It is classified as an extensible language, because it allows the user to define the mark-up elements. XML's purpose is to aid information systems in sharing structured data, especially via the Internet, to encode documents, and to serialize data.

HW platform: Group of HW components working together that provides the required support to the SW components and provided the required specific functionality required to perform the required application(s).

HW/SW partition: Process of dividing the system functionality in SW and HW components.

Instruction Set Simulator (ISS): Simulation model, usually coded in a high-level programming language, which mimics the behavior of a mainframe or microprocessor by "reading" binary instructions and maintaining internal variables which represent the processor's registers.

Metric: Set of units which can be used to specify anything which can be measured, along with the procedures to carry out measurements and the procedures for the interpretation of the assessment in the light of previous or comparable assessments.

Metric Definition file: XML file listing the system metrics the simulation must measure and report.

Network on Chip (NoC): Is a new approach to System-on-a-chip (SoC) design. NoC-based systems can accommodate multiple asynchronous clocking that many of today's complex SoC designs use. The NoC solution brings a networking method to on-chip communication and brings notable improvements over conventional bus systems

Node: Active electronic device or group of devices attached to a network, and capable of sending, receiving, or forwarding information over a communications channel. A node is a connection point, either a redistribution point or a communication endpoint.

Platform: Sort of hardware architecture or software framework (including application frameworks), that allows software to run. Typical platforms include a computer's architecture, operating system, programming languages and related runtime libraries or graphical user interface.

Real-Time Operating System (RTOS): Multitasking operating system intended for real-time applications. Such applications include embedded systems , industrial robots, spacecraft, industrial control, and scientific research equipment. A RTOS facilitates the creation of a real-time system, but does not guarantee the final result will be real-time; this requires correct development of the software. Key factors in an RTOS are therefore a minimal interrupt latency and a minimal thread switching latency.

SW platform: Group of generic SW components used to provide the required support to the SW applications.

System Configuration file: XML file defining the values required for the system configuration parameters to perform a simulation.

System Descriptions file: XML file describing the system. It contains a description of the HW platform, SW platform and SW application.

System Metrics file: XML file where the obtained estimation for the system metrics are reported.

SystemC: Set of library routines and macros implemented in C++, which makes it possible to simulate concurrent processes, each described by ordinary C++ syntax. Instantiated in the SystemC framework, the objects described in this manner may communicate in a simulated real-time environment, using signals of all the datatypes offered by C++, some additional ones offered by the SystemC library, as well as user defined.

SCoPE: SystemC framework for system modeling based on approximately (loosely) timed descriptions of the system components.

Timed simulation: Simulation performed using timed models of the system components. The resulting simulation considers both the system functionality and the associated timing.

Transaction-Level Modeling (TLM): High-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture. Transaction requests take place by calling interface functions of these channel models, which encapsulate low-level details of the information exchange. Communication mechanisms are modeled using channels, such as busses or FIFOs.

TLM2: The second major version of the OSCI Transaction Level Modeling standard.