Grupo de Ingeniería Microelectrónica

Dpto. de Tecnología Electrónica, Ing. de Sistemas y Automática, E.T.S.I.I.T.

Universidad de Cantabria

Memoria de tesis para la obtención del título de doctor

Estimación de prestaciones para Exploración de Diseño en Sistemas Embebidos Complejos HW/SW

Autor: Héctor Posadas Cobo

Director: Eugenio Villar Bonet

Febrero 2011

Sections

English Section

Summary i

Conclussions xv

Sección en Español

Agradecimientos 1

Índice detallado 4

I Introduccion 7

II Estado del arte 22

III Aportaciones 68

IV Resultados y conclusiones 243

Referencias 274

Doctoral Thesis:

Early Performance Estimation for Design Exploration in Complex HW/SW Embedded Systems

Author: Héctor Posadas

Director: Eugenio Villar

February 2011

Summary

Many advanced consumer products such as mobile phones, PDAs and media players are based on System on Chip (SoC) solutions consisting of a highly integrated chip and associated software. SoCs combine hardware IP cores (function specific cores and accelerators) with one or several programmable computing cores (CPUs, DSPs, ASIPs).

The increasing complexity, heterogeneity and flexibility of the SoCs make their development process to require large design efforts, especially for multi-processor SoCs (MpSoC). First, the high interaction among all the SoC components results in large number of cross-effects to be considered. Additionally, the huge number of design possibilities of complex SoCs makes very difficult to find optimal solutions. As a consequence, design decisions can no longer depend only on designers' experience. New solutions for early modeling and evaluating all the possible system configurations are required.

These solutions require very high simulation speeds, in order to allow analyzing the different configurations in acceptable amounts of time. Nevertheless, sufficient accuracy must be ensured, which requires considering the performance and interactions of all the design components (e.g. processors, busses, memories, peripherals, etc.).

Static solutions have been proposed to estimate the performance of electronic designs. However, they usually result too pessimistic and difficult to scale in order to be applied to very complex designs. Instead, performance of complex designs where no very hard real-time constrains have to be met can be more easily evaluated with simulation based approaches.

Simulations at different abstraction levels have been proposed for this goal, providing different tradeoffs between accuracy and speed. As early evaluation of complex designs requires very high simulation speeds, only the use of faster simulation techniques can be considered. Among them, simulations based on instruction set simulators (ISSs) and binary translation are the most important ones. However, none of them really provides the required trade-off for early evaluation.

ISSs are usually very accurate but too slow to execute the thousands of simulations required to evaluate complete SoC design spaces. ISS-based simulations usually can take hours, which means that the execution of thousand of simulation can require years, something not acceptable in any design process.

Simulations based on binary translation are commonly faster than ISSs. However, these solutions are more oriented to functional execution than to performance

estimation. Effects as cache modeling are usually not considered when applying binary translation. Furthermore, this simulations also result too slow to explore large design spaces.

Additionally, in both cases, the simulation requires a completely developed SW platform. OS and compiler ports for the target platform are needed. However, these ports are usually not available early in the design process. Then, these simulation techniques are not only too slow but also difficult to perform.

As a consequence, faster and more flexible simulation techniques, capable of modeling the effect of all the components that impact on system performance, are required. In this thesis, a complete simulation infrastructure capable of accomplishing these requirements has been developed.

The infrastructure is based on a native co-simulation solution. Native co-simulation is a novel co-simulation technique that combines native execution of annotated SW code with time-approximate models of the HW platform.

In current SoCs, most of the functionality is implemented as SW, thus, speeding-up the modeling of the SW part is a key aspect to achieve fast simulation speeds for the entire systems. The faster solution possible for executing the SW is to natively compile and execute the SW in the host computer, instead of using a cross-compilation tool set.

However, this kind of simulation does not provide performance information, and its interaction with a model the target HW is not possible. To solve that, in native co-simulation the code is previously annotated with performance information. Additionally, an operating system model is applied in order to control the execution of the SW, provide services to the applications and handle its interaction with the HW platform. As a result of this native execution, slow ISSs or binary translation engines are no longer needed, obtaining much faster simulation speeds.

Finally, it is required to provide HW platform models capable of modeling the functionality and performance effects of the target platform without slowing down the simulation speed. To do so, time approximate models of HW platform components (such as buses, memories, DMAs, NoCs, network interfaces, etc) based on SystemC/TLM standards are required.

In that context, during the thesis, different approaches for modeling the SW of the SoCs and its interconnection with the HW components have been explored. These solutions have been integrated into a new tool called SCoPE (System Co-simulation and Performance Estimation), capable of handling the modeling and evaluation of the entire electronic designs.

The goal of SCoPE is to provide a tool capable of assisting the designer during the initial design steps. More specifically, SCoPE has been developed to provide the following services to the designers:

- Simulate the initial system models to check the complete functionality, before the platform is available, including timing effects.
- Provide performance estimations of the system models to evaluate the design decisions taken.
- Provide an infrastructure to start the refinement of the HW and SW components and their interconnections from the initial functional specification
- Work as a simulation tool integrated in design space exploration flows together with other tools required in the process

The first goal is to provide the designer with information about the system performance in terms of execution time and power consumption to make possible the verification of the fulfilment of the design constraints. This verification can be performed in two ways. First, SCoPE reports metrics of the whole system performance at the end of the simulation, in order to enable the verification of global constraints. This solution allows “black box” analysis, where designers can execute several system simulations running different use cases, to easily verify the correct operation in all the working environments expected for the system.

A second option enabled by SCoPE is to perform the verification of the system functionality and the checking of internal constraints. These internal constraints must be inserted in the application code using assertions. For that purpose, the use of the standard POSIX function “assert” is highly recommended. SCoPE offers to the designer functions that provide punctual information about execution time and power consumption during simulation. Using that functions, internal assertions can check the accomplishment of parameters as delays, latencies, throughputs, etc.

A second goal of SCoPE is to provide useful information to guide the designers during the development process. The co-design process of any system starts taken decisions about system architecture, HW/SW partitioning and resource allocation. To take the optimal decisions SCoPE provides a fast solution to easily evaluate the performance of the different solutions considered by the designer. Task execution times, CPU utilization, cache miss rates, traffic in the communication channel of power consumption in some HW components are some of the metrics the designer can obtain from SCoPE to analyze the effects of the different decisions in the system.

Another goal of SCoPE is to provide the designers with a virtual platform where the development of all the components of the system can start very early in the design

process. In traditional development flows, some components, such as SW components, cannot start their development process until a prototype of the target platform is built. However, it increases the overall design time since HW and SW components cannot be developed in parallel.

To reduce the design time, SCoPE provides an infrastructure for HW/SW modeling where the design of the SW components can be started. To enable that, SCoPE provides a fast simulation of the SW components considering the effects of the operating system, the execution time of the SW in the target platform and enabling the interaction of the SW with a complete HW platform model. Even the use of interruptions and drivers can be modelled in the simulation. The execution of the SW is then transformed in a timed simulation, where the use of services such as alarms, timeouts or timers can be explored in order to ensure certain real-time characteristics in the system.

Furthermore, the simulation of the SW using a native execution improves the debugging possibilities. Designers can directly use the debuggers of the host system, which has a double advantage: first, it is not necessary to learn how to use new debugging tools; second, the correct operation of the debuggers are completely guaranteed, and does not depend on possible errors in the porting of the tool-set to the target platform. Additionally, designers can easily access to all the internal values of both the SW and HW components, since all are modelled using a C++ simulation.

Finally, SCoPE has been developed of be used as an evaluation tool ready to be integrated in design space exploration (DSE) flows. DSE tools usually analyze the design space selecting the most interesting experiments among all the space population in order to reduce the number of evaluations required to find the optimal solutions. Then, an additional tool is required in order to evaluate the performance of the selected configurations. SCoPE can be used to obtain the performance estimations of all these selected experiments, providing the explorer tool with the needed information to obtain the final Pareto curves.

The design space exploration tools selects the most interesting points based on mathematical equations, but they has no intelligence or electronic knowledge to generate the corresponding models of the selected configuration. To solve that, SCoPE contains an automatic model generator. Using configurable XML descriptions, the exploration tool only require selecting valid values for the configurable parameters. Then SCoPE takes these values and generates the corresponding system model for its evaluation. This solution not only avoids requiring large intelligence in the explorer tool, but also reduces the exploration times, since no recompilation is required before each simulation.

In order to achieve all these goals, SCoPE implements a modeling infrastructure capable of supporting complete native co-simulation. SCoPE provides novel solutions to enable automatic annotation of the application SW, a complete POSIX-based RTOS model, models of most common HW platform components and an infrastructure for native execution of the SW and its interconnection with the HW platform. Additionally, SCoPE includes an input/output interface based on XML files. Through these files the user can be describe configurable systems obtaining system metrics. These files follows a format developed to enable compatibility between SCoPE and several design-space exploration tools, in order to use SCoPE as evaluation tool in the exploration flows. All these features are briefly explained below, together with the references to the corresponding publications, where more details about the proposed techniques can be found.

SW modeling solutions have become one of the most important areas of the thesis. As a stated before, it has been considered that the fastest possible execution of that functionality is the direct compilation and execution of the code in the host computer. Thus, the goal is to provide a modeling solution capable of evaluating system performance, but maintaining a similar execution speed, as long as possible. Specially, the modeling solution has to overcome two main limitations of functional execution with a minimum simulation overhead. First, functional executions do not consider any timing effect of executing the code in the target platform. As a consequence, no performance information and no constraint checking are available. Second, these executions cannot interact with the functionality implemented in HW components in the target platform. Thus, the simulation of the entire system functionality and the verification of the HW/SW integration are not possible.

To solve the first limitation, the solution proposed is to automatically modify the application SW in order to model performance effects. These performance effects include the execution of the code in the target processor core and the operation of the processor caches.

The general solution applied for that modeling is based on estimating the effects during SW execution and apply them to the simulation, just before the points where the SW tasks start communications with the rest of the system, usually system calls. Two main solutions have been explored for obtaining the estimations: the use of operator overloading and static annotation of basic-blocks.

The fist solution relies on the capability of C++ to automatically overload the operators of the user-defined classes. Using that ability, the real functional code can be extended with performance information without requiring any code modification. New C++ classes (generic_int, generic_char, generic_float, …) have been developed to replace the basic C data types (int, char, float, …) . These classes replicate the behavior

of the basic data type operators, but adding to all the operator functions the expected cost of the operator in the target platform, in terms of binary instructions, cycles and power consumption. The replacement of the basic data types by the new classes is done by the compiler by including an additional header with macros of the type:

"#define int generic_int"

A similar solution is applied to consider the cost of the control statements.

To apply that, a table with the cost of all the operators and control statements in the target platform must be provided by the user.

This solution has demonstrated to be easy to implement, and very flexible to support additional evaluations, since all the information is managed dynamically, including the data values. In that context, preliminary applications of the technique to evaluate different HW implementations have been explored, as ASAP or sequential resource scheduling.

More information about this technique can be found in *"System-Level Performance Analysis in SystemC", Proc. of DATE'04* and *"Single Source Design Environment for Embedded Systems Based on SystemC", DAES Journal, 2004.*

Nevertheless, this solution has several limitations. First, compiler optimizations are not accurately considered. Only a mean optimization factor can be applied. Furthermore, the use of operator overloading for all the data types implies a certain overhead, which slows down the simulation speed. As a consequence, other alternative techniques based on static annotation have been explored.

Solutions based on static annotation divides the performance modeling in two steps. First, the source code is statically analyzed, obtaining performance information for each basic block of the source code. After that, this information is annotated in the code, and during the simulation the cost of each basic block executed is accumulated and annotated at system calls.

For the static analysis, a parser based on an open-source C++ grammar has been implemented. Two versions of that parser have been created. In the first one, the parser analyzes the source code, obtaining the number and type of operators used on each basic block, as long as the control statements at the beginning of each block. Using that information and the table with the cost of each operator used for the previous technique it is possible to obtain the cost for the entire basic block. Then this cost is applied in the source code in the following way:

"segment_cycles += 120; segment_instructions += 20;"

As a result, during simulation the variables segment_cycles and segment_instructions contains the total cycles and instructions required in the target platform by the executed code. At each system call, the number of cycles is transformed into time and a SystemC "wait" is applied to model the processor execution time. Additionally, the variables are set to "0", to start the performance estimation of a new segment.

This solution requires more development effort than the operator overloading technique, especially for the implementation of the parser using the yacc/lex grammar. However, the simulation speed is really improved, achieving simulation times very close to the functional execution times (only two or three times slower). The main limitation of the technique is, again, the impossibility of accurately considering the compiler optimizations, since no analysis of the compiler output is performed. More information can be found in *"SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems", in the book L. Gomes and J. M. Fernandes (Eds.): "Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation", IGI Global. 2009*

To solve that drawback a second implementation has been performed. The third solution proposed is capable of maintaining the qualities of the previous annotation technique, but providing more accurate results, and including compiler optimizations. For that solution, the parser is executed in twice. At the first time, the code is parsed and annotated including volatile labels *("volatile label_xxx"*) at the borders of all the basic blocks. This code is cross-compiled for the target platform, and the output is analyzed in order to know the assembler instructions on the final binary code for each basic block (between the labels). Then, this information is used by the parser to increase the number of cycles and instructions properly. This solution has been presented in *"Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-Simulation ", GLSVLSI'10.* Results demonstrate an accuracy of about a 10% of error, maintaining the simulation speed of the previous solution.

Nevertheless, the performance of the SW in the target platform does not only depend on the binary instructions executed. Processor caches also have an important impact on it. Common cache models are based on memory access traces. However, in native co-simulation no traces about the accesses in the target platform are obtained. As a consequence, new solutions for modeling both instruction and data caches have been explored and included in SCoPE.

The modeling of instruction caches is based on the fact that instructions are placed sequentially in memory, in a place known at compilation time. Knowing the

amount of assembler instruction for each basic block it is possible to obtain a relative address for the instructions with respect to the beginning of the "text" section of the "elf" file. This information is used as variables' address to access the cache model, instead of the real access trace. Additionally, the use of static structs has been applied in order to speed-up the simulation speed, achieving a similar error and overhead for instruction cache modeling than for the static time annotation. More information can be found in *"Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-Simulation ", GLSVLSI'10.*

For data caches, the solution proposed uses corrected host addresses for each data variable used in the code. Additionally, global arrays handling information about the status of all the possible memory cache lines are used to improve the simulation speed maintaining the balance of the two previous techniques. The technique is described more in detail in *"Fast Data-Cache Modeling for Native Co-Simulation ", ASP-DAC'11 and "Obtaining Memory Address Traces from Native Co-Simulation for Data Cache Modeling in SystemC", DCIS'10.*

A final issue related to modeling the performance of the application SW is how to consider pre-emption. With the proposed modeling solutions, the segments of code between function calls are executed in "0" time, and after that, the time estimated for the segment is applied using "wait" statements. As a consequence, pre-emption events are always received in the "wait" statements. Thus, the segment has been completely executed before the information about the pre-emption arrives. As a consequence, the task execution order and the values of global variables can be wrong. In order to solve these problems, several solutions have been proposed in *"Real-time Operating System modeling in SystemC for HW/SW co-simulation". DCIS'05.* The final solution applied is to use interruptible "wait" statements. This approach solves the problems in the task execution order. Additionally, it is considered that possible modifications in the values of global variables are not a simulation error but an effect of the indeterminism resulting of using unprotected global variables. In other words, it is not really an error but only a possible solution. For more details, see *"SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems", in the book L. Gomes and J. M. Fernandes (Eds.): "Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation", IGI Global. 2009.*

Once solved the modeling of the application SW, the next step is to provide a model of the operating system. For that purpose, a model based on the POSIX API has been developed. The model uses the facilities for thread control of SystemC to implement a complete OS. Threads, mutexes, semaphores, message queues, signals, timers, policies, priorities, I/O and other common POSIX services are provided by the model. This work has been presented in *"POSIX modeling in SystemC" ASP-DAC'06*

and "RTOS modeling in SystemC for Real-Time embedded SW simulation: A POSIX model", DAES Journal, 2006.

Special interest has the modeling of separated memory spaces in the simulation. As SystemC is a single host process, the integration of SW components containing functions or global variables with the same names in a single executable, or the execution of multiples copies of components that use global variables result in name collisions. To solve that, an approach based on the use of protected dynamic variables has been presented in *"Modeling Separate Memory Spaces in Native Co-simulation with SystemC for Design Space Exploration ", 2PARMA workshop, 2010.*

However, the OS model is not only in charge of managing the application SW tasks. The interconnection between the native SW execution and the HW platform model is also performed by this component. For that goal, the model provides functions for handling interrupts and including device drivers following the Linux kernel 2.6 interfaces. More information can be found in *"TLM interrupt modeling for HW/SW co-simulation in SystemC", DCIS'06.*

Additionally, a solution capable of detecting and redirecting accesses to the peripherals directly through the memory map addresses as been implemented. Most embedded systems access the peripherals by accessing their registers directly through pointers. However, in a native simulation, pointer accesses do not interact with the target HW platform model, but with the host peripherals. In fact, accesses to peripherals results in segmentation faults, since the user code has no permission to perform this kind of accesses. To solve that, these accesses are automatically detected and redirected using memory mappings ("mmap()"), interruption handlers, and code injection, in order to work properly (*"Automatic HW/SW interface modeling for scratch-pad & memory mapped HW components in native source-code co-simulation", ISORC'09).*

Furthermore, a TCP/IP stack has been integrated in the model. For that purpose, the open-source, stand-alone lwIP stack has been used. The stack has been adapted for its integration into the SCoPE environment both for connecting different nodes in the simulation through network models, and for connecting the simulation with the IP stack of the host computer, in order to communicate the simulation with other applications. See *"Native Co-Simulation of TCP/IP-Based Embedded Systems in SystemC", DCIS'10.*

The OS model has been also extended to support dual operation as general-purpose and real-time operating system. For that goal, the dual Linux-based implementation developed by Thales Communication in the Hyades project has been analyzed, providing equivalent facilities in the model. Thus, the execution of SW tasks in both environments can be explored in order to obtain the optimal configurations. This

work has been presented in *"Early Modeling of Linux-based RTOS Platforms in a SystemC Time-Approximate Co-Simulation Environment", ISORC'10.*

Regarding the HW platform modeling, an infrastructure based on the SystemC TLM 2 standard has been implemented. A time-approximate bus model based capable of modeling how the bandwidth is divided among all the current transfers is provided in SCoPE. Additionally generic models for memory, DMA, network interfaces and bridges are included in the tool, together with a socket capable of connecting the low level functions of the OS model with the system buses. For network modeling, the network simulator Sicosys (http://www.atc.unican.es/SICOSYS/) has been modified and integrated in the SCoPE environment.

TLM templates capable of automatically handling the bus protocol has been also created, in order to make easier the development and integration of application-specific HW components by the users, as described in *"Protocol Bus Modeling using inheritance with TLM2.0", FDL'07.*

Summarizing, SCoPE is capable of modeling complete embedded systems with very high simulation speed, of about only one order of magnitude slower than functional executions and two or three order faster than ISS-based techniques. As a result the simulation can obtain performance results with 10-20% or estimation errors.

The very high simulation speed achieved has enabled its use in design space exploration flows. SCoPE has been integrated together with the explorer tools m3Explorer (http://home.dei.polimi.it/zaccaria/multicube_explorer_v1/Home.html) and modeFRONTIER (http://www.esteco.com/home.html).

For that purpose the SCoPE interfaces has been improved, in order to be inserted in an automatic exploration flow. The tool has been extended to accept the system description in a friendly format and automatically generate the system model. Two input XML files and a XML output file have been defined to provide the system description and return the simulation results.

The two input files are the following:

- a file describing the system and its configuration options, called System Description file
- a file of pairs identifier-value, fixing the selected configuration for each experiment, called System Configuration file

The XML System Description file includes information about the HW components, the HW architecture, and the SW tasks. A simple XML language has been developed to easily explain the proposed way to describe modifiable platforms. The language guarantees fast model creation and efficient system simulation.

Using this solution the external DSE explorer only has to indicate to the simulator the configuration to be analyzed each time by generating the corresponding System Configuration file. The simulation tool interprets the file, builds the system model and performs the simulation. This means that no user interaction or model recompiling is required once the exploration process starts.

The tool generates an output file, when the simulation finishes. This file contains the values of the metrics obtained during the simulation. The information returned is used by the DSE explorer to perform the search of the best solutions applying response surface modeling techniques.

To simulate each one of the configurations selected by the DSE tool, different system models are automatically created by SCoPE. The complete model description for each configuration is obtained applying the values of the XML System Configuration file to the XML System Description.

To generate the system model, the simulator dynamically creates instances of the required models and builds a platform model by interconnecting them as specified in the XML files. The generic component models developed to model the target HW platforms are fully configurable. These models contain a long range of configuration details to describe their functionality. Response times, delays, area, mean power consumption, power for access, frequency, memory size, IRQ or associated memory map addresses are some of the configuration possibilities.

To instantiate a component in the system model, all this parameters must be set. Parameter values are obtained from the values indicated in the corresponding clause of the XML System Description file. These parameters can be either defined as explicit values (“200MHz”, “500MB”) or identified as configurable values.

The instantiated components must be interconnected to create an executable system model. To simplify the interconnection work, TLM techniques have been used. TLM accurately describes the system communication architecture down to the level of individual read and write transactions. The use of transfers instead of signals, reduce the complexity when automatically interconnecting the system components.

To allow easy automatic interconnection of the system components, all component models have been created using a generic template provided by the

simulation engine. This template is oriented to ensure interface compatibility without limiting the component communication requirements. Ensuring that both ends of each interconnect have compatible interfaces, the automatic connection is possible.

To complete the HW platform generation, it is required the creation of the memory maps and ensure correct interrupt delivering. Each time a component is connected to a bus, its associated memory area is integrated in the memory map, ensuring that it has not been used before. The solution is similar for networking communication. Network models require the node identifier in order to configure the internal routing protocols properly.

OS models and SW tasks are finally added to the simulation as described in the XML files. An OS is mapped to a processor or group of processors (for SMP systems). SW tasks are associated to an operating system, and thus mapped to the system processors where the OS runs. SW tasks are defined in the XML System Description file indicating the name of the main function of the task. To load the main function, the dynamic library management is used, by calling the dlopen and dlsym function. Additionally, other parameters like the OS where it will run, the priority, the policy and the main function arguments can be defined. All these elements can be parameterized, so the DSE flow can explore the best configuration for the SW tasks.

More information about the file formats, the automatic generation of the system models and the integration of SCoPE with other exploration tools can be found in *“Automatic generation of modifiable platform models in SystemC for Automatic System Architecture Exploration”, DCIS’09 and "M3-SCoPE: Performance Modeling of Multi-Processor Embedded Systems for Fast Design Space Exploration", in the book C. Silvano, W. Fornaciari & E. Villar (Eds.): "Multi-objective Design Space Exploration of Multiprocessor SoC Architectures: the MULTICUBE Approach", Springer Ed.*

The proposed approach has been applied during the thesis to two main platforms, an OpenRISC 1500 platform and an ARM9 platform, in order to evaluate the accuracy of each on the techniques presented above. The OpenRISC platform has been developed in the University of Cantabria. It was presented in the paper *"Platform based on open-source cores for industrial applications", DATE’04.* The OpenRISC processor and a HW platform, containing bus, memory, a keyboard controller, serial port and a basic graphic card has been developed in SystemC and translated to Verilog for real implementation. With respect to the thesis, two ISSs (one cycle-accurate and another considering only instructions) has been developed together with Verilog components capable of obtaining a complete set of metrics about the performance of the real system. All these elements have been used to check and optimize the estimation approaches.

The ARM9 platform has been used to compare the estimation results of the different modeling solutions, in order to obtain the error when applied to one of the most popular processors in the embedded world.

Nevertheless the application of SCoPE has not been limited to the internal use in the University of Cantabria. The tool has been used by several partners in different European projects, especially by the company Marvel Hispania (previously DS2), where it has been fully integrated in the company design process. The application of SCoPE to the design of one of the company products has been presented in *M. Martínez, D. Ferrúz, H. Posadas, E. Villar "High-level modeling and exploration of a powerline communication network based on System-on-Chip", in the book C. Silvano, W. Fornaciari & E. Villar (Eds.): "Multi-objective Design Space Exploration of Multiprocessor SoC Architectures: the MULTICUBE Approach", Springer*

As a summary of the final results achieved, the following tables show the estimation accuracy of the SW modelling, and the simulation times for a list of examples:

	Number of instructions executed					
	Without optimizations (-o0)			With optimizations (-o2)		
	Skyeye	SCoPE	Error (%)	Skyeye	SCoPE	Error (%)
Bubble 1000	30504511	30504511	0	4010006	4510501	12,4812
Bubble 10000	5200180007	5200180007	0	400120008	400130013	0,0025
Vocoder	13466069	14066581	4,45945	6599330	8338713	26,357
Factorial	2747041	2996535	9,08228	1498521	1498518	0,0002
Hanoi	18481575	17695142	4,25523	13107284	11141209	14,9999
H264	5601674012	5800347641	3,54668			

Table 1. Comparison of number of instruction executed for an ARM9 platform

	Fallos de cache de instrucciones					
	Without optimizations (-o0)			With optimizations (-o2)		
	Skyeye	SCoPE	Error (%)	Skyeye	SCoPE	Error (%)
Bubble 1000	15	16	6,666666667	6	5	16,66666667
Bubble 10000	25	27	8	7	7	0
Vocoder	8	7	12,5	5	4	20
Factorial	20	18	10	12	10	16,66666667
Hanoi	46074	46761	1,491079568	25842	28607	10,69963625
H264	677865	589944	12,97028169			

Table 2. Comparison of instruction cache misses ARM926t platform

	Fallos de cache de datos					
	Without optimizations (-o0)			With optimizations (-o2)		
	Skyeye	SCoPE	Error (%)	Skyeye	SCoPE	Error (%)
Bubble 1000	126	127	0,79365	126	126	0
Bubble 10000	5199772	5209087	0,17914	5199310	5211595	0,23628
Vocoder	375	500	33,3333	375	500	33,3333
Factorial	38	45	18,4211	41	45	9,7561
Hanoi	6018	5908	1,82785	6026	5915	1,84202
H264	5944228	5950255	0,10139			

Table 3. Comparison of data cache misses ARM926t platform

	Skyeye	SCoPE		Without data cache		Without caches	
	Time	Time	Speed-up	Time	Speed-up	Time	Speed-up
Bubble 1000	0m2.186s	0m0.028s	x78	0m0.028s	x78	0m0.025s	x80
Bubble 10000	4m6.500s	0m3.486s	x71	0m2.792s	x88	0m1.92s	x130
Factorial	0m1.071s	0m0.014s	x76	0m0.014s	x76	0m0.012s	x90
Hanoi	0m9.426s	0m0.043s	x219	0m0.032s	x294	0m0.020s	x479
Vocoder 10	0m48.793s	0m0.262s	x187	0m0.185s	x263	0m0.105s	x464

Table 4. Comparison of simulation times between SCoPE and Skyeye

As can be shown, simulation speed-ups of two or more orders of magnitude can be achieved by assuming an acceptable error, below 20%.

Conclusions

In this thesis, several solutions have been developed in order to cover all the features required to create an infrastructure capable of obtaining sufficiently accurate performance estimation with very fast simulation speeds. These solutions are based on the idea of native co-simulation, which consists in the combination of native simulation of annotated SW codes with time-approximate HW platform models. All these techniques has been integrated in a simulation tool called SCoPE which can be used as an independent simulator or can be used integrated in different design space exploration flows.

The modeling solutions can be divided in three main groups: solutions for modeling in the native execution the operation of the application SW in the target platform; a complete operative system model; and an infrastructure for easily creating the HW platform models. All these solutions have been implemented as SystemC extensions, using the features of the language to provide multiple execution flows, events and time management. Additionally, an infrastructure for automatic interconnection with design space exploration tools has been developed.

The modeling of the application SW considers the execution times and power consumption of the code in the target platform, as long as the operation of the processor caches. Three different solutions for modeling the processor performance have been explored in the thesis (operator overloading, annotation based on source code analysis and annotation based on binary code analysis), in order to find an approach capable of obtaining accurate solutions with minimal simulation overheads and as flexible as possible, to minimize the effort required to evaluate different target processors and platforms. As a result of the study, the annotation based on binary code analysis has demonstrated to obtain the best results with minimal simulation overhead. Additionally, the technique is very flexible, since only requires a cross-compiler for the target platform capable of generating object files from the source code. No additional libraries, ported operating systems, or linkage scripts are required. Additionally, it has been demonstrated that cache analysis for both instruction and data caches can be performed obtaining accurate results with adequate simulation times.

A POSIX-based operating system model has been also developed on top of SystemC. The OS model provides all the services usually required in common POSIX environments, as long as support for memory space separation, drivers' integration and interruption handling. Additional extensions as a TCP/IP package and a real-time extension has been added to the model, which extends the modeling capabilities not only of the SCoPE tool, but also the SystemC language itself.

Additionally, a solution has been developed to enable the possibility of describing fully configurable target systems (including HW platform, SW components and allocation) in terms of XML files. These files can be read by the tool, which is capable of automatically generating the system models without requiring external re-coding or additional recompiling steps. This enable easy integration of the tool in deferent design exploration flows.

Finally, the tool has demonstrated to be useful and valid for its integration in the design process of an electronic company, providing interesting improvements in their design process. The company Marvell Hispania is currently using this technology, which is a nice example of a successful story.

Summarizing, this thesis demonstrates that the SystemC language can be extended to enable the early modeling and evaluation of electronic systems, and providing important information to help the designers during the first steps of the design process. These extensions allow using a SystemC-based infrastructure for functional simulation, performance evaluation, constraint checking and HW/SW refinement.

Agradecimientos

La realización de este trabajo no habría sido posible sin la ayuda de todos aquellos que han colaborado conmigo en cada una de las etapas llevadas a cabo. Por ello me gustaría comenzar este documento agradeciéndoles todo su esfuerzo y ayuda.

En primer me gustaría agradecer su apoyo a mi director de tesis, Eugenio Villar, a Pablo Sánchez y a Fernando Herrera, por sus aportaciones durante todo el desarrollo del trabajo. Así mismo, querría dar las gracias a todos los compañeros de laboratorio que han colaborado en el desarrollo e implementación de las distintas partes descritas en este documento, especialmente a Juan Castillo y David Quijano, por el esfuerzo de modificar la primera versión de la herramienta en una versión más estable y distribuible, además de por sus aportaciones en el modelado de caches de datos, parseado y modelado de drivers, a Jesús Adámez, por su colaboración en el chequeo de la interfaz POSIX, a Luis Díaz, por su colaboración en el modelado de caches de datos, a Gerardo de Miguel, por su trabajo en la generación de los archivos XML de salida, y a Juan Castillo, María Bolado, Carlos Sánchez y Pablo Huerta por su trabajo en el desarrollo de la plataforma OpenRISC 1500.

También me gustaría dar las gracias a aquellos que han utilizado, extendido y en general "sufrido" con el uso de SCoPE, realizando extensiones que no han sido incluidas en este documento: a Gerardo Caballero, por su interfaz Win32, a Daniel Calvo, por la integración de archivos IP-XACT y el plug-in térmico, a Roberto Varona, por su integración con AADL, a Sara Real, por su trabajo en los modelos ARM y el modelo de cache L2, a Patricia Botella, por la nueva implementación para generación de procesos dinámicos, a Álvaro Díaz, por su trabajo en la interfaz gráfica, y a Pablo González por la extensión de la técnica de estimación desde código binario.

Además he de tener un recuerdo especial para aquellos que han contribuido desde fuera de la universidad. Ahmed A. Jerraya, Lobna Kriaa, todos los integrantes del laboratorio SLS de TIMA y Francisco y Virginia por su apoyo durante mi estancia en Grenoble. Anne-Marie Fouilliart y Dominique Ragot de Thales Communications, por su información sobre la extensión de Linux de tiempo real, y el modelo CORBA. Cristina Silvano, Gianluca Palermo y Vittorio Zaccaria del Politécnico de Milán, y Carlos Kavka de ESTECO por su ayuda en la integración de SCoPE con M3Explorer y modeFrontier. A Geert Vanmerbeeck y Prabhat Avasare de IMEC por proporcionar el ejemplo de MPEG4 y la librería ARP. A la gente de NXP, por proporcionarnos información confidencial sobre tiempos y consumos de sus procesadores ARM. Y sobre todo a Marcos Martínez, Francisco Blasco y David Ferruz de DS2 (recientemente convertida en Marvell Hispania), por haber confiado en nosotros desde el primer momento y haber integrado plenamente la herramienta en el flujo de diseño de la compañía.

Y por supuesto a mis padres Fermín y Cristina, a mis amigos y a todos los demás compañeros y amigos que han estado trabajando en el laboratorio, aunque no hayan tenido una relación especial con este trabajo, especialmente a Pablo Peñil, Javi Barreda, Marga Díez, Jesús Pérez, Miriam, Cristina, Edu, Iván, Raul, …, y que se merecen mi agradecimiento tanto o más que los anteriores.

Nota:

Este documento no pretende ser una descripción o manual de usuario de SCoPE, por lo que no debe ser utilizado como tal. El objetivo es describir las ideas técnicas y soluciones desarrolladas durante el proceso de investigación, y no su implementación final. De hecho, no existe ninguna versión estable de SCoPE en la cual se integren todos los elementos descritos aquí.

Índice

SECCIÓN I: Introducción

1 Introducción 7

1.1 Reseña histórica 8

1.2 Motivación 16

1.3 Objetivos del trabajo 20

SECCIÓN II: Estado del Arte

2 Flujo de diseño de sistemas embebidos 22

2.1 Objetivos del flujo de diseño 23

2.2 Etapas del flujo de diseño 25

2.3 Localización del trabajo realizado en el flujo de diseño 28

3 Lenguajes y estándares 30

3.1 SystemC 31

3.2 TLM 35

3.3 POSIX: IEEE 1003.1 versión 3 40

3.4 **Linux** 42

3.5 CORBA 48

4 Análisis del estado del arte 52

4.1 Trabajos para exploración del espacio de diseño 52

4.2 Estimación de prestaciones 54

4.3 Técnicas de estimación dinámicas 56

4.4 Simulación nativa de código fuente 59

4.5 Modelado de la plataforma HW 64

4.6 Modelado de consumos 65

4.7 Limitaciones en el estado del arte 66

SECCIÓN III: Contribuciones

5 Infraestructura de simulación para estimación de prestaciones 68

5.1 Objetivos de SCoPE 68

5.2 Modelado de sistemas electrónicos con SCoPE 71

5.3 Nivel de abstracción 77

5.4 Elementos modelados y arquitectura interna de SCoPE 79

5.5 Estructura interna 87

5.6 Interfaces de entrada y salida 91

6 Modelado del SW de aplicación 94

6.1 Modelado del subsistema SW 94

6.2 Modelado del código de aplicación 94

6.3 Modelado de caches 125

6.4 Modelado del avance de tiempo de la ejecución SW 135

6.5 Estimación de consumos de SW de aplicación 144

7 Modelado de la plataforma SW 146

7.1 Modelado Abstracto de Sistemas Operativos 149

7.2 Paquetes integrados en el modelo de sistema operativo 188

7.3 Modelado de "middleware": CORBA 197

8 **Modelado de la plataforma HW** 200

8.1 Introducción 200

8.2 Modelado HW de los nodos de cómputo 202

8.3 Modelado de red 213

9 Exploración del Espacio de Diseño 217

9.1 Introducción 217

9.2 **Integración del simulador y la herramienta de exploración** 218

9.3 Archivos XML 219

9.4 Ejemplo de aplicación 238

9.5 Uso de SCoPE mediante la interfaz XML 241

SECCIÓN IV: Resultados y conclusiones

10 Plataformas sobre las que se ha probado la metodología 243

10.1 Plataforma OpenRISC 1500 243

10.2 Plataforma ARM 252

10.3 Utilización para Exploración del Espacio de Diseño 255

10.4 Estimaciones de componentes HW 265

11 **Conclusiones** 279

12 Participación en proyectos y diseminación 266

12.1 Proyectos donde se ha desarrollado el trabajo 266

12.2 Página Web y descarga 267

12.3 Uso de SCoPE en empresas 268

12.4 Diseminación 274

12.4.1 Demostraciones en congresos 274

13 **Referencias** 279

NOTA DE FUENTES

- Figura 1.1 : *Intel*
- Figura 3.2: *Swan, S. "An Introduction to System Level Modeling in SystemC 2.0". www.systemc.org, White Paper*
- Figura 3.3: "*TLM2 Lenguage Reference Manual", www.systemc.org*
- Figura 3.5.*IBM* *http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/figure1.gif*
- Figura 5.3: *IBM* *http://publib.boulder.ibm.com/infocenter/db2luw/v8/topic/com.ibm.db2.udb.doc/admin/00000189.gif*
- Figuras 10.3 y 10.9, 11.3, 11.5 y 11.6: *Silvano, C, Fornaciary, W & Villar, E."Multi-objective Design Space Exploration of Multiprocessor SoC Architectures". Springer*

SECCIÓN I: INTRODUCCIÓN

"Fallar en conocer la situación de los adversarios por economizar en investigación no es típico de un buen jefe militar ni de un gobernante victorioso. Lo que posibilita a un gobierno inteligente y a un mando militar sabio lograr triunfos extraordinarios es disponer de la información esencial."

Sun Tzu, El arte de la guerra (S. VI A.C.)

1 Introducción

La electrónica se ha convertido en las últimas décadas en una de las áreas de mayor crecimiento científico y económico en el mundo. La invención de los circuitos integrados a mediados del siglo pasado y su rápida aplicación comercial ha dado lugar a la aparición de todo un mundo de nuevos productos y nichos negocio. Este crecimiento económico ha fomentado y financiado un gran avance tecnológico, dando lugar a una fuerte espiral en la que economía y tecnología progresan simultáneamente.

Esta espiral de crecimiento ha dado lugar a modelos de negocio muy dinámicos, donde el tiempo de vida de los productos en el mercado es bastante bajo, especialmente si lo comparamos con otros sectores económicos. Pensemos, por ejemplo, que un coche diseñado hace 20 años, puede circular hoy en día por las carreteras sin mayor inconveniente. Sin embargo, es completamente inviable ejecutar programas actuales en un computador de la misma época.

El dinamismo de este mercado electrónico hace necesario que los nuevos componentes y sistemas puedan ser diseñados de manera rápida y eficiente. Además, dada la corta vida del diseño, los costes no recurrentes debidos al propio proceso de diseño adquieren gran importancia en el precio del producto final. Por estas razones, tanto el coste como el tiempo de desarrollo de los sistemas electrónicos se han convertido en un aspecto crítico en el proceso de diseño.

Para optimizar estos parámetros y gestionar adecuadamente la gran capacidad de crecimiento tecnológico actual, es necesario incrementar simultáneamente la capacidad de diseño de los ingenieros electrónicos. Esto implica desarrollar nuevas soluciones que permitan a los diseñadores utilizar todo el potencial tecnológico sin disparar el esfuerzo. Para proporcionar estas soluciones, las técnicas de diseño electrónico han ido evolucionando hacia niveles de abstracción cada vez más altos, reduciendo el esfuerzo de los niveles más bajos. Siguiendo esta tendencia, la presente tesis está orientada al desarrollo de nuevas técnicas de diseño de sistemas electrónicos en alto nivel. Más concretamente se centra en las técnicas de co-simulación, estimación y modelado de sistemas HW/SW con tiempos aproximados.

Comenzaremos la memoria de tesis mostrando la gran importancia que la electrónica ha tenido desde mediados del siglo pasado. Posteriormente veremos como se ha producido el desarrollo tecnológico hasta el momento actual. Esto nos permitirá presentar los tipos de sistemas que en la actualidad y a corto y medio plazo coparán el mundo electrónico y sus necesidades de diseño, para las cuales se han desarrollado las ideas y soluciones técnicas presentadas en los capítulos siguientes.

1.1 Reseña histórica

Desde el principio de los tiempos el poder de la información ha sido gran importancia. Conocer la posición de las tropas enemigas, el exceso o demanda de productos en tierras lejanas o el acceso a nuevas tecnologías o rutas comerciales han sido históricamente elementos fundamentales en el desarrollo civil y militar. Sin embargo no ha sido hasta el siglo XX cuando el hombre ha sido capaz de gestionar la información a gran escala. La aparición de la electrónica ha permitido manejar grandes cantidades de información en tiempos cada vez menores, obteniendo resultados insospechados hasta entonces. Como ejemplo histórico, consideremos los principios de la electrónica tal y como la conocemos hoy en día. Miremos a la segunda guerra mundial.

Mientras Alemania dedicaba todos sus esfuerzos en desarrollar armamento cada vez más sofisticado, Inglaterra, sin la capacidad militar alemana ni los enormes recursos rusos necesitaba una solución alternativa. Después del éxito estratégico proporcionado por el radar en la batalla de Inglaterra, la electrónica demostró ser quizá el arma más importante de la guerra en Europa occidental.

Las bombas diseñadas por Alan Turing permitieron romper el código enigma, siendo fundamentales tanto en el combate contra los submarinos alemanes en la batalla del Atlántico como en el bloqueo de los suministros que neutralizó al Afrika korps. Sin embargo, no fue hasta la operación Overlord cuando la electrónica alcanzó su punto culminante.

Año 1944, Francia ocupada. Las tropas alemanas habían sido confinadas a la Europa continental, y el siguiente paso era necesariamente un desembarco sobre Francia. La apertura de un segundo frente es imprescindible para la victoria aliada. Sin embargo establecer una cabeza de puente venciendo a las divisiones alemanas del muro atlántico parecía un suicidio.

El 1 de Junio, tras los recibir los informes de inteligencia, Hitler ordenó al mariscal Rommel la movilización de todas las tropas necesarias para repeler el desembarco aliado en Calais. Además, aprovechando las fuertes tempestades previstas para los días siguientes, ordenó también preparar un juego de guerra en Rennes para los generales. Tras culminar los preparativos, el día 4 Rommel partió hacia Berchtesgaden para informar al Führer y recibir las últimas indicaciones. Contando con la tormenta y el desconocimiento enemigo, gran parte de la costa francesa había quedado desprotegida, para preparar al máximo la esperada batalla.

Sin embargo, la madrugada del día 6 Eisenhower ordenó a sus tropas embarcar, y tras esperar más de una hora a una tregua de la enorme tempestad, lanzó un ataque total sobre la

costa de Normandía. Con las divisiones alemanas atrincheradas en Calais, y los generales alejados del frente, el desembarco aliado resultó un gran éxito.

Este éxito se debió a que los tres supuestos alemanes habían fallado: que el ataque se realizase en Calais, que los aliados no se atrevieran a atravesar la tormenta y, sobre todo, el desconocimiento enemigo de sus movimientos. El nuevo Mark II Colossus acababa de romper la codificación que el propio Hitler utiliza para comunicarse con su alto mando, dando a los aliados gran cantidad de información sobre las operaciones alemanas. Eisenhower conocía todas las operaciones alemanas y sabiendo que esa era su gran oportunidad no la desaprovecho. Además, al conocer las ordenes secretas alemanas durante el transcurso de la batalla, los aliados pudieron preparar respuestas adecuadas a todos los intentos de replica.

La importancia del Colossus en la victoria fue tan grande, que al final de la guerra, en 1946, Churchill ordenó destruir inmediatamente ocho de los diez Colossus activos para evitar que los detalles sobre su funcionamiento cayeran en manos de espías extranjeros.

Estos primeros éxitos hicieron que tanto la informática como la electrónica recibiesen un impulso muy importante durante las décadas siguientes. En el mundo de la electrónica, unos pocos años después del final de la guerra, la integración de circuitos fue conceptualizada por el científico de radares Geoffrey W. A. Dummer (1909-2002), que estaba trabajando para la “Royal Radar Establishment” del Ministerio de Defensa Británico. El primer circuito integrado fue desarrollado poco después, en 1958 por el ingeniero Jack Kilby (1923-2005) tras haber sido contratado por la firma Texas Instruments. Se trataba de un dispositivo de germanio que integraba seis transistores en una misma base semiconductora para formar un oscilador de rotación de fase. (En el año 2000 Kilby fue galardonado con el Premio Nobel de Física por la contribución de su invento al desarrollo de la tecnología de la información).

A partir de aquí, los circuitos integrados comenzaron un desarrollo espectacular, que no se ha detenido hasta hoy. Ya en un artículo de 1950, Alan Turing predijo que los computadores tendrían un billón (10^9) de palabras de memoria al final del siglo [Turing50]. Pero la frase que más claramente muestra el crecimiento de la electrónica se debe a Gordon E. Moore. En el “Electronics Magazine” de 19 de Abril de 1965, Moore afirmó que el número de transistores por circuito integrado se doblaría cada año, en un artículo titulado "Cramming more components onto integrated circuits”:

> "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer." [Moore65]

Esta afirmación fue denominada “ley de Moore” por el catedrático de Caltech Carver Mead, denominación que perdura hasta hoy [Mead70]. Moore modificó la ley en 1975, indicando que el número de transistores se doblaría cada dos años [Moore75]. Finalmente,

esta ley fue ajustada indicando que el tiempo requerido para doblar la capacidad de los circuitos es de 18 meses [Kanellos03].

A la vista de los acontecimientos se puede decir que los incrementos exponenciales en las capacidades de integración de los circuitos integrados han permitido una evolución tecnológica a gran escala. En 40 años los chips electrónicos han pasado de los mil transistores a finales de los 60 a mil millones de transistores en la actualidad.

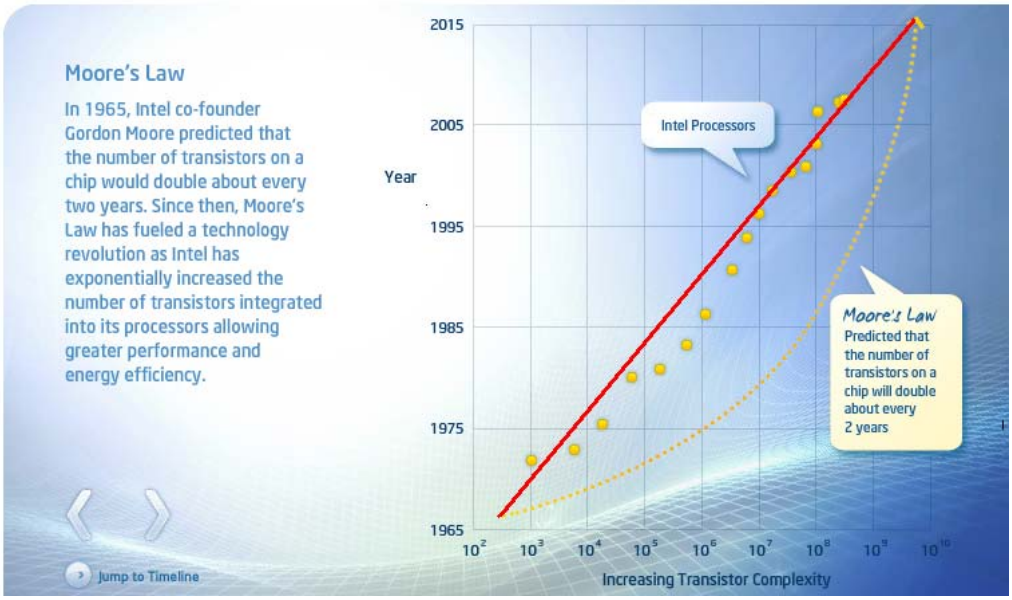


Figura 1.1: Evolución del número de transistores por chip (Intel)

Este aumento del número de transistores disponibles por chip ha dado como resultado un aumento de la complejidad de los circuitos y sistemas electrónicos. Como consecuencia se ha producido un aumento tanto en el número y complejidad de las tareas realizadas como de las unidades funcionales que se pueden integrar en un chip. Los circuitos integrados han pasado de tener un simple microprocesador o un pequeño HW específico a contener sistemas enormemente complejos enteros, compuestos por múltiples procesadores, módulos HW, Entrada/Salida, elementos de interconexión, etc. todos en el mismo chip, dando lugar a los sistemas embebidos o empotrados.

Para gestionar el aumento de capacidad tecnológica ha sido necesario aumentar la capacidad productiva de los diseñadores. Para ello las técnicas de diseño han evolucionado en consonancia, desde los métodos iniciales de diseño de circuitos integrados, válidos para implementar diseños de unos miles de transistores, hasta las técnicas actuales que permiten crear sistemas de cientos de millones de transistores.

En los primeros circuitos integrados, el diseño electrónico se realizaba dibujando manualmente las máscaras que debía utilizarse en la fabricación de los circuitos. Los

componentes de óxido, polisilicio o metal necesarios para implementar los transistores e interconectar el circuito eran trazados uno a uno.

Posteriormente se pasó a diseñar el circuito en base a transistores primero, y puertas lógicas después. Los transistores y puertas lógicas se podían transformar mediante herramientas automáticas en sus componentes primarios y de esa forma generar las máscaras necesarias en el proceso de fabricación. La idea de utilizar elementos básicos que se pudieran traducir automáticamente al ser meras copias unos de otros evitó una gran parte del esfuerzo de diseño inicial. Además, las herramientas de diseño eran capaces de ubicar cada una de las partes del circuito de la forma más óptima. De esta manera no era necesario planear totalmente el dibujo del circuito, sino que sólo se requería realizar una descripción funcional en base a puertas lógicas del circuito [Sangiovanni04].

Conforme continuó el crecimiento de los sistemas electrónicos, las descripciones con puertas lógicas se volvieron demasiado complejas. Como consecuencia, se desarrollaron los lenguajes de descripción HW (HDL). Estos lenguajes permiten diseñar los sistemas describiendo mediante código las transferencias entre los registros del dispositivo. En consecuencia, este tipo de descripciones se denominan descripciones de nivel de transferencia entre registros (RTL).

Desde estas descripciones RTL, las herramientas de diseño pueden generar automáticamente circuitos con puertas lógicas capaces de realizar la funcionalidad descrita, y de ahí las máscaras necesarias para su fabricación. Los HDL además permiten la simulación de los circuitos diseñados de forma sencilla.

No obstante, la utilización de descripciones RTL de todos los componentes del sistema implica un alto coste computacional en las simulaciones. Si bien estas descripciones pueden ser necesarias para los nuevos componentes a diseñar, otros componentes predefinidos no necesitan tal grado de detalle. El componente paradigmático de este caso es el procesador. En sistemas con componentes HW/SW el modelado interno del procesador a nivel RTL produce simulaciones muy exactas, pero tiene dos inconvenientes: en primer lugar requiere altos tiempos de simulación y, en segundo lugar, el código RTL no suele estar disponible por problemas de propiedad intelectual [Sewal07].

Para solventar estos problemas se ha propuesto la utilización de simuladores funcionales de los procesadores. Estos simuladores reciben los códigos binarios desde la memoria del sistema, e interpretan y ejecutan las operaciones binarias. Para ello tienen en cuenta detalles internos del procesador, como el número de registros o el estado de la cache, pero las descripciones se realizan a nivel funcional, no mediante señales. Estos simuladores están realizados a partir de las descripciones del conjunto de instrucciones del procesador, por lo que reciben el nombre de simuladores de conjunto de instrucciones (Instruction Set Simulator, ISS) [Skyeye]. La utilización de ISSs junto con descripciones de precisión de ciclo del HW ha sido una solución ampliamente utilizada para simular conjuntamente SW y HW hasta el momento actual [Synopsys] [Benini05].

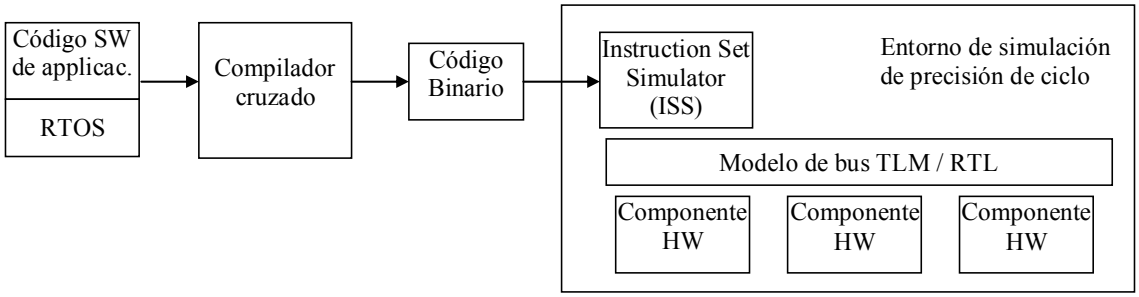


Figura 1.2: Simulación de sistemas electrónicos mediante ISS + RTL

Aunque los ISSs permiten simulaciones más rápidas que las descripciones RTL, aún están tremendamente alejados de las velocidades de ejecución del código puramente funcional. Una ejecución funcional del código SW se realiza compilando y ejecutando el código directamente sobre un computador convencional, llamado nativo o "host" (e.g. un PC), aunque no sea esa la plataforma para la que esté diseñado.

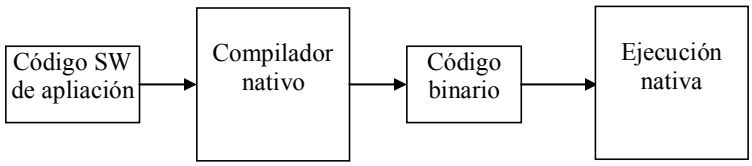


Figura 1.3: Ejecución funcional del código de aplicación

Esta ejecución se realiza aproximadamente a una velocidad de una instrucción ensamblador por ciclo del procesador nativo. Por el contrario, la ejecución con un ISS requiere la actualización de todos los componentes internos del modelo del procesador destino por cada instrucción ensamblador. Esto puede significar la ejecución de miles de instrucciones o más, lo que implica que cada instrucción binaria requiere 100 instrucciones en el ordenador de la simulación. La velocidad del ISS es, pues, unos 4 ó 5 órdenes de magnitud más lenta que la ejecución funcional. Por contra, la ejecución funcional no genera información sobre el tiempo de ejecución en la plataforma destino, sobre consumos o sobre los accesos al bus.

En resumen estas dos técnicas presentan balances muy dispares, surgiendo la pregunta de si serán suficientes para abordar adecuadamente los próximos diseños electrónicos o habrá que desarrollar tecnologías intermedias.

1.1.1 Los sistemas embebidos integrados: SoCs y MPSoCs

Conforme el mercado de la electrónica evoluciona, los productos se vuelven cada vez más elaborados, siendo capaces de proporcionar más servicios de mayor complejidad. Operaciones matemáticas complejas, operaciones simples con grandes cantidades de datos, sub-sistemas de decisión y aprendizaje o elementos de procesado de señal, entre otros, se combinan para satisfacer las necesidades de los usuarios.

Para implementar eficientemente esta gran diversidad de tareas se necesita tanto aumentar la capacidad de cómputo como combinar diferentes tipos de recursos de cómputo, utilizando para cada tarea el recurso más apropiado. Por esta razón, los sistemas electrónicos tienen como integrantes fundamentales una o varias unidades procesadoras de propósito general (GPP) junto con unidades de cómputo de propósito más específico, como tratamiento digital de señal (DSP). Además, estas unidades están rodeadas de dispositivos HW encargados de la realización de tareas específicas. En definitiva, los requerimientos impuestos para obtener una relación adecuada entre coste y rendimiento implican la necesidad de manejar sistemas realmente heterogéneos.

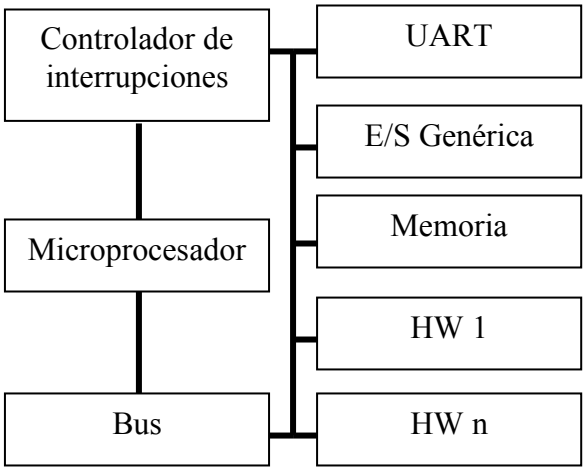


Figura 1.4: Ejemplo de componentes de un SoC

En muchas ocasiones, estos sistemas se encuentran embebidos en sistemas mayores; en otras, las características de la electrónica de consumo requieren de ligereza y manejabilidad. Con objeto de alcanzar los reducidos tamaños, pesos y consumos requeridos en estos ámbitos, las capacidades de integración de los circuitos electrónicos se vuelven críticas. La gran cantidad de transistores por chip disponibles en la actualidad ha permitido que procesadores y componentes HW se puedan integrar en un solo chip, dando lugar a compatos sistemas en chip (SoC). Cuando se integran varios procesadores en el mismo chip, el sistema se denomina SoC multi-procesador (MPSoC).

La integración gran cantidad de componentes en un solo chip hace que estos sistemas tengan unas características especiales, distintas de las de los sistemas electrónicos más tradiciones. En primer lugar, hay que considerar que en la mayoría de los casos los SoC están diseñados para realizar un funcionamiento autónomo, a diferencia de otros sistemas, como por ejemplo los PCs, donde el usuario está permanentemente pendiente del monitor.

Un SoC es además un sistema con un alto grado de heterogeneidad por la combinación de hardware a medida y software embebido. La interacción entre los componentes HW y SW del sistema es muy alta, de tal forma que las técnicas de diseño deben estar específicamente diseñadas para gestionar adecuadamente dicha heterogeneidad. Siguiendo con el ejemplo del

PC, las técnicas de diseño comúnmente utilizadas para realizar la interacción de un procesador con sus periféricos, pueden ser muy ineficientes en un SoC.

Como consecuencia de estas diferencias en arquitectura y características, las metodologías de diseño requeridas para el diseño de SoC han de ser distintas a las de otros sistemas electrónicos más tradicionales. Para definir las nuevas metodologías hay que tener en cuenta que un sistema embebido se define por las siguientes características:

- Concurrencia y tiempo real. Los componentes del sistema responden en función de eventos externos, que pueden llegar a la vez, por lo que deben operar simultáneamente. Además el tiempo de respuesta debe ser limitado para que la reacción al estímulo sea eficaz.
- Fuerte interacción HW/SW. Los sistemas empotrados interaccionan con el entorno a través de dispositivos E/S específicos por lo que las comunicaciones HW/SW son críticas.
- Fiabilidad y seguridad. El sistema debe ser fiable y seguro frente a errores, ya que suele requerir un comportamiento autónomo.
- Bajo consumo. La integración de los SoC en otros sistemas, o su uso para dispositivos portátiles, conlleva grandes limitaciones en el consumo energético. Por un lado, en muchas ocasiones los sistemas no tienen acceso a una gran fuente de energía, como un generador o un cable eléctrico. Deben funcionar con baterías de capacidad reducida. Por otro lado, su uso en estos entornos implica una limitación del calor que pueden generar.
- Bajo coste y reducido tiempo de desarrollo. La rápida evolución tecnológica ha hecho que el mercado de la electrónica sea uno de los más dinámicos y de márgenes más ajustados, haciendo que pequeñas diferencias en el coste de los diseños y reducciones en el tiempo de desarrollo sean muy importantes.

Además de estas características generales, las arquitecturas HW y SW presentan sus propias particularidades. En cuanto al desarrollo del SW para un SoC, considerando que un SoC suele requerir de un funcionamiento autónomo, la interacción con el usuario esta restringida. Operaciones como la carga y ejecución de nuevos programas o la actualización del sistema no son nada habituales. Esto significa, que elementos imprescindibles en un PC, como una Shell, pasan a ser innecesarios.

Finalmente hay que tener en cuenta que la integración de múltiples procesadores accediendo simultáneamente a las unidades de memoria y a los periféricos hace que las arquitecturas utilizadas tradicionalmente, basadas en buses, dejen de ser válidas. Los buses representan una manera sencilla de comunicar los componentes de un sistema entre si, sin embargo presentan una baja escalabilidad. Conforme aumenta el número de maestros conectados al bus, su rendimiento cae drásticamente. Para reducir este problema, la propuesta más habitual es sustituir las comunicaciones basadas en bus por redes. La localidad espacial hace que estas redes presenten ciertas diferencias con las redes convencionales, pensadas para comunicaciones a ciertas distancias. Para diferenciar estos nuevos tipos de redes, se utiliza el término redes en chip (NoC).

La posibilidad de combinar buses y redes da una gran flexibilidad al diseño de los MpSoC. Desde arquitecturas específicas multiprocesadoras, a configuraciones multi-computador, basadas en la idea de cluster, es posible definir una gran cantidad de implementaciones arquitecturales [Dally07].

De modo general vamos a considerar dos tipos de arquitecturas basadas en NoC: arquitecturas basadas en múltiples nodos computadores y arquitecturas donde cada componente HW se conecta directamente a la red. En lo sucesivo llamaremos a las primeras arquitecturas multi-computador, y las segundas, arquitecturas multiprocesador.

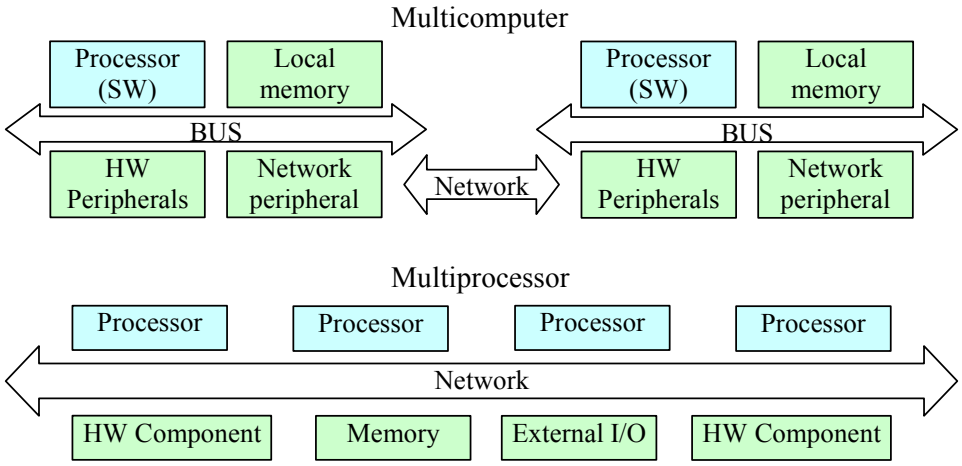


Figure 1-5: Prototipos de plataforma MpSoC contemplados

En plataformas especificas de multiprocesador, los procesadores están conectados directamente a la red, formando una única unidad funcional. Los procesadores comparten la memoria, los periféricos, el sistema operativo y hasta el mapa de memoria. No hay memoria local a cada nodo, más allá de la cache de primer nivel. En su lugar, la memoria y los periféricos están distribuidos por la infraestructura de comunicaciones. Esto da lugar a la aparición de sistemas realmente simétricos, ya que es completamente indistinguible para las aplicaciones SW saber en que procesadores están, pudiendo realizar las mismas operaciones en cada uno de ellos.

La existencia de una única entidad multi-procesadora permite el uso de sistemas operativos realmente distribuidos entre los procesadores. Las tareas pueden ejecutarse en cualquier procesador, incluso redistribuyéndose entre distintos puntos de ejecución. No es necesario mover los espacios de memoria de código o datos entre distintas unidades de memoria. De esta forma la carga computacional puede ser redistribuida óptimamente entre todos los procesadores. Los distintos hilos de un mismo proceso pueden estar ejecutando en simultáneamente sin problemas.

Sin embargo, estas infraestructuras tienen una carga de comunicaciones muy alta. Aunque no es necesario mover los espacios de memoria de los procesos, las transferencias de datos son muy altas. Hay que tener en cuenta, que al permitir la ejecución en paralelo de

múltiples hilos que actúan sobre el mismo espacio de memoria los sistemas de mantenimiento de coherencia entre los procesadores requieren un esfuerzo muy alto. Además, al mover dinámicamente las tareas, las caches deben ser cargadas y descargadas continuamente. Por último, dado que son sistemas teóricamente simétricos, la distribución de las tareas puede hacer que los datos tengan que atravesar el chip constantemente de un extremo a otro.

Todo esto no solo supone un problema a la hora de requerir una infraestructura de red muy potente, sino que además requiere un porcentaje importante de área en el chip e implica un consumo de energía bastante elevado. Aunque estas redes emplean protocolos simples que permiten minimizar las sobrecargas, como las arquitecturas CMP (continuous multicast push), los requerimientos siguen siendo altos. Estas redes complejas, comúnmente requieren entorno a un 10% del chip.

En el otro extremo encontramos las arquitecturas basadas en múltiples nodos de computación. En estos sistemas, cada nodo se comporta como un procesador más o menos independiente, de forma que cada procesador posee su propia memoria y periféricos conectados al bus local. La red es por tanto utilizada únicamente para transferir información entre los nodos de cómputo, comunicaciones que sólo se realizan en caso necesario, normalmente explicitas en el código. Esto produce una reducción drástica de los requerimientos de comunicación a lo largo del chip.

Por el contrario, las tareas no pueden moverse de procesador. Los mapas de memoria son independientes, con sistemas operativos normalmente independientes. Esto significa que la memoria de un nodo no es visible desde otro nodo, y que además el sistema operativo no tiene capacidad de planificar la ejecución de las tareas en un nodo distinto al que se encuentran inicialmente. Así, la reducción de potencia y área requerida por la red se compensa por una mínima capacidad de balance de la carga de trabajo.

En conclusión, considerando todas estas posibilidades, ventajas e inconvenientes, el diseño de la arquitectura de los MpSoC es uno de los puntos más importantes en el diseño de estos sistemas. Además, hay que considerar que los parámetros de rendimiento de los sistemas, como tiempo, tamaño, consumo o coste, pueden ser tan importantes como los requerimientos funcionales [Sangiovanni01].

1.2 Motivación

1.2.1 Metodologías de diseño para SoC y MpSoC

A la luz de las especiales características de los sistemas SoC y MpSoC, para su diseño óptimo es necesaria una evolución de los métodos de diseño tradicionales de sistemas electrónicos. Un dato que corrobora esta complejidad es que se estima que el coste del desarrollo del software embebido ha pasado a representar la mayor parte del coste de desarrollo de un SoC [ITRS07]. Esto es especialmente importante en sistemas con varios procesadores en el mismo chip, ya que al tener más elementos de cómputo, más parte del sistema se puede implementar en SW.

Por tanto, el correcto diseño de estos sistemas embebidos y de tiempo real (RT/E) requiere metodologías más complejas que las utilizadas para desarrollar los elementos software y hardware por separado. La fuerte interacción entre los elementos software y hardware conlleva la necesidad de desarrollar nuevas metodologías de diseño capaces de manejar ambos flujos de diseño de manera coordinada dado que la estimación y verificación de las prestaciones del sistema embebido completo es un hito de gran importancia.

Las metodologías de diseño de sistemas embebidos deben estar preparadas para garantizar el cumplimiento de los requerimientos funcionales, de tiempo y consumo impuestos. La correcta verificación de este cumplimiento es especialmente importante dados los requisitos de fiabilidad y seguridad impuestos. Además la fuerte interacción HW/SW hace que las verificaciones deben realizarse considerando el sistema completo, no siendo suficiente con el análisis de cada parte por separado.

Como se ha dicho anteriormente, la solución más comúnmente utilizada para co-simulación de sistemas se basa en simular la funcionalidad del sistema en cada ciclo de reloj a través de simuladores de instrucciones (ISS) y descripciones RTL. El código SW es compilado e introducido en el simulador de instrucciones y ejecutado como si estuviera en un procesador real. Sin embargo, estos simuladores resultan excesivamente lentos para modelar sistemas complejos y requieren de simulaciones largas para comprobar la funcionalidad de los diseños.

En sistemas completos como un SoC, estas simulaciones conllevan una cantidad ingente de cómputo. En primer lugar, depender en exclusiva de estas simulaciones significa que cualquier modificación en las decisiones tomadas inicialmente implica unos tiempos prohibitivos. La necesidad de utilizar códigos binarios también limita la flexibilidad de los sistemas de diseño, la capacidad de reutilización de los bancos de pruebas entre las distintas etapas del proceso de diseño y la disponibilidad de todas las herramientas de desarrollo necesarias para la ejecución de código SW desde el primer momento, incluso cuando aún no esté clara la adecuación de la plataforma procesadora a los requerimientos de sistema.

Además la disponibilidad de múltiples tipos de recursos de cómputo en el chip implica que no solo es necesario diseñar cada componente sobre un recurso, sino que al haber varios tipos de recursos disponibles, también hay que decidir el tipo de recurso más adecuado para cada parte del sistema. Decidir si cada parte debe ser implementada en HW o programada como SW y cual debe ser la arquitectura general de sistema a partir de una especificación de alto nivel son tareas cruciales en el proceso de diseño.

En consecuencia en esta tesis se han de considerar dos necesidades a resolver para facilitar el diseño de sistemas MpSoC. En primer lugar, conseguir simulaciones rápidas que permitan la verificación de las cualidades funcionales y de rendimiento del sistema completo. En segundo lugar utilizar estas simulaciones para realizar la exploración del espacio de diseño y encontrar las mejores soluciones de partición y mapeo HW/SW, así como de configuración de los parámetros de la plataforma.

1.2.2 Exploración del espacio de diseño

Para optimizar el diseño del sistema, es necesario evaluar todas las posibilidades del espacio de diseño. Por ello es necesario disponer de técnicas capaces de simular todas las combinaciones posibles, obteniendo la información necesaria para poder seleccionar las mejores.

El crecimiento de las combinaciones es exponencial respecto al número de parámetros a optimizar, con lo que se necesitan técnicas de evaluación realmente eficaces. Para resolver este problema se necesitan dos elementos: herramientas capaces de minimizar el número de puntos a simular y simuladores realmente rápidos.

Las técnicas de Exploración del Espacio de Diseño (DSE) resuelven el problema del número de puntos a simular mediante dos etapas. En la primera se identifican los puntos más importantes del espacio de diseño, y se realizan las simulaciones necesarias para evaluar estos puntos. A partir de estos datos, las herramientas de exploración extrapolan la información para estimar el rendimiento del sistema en el resto de puntos. La primera parte, encargada de seleccionar los puntos se denomina diseño de los experimentos (DoE). La segunda parte, encargada de obtener las funciones matemáticas con las que estimar los puntos no simulados se denominan métodos de obtención de superficies de respuesta (RSM).

Estas técnicas de DSE han sido aplicadas a muchos otros ámbitos de la ingeniería, como automoción, aeroespacial o control industrial, alcanzando una gran madurez. Esta es una de las razones por las que están siendo de gran interés en diseño de sistemas en chip.

Aun cuando el DoE reduce en gran medida el número de puntos a estimar, para realizar una exploración adecuada todavía se necesita obtener resultados de muchos casos, normalmente del orden de miles de puntos. Por esta razón en diseño electrónico es necesario disponer de simulaciones rápidas que permitan evaluar esos puntos en un tiempo razonable. En consecuencia, se necesita desarrollar técnicas que permitan la ejecución de simulaciones rápidas, que permitan probar el mayor número de casos posibles.

En estas simulaciones de deberá modelar los sistemas con todas sus características funcionales y no funcionales para que pueda ser verificado el cumplimiento de los requerimientos durante toda la etapa de diseño. La utilización de técnicas de diseño que cumplan estos parámetros reducirá además el esfuerzo de ingeniería, permitiendo un mejor ajuste en el precio de los sistemas.

1.2.3 Lenguajes de alto nivel para modelado de sistemas

Con objeto de considerar los parámetros funcionales y de rendimiento durante todo el proceso de diseño, se está realizando un gran esfuerzo investigador en desarrollar técnicas que permitan simular completamente la funcionalidad del sistema para, a partir de ellas, obtener estimaciones de tiempo, área o potencia en las primeras etapas del diseño. Estas estimaciones deben ser utilizadas para tomar las decisiones de diseño correspondientes a la partición del

sistema en HW y SW y a su posterior refinado. De la misma forma, el cumplimiento de las especificaciones iniciales debe ser verificado durante el proceso de refinado del sistema.

Para permitir una simulación rápida y conjunta de todo el sistema, tanto antes como después de la partición HW/SW del sistema, se han desarrollado técnicas de generación de especificaciones ejecutables y de co-simulación. Para la generación de especificaciones ejecutables de los sistemas electrónicos y el có-diseño HW/SW han surgido varios lenguajes que tratan de aunar las capacidades de los lenguajes de diseño SW y HW. De los lenguajes presentados en los últimos años dos han sido lo que han obtenido más aceptación.

SystemVerilog es un lenguaje que basándose en Verilog ha introducido una serie de extensiones para facilitar la ejecución de código C con el objeto de permitir la introducción de algoritmos diseñados en C en la simulación de un sistema con sus partes HW claramente definidas. Además, la integración de código C en la simulación del sistema permite la reutilización de los bancos de pruebas utilizados para la verificación de los algoritmos C. La integración de código C es especialmente útil para modelar condiciones de prueba que permitan verificar el sistema. C es un lenguaje menos restrictivo y por tanto mas potente que Verilog, y su utilización para diseñar las partes de la simulación que no van a ser sintetizadas es muy apropiado.

De la misma forma que SystemVerilog representa un intento de un lenguaje de descripción HW de integrar código originalmente SW, también se ha propuesto el caso contrario. SystemC es una librería de C++ que proporciona un núcleo de simulación y unas construcciones similares a las de los lenguajes de descripción HW. Dado que es una librería de C++, SystemC acepta directamente cualquier código desarrollado en este lenguaje, el más habitual en el diseño de SW para sistemas electrónicos. Sin embargo, el lenguaje no es tan eficaz cómo un lenguaje HW original, no hay tantas herramientas para la síntesis de código SystemC, apenas hay diseños que puedan ser reutilizados y los diseñadores HW no están tan familiarizados con él como con VHDL o Verilog.

En consecuencia, ambos lenguajes no son completamente equivalentes. Mientras que SystemVerilog es muy útil para las etapas finales del refinado de sistemas electrónicos complejos, dado que el código Verilog obtenido es optimo y directamente implementable, SystemC es más adecuado para las primeras etapas de refinado. La capacidad de integrar módulos SW, HW y especificación en una misma simulación es muy interesante cuando se esta decidiendo la arquitectura del sistema. El hecho de que en los tres casos se use el mismo lenguaje como base permite además una gran flexibilidad, haciendo que los módulos de diseño puedan ser re-mapeados con relativa facilidad. Por esta razón hemos utilizado SystemC como lenguaje sobre el que desarrollar el trabajo.

No obstante, de simular conjuntamente todos los componentes del sistema, a obtener además todos sus métricas de rendimiento hay un gran trecho que recorrer.

1.3 Objetivos del trabajo

El objetivo de la tesis es aportar soluciones que faciliten la estimación y simulación de código SW en un entorno de co-diseño HW/SW de alto nivel. Estas facilidades deben servir para facilitar las primeras etapas de refinado SW a partir de la especificación. En estas etapas, el código fuente refinado debe ser simulado conjuntamente con el resto del sistema, de tal forma que se puedan verificar tanto sus características funcionales como no funcionales: tiempo, consumo,...

SystemC permite la ejecución directa de Código C++ en la simulación, así como la integración y el modelado de código HW. Sin embargo, SystemC presenta una gran carencia para ser considerado como un lenguaje de co-diseño HW/SW completo. Aunque permite la ejecución funcional de código SW no proporciona soluciones que permitan un completo modelado del mismo. Para verificar el correcto comportamiento de un sistema electrónico complejo no solo es necesario ejecutar la funcionalidad, sino que también hay que analizar otros efectos de su comportamiento. Por ejemplo, en un ABS de un coche, no es suficiente con comprobar que el sistema se activa correctamente cuando se dan las condiciones adecuadas, sino que además es imprescindible garantizar que lo haga en un tiempo mínimo, ya que si tarda demasiado no conseguirá evitar los golpes, haciéndolo inútil.

La primera carencia de SystemC para el modelado SW es que el lenguaje no aporta facilidades para realizar un modelado temporal del código. El código C se ejecuta sin información de tiempo, de forma que no se sabe a priori cuanto va a tardar en ejecutarse una secuencia de código. SystemC tampoco tiene integrada ninguna facilidad para el control de concurrencia. SystemC no modela el efecto de la secuencialidad subyacente generado cuando se intentan ejecutar en concurrentemente varias tareas en un mismo procesador: no tiene ningún método de modelado de planificación, asignación de prioridades o políticas.

De la misma forma, cualquier llamada al sistema en un código SystemC se modela con una llamada al sistema operativo del ordenador donde está siendo ejecutada la simulación (host). Sin embargo, esto no permite modelar los efectos no funcionales del sistema operativo, ya que los recursos requeridos para sus operaciones ni se añaden al modelado en la simulación ni tiene necesariamente una relación directa con el coste del OS final en la plataforma destino. De la misma forma, los efectos de sincronización y gestión de tiempo de las llamadas al sistema operativo, como temporizadores o esperas temporales condicionales (“timeouts”) no pueden ser correctamente modeladas. Estas dependerán del funcionamiento del “host”, por ejemplo, avance del reloj del “host”, y no del tiempo que haya avanzado internamente el núcleo de simulación.

Por último, no se pueden modelar adecuadamente las interacciones HW/SW. Si el SW intenta acceder directamente a una dirección de memoria de un periférico, el OS del “host” dará un error, y si cargamos un controlador de dispositivo, intentará acceder a los periféricos reales del “host” y no a los que hayamos introducido en nuestra simulación.

En conclusión, el objetivo de esta tesis es identificar los problemas existentes y proporcionar soluciones para que el código SW pueda ser modelado adecuadamente con el resto del sistema, y se puedan verificar los requisitos tanto funcionales como no funcionales del mismo.

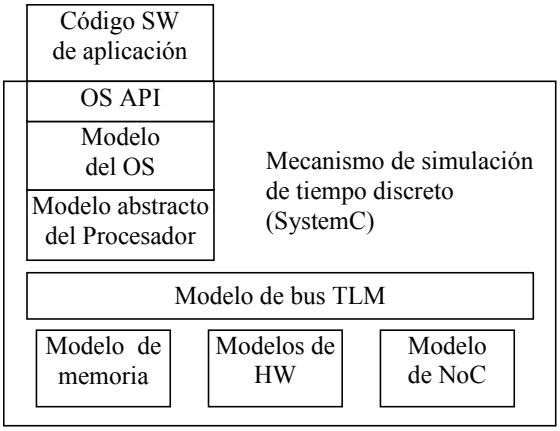


Figura 1-6: Co-simulación nativa

Además, hemos de considerar que existen varias herramientas capaces de realizar las tareas de DSE en ámbitos generales. Rara aplicarlas a la exploración de sistemas MpSoC es necesario integrar los simuladores que permitan evaluar los puntos seleccionados. Por ello, en la presente tesis no se proponen nuevas soluciones de DoE o RSM, pero sí se ha acometido la integración de las soluciones propuestas durante esta tesis para la simulación de sistemas MpSoC en dos herramientas de exploración: modeFRONTIER y m3Explorer.

SECCIÓN II: ESTADO DEL ARTE

"Los procesos de síntesis se basan en conducir por orden los pensamientos, comenzando por los más simples e ir avanzando poco a poco, como por grados, hasta los más compuestos"

René Descartes

2 Flujo de diseño de sistemas embebidos

Hasta la reciente aparición de los lenguajes de descripción a nivel de sistema, los flujos de diseño utilizados para el desarrollo de sistemas electrónicos se han caracterizado por una gran separación entre el diseño SW y HW. Las diferentes características de unos y otros componentes ha fomentado el uso de lenguajes distintos para la programación del SW (C, Ada, …) y la descripción del HW del sistema (VHDL, Verilog). Esto ha llevado a la especialización de los diseñadores en uno de los dos campos, dando lugar a grupos de trabajo independientes. Como resultado, los procesos de integración de SW y HW al final del diseño resultan muy costosos. Incluso, en muchos diseños ni si quiera pueden realizarse en paralelo. En muchas ocasiones es necesario disponer de la plataforma HW para comenzar el diseño SW, produciendo de esta forma un gran aumento del tiempo de diseño del sistema.

Por otra parte, los encargados de realizar la especificación y diseñar los algoritmos iniciales del sistema utilizan también sus propios lenguajes y herramientas en función de sus necesidades (UML, Matlab/Simulink, …). Así pues, cada grupo de trabajo involucrado en el desarrollo de un diseño utiliza su lenguaje, aumentando las distancias entre ellos. La ausencia de etapas intermedias entre la especificación textual, los desarrollos algorítmicos y la implementación del sistema hace que estas etapas tuviesen bastante independencia entre ellas. De esta forma, cada una de las etapas apenas reutiliza elementos de las anteriores, además de requerir un gran esfuerzo de recodificación, al cambiar tanto los niveles de abstracción de las descripciones como los lenguajes utilizados en cada una de las etapas. Por ello se está realizando un gran esfuerzo en el desarrollo de entornos integrados[Popovichi10].

Únicamente el lenguaje C/C++ ha demostrado ciertas capacidades para cubrir los tres ámbitos. Este lenguaje, que es ampliamente utilizado en desarrollo SW, tiene también bastante aceptación para realizar algunas tareas en las primeras etapas del diseño de sistema, como el diseño de los algoritmos y modelos de referencia [GSM, H264], e incluso algunas herramientas lo han propuesto como lenguaje para diseño HW [ImpulseC, CatapultC]. Sin embargo, las limitaciones del lenguaje en la descripción de arquitecturas, modelos de computación o características HW, han hecho que estos intentos de unificación no hayan acabado de fructificar.

La resolución de este de heterogeneidad entre las etapas de diseño ha venido de la mano de la aparición de nuevos lenguajes de descripción de sistema (SystemC, SpecC, SystemVerilog). El objetivo de estos lenguajes es integrar la gran capacidad del lenguaje C junto con otros lenguajes o extensiones orientados al diseño HW o de sistema, de forma que se pueda obtener un único lenguaje utilizable en todas las etapas del proceso de diseño. De esta forma sería posible realizar flujos de diseño integrados.

Para obtener flujos de diseño integrados, junto con la aparición de estos nuevos lenguajes, y apoyándose en sus nuevas capacidades descriptivas, se han propuesto nuevas etapas en los flujos de diseño [Bailey09]. Estas etapas tienen por objeto reducir la distancia entre el desarrollo de las especificaciones y las implementaciones HW y SW. De esta forma se hace más natural la evolución del diseño, reduciendo el esfuerzo necesario para llevar a cabo las tareas de creación del sistema.

Sin embargo, la tarea de generar lenguajes capaces de cubrir completamente el flujo es extremadamente compleja, y aunque estos nuevos lenguajes han demostrado unas capacidades muy superiores a sus antecesores, aún no son aptos para la realización del flujo completo de diseño. En este contexto, esta tesis tiene como objetivo proporcionar soluciones que resuelvan algunas de las limitaciones que aún tienen estos lenguajes, especialmente SystemC. Con objeto de conocer las metodologías de diseño donde deberán aplicarse los resultados de la tesis, comenzaremos describiendo el flujo de diseño y las etapas en las que se engloba el resto de la misma.

2.1 Objetivos del flujo de diseño

Un proceso de diseño de un sistema complejo comienza habitualmente con la creación de modelos algorítmicos, en muchas ocasiones en código C, generados a partir de las especificaciones textuales del diseño. Estos códigos describen las transformaciones necesarias para que a partir de los datos de entrada recibidos por el sistema, éste genere las salidas necesarias. Por ello se trata habitualmente de un código transformativo.

Sin embargo este código transformativo esta conceptualmente lejos de los códigos utilizados para describir la implementación o los programas SW que se ejecutarán en el sistema. En estos últimos códigos, el procesamiento de los datos no se realiza de manera transformativa. Cada componente del sistema genera unas salidas en función de sus entradas, salidas con las que se alimenta a los otros módulos del sistema, y así sucesivamente. En esta estructura, no todo el sistema computa en cada momento los resultados de una única entrada. Los datos intermedios obtenidos son consecuencia de la secuencia de entradas anteriores del sistema, siendo a priori difícil establecer relaciones unívocas de cada salida o dato intermedio con una entrada. Por tanto, los valores intermedios generados por el sistema no se pueden ver como transformaciones directas de las entradas, sino como nuevos datos generados en función del entorno y el estado de la máquina. Por ello, el código transformativo inicial debe refinarse hasta obtener una descripción del sistema compuesta por múltiples códigos generativos.

Para realizar esta transformación es necesario realizar distintas tareas de refinado. En primer lugar, el código transformativo tiende a considerar el sistema como un todo, mientras que la implementación final se basa en la cooperación de múltiples componentes más pequeños, cada uno conteniendo una parte de la funcionalidad del sistema. Por ello, una de las primeras tareas a desarrollar es la subdivisión de la descripción inicial en múltiples tareas.

La separación en tareas concurrentes facilita a continuación la tarea de decidir el mejor emplazamiento para cada una de las partes del sistema: si ha de implementarse en SW o HW,

si es SW en qué procesador, … Además, la subdivisión permite la optimización de componentes más pequeños que el sistema completo, lo que requiere de un menor esfuerzo de diseño.

Una vez decidido el emplazamiento de cada tarea es necesario optimizarla en función de los recursos de los que puede hacer uso: funcionalidades del sistema operativo, co-procesadores, elementos de almacenamiento, etc. Además, hay que optimizar las comunicaciones entre las diversas tareas en función de los recursos de comunicación disponibles: buses, redes de comunicación, fifos, etc. De esta forma, al final se puede obtener un diseño de sistema basado en las descripciones generativas necesarias para su implementación.

En esta transformación, además del cambio de filosofía en la descripción se ha de transformar el código de especificación original a código VHDL o Verilog para los componentes HW. Las tareas a implementar en SW deben ser transformado a códigos que puedan ser aceptados por las herramientas de compilación disponibles de la plataforma destino. Es necesario planificar las tareas SW, poner esperas, canales de comunicación o puntos de sincronización utilizando para ello las llamadas al sistema operativo de la plataforma. Además, hay que generar los elementos necesarios para realizar las comunicaciones HW/SW, como controladores de dispositivo, manejadores de interrupciones, etc.

Aunque la aparición de nuevos lenguajes es un primer paso en la optimización de los procesos de diseño, es necesario definir como deben ser utilizados estos lenguajes. Se necesitan metodologías que permitan aprovechar las posibilidades de estos lenguajes y adaptarlas a los requerimientos de los diseñadores. Para obtener un procedimiento eficiente, el tránsito desde la especificación a la implementación debe hacerse de la forma más progresiva posible. Por ello, hay que definir las etapas intermedias adecuadas para que los cambios graduales hagan el proceso de diseño lo más sencillo posible.

En estas etapas, los requisitos de precisión y velocidad deben ir cambiando paulatinamente. Mientras que las primeras etapas se centran en el diseño de la estructura general del sistema, como la partición en tareas del sistema o la división HW/SW, las últimas etapas tienen como misión optimizar cada uno de los componentes más internos del sistema. Por esta razón, las primeras etapas, al ser más genéricas e influir en el sistema completo, necesitan técnicas que se centren más en la rapidez que en la precisión. Sin embargo, las últimas etapas, de optimizaciones locales deben tener una precisión alta, para lo cual es necesario utilizar técnicas más lentas.

La utilización de los nuevos lenguajes de descripción de sistema (SystemC, SystemVerilog) en estas etapas intermedias reporta múltiples posibilidades a los diseñadores. En primer lugar, la utilización de un lenguaje común permite reutilizar partes del sistema, reduciendo el coste de re-codificación. En segundo lugar, los lenguajes permiten simular tanto algoritmos transformativos secuenciales, como especificaciones ejecutables concurrentes, e incluso código RTL directamente sintetizable. Por esa razón, es posible realizar un refinado

parcial del diseño, simulando bloques a distintos niveles de desarrollo conjuntamente. Con ello se puede verificar los módulos RTL utilizando la especificación de los componentes SW o diseñar los códigos SW en base a un modelo equivalente de la plataforma HW. Además, la simulación de los componentes HW y SW conjuntamente evita la necesidad de disponer de una plataforma HW completamente desarrollada antes de poder comenzar el desarrollo del SW. El SW se diseña considerando los resultados de los accesos a los periféricos conforme estos se van optimizando, con lo que se facilita la labor de integración.

Además, se evita que la utilización de lenguajes distintos para HW y SW haga difícil la validación completa del sistema. La verificación es más sencilla, ya que se pueden reutilizar las técnicas de prueba y los bancos de test en todas las etapas del flujo de diseño. Incluso es posible comparar los elementos refinados con la especificación, utilizando esta última como modelo de referencia.

2.2 Etapas del flujo de diseño

Para realizar diseños mediante un flujo conjunto HS/SW y permitir un diseño más progresivo se han propuesto múltiples soluciones [Ghenassia05][OCP][Cai03]. Sin embargo, la implementación de estos flujos es aún objeto de una gran actividad investigadora. La metodología propuesta en esta tesis se basa en el uso de SystemC como lenguaje de descripción y en las técnicas de modelado a nivel TLM ya que SystemC-TLM se ha convertido en los últimos años en el lenguaje más utilizado para el diseño de embebidos a nivel sistema. Por esta razón para el desarrollo de la tesis se ha seguido la propuesta de etapas recogida en la especificación TLM de la OSCI [TLM2].

El flujo de diseño propuesto en SystemC-TLM se presenta en la figura siguiente:

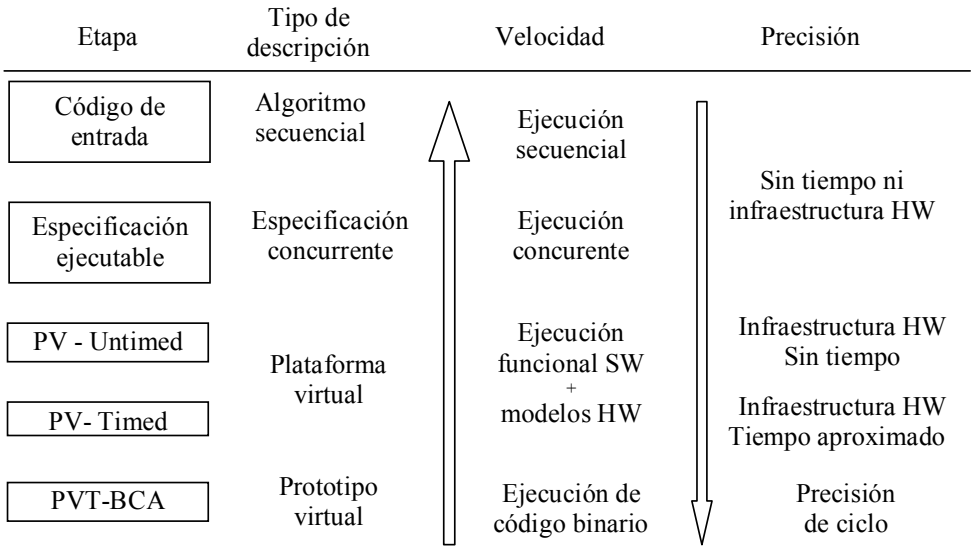


Figura 2-1: Etapas del flujo de diseño

2.2.1 Especificación

El diseño de un sistema empieza normalmente con la creación de una especificación ejecutable. Esta especificación, que en nuestro caso estará escrita sobre la base del lenguaje SystemC, estará compuesta de varios procesos concurrentes, que contendrán la funcionalidad del sistema. Estos procesos estarán comunicados utilizando técnicas de transferencia de alto nivel (como TL3 [OCP] o MLM [Perrier03]). Estos mecanismos de comunicación independizan la especificación de las posibles implementaciones futuras, dando más flexibilidad al proceso de diseño. Esta especificación sirve como descripción funcional donde los requerimientos del sistema pueden ser comprobados. Esta especificación es así mismo útil como modelo de referencia para las etapas futuras del diseño. Cualquier diseño refinado deberá comportarse de la misma forma que el modelo, garantizando de esta forma la equivalencia entre los resultados de las distintas etapas del diseño.

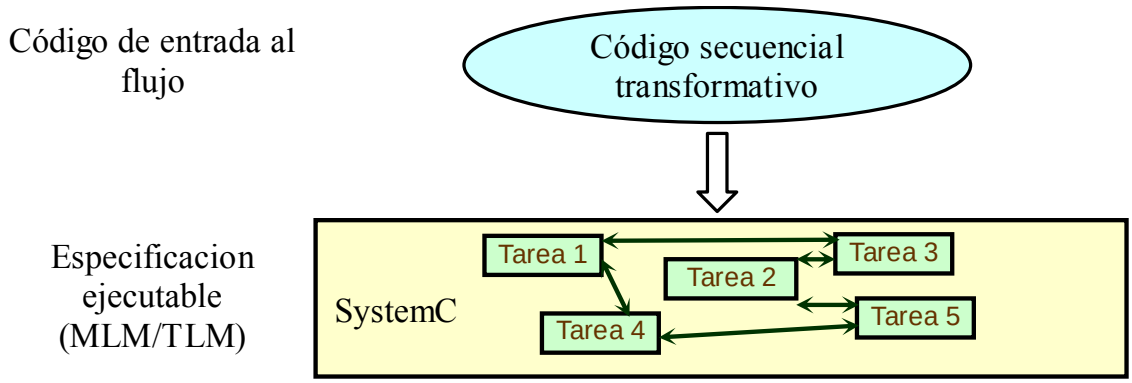


Figura 2-2: Especificación ejecutable

Sin embargo, la utilización de la especificación del sistema no debe limitarse a estas tareas. Esta especificación también puede ser utilizada para realizar las estimaciones de complejidad necesarias para decidir la partición HW/SW del sistema. Tras realizar una exploración de los posibles efectos de las distintas particiones posibles del sistema, se puede tomar adecuadamente la decisión de que procesos del sistema se implementarán en HW y que otros se ejecutarán como SW.

2.2.2 Arquitectura virtual

Una vez obtenida la decisión de la partición del sistema es el momento de comenzar el refinado. Para comenzar este proceso, las comunicaciones de los distintos procesos deben ser reemplazadas, eliminando las interfaces de comunicación abstractas de alto nivel. En su lugar, se han de colocar unas nuevas interfaces de comunicación que modelen adecuadamente los mecanismos de comunicación SW/SW, HW/HW o SW/HW.

Arquitectura virtual (TLM-PVT)

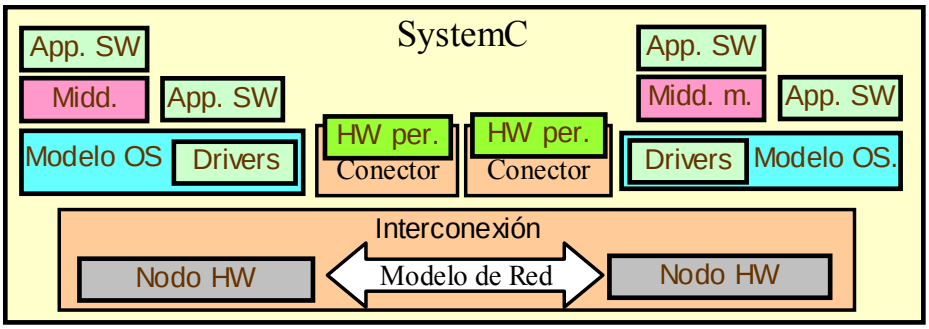


Figura 2-3: Arquitectura virtual

Las comunicaciones SW deben basarse en las APIs proporcionadas por la plataforma real sobre la que dicho código SW será finalmente ejecutado. Estas interfaces dependerán principalmente del sistema operativo utilizado, implementándose como llamadas al sistema, aunque también pueden tener en cuenta detalles de la plataforma HW para acceso a periféricos, o de posibles capas "middleware" intermedias. Las llamadas a las APIs "middleware" o del OS real deben sustituir a las interfaces de alto nivel utilizadas en la especificación. De esta forma se realizarán las primeras etapas de refinado de código de especificación a código HW y SW. Como consecuencia, la infraestructura de modelado del sistema debe proporcionar los modelos de OS y "middleware" necesarios, integrados dentro del entorno de simulación SystemC [Gerstlauer03] [Kriaa].

Estos modelos deben proporcionar las facilidades necesarias para la comunicación y sincronización de los componentes del sistema. Además, otros elementos, como las facilidades para planificación o gestión del tiempo deben ser también proporcionados.

Para comunicaciones HW/SW, el modelado de las interfaces necesaria requiere un proceso aún más complejo, dado que dichas comunicaciones implican parte del sistema operativo, desde la API hasta el HAL, atraviesan los elementos de comunicación de la plataforma HW, y las interfaces de conexión de los periféricos al bus.

En este sentido se han propuesto soluciones que tratan de evitar el modelado de toda la infraestructura, realizando comunicaciones directas entre los componentes utilizando modelos TLM-PV. Sin embargo, estas técnicas suelen emplearse más por la falta de modelos adecuados de los elementos necesarios (modelo de OS, modelo de bus,...) que por necesidades del nivel de abstracción.

En esta tesis, la propuesta presentada se basa en la utilización de modelos TLM-PVT de cada uno de los elementos intermedios que aparecen en la comunicación HW/SW. Dado que la partición, el mapeo y el refinado necesitan información del rendimiento de cada uno de los elementos del sistema, así como del sistema completo, con precisión suficiente, este tipo de modelado temporal de la comunicación es el más adecuado.

2.2.3 Prototipo virtual

Una vez culminada la transformación de la especificación del sistema en componentes HW y SW, es momento de realizar las pruebas definitivas antes de probarlo en la plataforma real. Para ello se requiere un modelo preciso de plataforma, que garantice su equivalencia con la plataforma real. Este modelo se denomina prototipo virtual. Una vez que el diseño ya ha sido convenientemente probado en el prototipo virtual, ya puede ser implementado en la plataforma real.

El prototipo virtual esta compuesto por simuladores de instrucciones (ISS), encargados de modelar los procesadores, y descripciones con precisión de ciclo del resto de la plataforma. Los ISS ejecutan código binario, de la misma forma que los procesadores reales. De esta forma, se puede verificar el diseño utilizando el código SW ya compilado, junto con el resto de infraestructura SW: OS, middleware,... La utilización de modelos de ejecución de instrucciones binarias da lugar a una simulación muy cercana a la realidad, pero con una alta carga computacional, lo que deriva en simulaciones lentas y poca flexibilidad.

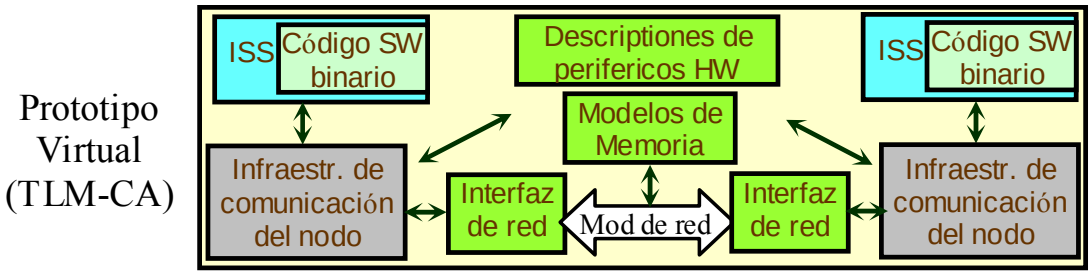


Figura 2-2: Prototipo virtual

El resto del modelo de plataforma, tanto los elementos de comunicación como los componentes HW se basan en modelos con precisión de ciclo o modelos RTL.

2.3 Localización del trabajo realizado en el flujo de diseño

La realización del modelado del sistema con prototipos virtuales es una técnica bastante madura. La ejecución de código binario sobre modelos del procesador (ISS) junto con descripciones de precisión de ciclo de los componentes HW puede realizarse mediante herramientas comerciales [Synopsys][ARMulator]. Por esta razón, durante la tesis el desarrollo de la etapa de prototipo virtual se ha dado por resuelta, para centrarse en las etapas superiores. Es en las etapas superiores donde existe una clara necesidad de nuevas metodologías que permitan el diseño eficiente de sistemas RT/E en plataformas complejas.

De las etapas superiores la tesis se centrará en el modelado de arquitecturas virtuales. Estos modelos son más rápidos dado que trabajan con tiempos aproximados y modelos simplificados. No obstante, todos los elementos que tienen efecto en el comportamiento del sistema han de ser modelados.

Como se ha dicho anteriormente, SystemC se ha erigido como uno de los más importantes [Grötker02] para el desarrollo de especificaciones y de las primeras etapas de co-simulación y co-diseño. En esencia, SystemC es una librería de C++ en la que se define un kernel de simulación y los tipos de datos y macros necesarios para realizar descripciones de componentes hardware y a nivel de especificación. Sin embargo, la integración del flujo de diseño software aun no esta completamente resuelta. Para conseguir que este flujo sea realmente utilizable se necesitan elementos que no están presentes en la versión actual del lenguaje: principalmente el modelado temporal de ejecución SW nativa y el modelado de sistemas operativos.

La tesis también se centra en la generación de una librería capaz de obtener datos sobre las prestaciones temporales de dichos sistemas a partir del código fuente, para poder verificar el cumplimiento de las restricciones temporales requeridas.

Las técnicas de estimación comunes se pueden dividir en dos grupos: estáticas y dinámicas. Las técnicas estáticas se basan en estimaciones de peor caso (WCET) [Malik97] [Hergenham00]. Estas técnicas no son adecuadas para su uso en la simulación SystemC dado que las estimaciones de peor caso son demasiado pesimistas. Dado que en cada momento conocemos exactamente el código que se está ejecutando y los valores de las variables utilizadas no es adecuado el cálculo apriorístico. Este cálculo no tiene en cuenta los datos que sí disponemos durante la simulación y al ignorar información importante no puede obtener un resultado óptimo.

Por esa razón son más adecuadas las técnicas dinámicas [Guiso01] [Brandolese01]. En nuestro caso podemos ir calculando los costes temporales a la vez que se va ejecutando la simulación, con lo que contamos con todos los datos necesarios para un cálculo adecuado del coste temporal del código simulado.

La librería permite la obtención de estimaciones dinámicas de los costes de tiempo de códigos, inicialmente descritos de forma atemporal, y la anotación de dichos tiempos en la propia simulación, de forma que la simulación se convierte en temporal.

Para poder realizar esta librería, se ha necesitado una plataforma sobre la que se pueda tener el mayor control posible, con la finalidad de obtener todos los parámetros necesarios para realizar el modelado de la plataforma procesadora y para poder comprobar y optimizar los resultados de la herramienta. La plataforma utilizada se basa en la arquitectura del procesador OpenRisc 1000 [OR1000], y será denominada OR1500. Además los resultados también han sido analizados sobre una plataforma ARM9.

3 Lenguajes y estándares

La tesis se centra en el modelado de sistemas embebidos HW/SW en alto nivel. Como sabemos, el diseño electrónico, tanto de HW como de SW, lleva realizándose durante unas 5 décadas décadas. Los trabajos que se han ido realizando durante ese tiempo en estas áreas han dado lugar a diversos estándares y lenguajes que ha permitido facilitar las tareas de diseño, reutilización e integración de los componentes en los sistemas. En el desarrollo de la tesis se ha ido utilizando algunos de estos estándares con objeto de crear una infraestructura de modelado lo más realista y útil posible. El uso de estándares se ha realizado con objeto de permitir el diseño y la integración de componentes trasladables al mundo real. En esta sección se presentan los más importantes.

En primer lugar, el diseño de la infraestructura de modelado no se ha realizado de cero. Como punto de partida se ha utilizado el lenguaje de descripción de sistemas más extendido en la actualidad para el diseño electrónico de alto nivel. SystemC es un lenguaje de descripción de sistemas electrónicos, estandarizado por el IEEE y que proporciona una base sólida sobre la que construir una infraestructura de modelado completa. Por ello, comenzaremos presentando este lenguaje.

Además del lenguaje, el segundo punto de partida necesario es definir cómo utilizar dicho lenguaje. Los mecanismos en los que se ha centrado la tesis para el modelado del sistema, especialmente de la plataforma HW, están basados en las metodologías de nivel de transacción (TLM). Más en concreto, el trabajo se ha basado en la infraestructura TLM proporcionada por el grupo de desarrollo de SystemC.

Para permitir el modelado de los componentes SW de los sistemas embebidos se ha desarrollado un modelo de plataforma SW. Esta plataforma tiene por núcleo un modelo de sistema operativo que proporciona a las aplicaciones los servicios requeridos. Para generar un modelo de sistema operativo flexible y compatible se ha implementado una interfaz para programación de aplicaciones (API) basada en el estándar POSIX.

Dado que POSIX define únicamente características de la API, con objeto de realizar modelos realistas de su implementación es necesario basarse además en un modelo de sistema operativo real. Como referencia para modelar el núcleo del sistema y la capa de comunicación con el HW se ha tomado el kernel Linux 2.6.

Por último, para permitir la integración de componentes reubicables, y facilitar la exploración del mapeo óptimo de componentes, se proporciona una infraestructura intermedia (middleware) para computación distribuida basada en un modelo de ORB (CORBA).

Todos estos estándares serán presentados en las siguientes secciones, antes de entrar a presentar como se ha realizado su modelado en alto nivel.

3.1 SystemC

La tendencia actual de las nuevas metodologías de diseño indica una clara inclinación a comenzar el proceso de refinado del sistema a partir una especificación ejecutable. Esta descripción de alto nivel, sirve como modelo sobre el cual comparar las diferentes etapas del proceso. Una de las técnicas que están teniendo más aceptación en el diseño electrónico es escribir esta especificación sobre la base del lenguaje C++ utilizando SystemC. De esta forma, códigos y algoritmos C pueden ser fácilmente integrados en la nueva especificación, que puede ser ejecutada como un programa C normal. A partir de esta especificación se puede desarrollar los diversos componentes del sistema de manera eficiente. Para ello se aplica un proceso de refinado consistente en transformar la descripción inicial paulatinamente hasta obtener unos elementos HW y SW que puedan ser integrados en el diseño final.

Sin embargo, el uso de una nueva metodología, nuevos lenguajes, paradigmas y herramientas, suele implicar un descenso de la productividad debido al proceso de adaptación que deben superar los diseñadores. En este sentido, el uso de una extensión de C++ como lenguaje de descripción, proporciona grandes ventajas respecto a otros lenguajes propuestos para el mismo fin (como lenguajes basados en Verilog).

En primer lugar, una gran parte de los diseñadores, especialmente los encargados de realizar la especificación de los sistemas y algoritmos a utilizar en los diseños, están ampliamente familiarizados con C/C++. Por ello, su aprendizaje requiere un esfuerzo relativamente bajo. Además gran parte de las herramientas necesarias durante el proceso de diseño ya son conocidas por estos diseñadores. Entornos de desarrollo, compiladores o depuradores utilizados para el desarrollo de programas C++ pueden ser reutilizados para el flujo de diseño en SystemC.

De esta forma, el proceso de aprendizaje se reduce al conocimiento de la semántica adicional introducida por SystemC, conservando la sintaxis. Los conceptos de jerarquía, estructura y encapsulación necesarios para la correcta especificación de un sistema se pueden entender más fácilmente al comprobar su relación con los paradigmas que hacen de C++ un lenguaje orientado a objetos. La idea de interfaces o la comparación entre clases y módulos pueden ayudar a un mejor entendimiento de los nuevos elementos introducidos por SystemC.

Además, C es uno de los lenguajes existentes más eficientes y de más potencia descriptiva. Esto redunda en una mayor facilidad para describir los sistemas y los bancos de prueba de modo que los modelos resultantes sean rápidos y manejables.

SystemC es un lenguaje que permite al diseñador trabajar con modelos de las sistemas embebidos HW/SW a distintos niveles de abstracción, desde especificaciones ejecutables hasta modelos con precisión de ciclo o incluso descripciones a nivel de transferencia de registros (RTL). Estos modelos pueden ser usados para evaluar eficientemente el sistema, obteniendo predicciones de su comportamiento funcional. Incluso, utilizando las convenientes extensiones se pueden realizar análisis de rendimiento, tiempo, consumo, con objeto de

obtener la información necesaria para validar y/u optimizar el diseño, explorar distintos algoritmos o realizar adecuadamente el mapeo HW/SW de los componentes del sistema.

SystemC está formado por un conjunto de librerías y macros implementadas en C++ que hacen posible una descripción arquitectural de los sistemas basada en procesos concurrentes con la sintaxis del lenguaje C++ ordinario. El lenguaje proporciona también un kernel de simulación capaz de desplegar simulaciones dirigidas por eventos. Así los objetos descritos pueden modelarse en una simulación concurrente. Dichos objetos se comunican e interaccionan usando señales y canales de comunicación ofrecidos por la librería de SystemC. Así mismo también pueden utilizarse otras definidas por el usuario.

Sin embargo, la mayoría de las extensiones proporcionadas por SystemC se limitan a la generación de especificaciones y descripciones hardware. El modelado completo de los sistemas SW apenas esta esbozado en SystemC. SystemC carece de modelos de procesadores o de sistema operativo. Aunque el código SW, escrito también en C++, se puede ejecutar conjuntamente con el resto del modelo, no es posible simular su comportamiento temporal, emular políticas o prioridades para la planificación de las tareas, integrar temporizadores, interrupciones o controladores de dispositivo, etc. Es por eso que el desarrollo de todas una infraestructura para modelado de SW es necesaria para la utilización de SystemC como entorno de desarrollo de sistemas HW/SW. Como consecuencia, este es un área que ha experimentado un gran desarrollo en los últimos años, que se espera continúe en el futuro.

3.1.1 Elementos de SystemC

Para permitir la creación de modelos de los sistemas a distintos niveles durante el proceso de diseño la librería SystemC proporciona una serie de construcciones que se añaden al lenguaje C++. Estas construcciones, comunes en los lenguajes de descripción HW comunes, como VHDL o Verilog, no están presentes en C++ y han de ser añadidos para su utilización como lenguaje de modelado de sistemas HW/SW. Las construcciones tienen por objeto ayudar al diseñador a describir elementos como la arquitectura del sistema, la jerarquía de módulos, el modelado temporal, la emulación de paralelismo o el comportamiento reactivo.

Los elementos proporcionados por SystemC son principalmente los siguientes:

- Módulos: Elementos de SystemC que permiten la descripción jerárquica del sistema. Proporcionan la idea de contenedor, encerrando la funcionalidad necesaria para realizar una determinada tarea. Como entidad jerárquica puede contener procesos y otros módulos, además de puertos, canales, variables o funciones.

- Canales de comunicación: SystemC proporciona diferentes semánticas de comunicación que pueden ser utilizadas para comunicar tanto los componentes internos de un módulo entre si como los módulos con otros elementos del exterior. Los diferentes canales proporcionan semánticas útiles en los diferentes niveles de abstracción del proceso de diseño. Desde FIFOs hasta señales. Además se permite la

generación de canales definidos por el usuario, con su propia semántica e interfaces. Estos canales pueden describir desde sistemas de comunicación punto a punto como protocolos de canales complejos, como buses, redes, etc.

- Puertos: Permiten la comunicación de los módulos con el exterior. Se utiliza para permitir el acceso desde los elementos internos a los módulos a los canales externos, que no son visibles desde ese ámbito. SystemC proporciona puertos unidireccionales y bidireccionales, soportando una o varias interfaces de comunicación.

- Interfaces: Las interfaces definen las funciones que proporciona o requiere un determinado elemento de comunicación. Canales y puertos se desarrollan en base a interfaces, de tal forma que solo pueden ser conectados canales y puertos con interfaces compatibles.

- Tipos de datos: mientras que en los lenguajes SW los tipos de datos se restringen a enteros o flotantes de 1, 8, 16, 32 o 64 bits, los lenguajes HW necesitan de una mayor capacidad descriptiva. Esto se debe a que muchas operaciones, en un procesador tiene el mismo coste de las arquitecturas HW independientemente del tamaño de los datos. Por ejemplo, en SW cuesta lo mismo sumar dos palabras de 17 o de 32 bits. Sin embargo en descripciones HW, el tamaño de los datos es crítico. En el ejemplo anterior, el segundo caso requeriría el doble de tiempo y tendría un aumento exponencial de área.

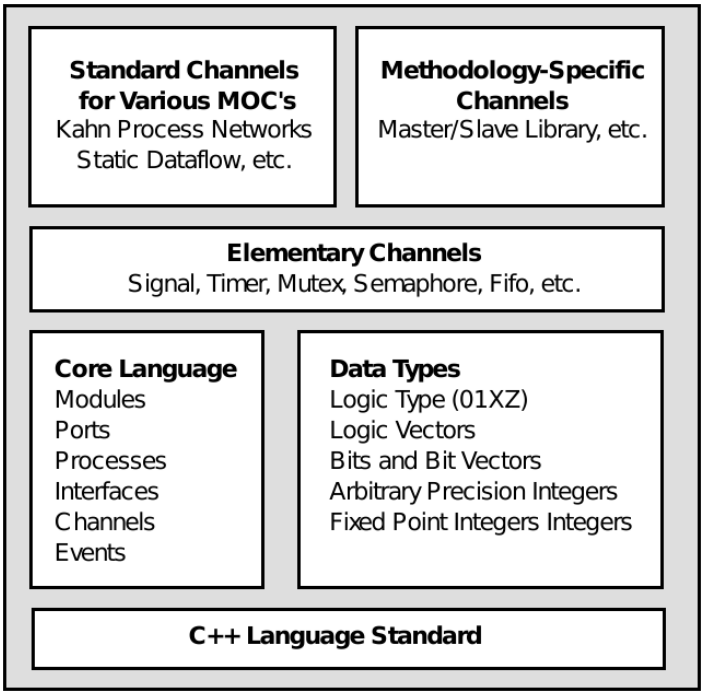


Figura 3-2: Estructura del lenguaje SystemC

- Procesos: Los procesos se usan para ejecutar la funcionalidad y modelar la concurrencia entre las distintas tareas del sistema. SystemC proporciona tres tipos de procesos: métodos, hilos, e hilos con reloj (SC_METHOD, SC_THREAD y SC_CTHREAD).
  Los métodos se comportan como los procesos de VHDL o los "always" de Verilog. Comienzan y acaban de una pasada, de forma que cada vez que deben ser ejecutados se realiza una nueva llamada. Cada llamada se produce cuando se produce un evento en la lista de señales asignada a tal efecto para cada módulo.
  Los hilos por contra se arrancan una sola vez y pueden ser suspendidos y reactivados cuando se considere necesario.

- Eventos (sc_event): Se utilizan para controlar la ejecución de los hilos. El evento se envía a un hilo mediante los métodos "evento.*notify()*". El hilo a su vez contiene funciones "*wait*()" que suspenden el proceso por un determinado tiempo o hasta la recepción de un evento. Los eventos se usan además para implementar las listas de sensibilidad de los métodos y los canales con bloqueo.

Aparte de las construcciones citadas anteriormente, el otro elemento principal que proporciona SystemC es un núcleo de simulación. Este núcleo trata de alterar los mecanismos de ejecución propios de C++ para acercarlos más a las técnicas de simulación utilizadas en lenguajes HW (VHDL). Para ello realiza una simulación dirigida por eventos y basada en un doble eje temporal. El primer eje se utiliza para describir el avance del tiempo. La simulación solo avanza en este eje hasta que todos los procesos están bloqueados por otros procesos o se encuentran en una espera de tiempo. De esta forma se pueden realizar estimación de tiempo y rendimiento del sistema y verificar el cumplimiento de las condiciones impuestas al diseño.

El segundo eje se utiliza para permitir la simulación de múltiples estados de cada tarea en el mismo punto del eje temporal. Esto se puede usar tanto para modelar operaciones que se deben realizar en tiempo cero, como por ejemplo las distintas iteraciones necesarias para que las descripciones HW a nivel RTL lleguen a un punto de estabilidad cuando llega un nuevo ciclo de reloj.

El mecanismo de simulación se basa en le ejecución repetida de dos etapas. Durante la etapa de evaluación se ejecuta el código de los procesos que no están bloqueados, leyendo los valores actuales de los canales y generando los valores nuevos. Durante la etapa de actualización se cargan los valores actuales de los canales con los nuevos valores, se aumenta el valor del eje delta o del eje temporal y se decide qué procesos se ejecutarán en el ciclo de evaluación siguiente.

Por último SystemC proporciona ciertos servicios para facilitar el depurado del diseño. Los elementos SystemC tienen chequeos en tiempo de ejecución. SystemC permite la obtención de formas de onda de las señales deseadas en varios formatos. Además existen puertos especiales que se pueden utilizar para acceder a los datos internos de los módulos.

Todos estos elementos son muy útiles para diseñar sistemas HW/SW complejos. Además estas funcionalidades alivian parcialmente las dificultades de análisis de sistemas con un modelado de paralelismo tan alto.

3.1.2 Limitaciones del lenguaje

Aunque SystemC es un lenguaje muy potente para el desarrollo de sistemas embebidos, presenta ciertas limitaciones.

En primer lugar se necesita un mecanismo que permita transformar automáticamente la especificación del componente, basada en elementos SystemC a un código software con elementos del sistema operativo elegido para el diseño [Herrera03]. Esta parte no ha sido desarrollada en profundidad durante la tesis. No obstante, no ha sido perdida de vista, ya que ha sido desarrollada por otros integrantes del grupo de trabajo de la Universidad de Cantabria. Diversas colaboraciones y publicaciones conjuntas han surgido a tal efecto.

La otra tarea a realizar es permitir la simulación de ese código SW, con primitivas del sistema operativo, en SystemC. Esto implica no solo una ejecución funcional, sino un modelado completo de su comportamiento. Para ello es necesario extender el lenguaje SystemC con el objeto de incluir primitivas de un sistema operativo de tiempo real (RTOS) que permitan la ejecución y el refinado de módulos software. Para ello se estudiará la especificación POSIX [POSIX] y se desarrollarán los elementos que permitan la ejecución de módulos refinados para dicho sistema de manera correcta.

Una vez resuelto este problema, es necesario transformar la ejecución funcional del código SW, que no incluye información de tiempo, en una simulación temporal. Esto implica que no es posible realizar estimaciones de rendimiento del sistema. Además, las interacciones del SW con los componentes hardware, cuya simulación tiene una fuerte dependencia temporal, no se realiza de manera correcta. Por último, la planificación de tareas que se ejecutan en tiempo “0” deja de tener sentido. Si el SW se no necesita tiempo para ejecutarse, el procesador está siempre libre, con lo que no es necesario un sistema de planificación.

Para resolver estas limitaciones, en la tesis se ha trabajado es el desarrollo de un método capaz de modelar sistemas operativos y de proveer características temporales a los componentes SW para poder realizar co-simulaciones precisas de las que poder obtener información del rendimiento del sistema. Esta información posibilitará el refinado y optimización del sistema en las primeras etapas del flujo de diseño.

3.2 TLM

Conforme más grandes son los sistemas más difícil es modelarlos y analizarlos en su conjunto. Para solventar ese problema la solución más común es el aumento de nivel de abstracción, que propone una perdida de parte de la información de detalle en el grado de descripción de los componentes del sistema, simplificando los modelos. A cambio es posible

obtener herramientas más rápidas y flexibles con las que poder realizar las exploraciones necesarias durante las primeras etapas de diseño.

Para aligerar estos análisis de alto nivel se prescinde de información en los procesos de comunicación, se utilizan algoritmos sencillos en lugar del código de implementación, se reduce la precisión del modelado no funcional del sistema, etc. Esto permite que los modelos de alto nivel mantengan su funcionalidad, pero con un esfuerzo de computo menor.

Esta es la razón por la que el modelado a nivel de transacción (TLM) está teniendo un gran impacto en la actualidad. Esta técnica propone simplificar el modelado de las comunicaciones realizado en niveles inferiores, como RTL (nivel de transferencia de registros). En RTL cada línea necesaria en un proceso de comunicación se modela separadamente, considerando en qué momento se producen todos los eventos del proceso. Por ejemplo, en un bus, una transferencia de una palabra requiere de la subida de una señal de activación del bus, colocar la dirección, colocar el dato, indicar si es lectura o escritura y esperar la confirmación del periférico. Todo este proceso depende del mantenimiento de una determinada ordenación y el cumplimiento de unos márgenes temporales bastante estrictos.

Sin embargo, este nivel de detalle es muy costoso computacionalmente. En determinados casos puede incluso que la mayoría de esta información sea irrelevante, y que solo produzca una ralentización del modelado del sistema. Por ejemplo, si se quiere comprobar que juntando una serie de componentes el diseño cumple la tarea requerida para una serie de entradas, puede que esto no sea necesario. En este caso se necesitará saber si los algoritmos de estos componentes resuelven el problema, pero el nivel detalle en el modelado de sus comunicaciones es innecesario. En la verificación funcional se puede suponer que las comunicaciones son correctas y que el protocolo de bus será respetado. De esta forma solo es importante que el dato pase de un componente a otro, y no los valores de las señales o los tiempos entre ellas.

Por esta razón TLM propone que cada comunicación se realice como una única operación, en lugar de requerir un elemento por cada señal utilizada. Cada vez que se quiere transferir un dato, se realiza una llamada a una función “get” o “put” del canal de comunicación correspondiente, enviando o recibiendo el dato en cuestión. Así, con una sola llamada a función se modela todo el protocolo de comunicaciones de una transferencia de datos.

Esta solución, que a priori puede parecer extraordinariamente simple, tiene sin embargo gran versatilidad. En función de los detalles que se expliciten o se omitan en el modelado de dichas transferencias, se pueden realizar descripciones a distintos niveles de abstracción. Cambiando el detalle en el modelado de tiempos o la cantidad de información de cada transferencia se puede variar la precisión y la velocidad de la simulación del modelo. De esta forma se han definido varios niveles dentro del paradigma TLM.

Actualmente, TLM esta siendo usado en la industria para resolver una gran variedad de problemas prácticos que se presentan durante el proceso de diseño de sistemas complejos.

Estos problemas cubren desde el desarrollo de especificaciones ejecutables en alto nivel de los sistemas hasta el proceso de refinado de los componentes HW y SW. En alto nivel, TLM es útil para realizar análisis arquitecturales rápidos y eficientes, así como para generar especificaciones ejecutables ligeras. Para el desarrollo de las aplicaciones SW, TLM es apropiado ya que proporciona la capacidad de realizar modelos de plataforma eficientes, donde el código puede ser probado y analizado para comprobar la corrección tanto funcional como del resto de parámetros no funcionales: tiempo, consumo, memoria, … Finalmente, en el diseño HW se aplica TLM para realizar la verificación funcional, realizando un eficiente modelado e integración de los entornos. Además se facilita el proceso de integración HW/SW.

De esta forma, cada uso de TLM requiere diferentes niveles de precisión temporal o modelado del uso exclusivo de los recursos. Por ejemplo, en el desarrollo SW se puede suponer que la plataforma se hará cargo de que dos procesadores no puedan acceder simultáneamente al mismo bus.

Para soportar y facilitar el manejo de estas técnicas en SystemC, sus responsables han generado una especificación estándar de funciones para la descripción de modelos TLM. Esta especificación está compuesta por declaraciones de funciones y estructuras adecuadas al modelado de transferencias. El objetivo es estandarizar el uso de TLM de tal forma que diseñadores diferentes puedan utilizar conjuntamente sus componentes sin un gran esfuerzo de adaptación de comunicaciones. En paralelo otros grupos han desarrollado otras propuestas basadas en TLM para sus entornos de aplicación.

En esta tesis se ha desarrollado un trabajo de modelado de plataforma basado en TLM, utilizando el estándar TLM2 de la OSCI.

3.2.1 Estándar OSCI TLM2.0

Un problema importante que se presenta en el diseño de sistemas complejos, es poder integrar elementos preexistentes distintos en el flujo de diseño. Sin embargo, esto es normalmente difícil ya que la interoperatividad entre los distintos modelos esta muy limitada.

Si bien es cierto, que a nivel de señales, los principales estándares de buses definen como deben ser las interfaces de los componentes para ser conectados entre sí, esto no se ha extendido a otras áreas de diseño. Es necesario definir que nombres de funciones, que tipos de comunicación y que estructuras de datos deben ser transferidas para que descripciones a otros niveles, o incluso componentes a bajo nivel que no se conecten a través de un bus estándar puedan interactuar directamente en un modelo.

El principal objetivo del estándar TLM 2.0 es resolver estos problemas definiendo una interfaz sólida y proponiendo las estructuras de datos necesarias para realizar modelados de comunicación rápidos y eficientes. El objetivo es que dos modelos escritos por personas diferentes, sin ningún conocimiento respecto al otro, puedan interactuar sin necesidad de realizar cambios en las descripciones.

La interfaz se basa en una serie de elementos escritos en forma de plantillas que proporcionan un entorno de trabajo genérico que permite la adaptación de la interfaz a los tipos de datos más adecuados en cada entorno de trabajo que deba ser modelado. Los objetivos de TLM2.0 son los siguientes:

- Proporcionar estructuras y tipos de datos genéricos para el modelado de transacciones
- Recomendar tipos de datos para ser usadas en las plantillas de la interfaz
- Proponer estructuras de datos que ayuden a modelar los protocolos de las distintas infraestructuras de comunicación.

TLM esta recomendado para ser usado en las siguientes partes de proceso de diseño:

- SW
  - Desarrollo de la aplicación
  - Análisis de rendimiento
  - Análisis arquitectural
- HW
  - Análisis de rendimiento
  - Análisis arquitectural
  - Refinado e Implementación
  - Verificación funcional
- Integración HW/SW

TLM 2.0 defines dos estilos de descripción para modelos a nivel de transacción, dependiendo de la dependencia entre la información temporal y los datos transferidos.

3.2.2 Estilos de descripción TLM propuestos por la OSCI

En un modelo descrito en TLM, al contener varios procesos se requieren mecanismos por lo cuales los procesos entren y salgan de ejecución, permitiendo la evolución de todos ellos. Para ello cuando un proceso está en ejecución se necesita definir los mecanismos por los cuales devolverá el control al núcleo de simulación para que otro proceso pueda ser ejecutado.

En SystemC, el mecanismo de simulación no realiza expulsiones forzadas de las tareas. Esto significa que deben ser las propias tareas las que devuelvan el control al núcleo, utilizando una función o recurso que pueda cumplir este propósito. Para ello existen dos posibilidades, o se introducen puntos de sincronización explícitos en el código de descripción, o se utiliza un estilo descriptivo que fuerze a los procesos a devolver el control en ciertos puntos.

El uso de sincronización completa es habitual en descripciones realizadas siguiendo modelos de computación como CSP o KPN. En ellos todas las comunicaciones se realizan mediante elementos de sincronización utilizando “fifos”, semáforos u otras primitivas similares. Esto permite la descripción de modelos completamente atemporales, donde la

simulación no necesita de ningún avance de tiempo. Esto suele ser útil para las descripciones de alto nivel de los sistemas.

Sin embargo, a niveles inferiores esto no suele ser posible. Cuando se quiere modelar sistemas con múltiples tareas SW, estas no están siempre enlazadas por sincronizaciones. En ocasiones el avance del tiempo es necesario para modelar el comportamiento del sistema, en el que existen temporizadores, relojes, accesos por encuesta, etc. Por tanto estos modelos requieren simulaciones temporales, donde el avance del tiempo juega un papel muy importante.

En el estándar TLM2.0 se consideran tres diferentes estilos descriptivos: atemporal (Untimed), de tiempo relajado (Loosely-timed) y de tiempo aproximado (Appoximately-Timed). No obstante la técnica de modelado de transacciones puede ser adaptada a otros estilos modelando en otros niveles de abstracción. Cada uno de los estilos descriptivos tiene un objetivo principal y se define por usar unas determinadas funciones de la API TLM con un determinado significado.

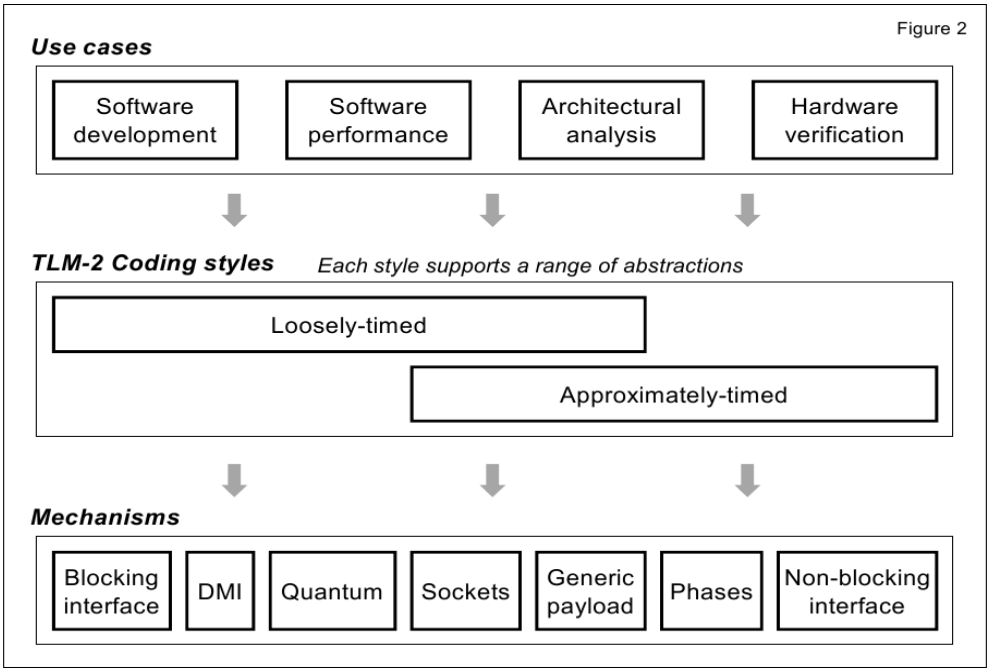


Figura 3-3: Aplicación y mecanismos de comunicación de los estilos TLM

La metodología recomendada para el desarrollo de sistemas utilizando el nivel TLM es comenzar realizando las descripciones de alto nivel mediante descripciones atemporales. Desde estas descripciones, cada componente debe ser traducido a una descripción temporal relajada. Una vez que la estructura del sistema (comunicaciones, mapeo HW/SW, arquitectura) ha sido fijada se ha de pasar a descripciones de tiempo aproximado. Con estas últimas descripciones se puede comprobar con más precisión el cumplimiento de todas las restricciones no funcionales de la especificación.

3.3 POSIX: IEEE 1003.1 versión 3

POSIX es el acrónimo de Portable Operating System Interface. La X tiene como objeto recordar al sistema UNIX, del cuya filosofía nació el estándar. El término POSIX fue sugerido por Richard Stallman en respuesta a la demanda del IEEE, que buscaba un nombre fácil de recordar.

El estándar POSIX esta compuesto por una familia de sub-estándares compuesto de llamadas y facilidades del sistema operativo definidos por el IEEE y especificados formalmente en el IEEE 1003. Persiguen generalizar las interfaces de los sistemas operativos para que una misma aplicación pueda ejecutarse en distintas plataformas.

Los servicios a nivel de programa requeridos incluyen definición de estándares básicos de entrada/salida, planificación, comunicación y sincronización, terminal, y servicios de red, entre otros. POSIX también especifican una API para las librerías de gestión de hilos, que es muy popular y muy utilizada en muchos sistemas operativos.

Los más subconjuntos más comunes son:

- POSIX.1 Interfaz de sistema para programas de aplicación (API) en lenguaje C
- POSIX.1b Rectificación 1 del API: Extensión de tiempo real en C
- POSIX.1c Rectificación 2 del API: Extensión de hilos de control
- POSIX.2 Intérprete de comandos y útiles
- POSIX.4 Ahora llamado POSIX.1c
- POSIX.5 POSIX.1 en lenguaje ADA
- POSIX.6 Seguridad
- POSIX.7 Administración del sistema

En esta tesis, para realizar el modelado de un sistema operativo estándar se ha optado por centrar el trabajo en una especificación de sistema operativo los más extensa, general y comúnmente aceptada posible. Por esa razón se ha decidido analizar y utilizar la última versión del estándar POSIX. No obstante, dado que este estándar contiene innumerables opciones y extensiones es necesario definir previamente que partes serán objeto del modelado y cuales no.

El análisis del estándar POSIX se puede dividir en las siguientes secciones:

- Concurrencia
- Planificación
- Sincronización y comunicación
- Librería C
- Gestión de tiempo
- Gestión de memoria
- Ficheros y entrada/salida

En cuanto a concurrencia se decidió modelar la concurrencia tanto a nivel de proceso como de hilo, permitiendo la creación de procesos e hilos dinámicos.

La planificación se decidió implementar de la forma más extensa posible, permitiendo los dos niveles de planificación, tanto entre procesos como entre los hilos de un mismo proceso. Además se deben modelar las políticas de planificación “Round-Robin”, “Fifo” y la política por defecto que no es de tiempo real. La política de “Servidor Esporádico”, indicada como opcional en el estándar no se ha considerado importante en el modelo.

Para comunicación y sincronización se han considerado tanto los canales como las señales. En cuanto a los canales se deben implementar “mutexes”, semáforos, variables condicionales, “rwlocks” y colas de mensajes, con tantas opciones como sea posible (e.g. herencia de prioridades o los avisos de temporizadores por señal). La implementación de la memoria compartida se estudiará conforme a la evolución de la librería. En cuanto a las señales se han implementado todas sus opciones, incluyendo las señales de tiempo real.

En cuanto a la librería de funciones C (e.g. las funciones matemáticas o de manejo de cadenas de caracteres), estas deben ser aportadas para facilitar la programación y poder ejecutar código real. No obstante, estas funciones ya están presentes en SystemC, como parte de los recursos para la compilación de C/C++, con lo cual no es necesario que sean implementadas de nuevo.

Para la gestión de tiempo han sido modelados los relojes de tiempo real, monotónico y todos los relojes por proceso y por hilo, así como temporizadores y alarmas con tantas opciones como sea posible. Por último han sido implementadas las funciones para dormir el flujo de ejecución como “sleep”, “nanosleep” o “usleep”.

Finalmente no se realiza un control de ficheros específico, al estar por debajo el control del host. Lo que si es necesario es modelar funciones de entrada/salida para permitir el modelado de comunicaciones entre procesadores distintos o incluso entre procesos distintos en la simulación SystemC. Para ello se proporcionan varias funciones del estándar POSIX, como las funciones “create”, “open”, “close”, “read” y “write”, tanto en sus opciones de funcionamiento bloqueante como no bloqueante, por lo que un cierto control de los descriptores de ficheros si será necesario en la librería de modelado.

Algunos sistemas operativos que tienen el certificado de conformidad POSIX son:

- A/UX
- AIX
- BSD/OS
- HP-UX
- INTEGRITY
- Irix
- LynxOS
- Mac OS X

- MINIX
- OpenVMS
- QNX
- RTEMS (POSIX 1003.1-2003 Profile 52)
- Solaris
- OpenSolaris
- UnixWare
- VxWorks

Además, hay muchos otros sistemas que aunque no están certificados cumplen en su mayor parte con el estándar, como:

- Nucleus RTOS
- FreeBSD
- Linux (most distributions — see LSB)
- NetBSD
- BeOS
- OpenBSD
- SkyOS
- Syllable
- VSTa

Finalmente, hay otros sistemas que, aunque no están diseñados como POSIX, contiene extensiones que proporcionan cierto grado de compatibilidad:

- eCos
- Plan 9 de Bell Labs APE
- Symbian OS con PIPS (PIPS es POSIX en Symbian)
- Windows con kernel NT (usados en Windows NT, 2000, 2003; XP, Vista): sólo en algunas ediciones o con determinadas aplicaciones instaladas.

En conclusión, la elección del estándar POSIX como API para generar el modelo de sistema operativo asegura una gran generalidad. Muchos sistemas y códigos SW están basados en esta API. Sin embargo, los detalles de implementación del núcleo del sistema operativo y sus mecanismos de comunicación con el HW no vienen especificados en el estándar. Para mantener la reusabilidad del modelo, su implementación se ha basado en el núcleo del sistema operativo Linux, uno de los de mayor uso en la actualidad.

3.4 **Linux**

Linux es un sistema operativo que fue creado al fusionar las utilidades y librerías del proyecto GNU con el núcleo de sistema operativo desarrollado inicialmente por Linus Torvalds. Es un sistema operativo, compatible Unix, que implementa gran parte de la

funcionalidad descrita en el estándar POSIX. Dos características muy peculiares lo diferencian del resto de sistemas que podemos encontrar en el mercado. La primera, es que es libre, esto significa que no tenemos que pagar ningún tipo de licencia a ninguna casa desarrolladora de software por el uso del mismo. La segunda, es que el sistema viene acompañado del código fuente.

El sistema operativo lo forman el núcleo (kernel) más un gran número de programas y bibliotecas que hacen posible su utilización. Linux se distribuye bajo la licencia pública GNU (GPL) por lo tanto, el código fuente tiene que estar siempre accesible y cualquier modificación ó trabajo derivado tiene que tener esta licencia.

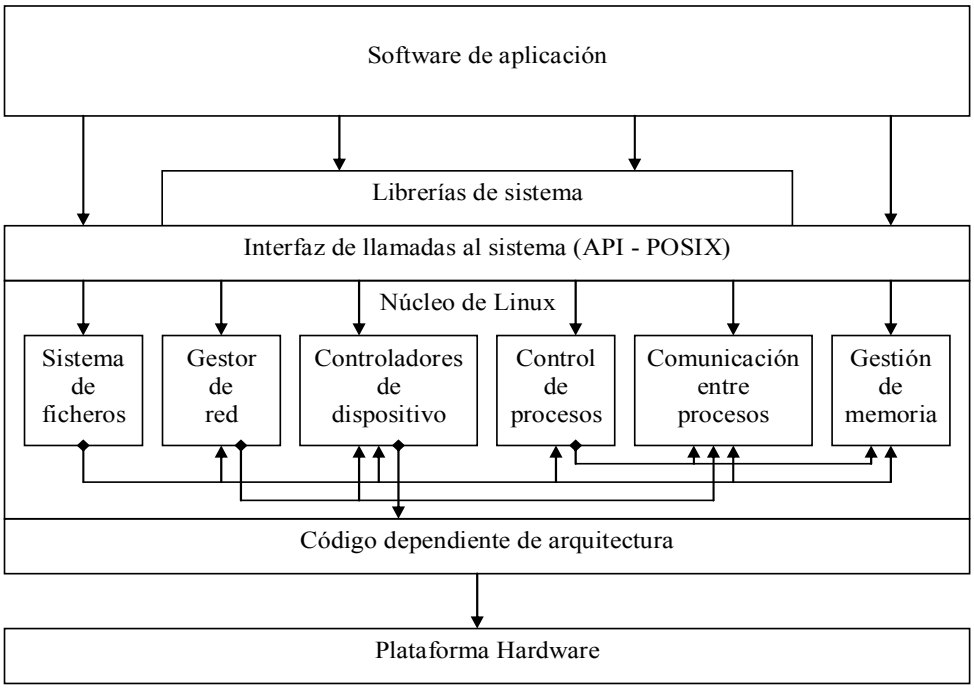


Figura 3-4: Arquitectura de un sistema Linux

El sistema ha sido diseñado y programado por multitud de programadores de todo el mundo. El núcleo del sistema sigue en continuo desarrollo bajo la coordinación de Linus Torvalds, la persona de la que partió la idea de este proyecto, a principios de la década de los noventa. Hoy en día, grandes compañías, como IBM, SUN, HP, Novell y RedHat, entre otras muchas, aportan a Linux grandes ayudas tanto económicas como de código.

Día a día, más y más programas y aplicaciones están disponibles para este sistema, y la calidad de los mismos aumenta de versión a versión. La gran mayoría de los mismos vienen acompañados del código fuente y se distribuyen generalmente también bajo los términos de la licencia GPL.

Las principales características de Linux son:

- Es un sistema multitarea, multiusuario, multiplataforma y soporta opciones multiprocesador (SMP)
- Carga de ejecutables por demanda y protección de la memoria entre procesos
- Librerías compartidas de carga dinámica y librerías estáticas.
- Compatible con POSIX, System V y BSD a nivel fuente (control de tareas POSIX)
- TCP/IP, incluyendo "ssh", "ftp", "telnet", "NFS", etc. y diversos protocolos de red incluidos en el kernel: TCP, IPv4, IPv6, AX.25, X.25, IPX, DDP, Netrom, etc.

En el ámbito de la tesis se ha utilizado la arquitectura Linux, junto con la especificación POSIX para implementar de manera realista los mecanismos de gestión de procesos e hilos, planificación, comunicación y sincronización de tareas. En casos de conflicto se ha seguido el estándar POSIX.

Además, se la arquitectura Linux se ha utilizado para definir uno de los elementos más importantes del modelado de sistemas HW/SW, que no esta definido en el estándar POSIX: la infraestructura de soporte de comunicaciones entre HW y SW. Con el objetivo de permitir la utilización de interfaces de comunicación con el HW reales se ha integrado un soporte que permite la integración de controladores de dispositivo, manejo de interrupciones y acceso a periféricos basada en las funciones y estructuras de datos incluidas en Linux.

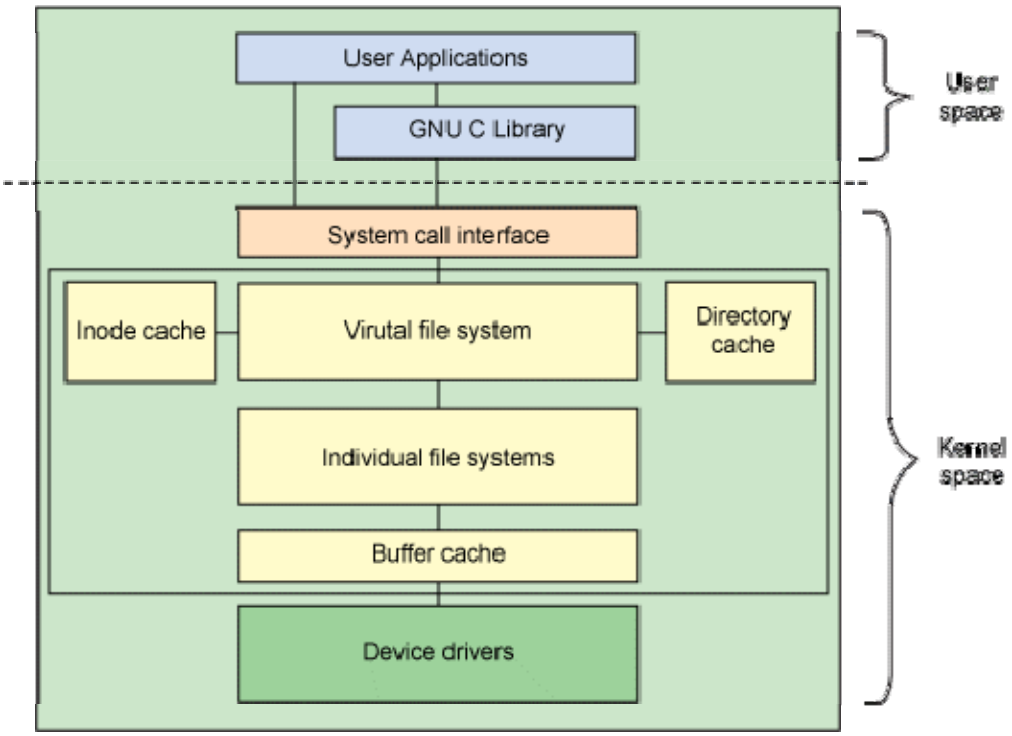


Tabla 3-5: Mecanismo de comunicación entre el SW y la plataforma HW

Linux proporciona tres tipos de soportes, en función del tipo de dispositivo HW con el que se va a establecer la comunicación: soporte para dispositivos de carácter, de bloque y de red. En la implementación actual se ha incluido el soporte para dispositivos de carácter.

Adicionalmente se ha implementado funcionalidad mínima para permitir el desarrollo de un controlador de dispositivo de red. La implementación de este controlador ha sido necesaria con objeto de proporcionar los servicios necesarios para la integración de una pila TCP/IP.

3.4.1 Extensiones de tiempo real para Linux

Un sistema embebido de tiempo real es un sistema que interacciona repetidamente con su entorno físico y debe recibir/generar estímulos dentro de un plazo de tiempo determinado. Esto es, para que el funcionamiento del sistema sea correcto no basta con que las acciones sean correctas, sino que tienen que ejecutarse dentro del intervalo de tiempo especificado. Cuando se requiere que normalmente se cumplan dichos tiempo, pero se permiten violaciones puntuales se dice que es un sistema de tiempo real relajado. Cuando no se permite ninguna violación de estos tiempos, el sistema es de tiempo real estricto.

Un sistema operativo Linux, y en general la mayoría de los sistemas POSIX, proporcionan al usuario funciones de control de tiempo: esperas, temporizadores, relojes, alarmas, etc. Esto hace que dichos sistemas tengan cualidades de tiempo real. Sin embargo, estos sistemas operativos no garantizan que en todo momento las operaciones de tiempo son exactas, sin ningún tipo de desviación temporal. Acciones como interrupciones, determinadas secciones críticas u operaciones internas del sistema operativo pueden provocar inexactitudes en las funciones de tiempo, añadiendo retrasos imprevistos [Shiva06]. Al no poder garantizar el cumplimiento estricto de los tiempos en el 100% de los casos, estos sistemas operativos no pueden ser utilizados en diseños de tiempo real estricto.

Los sistemas operativos de tiempo real estricto, suelen ser sistemas muy reducidos, con objeto de que los retrasos y sobrecarga producidos por su ejecución sean mínimos. Sin embargo, como consecuencia, los servicios proporcionados por estos sistemas operativos son muy inferiores a los sistemas operativos convencionales, como Linux.

Con el incremento de la potencia de cómputo de los sistemas embebidos, la funcionalidad SW ha sido incrementada paulatinamente. Este incremento de funcionalidad suele ir asociada a la necesidad de mayor soporte por parte del sistema operativo. En consecuencia, es habitual encontrarse con sistemas que integran tareas o secciones ligeras con requerimientos de tiempo real estricto, ejecutando junto con otras tareas más complejas que permiten mayores márgenes temporales. Para dar servicio a ambos tipos de tareas se necesita un sistema operativo complejo con cualidades de tiempo real estricto. Por ello se ha realizado en los últimos años un gran trabajo para extender los sistemas Linux con capacidades de tiempo real estricto [Hu06].

Las primeras propuestas de extensión de Linux se realizaron a través de parches para el sistema operativo. En [Molnar] se propuso un parche que añade tres elementos para gestión de tiempo real estricto: hilos específicos para la gestión de IRQs, “mutexes” de tiempo real, y temporizadores de alta resolución [Dietrich05].

En [Leyva06] se propone el manejo de interrupciones para tiempo real mediante tareas específicas. Esta gestión permite enmascarar selectivamente las interrupciones deseadas, mediante la asignación de prioridades. Esto permite limitar las latencias debidas a expulsiones por interrupciones.

UTIME [Niehaus97] fue el primer proyecto que introdujo gestión de tiempo con precisión inferior al quantum temporal mínimo de los sistemas convencionales, del orden de milisegundos. Posteriormente se generó un parche para introducir los temporizadores de alta resolución (HRT) [HRT]. El proyecto Ktimers[Gleixner05] también propone soluciones para proporcionar alta resolución temporal.

La implementación de "mutex" robustos para tiempo real ha sido tratada en varios proyectos, como Fusyn[Perez04] con participación de Intel, y Futex[Franke02], con participación de Montavista y Bull.

Posteriormente, estas y otras propuestas han sido integradas produciendo sistemas operativos de tiempo real estricto, en lugar de meros parches. Adeos [Yahmour] propone un entorno flexible para la compartición de recursos HW sobre múltiples sistemas operativos. Adeos permite la ejecución de múltiples componentes de kernel, llamados dominios sobre el mismo hardware. Adeos se encarga de la gestión de los entornos y las interrupciones.

TimeSys[Timesys] proporciona una variante de Linux con un modelo avanzado de gestión de interrupciones y bloqueo de recursos compartidos. Las rutinas de interrupción son manejadas como objetos planificables, y los "spinlocks" son reemplazados por "sleepmutex" con soporte de herencia de prioridad.

Concurrent Computer Corporation [RedHawk] ha desarrollado un sistema basado en Linux para plataformas SMP garantizando tiempos de respuesta inferiores a los milisegundos. Dicho sistema se llama RedHawk. En el las prioridades e interrupciones de alta prioridad tienen procesadores reservados, con lo que se garantiza su ejecución en tiempo óptimo, sin intromisiones externas.

RTLinux y su evolución, RTMS [RTMS], son sistemas operativos de tiempo real estricto que combinan dos núcleos, uno de propósito general (Linux), y uno propietario de tiempo real. El sistema de propósito general se ejecuta como la tarea de menos prioridad del sistema de tiempo real estricto.

RTAI [Dozio03] presenta una aproximación similar a RTLinux, con doble sistema operativo. Para ello utiliza la infraestructura proporcionada por el sistema Adeos.

Con objeto de extender RTAI, en el proyecto Hyades-ITEA ha sido desarrollado un sistema operativo de tiempo real estricto para sistemas SMP. Este sistema, diseñado por Thales con la colaboración de Mandrakesoft e Intel. Este sistema ha sido seleccionado en la tesis como base para realizar la extensión del modelo de sistema operativo POSIX/Linux con servicios de tiempo real estricto.

3.4.1.1 Extensión de tiempo real: Projecto HYADES

El proyecto Hyades-ITEA propone una versión de Linux extendida para sistemas multiprocesador SMP de alto rendimiento con requerimientos de tiempo real [Chanteperdrix04]. Para ello modifica algunos de los servicios proporcionados por un kernel de Linux convencional.

La gestión de interrupciones es uno de los puntos más controvertidos en los sistemas Linux para tiempo real. La influencia de los manejadores de interrupción en la ejecución de tareas críticas no es óptimo. Dichos manejadores no están pensados para reducir el tiempo de manejo de la interrupción lo mas posible acercándose al tiempo de gestión HW de la interrupción (Cambio de registros, salto al vector de interrupción, ...). Por tanto, en tareas de tiempo real duro, los manejadores convencionales de Linux pueden suponer sobrecargas y retrasos inaceptables.

Además, los accesos y operaciones del kernel de Linux tampoco están optimizados en tiempo. Independientemente de que la tarea que acceda sea crítica o no las llamadas hacen las mismas operaciones, realizando gran cantidad de cálculos y comprobaciones. Por ello, estos accesos tardan demasiado tiempo y son ineficaces cuando son ejecutadas por tareas críticas.

No obstante los mayores problemas aparecen en sistemas SMP. En estos sistemas, los eventos producidos en un procesador no son manejados por el resto. Esto significa que la gestión de prioridades no es correcta en el sistema SMP en conjunto, aunque en un procesador el orden de ejecución y la inversión de prioridades estén adecuadamente manejadas. Por ejemplo, si en un sistema dual, llega una interrupción a un procesador que despierta dos tareas críticas, se ejecutará una a continuación de la interrupción en ese mismo procesador. Sin embargo, la otra, que debería ejecutarse también inmediatamente en el otro procesador, permanecerá en espera hasta que la tarea en curso decida abandonar el procesador. Esto se debe a que nada indica al procesador que lanzar el planificador. Por tanto, se producirá un caso de inversión de prioridad que debe ser evitado.

Para resolver estos y algunos otros problemas en que aparecen en el kernel básico de Linux, la propuesta realizada en el proyecto Hyades es implementar una estructura de co-kernel y añadir una gestión auxiliar de interrupciones que envuelvan al kernel original.

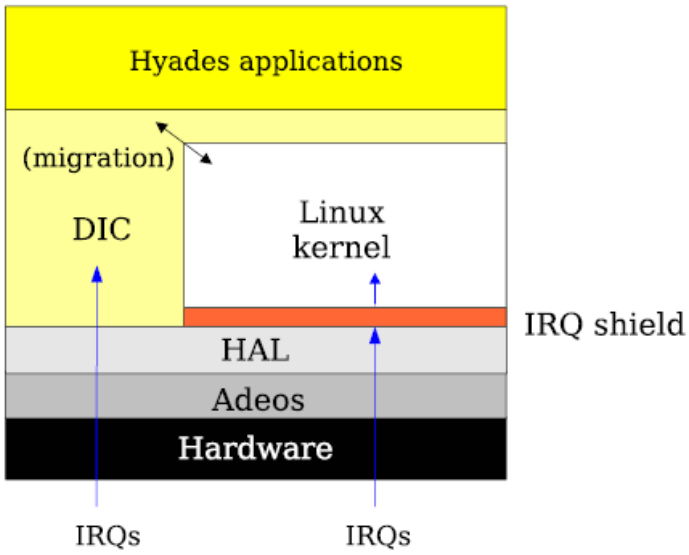


Figure 3-6: Arquitectura del sistema Hyades

De esta forma, se integran dos núcleos en el OS: el estándar de Linux, con su planificación original, y otro para cómputo intensivo, con una planificación completamente determinista (DIC). Este último gestionará las tareas de tiempo real.

El DIC implementa sus propias funciones de API, en número y costo mínimo. No obstante, algunas funciones pueden ser llamadas desde tareas que se encuentren ejecutando sobre el otro kernel. En ese caso, la tarea será automáticamente movida de kernel antes de realizar la llamada. Para ello se ha colocado una capa por encima de la API original del kernel de Linux que realiza estas transferencias.

También se coloca un gestor (Adeos) por debajo del HAL original que maneja directamente las interrupciones y traps, para minimizar el retardo de los manejadores, y decidir si las interrupciones son críticas o no. Además Adeos se encarga de gestionar el uso de recursos exclusivos manteniendo el orden de prioridades en los eventos.

Las interrupciones críticas son manejadas por el DIC, interrumpiendo incluso a las tareas de tiempo real crítico. El resto de interrupciones son manejadas por el kernel original, que las recibe de forma controlada. Cuando una tarea de tiempo real esta ejecutando en el kernel original, la interrupción es bloqueada. Esto se llama “IRQ shielding”.

3.5 CORBA

3.5.1 Software intermedio: los ORB

Los ORB (Object Request Broker) son tecnología intermedia (“middleware”) que se coloca entre la aplicación y la plataforma (e.g. el sistema operativo), para independizar la aplicación de la arquitectura del sistema. Un mecanismo de gestión de peticiones de objetos

se encarga de realizar las comunicaciones entre objetos de tal forma que los objetos no necesitan saber donde están los componentes con los que se relacionan. De esta forma, los ORBs proporcionan mayor interoperabilidad entre los sistemas basados en objetos distribuidos, ya que permiten conectar objetos independientes automáticamente, incluso provenientes de diferentes fuentes, sin necesidad de alterar ni conocer el código, independiente de que estos sean ejecutados conjuntamente en un procesador o cada uno corra en un extremo distinto del sistema [Wade94].

Para ello, la tecnología ORB proporciona herramientas que generan automáticamente las interfaces necesarias, de tal forma que todos los detalles de comunicación son ocultados al usuario y encerrados en el ORB [Cobb95]. EL objetivo es permitir la comunicación entre los distintos objetos del código de aplicación atravesando internamente los límites impuestos por la arquitectura HW, la infraestructura SW y las limitaciones del código propietario. Las funciones principales de un ORB son definir las interfaces de comunicación, controlar el estado y la posición de los objetos en el sistema y permitir la comunicación entre objetos cliente y servidor remotos. El ORB se encarga de comunicar los objetos enviando la información de las peticiones de los clientes a los servidores y devolviendo los resultados de los servidores a los clientes. Para ello proporciona una lista de servicios y facilidades para conectar los clientes y los servidores [CORBA96, Steinke95]. La figura siguiente muestra someramente su funcionamiento:

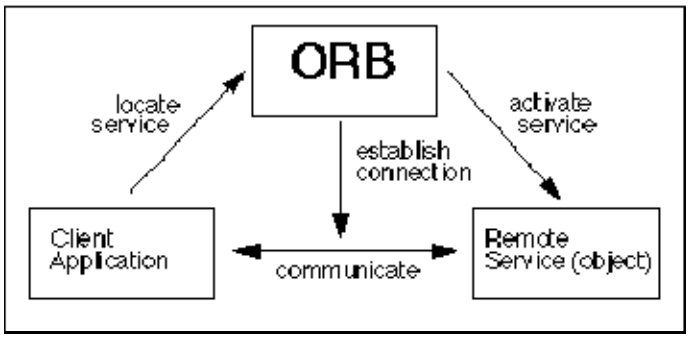


Figure 3-7: Funcionamiento de un ORB

Un ORB tiene como objetivo aparentar a la vista de los clientes que los servidores están localmente en el mismo ámbito, aunque puedan ser diferentes procesos o estar ejecutando en diferentes procesadores [Reddy95]. Sin embargo, las capacidades de un ORB no se limitan a esto. También debe proporcionar interoperabilidad entre distintas plataformas. El ORB oculta completamente la implementación de los objetos, de tal forma que un cliente y un servidor pueden tener diferentes lenguajes de programación, diferentes sistemas operativos, arquitecturas del sistema o de procesador, diferentes conjuntos de instrucciones o "endianismo" o estar en diferentes redes.

Para implementar estas técnicas existen varias posibilidades: integrar las funciones de comunicación en los objetos, generar procesos independientes o integrarlo como una capa adicional del sistema operativo. Cada tecnología de ORB tiene sus propias particularidades al respecto. De estas, las tres más importantes son:

- La especificación de una Arquitectura Común de ORB (Common Object Request Broker Architecture - CORBA) generada por el Object Management Group's (OMG).
- El modelo de Microsoft de componentes-objeto, Component Object Model (COM)
- Las técnicas de ejecución remota de Java. Integrada en la máquina virtual de Java, el sistema de invocación de métodos remotos (Remote Method Invocation (RMI)) es un sistema que proporciona capacidades de ORB como extensión nativa de Java [RMI97].

Algunos otros ejemplos de ORB son:

- MICO: Una implementación Open-Source de un ORB.
- ORBacus de Object Orientated Concepts (hoy en día fusionado con Iona).
- Visibroker de Borland.
- OrbixWeb de Iona.
- Component Broker de IBM.

El uso efectivo de un determinado ORB en el diseño requiere un análisis adecuado de las necesidades arquitecturales de la aplicación y de cómo cada ORB puede resolverlas [Abowd 96]. Entre las características a analizar están la disponibilidad para la plataforma particular, el soporte para los lenguajes de programación a utilizar o el rendimiento del ORB.

Finalmente, cabe mencionarse que bajo TCP/IP, CORBA emplea el protocolo llamado IIOP ("Inter Orb Protocol") para ejecutar métodos/funciones en los diversos objetos del sistema.

3.5.2 El estándar CORBA

CORBA (Common ORB Architecture), es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

CORBA fue definido y está gestionado por el Object Management Group (OMG) que define las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida.

En un sentido general CORBA "envuelve" el código escrito en otro lenguaje en un paquete que contiene información adicional sobre las capacidades del código que contiene, y sobre cómo llamar a sus métodos. Los objetos que resultan pueden entonces ser invocados desde otro programa (u objeto CORBA) desde la red. En este sentido CORBA se puede considerar como un formato de documentación legible por la máquina, similar a un archivo de cabeceras pero con más información.

CORBA utiliza un lenguaje de definición de interfaces (IDL) para especificar las interfaces con los servicios que los objetos ofrecerán. CORBA puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo los tipos de dato CORBA

deben ser utilizados en las implementaciones del cliente y del servidor. Implementaciones estándar existen para Ada, C, C++, Smalltalk, Java y Python. Hay también implementaciones para Perl y TCL.

Al compilar una interfaz en IDL se genera código para el cliente y el servidor (el implementador del objeto). El código del cliente sirve para poder realizar las llamadas a métodos remotos. Es el conocido como "stub", el cual incluye un representante (proxy) del objeto remoto en el lado del cliente. El código generado para el servidor consiste en unos esqueletos (skeletons) que el desarrollador tiene que rellenar para implementar los métodos del objeto.

No obstante, CORBA es más que una especificación multiplataforma, también define servicios habitualmente necesarios como seguridad y transacciones.

En el ámbito de la tesis, se ha integrado un modelo CORBA en SystemC, basado en MICO y proporcionado por TIMA. Este middleware tiene como objeto permitir el modelado en SystemC de aplicaciones SW distribuidas, teniendo en cuenta el rendimiento del middleware, OS, procesador, plataformas HW de los nodos de comunicación y red de comunicaciones.

4 Análisis del estado del arte

El objetivo general de la tesis es desarrollar un entorno de simulación en el que se pueda verificar y evaluar el rendimiento de sistemas HW/SW complejos de manera rápida y eficiente, aún a costa de un cierto grado de precisión.

Para alcanzar estos objetivos se han propuesto históricamente varios procedimientos, que en función de sus puntos fuertes y débiles, han sido aplicados a distintos ámbitos de desarrollo. En esta sección se repasarán algunas de estas soluciones.

La primera división que se ha de realizar en los trabajos relacionados con el área de estudio, tanto para simulación como para estimaciones de rendimiento, es la existencia de técnicas de análisis tanto estáticas como dinámicas. Dentro de las dinámicas se puede realizar otra subdivisión entre simuladores basados en simulación nativa basada en código fuente y simulación basada en código binario obtenido por compilación cruzada.

Por último, dentro de las técnicas de análisis basadas en simulación nativa, hay que considerar el modelado de cada uno de los elementos del sistema, especialmente código SW de aplicación, sistema operativo y plataforma HW.

Estas técnicas, que tienen de por si aplicación en las tareas de verificación, evaluación y refinado del diseño, también están siendo especialmente importantes al estar siendo aplicadas en el ámbito de la exploración del espacio de diseño. Esta exploración ha de ser realizada en las primeras etapas de desarrollo con objeto de tomar las decisiones de implementación más adecuadas para el sistema en desarrollo, como la configuración de la plataforma HW, la partición HW/SW o la asignación de recursos. Por ello, aunque el desarrollo de algoritmos para exploración no es un área cubierta por esta tesis, si lo es la integración de la infraestructura de simulación desarrollada. Por ello se ha considerado interesante añadir al análisis del estado del arte un breve estudio sobre técnicas de exploración.

4.1 Trabajos para exploración del espacio de diseño

La exploración del espacio de diseño es una de las etapas más importantes de todo diseño electrónico. Si bien es posible obtener importantes mejoras en un diseño dedicando grandes esfuerzos a la optimización del HW y SW del sistema, una correcta elección del mejor punto en el espacio de diseño puede tener efectos incluso más importantes. Decidir qué parte del sistema ha de ser implementado en HW y cuál en SW, diseñar una plataforma HW optima y asignar adecuadamente los recursos en los que se ejecutará cada elemento son puntos fundamentales de cualquier buen diseño. En consecuencia, para buscar las soluciones que optimicen simultáneamente diversos parámetros, como consumo o rendimiento, en función de la gran cantidad de opciones de configuración que podemos tener, es necesario usar técnicas de exploración multi-objetivo.

En general, la optimización multi-objetivo es el proceso de optimizar simultáneamente dos o más parámetros que se encuentran en conflicto respecto a ciertos objetivos. Estas técnicas tienen cabida en aquellos casos en los que la optimización de todos los parámetros no lleva a una única solución. En esos casos, la mejora de determinados parámetros respecto a unos objetivos conlleva el empeoramiento de otros. De esta forma las técnicas de optimización multi-objetivo están diseñadas para encontrar el conjunto de soluciones que optimiza los factores entre todos los objetivos, de tal forma que para cada combinación de valores de n-1 objetivos se elige la solución que minimiza el objetivo restante. El concepto es fácilmente analizable mediante representaciones gráficas. Si dibujamos todos los puntos posibles en función de los objetivos, obtendremos una nube de puntos. De estos, los puntos óptimos son aquellos que conforman los bordes de la nube. Estos puntos óptimos constitullen lo que se denomina el "frente de Pareto".

Problemas de optimización multi-objetivo se encuentran en multitud de campos, como procesos de producción, finanzas, diseño aeroespacial, automoción, industria petrolera, etc. Maximizando el beneficio mientras se minimiza el coste, maximizar potencia minimizando consumos, o maximizando la robustez minimizando el peso son ejemplos comunes de problemas de optimización multi-objetivo. Por esta razón gran cantidad de trabajos se han realizado en ese campo, durante al menos las tres últimas décadas [Sawaragi85][Mueller09].

Sin embargo la aplicación de estas técnicas al diseño electrónico ha estado limitada en determinados aspectos. Las técnicas habituales de exploración requieren la evaluación de una buena parte de los puntos posibles del espacio de diseño para tener una idea de cómo es dicho espacio y poder así elegir un punto óptimo. Sin embargo, la aplicación de técnicas de exploración utilizando soluciones tradicionales de modelado de sistemas, como modelos de precisión de ciclo, presentan ciertos inconvenientes. Al ser estas soluciones bastante lentas, evaluar la cantidad necesaria de puntos del espacio de diseño requeridos para realizar la exploración de sistemas completos resulta en unos tiempos totales completamente inviables. Por ejemplo, para realizar una exploración de 6 ó 7 parámetros de configuración con 4 ó 5 valores posibles cada uno podría requerir de la evaluación de unos 10.000 puntos. Si cada simulación necesita de al menos una hora (tiempo relativamente bajo para una simulación de precisión de ciclo), el tiempo total de las simulaciones sería de un año. Sería necesario disponer de técnicas de simulación que permitan realizar las evaluaciones en cuestión de segundos en lugar de horas. Por eso la exploración se ha propuesto la utilización de simulaciones realizado a muy alto nivel, a nivel algorítmico [Mehra96][Knudsen96] [Peixoto97][Gerlach98][Henke02][Kaul99][Niemann96].

Los procesos de exploración desarrollados hasta la fecha a niveles más bajos se limitaban a exploración de algunos de los componentes del sistema por separado. Ejemplos de estos trabajos pueden verse en [Khare99] donde se trabajaba en la optimización interna de procesadores, mejorando el camino de datos o analizando modificaciones en el conjunto de instrucciones.

Sin embargo, la aparición de técnicas con mejor relación velocidad/precisión de simulación de sistemas complejos HW/SW ha reabierto el interés en la exploración del

espacio de diseño en los últimos años. En este sentido se han propuesto tanto mejoras de los algoritmos de exploración como soluciones para la adaptación a las nuevas técnicas de simulación a los procesos de exploración. En cuanto a los algoritmos de exploración, estos se basan habitualmente en la adaptación de las técnicas preexistentes a las particularidades del diseño electrónico [Beltrame06][Ascia07]. Mejoras en la selección de los puntos del espacio a simular [Sheldon07], generación de modelos predictivos [Ipek06] y técnicas de regresión [Joseph06] conforman los puntos más interesantes. Herramientas de exploración gratuitas [m3explorer] y comerciales [modeFRONTIER] han sido diseñadas y adaptadas para su utilización juntamente con los simuladores encargados de evaluar las distintas configuraciones de los sistemas electrónicos. No obstante, dado que esta tesis se centra en las técnicas de simulación, no ahondaremos más en esta área, y nos centraremos en el área de estimación de prestaciones.

4.2 Estimación de prestaciones

Ya sea para realizar exploraciones del espacio de diseño o simplemente para evaluar el rendimiento que los sistemas en desarrollo tendrán una vez implementados y dirigir el diseño, la estimación de las prestaciones durante las primeras etapas del proceso de diseño tiene suma importancia [Puschner89]. En muchos sistemas, parámetros como el tiempo de repuesta o el consumo son tan importantes como su funcionalidad, por lo que hay que tenerlos muy en cuenta durante todo el diseño. Por esta razón la estimación del rendimiento es un problema tradicional en la ingeniería de sistemas embebidos.

Para realizar la estimación de prestaciones de los sistemas de desarrollo se han propuesto diversas técnicas. Estas técnicas pueden dividirse en dos grandes grupos: técnicas dinámicas y técnicas estáticas.

Las técnicas dinámicas se basan en simular el sistema, proporcionando unos valores de entrada que modelen de forma adecuada el entorno en el que va a trabajar el sistema [Fummi04][Hwang08][Oyamada07][Kirchsteiger08]. De esta forma se obtiene información del rendimiento del sistema para cada situación simulada. Esta solución presenta el riesgo de que en un sistema mínimamente complejo es imposible simular todas las posibles situaciones de entrada, con lo que no podemos estar seguros de que no se producirá una determinada situación en la que el sistema responda de manera inesperada.

Las técnicas estáticas se basan en el análisis del flujo del código del sistema [Wilhelm08]. Considerando el camino de datos que ejecutará el código, se estima el tiempo y el consumo asociados. La idea básica es dividir el código en nodos, que representan por ejemplo bloques básicos. A partir de ahí se estima el coste de cada nodo y se realiza un análisis para comprobar el cuando se ejecuta cada uno. Sabiendo el coste de cada nodo en tiempo, potencia, etc. y el número de veces que se ejecuta se puede calcular el coste total del programa.

No obstante, hay que tener en cuenta que en la mayoría de las ocasiones, el flujo de ejecución depende de los datos. La decisión de si se toma un “if” o se repite un “for” depende

del valor de los datos. Sin ejecutar el código no es posible conocer los valores de las variables en cada punto del código. Además, cada vez que el programa pasa por un nodo lo hace con datos distintos. Como consecuencia, un análisis estático considerando todos los nodos una vez por cada posible ejecución, implica una cantidad inabordable de nodos a analizar. Por esa razón los análisis estáticos se hacen de forma separada de los datos. Mientras que las estimaciones dinámicas se basan en simulaciones aplicando diferentes bancos de pruebas, las estáticas no dependen de los bancos de pruebas.

La independencia de los datos de entrada hace que sea difícil evaluar el comportamiento medio del sistema. En consecuencia estos métodos de evaluación no suelen considerar el efecto de que determinadas combinaciones o secuencias de entradas sean más comunes que otras. Por ello, el análisis del comportamiento medio suele dar resultados pesimistas. Sin embargo, al ser capaz de analizar todos los casos posibles, conoce el coste de todos los casos, obteniendo los límites inferiores y superiores, delimitando el problema.

Como consecuencia, cada tipo de técnica se aplica a un entorno distinto. En sistemas en los que hay que garantizar que cualquier caso funcionará adecuadamente, se utilizan métodos estáticos, ya que son capaces de evaluar los peores casos. En sistemas donde se busca que el sistema funcione generalmente lo mejor posible, y se permiten ciertos fallos en puntos esporádicos se utilizan estimaciones dinámicas, ya que evalúan mucho mejor el funcionamiento medio.

En algunos casos se realizan estimaciones híbridas para mejorar los resultados obtenidos. En algunas técnicas estáticas [Brandolese01][Anantaraman04][Colin02] se realizan simulaciones parciales de los sistemas para obtener estimaciones de los nodos. A partir de esos datos se aplica el análisis de flujo de datos para obtener información de la simulación. El resultado tiene cualidades similares a las estimaciones estáticas, y se aplican en el mismo ámbito.

En otros casos [Schnerr08], se realiza un análisis estático de pequeños segmentos de código, se anota la información en el código y se realiza una simulación de forma que se va acumulando el coste de cada segmento de código ejecutado para finalmente obtener el coste total. Estas técnicas se aplican en las mismas áreas que las estimaciones puramente dinámicas.

Para realizar evaluaciones y exploraciones de sistemas con fuerte dependencia HW/SW, las técnicas dinámicas proporcionan una mejor solución que las estáticas. Como se dijo antes, las técnicas estáticas mantienen independencia entre el análisis del código y el entorno, mientras que las dinámicas dependen de la interacción con el entorno. Para el código SW, los componentes HW se ven, en buena medida, como elementos del entorno. Por tanto, si quedemos poner énfasis en las interacciones HW/SW las técnicas dinámicas basadas en simulación son las que proporcionan mejores resultados. Es por ello que han sido elegidas como solución para el desarrollo de la tesis.

4.3 Técnicas de estimación dinámicas

Las técnicas de simulación de sistemas electrónicos necesarias para los cubrir los objetivos de la tesis, han de mantener un equilibrio entre velocidad y precisión. Para ello se necesita que los modelos de cada uno de los componentes involucrados en la simulación estén realizados al nivel de abstracción adecuado. Para alcanzar este objetivo, el uso de TLM se ha revelado como una solución ineludible [TLM]. No obstante, TLM puede ser utilizado aplicando varios tipos de descripción, especialmente, distinguiendo componentes HW y SW.

En TLM, los componentes HW se modelan mediante descripciones que aúnan comportamiento e información temporal, ya sea con precisión de ciclo o con tiempos aproximados. En caso de modelos de precisión de ciclo, los tiempos del HW están directamente integrados en función de dichos ciclos. En los modelos de tiempo aproximado, es habitual que los costes temporales y de consumo de las actividades tengan tiempos asociados. La elección del tipo de modelado temporal depende de la relación requerida entre precisión de la simulación y velocidad.

Los modelos con precisión de ciclo tienen una precisión similar a los modelos RTL. Aunque su ejecución es más rápida que el RTL, la simulación a este nivel es muy pesada, requiriendo grandes tiempos para la simulación de sistemas electrónicos. La simulación con modelos de tiempo aproximado es más ligera, ya que gran parte de los detalles internos de los componentes son obviados. Estos modelos son adecuados para la simulación completa del sistema y la evaluación de su validez en el entorno seleccionado. También son muy útiles cuando se quiere comprobar la integración de componentes concretos en el sistema completo. Por ejemplo, la evaluación del funcionamiento del SW de un sistema es más eficiente si se realiza considerando una plataforma HW compuesta modelos de tiempo aproximado de sus componentes. Con este modelo de plataforma es posible considerar los efectos del HW en la ejecución SW sin ralentizar extraordinariamente la simulación.

En el caso del SW el problema es más evidente, ya que normalmente un código SW no lleva asociada información de su rendimiento. Un código “C/C++” puede incluir elementos temporales, como temporizadores, esperas o “timeouts”, pero no suele indicar cuánto tiempo de procesador va a requerir su ejecución. Esa información se obtiene a posteriori. Por ello es necesario disponer de técnicas que permitan estimar y añadir información sobre el coste del código SW en ejecución. El modelado de rendimiento de componentes SW se realiza a dos niveles: mediante simulación nativa o mediante simulación de código binario.

La simulación nativa [Hwang08][Schnerr08][Bouchima09] obtiene la información de rendimiento en función del análisis del código fuente a ejecutar. Ya sea antes o durante la simulación se estima el coste de segmentos del código a ejecutar y se integra esa información en la simulación dinámicamente.

La simulación de código binario se basa en ejecutar el código destino tras ser compilado para la plataforma destino. Dado que el código ha sido para una plataforma especial no puede ser ejecutado directamente sobre el computador donde se realiza la

simulación. Se necesita una herramienta que interprete y ejecute ese código binario. En este caso, la herramienta es además la encargada de generar la información de rendimiento. Para ello ha de poseer junto al conocimiento de qué significa cada instrucción binaria, cuánto tiempo y que efecto produciría en el procesador destino. La ejecución de ese código binario se realiza de tres formas principales: mediante modelos del procesador, mediante simulación compilada, o mediante traducción dinámica.

4.3.1 Modelos virtuales del procesador

La forma más realista de modelar la ejecución de un procesador es mediante un modelo virtual de dicho procesador. Este modelo integra una descripción de los componentes del procesador, como las unidades funcionales o los bancos de registros, junto con descripciones de su funcionamiento. El modelo recibe el código binario a ejecutar y lo utiliza como entrada para los elementos internos del modelo. La unidad de datos, la unidad de control los bancos de registros, etc. del modelo realizan las mismas operaciones que el modelo real, con lo que se modela el comportamiento del código en la plataforma destino. Estos modelos capaces de interpretar el código binario y modelar la ejecución de las instrucciones correspondientes de denominan simuladores del conjunto de instrucciones (Instruction Set Simulator, ISS).

La precisión del modelo puede variar en función del grado de detalle con el que se describen los componentes internos. Para simuladores con precisión de ciclo, detalles como las etapas de la unidad de datos han de estar debidamente integrados, mientras para que simuladores a nivel de instrucción no requieren esos detalles.

Los simuladores basados en modelos del procesador implican una baja velocidad de simulación a cambio de un alto grado de precisión cuando los comparamos con el resto de mecanismos de simulación utilizados en los niveles de abstracción considerados en la tesis. Estos simuladores son los más comunes en entornos profesionales, como CoWare Processor Designer [Coware], CoMET de VaST Systems Technology [CoMET], Synopsys Virtual Platforms [Synopsys], MPARM [Benini03].

Algunos trabajos de investigación proponen mejoras a estas herramientas. En [Schirner07] el modelo de procesador se integra junto con un RTOS.

4.3.2 Simulación compilada

El objetivo de la simulación compilada es mejorar el rendimiento de las simulaciones basadas en modelos virtuales del procesador, pero manteniendo su precisión. Para ello se considera que algunas operaciones costosas de los modelos virtuales pueden ser optimizadas realizando la adición en tiempo de compilación. Esta etapa se realiza por una herramienta llamada compilador de simulación.

Durante esta etapa se realizan algunas de las etapas del modelo que pueden llevarse a cabo en estático al no depender de los datos. Por ejemplo, la etapa de decodificación es

realizada en tiempo de compilación. En función de los resultados de la etapa de decodificación el compilador de simulación selecciona las operaciones nativas necesarias para simular la aplicación [Nohl02]. El grado de compilación varía en función de la arquitectura del procesador, las características de la aplicación y la solución utilizada. Como resultado se obtienen un código intermedio que puede ser ejecutado por un modelo virtual simplificado, más rápido. Simulaciones compiladas basadas en lenguajes de descripción arquitectural han sido desarrollados en los proyectos Sim-nML (FSim) [Hartoog97], ISDL (XSSIM) [Hadjiyiannis97] y MIMOLA [Leupers99].

4.3.3 Traducción binaria

La tercera solución utilizada para simular código binario compilado para una plataforma destino, es la traducción binaria [Gligor09]. Esta técnica se basa en traducir dinámicamente las instrucciones ensamblador del procesador destino en instrucciones ensamblador del computador nativo de la simulación. De esta forma no es necesario tener un modelo virtual del procesador que describa los componentes del procesador real. Como consecuencia, el código es simulado más rápido que con los dos métodos anteriores. Sin embargo, al no tener un modelo del procesador no se obtiene información de rendimiento del SW. Además, los sistemas actuales de traducción binaria no suelen ir enlazados a un modelo de plataforma, con lo que los efectos de caches, memorias o buses no son considerados en la simulación. Estos simuladores son utilizados para verificación funcional, más que para estimaciones de rendimiento. Algunos ejemplos son IBM PowerVM [PowerVM], QEMU [Qemu] o UQBT [UQBT].

Algunas de estas técnicas han sido mejoradas mediante re-compilación dinámica. En esta técnica determinados fragmentos de código que se ejecutan muchas veces son re-compilados de forma que no es necesario traducirlo cada vez.

Para resolver la ausencia de información temporal en [Schnerr05] se propone introducir dicha información en el código binario añadiendo instrucciones al principio y final de cada bloque básico. No obstante, al necesitar un análisis estático de cada bloque básico, la precisión no es comparable al resto de modelos basados en código binario, sino a los de simulación nativa.

Sin embargo, todas estas técnicas son bastante más lentas que la ejecución funcional, con las limitaciones que eso conlleva cuando se requiere realizar evaluaciones rápidas del diseño. En concreto, la traducción binaria es la que está teniendo más aceptación en los últimos tiempos, aunque los tiempos necesarios para obtener estimaciones de rendimiento precisas con dichas técnicas están aun unos 2 ó 3 órdenes de magnitud por debajo de la ejecución funcional.

Simulation type	Speed	Accuracy
Functional execution	100000	
Timed native co-simulation	10000	
Timed binary translation	1000	
ISS (instructions)	100	
ISS (cycle accurate)	10	
Pin accurate	1	

Figure 4-1: Relación de velocidad de distintas técnicas de simulación

4.4 Simulación nativa de código fuente

Para aumentar la velocidad de las simulaciones basadas en código binario, se han propuesto técnicas de simulación a partir de código fuente [Gerslauer10]. En ellas, el código se compila para ser ejecutado como un programa normal del ordenador donde se realiza la simulación. Por tanto no se necesita ninguna herramienta intermedia que permita ejecutar el código SW. Para obtener estimaciones de rendimiento de dicho código se realizan determinadas modificaciones al código mediante técnicas de instrumentación y anotación. Además, para poder modelar sistemas HW/SW completos, la simulación del SW se enlaza con modelos virtuales de la plataforma HW. Todas estas extensiones permiten que la estimación de rendimiento del sistema tenga la precisión suficiente para permitir la evaluación y exploración de los sistemas embebidos en desarrollo.

Para permitir el correcto modelado del SW del sistema es necesario disponer de una infraestructura que permita realizar el modelado temporal y funcional tanto del código de aplicación como de la infraestructura SW. El modelado de temporal del código de aplicación requiere de la obtención de estimaciones del tiempo de ejecución de dicho código en la plataforma destino. Una vez obtenidas dichas estimaciones, ha de incluirse la información temporal en la simulación con objeto de permitir el modelado temporal del SW de aplicación.

Para modelar la infraestructura SW es necesario considerar el efecto del sistema operativo, así como otros componentes de dicha infraestructura que puedan incluirse puntualmente en un sistema, como infraestructuras middleware, extensiones de tiempo real, etc. Todos estos serán desarrollados en la tesis, y por ello su estado del arte será estudiado en detalle en las siguientes secciones.

4.4.1 Estimación de tiempo de ejecución para simulación nativa

En simulación nativa, el código SW es ejecutado directamente, sin el soporte de un intérprete. Por esta razón, el tiempo de ejecución ha de ser proporcionado directamente en el código SW a ejecutar. Para ello la técnica habitual es dividir el código en fragmentos, estimar en estático el tiempo de cada uno de esos fragmentos y anotar esa información en el código.

Dicho código anotado se compila y ejecuta normalmente, junto con un mecanismo capaz de capturar la información de tiempo y aplicar los retardos asociados a la simulación.

Para obtener la información de tiempo de cada sección de código se han propuesto varias técnicas. Habitualmente se define la sección básica de código para la anotación como el bloque básico. Dado que un bloque básico se ejecuta siempre completo, se puede modelar temporalmente como una unidad sin introducir errores de estimación. Las técnicas tienen una precisión distinta en función del nivel donde se aplican: en código fuente, en código intermedio o en código objeto.

Las estimaciones en código fuente se basan en analizar las instrucciones C/C++ del bloque básico y estimar el número de ciclos que requerirá su ejecución en la plataforma destino. Esta técnica asocia un coste temporal a cada instrucción C que es función del compilador y de la plataforma destino. Mediante operaciones matemáticas simples se calcula el coste total de cada bloque de código en función de las operaciones C que incluye [Brandolese01]. Este nivel es, de los tres, el más independiente de la plataforma destino, ya que no necesita la utilización del compilador destino, sino únicamente de una base de datos con los costes de cada operación C, que puede ser proporcionada por el vendedor del producto. No obstante, en contraposición también es la menos precisa de las tres.

La estimación en código binario se basa en la obtención de equivalencias entre el código binario y el código fuente. Para segmento de código fuente se asocia la secuencia de código binario equivalente. Analizando el tiempo requerido por la ejecución de cada secuencia de código binario se obtiene el coste temporal de cada bloque básico de código C/C++ [Schnerr08]. El análisis puede hacerse mediante la ejecución de los segmentos en un simulador de instrucciones o mediante un análisis estático. En ambos casos la precisión es bastante alta, ya que tiene en cuenta todos los efectos y optimizaciones del compilador. Sin embargo presenta dos inconvenientes. En primer lugar, en algunos casos no es posible asociar secuencias de código fuente y secuencias de código ensamblador, como consecuencia de las optimizaciones del compilador. En segundo lugar, es necesario disponer del compilador destino para poder realizar estimaciones correctas. Además, en función del compilador, la traducción y las optimizaciones pueden cambiar de forma que los mecanismos de asociación entre código fuente y código binario no es completamente portable entre plataformas distintas.

La estimación a partir de código intermedio se basa en analizar los resultados intermedios de compilación para ir conociendo cuales son las operaciones intermedias del compilador y así poder asociar sin problemas el código fuente y el código binario [Kempf06][Hwang08][Bouchima09]. La precisión obtenida mediante estas técnicas es similar a las de código binario. Elimina las limitaciones de asociación pero es completamente dependiente del compilador destino. Dado que cada compilador utiliza sus propios formatos intermedios y realiza unos procesos de optimización distintos, el análisis por código intermedio tiene un bajo grado de portabilidad. Esto puede ser un problema para su aplicación a exploración del espacio de diseño.

Con objeto de conseguir la máxima portabilidad para la exploración del espacio de diseño en las primeras etapas de desarrollo, en la tesis se ha trabajado en estimaciones sobre código fuente, con algunos acercamientos al análisis por código binario.

4.4.2 Modelado de Caches

Las técnicas convencionales de estimación de código SW para simulación nativa tienen en cuenta el tiempo de ejecución de cada secuencia de código relativo al procesador. Sin embargo, los tiempos asociados a la arquitectura de memoria del sistema no son considerados. Estos tiempos han de ser evaluados a parte.

El modelado de una arquitectura de memoria convencional en sistemas MpSoC se divide en dos partes. Por un lado se modela la cache de primer nivel (L1) en función de la ejecución nativa de código SW. El resto de la arquitectura de memoria se modela como parte de la plataforma HW. Cada vez que en el modelo de cache L1 se produce un fallo de línea se genera un acceso TLM al elemento del modelo de plataforma HW correspondiente. A partir de aqui todos los accesos son modelados mediante transacciones.

La cache L1 no se considera una parte más del modelo de plataforma HW dado que los accesos no se producen por transferencias. Su modelado debe partir directamente de la estimación y ejecución del código SW. Para ello se han propuesto tres tipos de soluciones: estadísticas, basadas en análisis estáticos y basadas en anotación.

Las técnicas basadas en modelos estadísticos se basan en factores fijos independientes de la aplicación. Por esta razón estas técnicas permiten un modelado temporal del sistema, pero no son válidas para la exploración del espacio de diseño. Las caches son unos de los elementos más comunes de exploración al determinar los parámetros más apropiados para la aplicación objetivo. El tamaño de la cache o el número de vías son unos de los parámetros más importantes a decidir en el diseño de sistemas MpSoC.

Las técnicas basadas en análisis estáticos si permiten modelar el efecto de las caches en función de la aplicación. Estas técnicas han dado buenos resultados en estimación de rendimiento para análisis de peor caso [Mueller00]. En [FeWi99] se utiliza el método denominado “Abstract Interpretation” que calcula el WCET de una tarea prediciendo el comportamiento, tanto de la cache de instrucciones, como de la cache de datos. En [LuSt99], mediante el método “Symbolic Execution”, también se calcula el WCET de una tarea teniendo en cuenta el comportamiento, tanto de la cache de instrucciones, como de la cache de datos. En [WhMu97] se utiliza una técnica estática para predecir el comportamiento de una cache de datos. Estas técnicas pueden ser fácilmente adaptables a técnicas de estimación dinámica [Bjuréus01].

Todas estas técnicas intentan calcular el WCET de una tarea teniendo en cuenta el comportamiento de la cache, tanto de instrucciones, como de datos. Cualquiera de los métodos anteriormente citados proporciona una buena predicción del funcionamiento de la

cache de instrucciones; sin embargo, producen una sobre-estimación en el número de fallos cuando predicen los accesos a la cache de datos.

Predecir el funcionamiento de la cache de datos es bastante más complejo que predecir el funcionamiento de la cache de instrucciones. Esto es debido a las referencias indirectas (punteros y vectores de índices), cuya dirección no puede calcularse en tiempo de compilación.

Otro método que proporciona una buena predicción del comportamiento de la cache de datos es el conocido como CME: "Cache Miss Equations". Este método fue propuesto en [GhMM99] y permite predecir el comportamiento de la cache de datos de un programa resolviendo las denominadas ecuaciones de fallos de cache. Debido a sus limitaciones de aplicación fue extendido en [VeXu02] y posteriormente fue mejorado y modificado para generar patrones exactos de accesos a la cache de datos en [RaMu05].

Sin embargo, los métodos estáticos presentan un problema importante en la exploración de sistemas MpSoC. Estos métodos dan información de la eficiencia de la cache para una aplicación concreta, pero no proporcionan datos puntuales de su funcionamiento. Como indicamos anteriormente, para modelar el efecto de la arquitectura de memoria completa, es necesario que los fallos de cache generen las transacciones correspondientes en el momento correcto. Solo de esta forma se pueden modelar efectos como las colisiones en el bus, sobrecargas en las redes de comunicaciones o efectos del funcionamiento de la memoria principal. Además, los métodos estáticos no pueden tener en cuenta efectos como las des-habilitaciones de caches producidas por los cambios de contexto debidos al sistema operativo.

Por tanto son necesarias técnicas de estimación de cache completamente dinámicas. En [Schnerr08] se introduce una primera solución dinámica para el modelado de caches de instrucciones para simulación nativa, abriendo un importante tema de investigación, tanto para el modelado de cache de instrucciones como de datos. En esta tesis se han realizado trabajos en dicho ámbito (capítulo 6).

4.4.3 Modelado del sistema operativo

El segundo elemento fundamental para permitir un correcto modelado del SW en sistemas MpSoC es el sistema operativo [Zabel09][Becker10]. Las actividades de planificación, gestión de políticas y prioridades de ejecución o los elementos de comunicación y sincronización tienen un efecto muy importante en la ejecución de las tareas SW. Dado que en los ámbitos de aplicación de estas técnicas para diseño de MpSoC es muy importante el rendimiento y el tiempo de respuesta de los sistemas, los sistemas operativos utilizados suelen ser sistemas de tiempo real (RTOS).

Dada la importancia de los RTOS en el rendimiento del sistema es necesario que el modelo de MpSoC integre un modelo de sistema operativo adecuado. Diversas soluciones han sido proporcionadas con anterioridad para el modelado de sistemas operativos.

Algunos proveedores de sistemas operativos proporcionan simuladores como parte de su kit de desarrollo SW. Los simuladores de RTOS permiten el desarrollo y la emulación del código de aplicación sin necesidad de desarrollar los prototipos HW [ENEA][AXLOG]. Sin embargo, dado que la simulación depende el sistema operativo del ordenador donde se realiza la simulación estos sistemas carecen de la capacidad de realizar modelados temporales precisos. Además, estos simuladores están orientados al desarrollo de SW y no están preparados para su integración en entornos de co-diseño HW/SW.

Para conseguir entornos de co-simulación HW/SW con buenas relaciones entre precisión y velocidad para las primeras etapas del proceso de diseño es necesario realizar modelos del RTOS integrados en entornos y lenguajes de modelado en alto nivel. Varios modelos para SpecC [Tomiyama01][Gerstlauer03] y SystemC han sido propuestos [Hassan 05][He05][Schirner07]. Sin embargo, la mayoría de estas soluciones presentan funcionalidades limitadas e interfaces propias, que complican enormemente el modelado de códigos SW de aplicación reales [Gerstlauer03][He05][Yoo02] . Estos modelos se limitan a proporcionar capacidades de planificación de tareas. Posteriormente se han realizado modelos de sistemas operativos específicos. Estos RTOS eran muy ligeros y con funcionalidad reducid [Honda04][Hassan05].

Ante la necesidad de proporcionar modelos más completos para el modelado de sistemas operativos en MpSoC, en esta tesis se ha desarrollado un modelo muy completo de sistema operativo basado en la interfaz POSIX y la implementación del sistema operativo Linux.

4.4.4 Anotación de tiempos

Una vez estimados los tiempos de ejecución del código SW es necesario introducir esta información en la simulación para poder obtener un modelado realista. La anotación se realiza habitualmente en de dos formas: Introduciendo la información en el código fuente o a través del modelo de sistema operativo.

La introducción de las estimaciones temporales en el código fuente es la solución más inmediata La introducción de sentencias “wait” en cada secuencia de código SW hace que el control se bloquee en la tarea en curso, de tal forma que en la finalización de cada secuencia no solo modela el avance funcional sino también el avance temporal. [Tomiyama01][Yoo02][Gerstlauer03][Honda04][Hassan05].

Sin embargo esta solución presenta limitaciones en el modelado de efectos secundarios como interrupciones o expulsiones del sistema operativo. Una vez que comienza la llamada al “wait” de tiempo el control no retorna al sistema hasta su finalización. Por tanto, el modelado de eventos intermedios, como expulsiones, no es posible.

La solución propuesta habitualmente es permitir el modelado de estos eventos aun cuando la tarea en curso se encuentre bloqueada en un “wait”. Una vez que el “wait” finaliza se realiza un “wait” adicional con el tiempo que dicha tarea ha sido expulsada [He05]. Sin

embargo esta solución presenta algunos inconvenientes. El encadenamiento de expulsiones puede hacer que la gestión de esos "wait" adicionales se complique, al tener que distribuirse por todos las tareas bloqueadas. Si además consideramos la posibilidad de tener cambios dinámicos de prioridades o incluso movimiento de la tarea entre procesadores como consecuencia de las expulsiones, la implementación resultante puede ser muy costosa.

Otra complicación es que, al ser ejecutado el código antes de la llamada al "wait", y por tanto modelado antes de una posible expulsión, los datos con los que se realizan los cómputos pueden ser erróneos. Las tareas que se ejecuten durante la expulsión pueden modificar dichos datos.

Durante la tesis se han presentado soluciones para resolver estos inconvenientes (Capítulo 6).

4.5 Modelado de la plataforma HW

Además del modelado del SW, para una evaluación y exploración adecuada de los MpSoC también es necesario considerar el modelado de la plataforma HW. El modelado de la plataforma HW en modelos de alto nivel se basa fundamentalmente en el modelado del bus. El modelado del resto de componentes HW dependen del modelo de bus y suelen ser realizados con descripciones a nivel de comportamiento específicas para cada componente.

En exploración del espacio de diseño las simulaciones de la plataforma HW han de ser suficientemente rápidas para evaluar el mayor número de configuraciones posibles. No obstante, también han de tener la precisión suficiente para asegurar la corrección de las estimaciones. En el caso de infraestructuras de comunicación, como buses y redes, el aumento de velocidad ha de conseguirse minimizando el tiempo de simulación de cada transferencia. La aplicación de técnicas TLM hace que en lugar de necesitar una escritura en cada una de las señales del bus, una única llamada a función transfiera toda la información necesaria para modelar el acceso.

No obstante, el uso de TLM no define un único nivel de abstracción para el modelado de comunicaciones. En función de la precisión de los tiempos y la complejidad de los modelados de protocolos y colisiones el balance entre la velocidad y la precisión de los modelos puede ser adaptado a las necesidades del diseño. Modelos de precisión de ciclo y modelos de tiempo aproximado pueden ser creados sobre el paradigma TLM.

En [Ke07] se presenta un modelo de bus AMBA con precisión de ciclo de bus. Usando interfaces TLM sobre SystemC se demuestra el aumento de velocidad de estos modelos respecto de descripciones RTL. En [Caldari03] se presenta un modelo similar utilizando el lenguaje System-Verilog. En [Chung07] se presenta un modelo más complejo capaz de modelar diferentes protocolos de comunicación en base a diagramas de estados.

Sin embargo, en algunos trabajos se considera que los modelos de bus con precisión de ciclo pueden ser demasiado lentos para las primeras etapas del proceso de diseño,

especialmente para la exploración del espacio de diseño. Por esa razón se proponen modelos a un nivel de abstracción superior. Estos modelos se basan en modelos de carga. Cuando se va a transferir agrupaciones de datos, como paquetes Ethernet, transferencias de DMA o líneas de cache, en lugar de modelar la transferencia de cada palabra por el bus mediante un acceso, se realiza un único acceso con todos los datos indicando el tamaño de la transferencia. Para realizar el modelado temporal de estas transacciones, se ha de asociar un tiempo a la transacción completa. El modelado de grupos de transacciones simples como una unidad hace que se pierdan determinados detalles, con lo que la precisión de los modelos es un poco inferior a los modelos de ciclo. Por ello estos modelos se llaman de tiempo aproximado.

En [Schirner06] se propone una infraestructura de modelado abstracto de buses a diversos niveles de abstracción: a nivel de pin, de ciclo de bus y tiempo aproximado. Este último modelo es propuesto para largas exploraciones, sin embargo, sobrecargas de protocolo, arbitrio y colisiones no son modelados. Como consecuencia la precisión del modelo es limitada.

GreenSocs [GreenSocs] es un proyecto abierto para desarrollo de modelos en SystemC. Para modelo de plataformas se proporciona una infraestructura de modelado de bis denominada GreenBus. GreenBus proporciona modelos a niveles RTL, con precisión de ciclo y tiempo aproximado. Sin embargo, al igual que en el caso anterior, el modelo de alto nivel no modela con detalle efectos de funcionamiento del sistema.

Una solución propuesta para modelado rápido de comunicaciones considerando colisiones y árbitros es aplicar soluciones de teoría de colas. Aplicando todo el trabajo realizado en teoría de colas [Kelly75, Bose02, Dowdy04] se puede mejorar la precisión del modelo de bus. Sin embargo, la utilización de soluciones estadísticas es más adecuada para su aplicación en técnicas de estimaciones estáticas que dinámicas, ya que los efectos puntuales no pueden ser estimados, reduciendo la precisión de la simulación.

4.6 Modelado de consumos

Aunque la estimación y el modelado temporal son críticos en el modelado de sistemas MpSoC objeto de la tesis, otros factores no funcionales también son muy importantes en el proceso de diseño. De estos factores, el consumo es quizá el más importante. Por ello se han propuesto múltiples técnicas de estimación desde nivel de puerta lógica hasta estimaciones desde código fuente.

Las técnicas tradicionales de estimación de consumo se han realizado a bajo nivel, RTL o puerta lógica [Abrar04]. Estas estimaciones son muy precisas pero requieren unos tiempos de simulación extremadamente altos, y no son validos para las primeras etapas del proceso de diseño.

Para aumentar la velocidad de simulación se han propuesto nuevos modelos basados en TLM. La herramienta Sleep [Soulard07] propone una aproximación híbrida de modelado de potencia TLM basado en modelos de bajo nivel. Los modelos se usan para caracterizar

cada componente del sistema obteniendo métricas de potencia y energía. A partir de esta información se usa un modelo SystemC TLM de precisión de ciclo para obtener la estimación de consumo. Otras técnicas de estimación de consumo similares basada en SystemC TLM con simulaciones de precisión de ciclo se proponen por ejemplo en [Dhanwada05].

Sistemas de modelado de potencia basado en máquinas de estados se propone en [Bergamaschi03]. Maquinas de estados autónomas se utilizan como modelos separados de la descripción funcional. Los modelos se basan en descripciones TLM. La información de consumo utilizada es obtenida de las hojas de características de cada componente.

Aunque los modelos de precisión de ciclo son más rápidos que los modelos RTL o a nivel de puerta lógica, su velocidad no es suficiente para realizar largas exploraciones eficientemente. Por ello se han propuesto técnicas que aplican soluciones de más alto nivel al modelado de potencia.

En [Shafi03] se realizan caracterizaciones del consumo de cada instrucción del conjunto de instrucciones del procesador. Utilizando un simulador de conjunto de instrucciones se realiza una simulación basada en código binario. Conforme se van ejecutando las instrucciones ensamblador se va estimando el consumo en función de las instrucciones y el estado de la cache.

En [Sinha01] se realiza un estudio en el que se demuestra que obteniendo un consumo medio por instrucción ensamblador se puede obtener una estimación muy rápida del consumo de una aplicación con precisión suficiente para las primeras etapas del proceso de diseño.

En esta tesis se aplicará este principio para obtener estimaciones de consumo del código SW añadiendo a las técnicas de estimación de tiempo estimaciones de potencia. La gran velocidad de simulación conseguida permitirá la exploración de amplios espacios de diseño teniendo en cuenta el consumo del sistema.

4.7 Limitaciones en el estado del arte

Tras analizar el estado del arte, se puede concluir que, aunque las técnicas de simulación y estimación de rendimiento de sistemas embebidos han tenido un gran desarrollo, la mayoría de las técnicas propuestas no son validas para realizar largas exploraciones del espacio de diseño ni previsiones rápidas en los primeros pasos del proceso de diseño. Las técnicas de simulación basadas en modelos RTL, o de precisión de ciclo no tienen velocidad suficiente para estudiar los largos espacios de diseño disponibles al principio del proceso de diseño de un sistema MpSoC.

Las nuevas técnicas surgidas con objeto de obtener mayores velocidades de simulación, basadas en simulación nativa y tiempos aproximados, presentan limitaciones en cuanto a modelado de sistemas operativos y modelado de plataforma. Además las estimaciones de tiempo de ejecución y consumo aún requieren de grandes esfuerzos para

considerar elementos como el modelado de caches o gestionar optimizaciones y poder proporcionar simulaciones precisas.

Además, las técnicas propuestas no son fácilmente integrables en flujos de exploración junto con herramientas de selección de exploraciones y análisis de los espacios de diseño. No hay herramientas que integren simulaciones nativas, modelos de OS y modelos de plataforma de tiempo aproximado de forma adecuada para su uso en exploración. Para una exploración eficiente es necesario que el simulador sea capaz de aceptar descripciones de sistemas explorables, en los que las herramientas de análisis puedan introducir fácilmente las modificaciones a explorar. Configurar dinámicamente los componentes del sistema, gestionar las tareas SW, o Añadir, mover y quitar componentes del modelo de sistema, son tareas muy complejas de realizar en los sistemas de MpSoC existentes en la realidad.

En los próximos capítulos de este documento se presentarán las soluciones desarrolladas durante este trabajo, con objeto de resolver estas limitaciones.

SECCIÓN III: APORTACIONES

"La función intelectual de las dificultades es la de conducir a hombres y mujeres a pensar: cualquier nuevo avance científico ha salido de una nueva audacia de la imaginación."

John Deway

5 Infraestructura de simulación para estimación de prestaciones

El trabajo de investigación realizado durante el desarrollo de la presente tesis ha tenido por objeto la búsqueda de nuevas soluciones para la simulación de sistemas electrónicos en alto nivel. Sin embargo, estas soluciones no se han mantenido solamente a nivel teórico, como pruebas de concepto, sino que se han aplicado estas soluciones de manera práctica. Como resultado se ha desarrollado una herramienta de co-simulación HW/SW capaz de obtener simulaciones y estimaciones de rendimiento del sistema en diseño en alto nivel de manera rápida y suficientemente precisa para ser utilizado en las primeras etapas del proceso de diseño. Esta herramienta se ha denominado SCoPE: "System Co-simulation and Performance Estimation in SystemC".

Para presentar el trabajo se comenzará con una visión general de la herramienta, con objeto de describir todos los elementos estudiados y sus interrelaciones. Esto permitirá mostrar una visión de conjunto a partid de la cual exponer los detalles de cada elemento individual. Así, en los capítulos siguientes (6 al 9), se irán presentando los distintos ámbitos de modelado cubiertos: modelado del SW de aplicación y sistema operativo, modelado de la plataforma HW, interconexión y generación automática de los modelos de sistema, e integración en el proceso de exploración del espacio de diseño.

5.1 Objetivos de SCoPE

El objetivo de SCoPE es ser una herramienta capaz de ayudar al diseñador durante las primeras etapas de refinado de sistemas embebidos. La herramienta ha de ser utilizada una vez que la funcionalidad del sistema ha sido decidida y se han desarrollado los algoritmos que han de resolver las tareas requeridas. En ese momento ha de empezar el refinado del sistema.

Con objeto de que SCoPE sea una herramienta lo más útil posible en ese punto del diseño, la herramienta proporciona una serie de servicios con objeto de cubrir una serie de objetivos. En general, los objetivos principales a cubrir por la herramienta son los siguientes:

- Obtener estimaciones de rendimiento del sistema
- Proporcionar información para la realización del proceso de diseño
- Servir de plataforma para desarrollo de componentes HW/SW

- Servir como simulador para herramientas de exploración del espacio de diseño

El primer objetivo es la obtención de estimaciones de rendimiento, dado que el conocimiento de dichas métricas al principio del proceso de diseño puede resultar de gran importancia. Algoritmos repletos de operaciones con números en punto flotante de 64 bit, raíces cuadradas y gigabytes de datos almacenados no presentan mayor inconveniente cuando se evalúan en programas como Matlab. Sin embargo, una plataforma física razonable no puede soportar todos estos requerimientos de forma eficiente. Por esta razón, una de las primeras acciones a desarrollar durante el proceso de diseño es evaluar si la especificación teórica proporcionar al diseñador información sobre el rendimiento del sistema en términos de tiempo de ejecución y consumo para que sea posible verificar el cumplimiento de las restricciones de diseño.

Esta verificación debe realizarse de dos formas. En primer lugar SCoPE retorna las métricas de rendimiento total de sistema al final de la simulación para verificar el funcionamiento global del sistema. Esto permite realizar una verificación de "caja negra", en la que el diseñador ejecuta varias simulaciones del sistema conforme a determinados casos de uso y verifica que los resultados son válidos.

En segundo lugar, SCoPE permite la verificación del funcionamiento del sistema y del chequeo de las restricciones especificadas mediante aserciones. Para ello el diseñador ha de introducir en el código instrucciones "assert(x)" en las que se comprueben dichas restricciones. SCoPE ofrece al diseñador funciones para obtener el tiempo de ejecución y el consumo en momentos puntales del código con las que realizar las comprobaciones correspondientes. De esta forma es posible chequear tiempos de propagación entre entradas y salidas, tiempos de respuesta o ratios de generación de datos entre otros.

SCoPE tiene como segundo objetivo guiar al diseñador en el proceso de diseño del sistema. El proceso de co-diseño de un sistema comienza con la decisión de la arquitectura se ha de implementar, qué componentes han de ser mapeados en HW y cuales en SW, y a qué recurso será asociado cada uno.

Para realizar estas tareas, el procedimiento tradicional se basa en confiar en la experiencia del diseñador. Sin embargo, un mecanismo basado únicamente en la experiencia tiene pocas posibilidades de resultar óptimo. Con objeto de facilitar estas tareas, SCoPE proporciona al diseñador información del rendimiento de cada uno de los componentes del sistema. Tiempos de ejecución de tareas, utilización de las CPUs, cantidad de fallos de cache, sobrecarga de los canales de comunicación o potencia de consumida por los componentes HW son algunas de las informaciones que un diseñador puede obtener de SCoPE. Con ellas el diseñador puede evaluar los efectos de las decisiones de diseño tomadas, y optimizar los resultados.

Este mecanismo de optimización se basa en un procedimiento de múltiples evaluaciónes, en el que el diseñador lleva a cabo un blucle en el que repetidamente propone una configuración, la simulación obtiene el rendimiento esperable, y el diseñador analiza los

datos para proponer una configuración mejo. Para que este procedimiento sea viable, SCoPE tiene por objetivo obtener las estimaciones de la forma más rápida posible, manteniendo una precisión adecuada para realizar estas evaluaciones iniciales.

El tercer objetivo general de SCoPE es el de servir de plataforma sobre la que comenzar el desarrollo de los componentes HW y SW del sistema. En los flujos tradicionales, el desarrollo del sistema comienza con el diseño de los componentes HW del sistema. Una vez que estos han sido realizados, se construye una plataforma, habitualmente un prototipo. A continuación se diseñan los componentes SW sobre el prototipo HW.

Esta secuencialidad se debe a que en sistemas embebidos la interacción HW/SW es muy alta. Por una parte, el desarrollo del SW es altamente dependiente del comportamiento temporal del sistema. Por otra, la comprobación de la funcionalidad SW es muy complicada si no se integra la funcionalidad HW y la infraestructura SW en la simulación. Además, la verificación de sistemas sobre ISSs resulta complicada y tediosa, por lo que se necesita un mecanismo de ejecución más rápido, como un prototipo.

SCoPE proporciona una infraestructura para modelado HW/SW donde el diseño de HW y SW pueda realizarse en a la vez. Para ello es necesario generar una simulación temporal del sistema, de tal forma que se puedan integrar en el SW esperas, alarmas, temporizadores y otros elementos que garanticen el cumplimiento de requisitos del sistema como ratios de entrada y salida o calidad de servicio. SCoPE permite también el modelado de la infraestructura SW (sistema operativo y “middleware”), así como la integración de componentes HW en el modelo de plataforma. Todo esto se realiza con velocidades de simulación cercanas a las obtenidas mediante la ejecución funcional del código, de forma que un prototipo no es inicialmente requerido. Aunque este prototipo sigue siendo necesario en las últimas etapas de verificación, ya que la velocidad de simulación se consigue mediante simplificaciones que reducen la precisión del modelo, el uso de SCoPE puede acelerar enormemente el proceso de diseño.

Además, la posibilidad de depurar el sistema sobre una simulación C++ antes de pasar a utilizar una plataforma física puede resultar extremadamente útil. El acceso a la información interna del sistema puede resultar complicado cuando se depura sobre una plataforma física. Como consecuencia, la detección y corrección de errores requiere de grandes esfuerzos por parte de los diseñadores. Por el contrario, el análisis de las condiciones de error en una simulación C++ es mucho más simple, ya que es más fácil acceder a la información interna para conocer el estado de la ejecución.

Por último, SCoPE tiene como objetivo servir como simulador para procesos de exploración del espacio de diseño (DSE). Las herramientas de DSE analizan el espacio de diseño seleccionando los puntos de consideran más convenientes y comprobando sus características, para poder obtener finalmente grupos de puntos óptimos (sección 1.3.2). SCoPE, como herramienta de estimación de prestaciones de sistemas complejos, puede ser utilizada para obtener las características de cada uno de los puntos seleccionados.

La elección de cada punto a simular es realizada por la herramienta de DSE seleccionada. Sin embargo, estas herramientas no tienen ni inteligencia ni conocimientos en electrónica como para generar los códigos SystemC que describan cada una de las configuraciones elegidas. Por ello SCoPE ha sido extendido con un generador automático de modelos de sistema. Partiendo de descripciones XML del sistema, SCoPE es capaz de generar automáticamente los modelos simulables. De esta forma, la configuración del sistema tan solo requiere de la generación de un archivo con valores numéricos que indican las opciones seleccionadas. Además, con objeto de reducir el tiempo de exploración, SCoPE permite la ejecución de simulaciones paralelas así como la reconfiguración de la plataforma en tiempo de ejecución, evitando tiempos adicionales de compilación.

En resumen, SCoPE proporciona una infraestructura que, ejecutando sobre SystemC, extienda sus funcionalidades con nuevas cualidades que permitan el modelado de sistemas MpSoC en este nuevo nivel de abstracción. Para ello proporciona técnicas de estimación de prestaciones para simulación nativa con las que obtener la información temporal y de consumo necesaria para modelar los sistemas, proporcionar una infraestructura SW completa, con sistema operativo, controladores de dispositivo e infraestructura "middleware", así como modelos de bus, red y las interfaces correspondientes para modelar el comportamiento HW de la plataforma, integrando los módulos HW de aplicación específica y conectándolos con la simulación del código SW.

Más en concreto, SCoPE se ha desarrollado con el objeto de permitir el refinado de los componentes SW de un sistema HW/SW, especialmente aquellos más cercanos a los componentes HW, considerando los efectos funcionales y de rendimiento de la plataforma HW. Es por ello que el trabajo se ha centrando en el modelado SW sobre una plataforma HW.

Esto significa que la mayoría de los análisis se realizan sobre la componente SW de los sistemas, tratando la parte HW como un elemento capaz de alterar críticamente su funcionamiento. La plataforma HW se trata más como una parte de la infraestructura para el análisis del SW que como algo a analizar en detalle. No obstante también se han realizado exploraciones iniciales para aplicar algunas de las técnicas de estimación SW a los componentes HW, aunque su desarrollo en profundidad se propone como trabajo futuro.

5.2 Modelado de sistemas electrónicos con SCoPE

Para reducir los altos tiempos de simulación requeridos por los simuladores de código binario (ISSs), el trabajo realizado durante la tesis se ha centrado en el modelado de sistemas electrónicos mediante co-simulación nativa. La co-simulación nativa se basa en ejecutar el código SW directamente en una máquina nativa, sin necesidad de compilación cruzada, juntamente con descripciones HW. Como consecuencia, se obtienen simulaciones rápidas de sistemas complejos. Para conseguir esa velocidad, los elementos que componen el modelo de plataforma poseen un grado de detalle limitado.

La técnica es capaz de proporcionar información temporal sobre la ejecución del sistema con una razonable precisión, teniendo en cuenta los canales físicos de comunicación y

los componentes de la plataforma SW. Permite ejecuciones rápidas y con la precisión suficiente para servir de base a las primeras etapas de exploración y evaluación del diseño. En este nivel, las especificaciones de alto nivel empiezan a transformarse en descripciones HW/SW, teniendo en cuenta información temporal, soporte de sistema operativo, comunicaciones HW/SW, efectos producidos por las contenciones en el uso de determinados recursos, etc.

El balance entre velocidad y precisión se consigue utilizando simulaciones de tiempo aproximado. Para realizar simulaciones adecuadas, es necesario que todos los elementos del modelo tengan una precisión y un grado de detalle similar. La utilización de un modelo muy preciso de un componente del sistema en una simulación de tiempo aproximado tiende a provocar un aumento mayor en el tiempo de simulación que en la precisión de la simulación. Debido a la alta interrelación de todos los componentes de un sistema embebido, los pequeños errores derivados del modelado simplificado de cada componente influyen necesariamente en el modelado de los demás reduciendo su posible precisión inicial. Por contra, un modelado muy detallado de un componente provoca una sobrecarga en simulación sin un aumento de la precisión global debido a los errores introducidos por el resto de componentes, lo que resulta en un gran lastre para la eficiencia de la simulación.

Para modelar y de forma eficiente los modelos, SCoPE trata de manera distinta los componentes HW y SW. Las diferencias en el tratamiento de los componentes se basan en las diferentes características que, en general, tienen las descripciones de los componentes HW y SW. Por una parte, mientras que los modelos de componentes SW se reciben sin ningún tipo de información temporal o de consumo, de forma que es la herramienta la encargada de realizar todo el análisis, los componentes HW deben ser provistos con toda esta información. Esto significa que los tiempos de retardo de todos los componentes de la plataforma deben estar incluidos en los propios modelos.

Por otra parte, también los mecanismos de modelado de bajo nivel correspondientes son muy distintos. La plataforma HW del sistema se modela desde un enfoque concurrente: todos los componentes HW actúan simultáneamente. Sin embargo, el subsistema SW se modela de manera secuencial: un ISS ejecuta una instrucción tras otra. En consecuencia, los componentes HW son mucho más dependientes del núcleo de simulación de SystemC, mientras que los elementos SW pueden ser más fácilmente independizados del mismo. Esto ha sido aprovechado con objeto de optimizar la velocidad de simulación de los sistemas.

El modelado de los componentes SW se realiza en dos etapas. En primer lugar se analizan los componentes en sí mismos, y a continuación se integran en el resto del sistema. Para realizar estas estimaciones la herramienta creada utiliza cierta información sobre las características de la plataforma que ejecuta los componentes del sistema.

Para realizar una simulación realista del sistema, es necesario transformar los componentes SW, inicialmente carentes de información temporal, en modelos temporales. En estos modelos se integra la ejecución funcional propia del código con otra información no funcional. Para cada segmento de código se añade al código fuente información de cuanto

tiempo requiere en un determinado procesador y cuanta potencia va a consumir. Esto significa que la simulación del código SW, inicialmente atemporal, se transforma en una simulación temporal.

Para estimar los tiempos a anotar en el código SW de aplicación se requiere de información del procesador sobre el que van a ejecutar. Es el procesador y los elementos asociados, como las caches, los principales responsables de que de un determinado código se obtenga un cierto rendimiento. Estos definen la velocidad de ejecución, el número de instrucciones ejecutadas o el número de accesos a memoria.

También es necesario modelar el resto de componentes de la plataforma SW: el sistema operativo y el "middleware" utilizados. La definición de políticas y prioridades en las tareas o el uso de los mecanismos de comunicación y sincronización más adecuados son unas de las tareas más importantes en un proceso de refinado. Sin embargo todas ellas dependen de la existencia de una infraestructura que aporte los servicios necesarios para su utilización y verificación. Adicionalmente, la disponibilidad de una infraestructura SW adecuada posibilita el refinado completo del SW de aplicación, incluyendo los servicios y llamadas al sistema que se utilizarán en la realidad.

Otro elemento de vital importancia en el proceso de co-diseño de sistemas complejos es la creación de las interfaces HW/SW. Para poder analizar este punto debemos tener en cuenta que el proceso de diseño propuesto anteriormente parte de la definición de una especificación conjunta que posteriormente se divide en componentes HW y SW. A nivel de especificación todos los componentes del sistema se comunican entre sí utilizando canales de comunicación de alto nivel. Sin embargo, a la hora de implementar el diseño, es necesario sustituir estos canales por soluciones equivalentes que puedan ser utilizadas en la plataforma destino. Para comunicaciones HW/HW la conectividad a través de señales punto a punto o buses puede ser fácilmente implementada. En comunicaciones SW/SW, las facilidades de comunicación del sistema operativo o el "middleware" pueden resolver este problema.

Sin embargo, no hay una forma sencilla y universal de comunicar los componentes HW y SW entre sí. Dado que ambos tipos de componentes tienen formas de funcionamiento heterogéneas, la sustitución de los canales de comunicación no significa únicamente la transferencia de datos o señales de control, sino la conexión de dos entornos completamente distintos. Es por ello que se necesitan mecanismos complejos.

El modelo de plataforma HW necesario para conectar las aplicaciones SW y los controladores de dispositivo con los componentes HW puede realizarse de varias maneras. Teniendo en cuenta el nivel de abstracción en el pretende enclavarse SCoPE, dos soluciones han sido consideradas. La primera opción es realizar conexiones directas, punto a punto, entre componentes SW y HW. Esto permite una gran velocidad de simulación y gran simplicidad. Su implementación es muy cercana a la especificación y por tanto la generación de estos modelos es sencilla. Sin embargo esta aproximación está bastante alejada de la implementación final, con la consiguiente pérdida de precisión.

La segunda opción es generar un modelo completo integrando todos los componentes implicados en el proceso de comunicación, tanto los componentes de la plataforma HW como del sub-sistema SW. Los componentes SW deben gestionar el acceso a los periféricos realizando los accesos adecuados a través del modelo de bus, así como gestionar las interrupciones. Como consecuencia, la integración de controladores de dispositivo, interrupciones, interfaces de comunicación de bajo nivel o el análisis de la influencia de retardo en buses, colisiones o el uso de DMAs deben realizarse para obtener una simulación realista en SCoPE.

Recordemos que en el proceso de diseño, SCoPE se encuentra en un punto intermedio entre la especificación y el modelado basado en ISS. Esto significa que si el esfuerzo de adaptación no se realiza durante la etapa de diseño deberá realizarse durante la etapa siguiente. El desarrollo completo de los elementos de conexión HW/SW sobre modelos basados en ISS, esto es, sobre código binario, puede resultar complejo. La mejor solución es disponer de herramientas que permitan realizar parte del esfuerzo de integración de las interfaces HW/SW antes de pasar a código binario. SCoPE es una herramienta adecuada a tal efecto. Esta aproximación además permite que el modelado de la influencia de la plataforma HW en la simulación SW sea lo más realista posible.

Por otra parte, para realizar este modelado completo de las comunicaciones HW/SW es necesario hacer uso de parte de la infraestructura HW, como buses, DMAs o módulos de memoria, y de parte de la infraestructura SW, como funciones del HAL (Hardware Abstraction Layer) del sistema operativo, controladores de dispositivo, “middleware” específico, etc. Es por eso que el modelado de la plataforma SW no se puede limitar tan solo a la implementación de la API de usuario (en nuestro caso POSIX), sino que el modelo del sistema operativo debe integrar la funcionalidad necesaria para que los controladores de dispositivo y el resto de facilidades necesarias para realizar las comunicaciones HW/SW puedan ser modelados junto con el resto del sistema.

Para poder comunicar los controladores de dispositivo con la plataforma HW, estos hacen uso de las funciones de bajo nivel del modelo del sistema operativo (HAL). Estas funciones son las encargadas de escribir y leer en las posiciones de memoria del bus, o manejar las interrupciones recibidas lanzando los manejadores correspondientes. Sin embargo, estas funciones siguen siendo parte de la plataforma SW. Para comunicarlas con el HW es necesario un elemento que haga las veces de modelo HW de la interfaz física del procesador. Este modelo gestiona las peticiones de las funciones del HAL del sistema operativo generando las llamadas al bus adecuadas. Para ello el modelo usa los mecanismos de comunicación TLM descritos en el capítulo 3 sobre los que se basa el modelo completo de la plataforma HW.

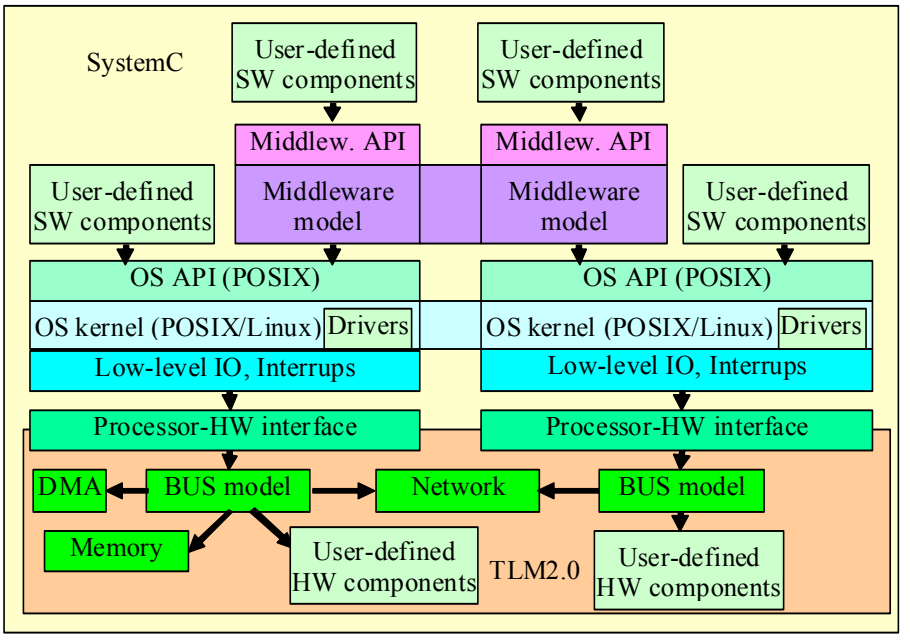


Figure 5-1: Plataforma SW propuesta

Para que estas peticiones lleguen adecuadamente a los componentes HW diseñados por el usuario es necesario modelar los componentes intermedios. El bus es el elemento principal para este propósito. Por ello se ha integrado en SCoPE un modelo abstracto de bus descrito a alto nivel aunque con cierto modelado de retardos y colisiones. También son necesarios otros componentes estándar como memorias, DMAs, etc. que han sido generados e integrados en SCoPE. Además, para el modelado de sistemas multiprocesador más complejos se han integrado modelos de redes de comunicaciones con sus correspondientes interfaces de red necesarias para conectar los nodos basados en bus al modelo general de red.

Por último se debe facilitar la integración de los componentes HW a dicha infraestructura de modelado de plataformas. A tal efecto se han generado una serie de interfaces genéricas que permiten una fácil conexión de los componentes HW generados por el usuario al bus del sistema.

5.2.1 Arquitectura de plataforma soportada

SCoPE es una herramienta diseñada para que el diseñador pueda modelar su sistema embebido. Sin embargo, la cantidad de arquitecturas que pueden encontrarse en sistemas MpSoC es extremadamente alta. Por esta razón no es posible realizar una herramienta precisa capaz de manejar cualquier sistema imaginable. En su lugar, SCoPE ha sido diseñado para modelar las arquitecturas más habituales, permitiendo la mayor flexibilidad posible.

El modelo de arquitectura manejado por SCoPE combina sistemas multi-computador y multi-procesador, utilizando tanto buses como redes. En este modelo de arquitectura se permite un número variable nodos de cómputo independientes (arquitectura multi-

computador) incluyendo en cada nodo una cantidad variable de procesadores (arquitectura de multi-procesador simétrico).

Esto optimiza la capacidad de explorar las diversas soluciones de balanceo entre la carga computacional y los requisitos de área y consumo de las comunicaciones en el chip. Esta solución, de múltiples entornos de procesadores simétricos, es por tanto una solución que nos permite modelar múltiples posibles arquitecturas. En este modelo arquitectural, las comunicaciones entre nodos se realizan por medio de una red, mientras que las comunicaciones internas a los nodos se simulan mediante un modelo de bus.

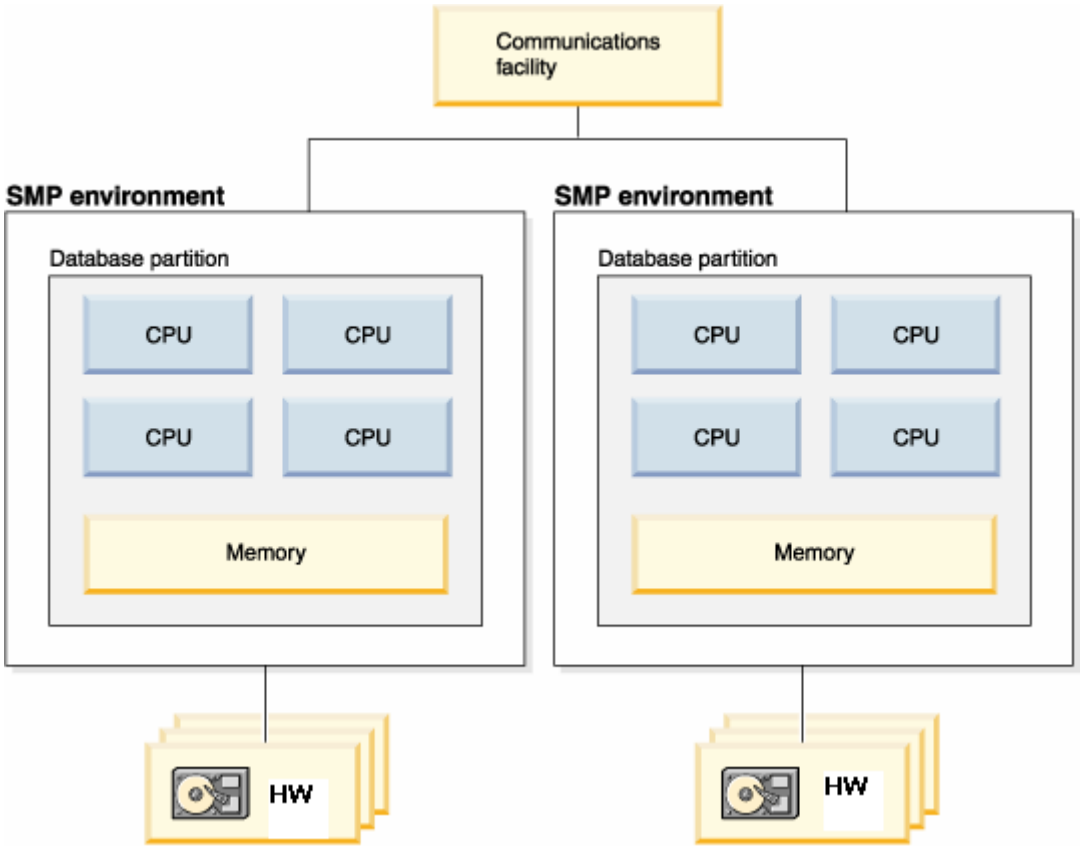


Figura 5-3: Modelo de genérico de arquitectura soportado

Con esta solución, cada nodo tiene múltiples procesadores con una memoria y un sistema operativo común. Esto significa que la carga computacional puede redistribuirse entre los procesadores del nodo sin problemas. El sistema operativo tiene capacidad para mapear las tareas a los procesadores del nodo tanto estática como dinámicamente. Aunque sigue sin poder moverse tareas entre nodos distintos, la posibilidad de gestionar las tareas para utilizar óptimamente la capacidad computacional eliminando la existencia de procesadores vacíos es suficientemente alta si las tareas han sido adecuadamente distribuidas entre los nodos.

SCoPE contiene los modelos de procesador, bus y red necesarios para construir modelos de cualquiera de las tres arquitecturas propuestas anteriormente. Además es posible adaptar el modelo de sistema operativo para gestionar los entornos de forma que las tareas puedan moverse entre los procesadores que formen un grupo simétrico, si se desea. La principal limitación actual es la ausencia de un sistema de modelado del manteniminento de coherencia en cache. Debido al mecanismo de simulación de SCoPE (toda la simulación es un único proceso en el computador nativo), el correcto valor de los datos está siempre asegurado, sin embargo los efectos en la estimación de rendimiento no son considerados. Este trabajo se propone como una posible extensión futura.

5.3 Nivel de abstracción

Para realizar un modelado óptimo de sistemas MpSoC en alto nivel, uno de los puntos más importantes es equilibrar la relación precisión / velocidad de todos los componentes del sistema. No es adecuado obtener resultados muy precisos de algunos componentes, requiriendo un algo esfuerzo de simulación, si el resultado depende de otros componentes cuyo modelado implica una gran fuente de error. Es por ello que una definición adecuada del nivel de abstracción que deben tener todos los componentes del modelo, es un punto crítico en la generación de un entorno de modelado eficiente.

SCoPE es una herramienta orientada al modelado de sistemas tanto funcional como de prestaciones para ser utilizado en las primeras etapas de diseño, donde se realiza el paso de la especificación inicial y las descripciones algorítmicas a las primeras soluciones de implementación. En estas etapas hay que tener en cuenta que se trabaja con códigos aún no definitivos, por lo que no tiene sentido realizar análisis exactos de descripciones que durante el proceso de diseño aún han de variar significativamente. Es por esto que obtener una precisión extrema no es uno de sus objetivos.

El objetivo no debe ser realizar mejoras como optimizar el número de ciclos de reloj de respuesta a un evento, sino decidir la mejor arquitectura para el sistema, decidir los componentes más adecuados, como procesadores o memorias, decidir la partición HW-SW o el mapeo de tareas a recursos de la forma más eficaz posible. La rapidez de simulación es por tanto realmente el punto crítico en este nivel del flujo de diseño. Por ello, errores del 10-20% en las estimaciones son perfectamente razonables. Trabajar con esos márgenes de error permite eliminar el modelado de multitud de detalles de grano fino, con lo que la simulación se hace bastante más sencilla y, en consecuencia, rápida.

Por esta razón se requiere un nivel de abstracción que permita obtener estimaciones adecuadas reduciendo en todo lo posible los elementos superfluos del modelado. Con este objetivo SCoPE se ha emplazado dentro de la filosofía TLM, como un modelo PVT utilizando elementos descriptos con tiempos aproximados. Para explicar un poco más esta decisión analicemos el modelo del sistema por partes.

En primer lugar la gestión de transferencias como una única entidad (TLM) en lugar de modificaciones de múltiples señales (RTL) implica un gran salgo en velocidad. Un

ejemplo paradigmático es una transferencia por un bus. En un modelo RTL es necesario definir señales para pedir una transferencia, para indicar que el bus esta ocupado, el tamaño de palabra utilizado, la dirección destino, el dato, el tipo de ráfaga, o esperar la confirmación de que la operación se ha realizado con éxito. Esto implica una gran cantidad de operaciones, más aún si consideramos lenguajes como VHDL donde el manejo de tal número de señales acaba produciendo una ejecución de gran cantidad de ciclos delta por cada ciclo de reloj, implicando re-ejecución de código y gran número de cambios de proceso.

En cambio, el modelado TLM propone que el acceso al bus pueda ser modelado simplemente como "is_ok = write (data, address);". Esto implica que una única operación, una llamada a función, es suficiente para completar la transferencia, con lo que la velocidad de simulación es enormemente mejorada. Además, si sabemos que el acceso al bus tarda, por ejemplo, tres ciclos de media, es suficiente con hacer que el "write" realice una parada de tres ciclos antes de continuar con la simulación para obtener un modelado que mantenga la precisión en tiempo.

Además, esta solución permite agrupar las transferencias en bloques que se envían de mediante un único acceso. Por ejemplo, para transferir un paquete Ethernet de 1 kbyte de la memoria a la tarjeta de red se necesitan múltiples accesos al bus (en un bus de 32 bits son 256 accesos). Sin embargo, podemos evitar realizar los 256 accesos en el modelo, requiriendo uno solo. Si realizamos una transferencia del tipo "is_ok = write (buffer, size, address)" y enviamos el Kbyte en una sola llamada podemos reducir enormemente el tiempo de simulación. Además, si sabemos que cada acceso requiere 3 ciclos podemos realizar una única espera de 768 ciclos. Esto reduce el número de cambios de proceso en el núcleo de simulación de SystemC ya que el planificador no ha de activar el hilo 256 veces, sino una sola, incrementando aún más la ganancia en velocidad de simulación.

Estas técnicas de optimización deben ser aplicadas también en el modelado interno de los componentes del sistema, no solo en sus comunicaciones. Por ejemplo, un modelo RTL de un procesador es extremadamente complejo. Una primera evolución es utilizar un simulador del conjunto de instrucciones (ISS) que vaya leyendo cada instrucción binaria y ejecutando las operaciones requeridas sobre los registros del procesador o el exterior. De esta forma, si sabemos cuantos ciclos requiere cada operación binaria podemos hacer que cada vez que se ejecuta una instrucción se espere el número de ciclos correspondiente y así mantener la información temporal en la simulación.

Esta solución puede ser llevada aún más lejos. El código binario se genera mediante un compilador que sustituye las operaciones C por sus instrucciones máquina equivalentes. Si sabemos cuantas operaciones binarias requiere cada operador C podemos ejecutar el código C y tras cada operador anotar el tiempo que requiere. De esta forma ni si quiera necesitamos un modelo de procesador con su banco de registros y sus componentes internos. Podemos ejecutar el código fuente junto con la información temporal directamente, con el gran avance en velocidad de simulación correspondiente.

Aún más. El código C se ejecuta secuencialmente en el procesador hasta que se hace una llamada al sistema o es expulsado por efecto de una interrupción. Por tanto, si en lugar de esperar los ciclos correspondientes tras cada operación los vamos acumulando y los esperamos de una sola vez cuando la tarea abandona el procesador los cambios de proceso en la simulación son reducidos enormemente, con lo que la simulación se acelera definitivamente.

Hay que tener en cuenta que en estas simplificaciones siempre se pierde un cierto grado de detalle. Por ejemplo, si en lugar de utilizar un modelo RTL utilizamos un ISS a nivel de instrucción, no podremos modelar las paradas en el pipe como consecuencia de dependencia de datos entre instrucciones. Sin embargo, estos efectos puntuales se producen relativamente poco durante la ejecución con lo que el efecto sobre el resultado final es bastante reducido. Además, sobre ejecuciones suficientemente largas (simular un segundo de plataforma real con un procesador a 1GHz implica ejecutar 10^9 operaciones binarias), se puede aplicar un factor de corrección estándar que puede limitar el error a un mínimo porcentaje. Así pues podemos alcanzar el grado de error 10-20% propuesto anteriormente.

Resumiendo, SCoPE basa su ejecución en la utilización de modelos de alto nivel, obteniendo una relación directa entre el tiempo que requiere cada operación de alto nivel (o macro-operación) y el número y coste de las operaciones simples que esa macro-operación implica en el sistema real.

5.4 Elementos modelados y arquitectura interna de SCoPE

Los elementos integrados en SCoPE se pueden dividir en dos grandes grupos considerando la parte de sistema que deben modelar: modelado de la plataforma HW y modelado de la plataforma SW. Estos elementos, que son aquí presentados con objeto de obtener una primera visión de conjunto de SCoPE serán desarrollados en profundidad en los siguientes capítulos de la tesis.

Todos los elementos de SCoPE están diseñados para simular sobre una infraestructura SystemC. Los componentes de SCoPE se ejecutan sobre SystemC haciendo uso de las facilidades aportadas por la propia librería de simulación y de la librería auxiliar de TLM proporcionada por el mismo organismo que mantiene SystemC (la OSCI). SCoPE extiende las posibilidades de modelado de SystemC sin modificar el mismo.

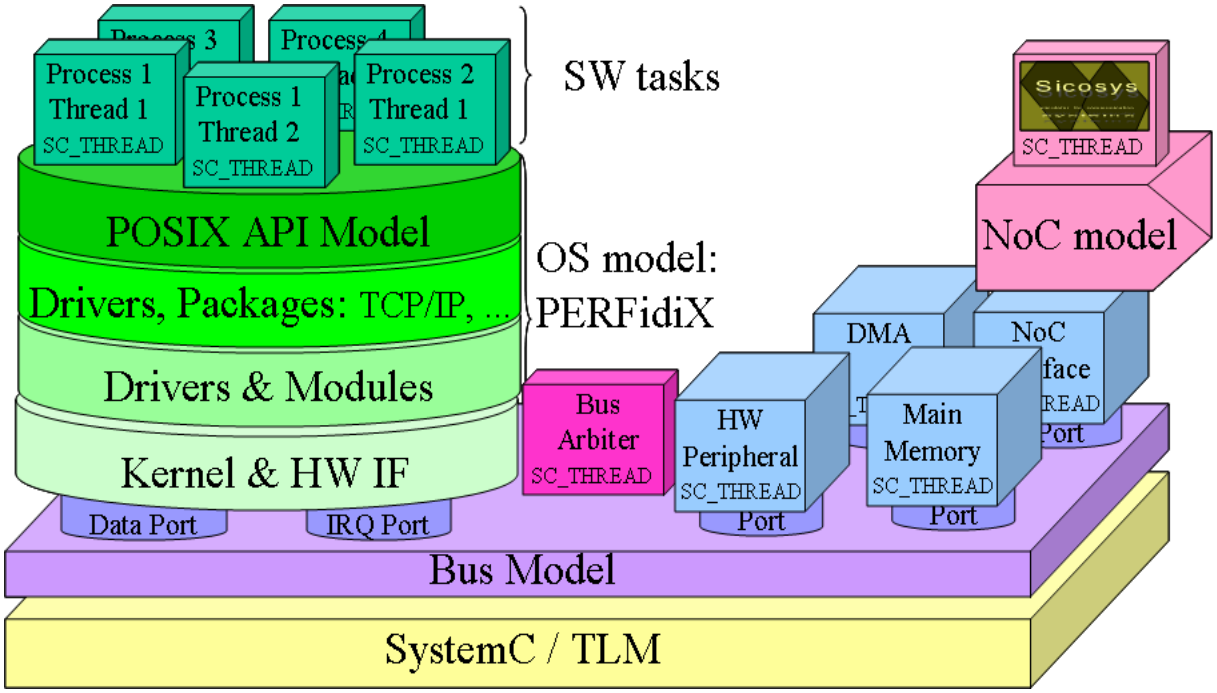


Figura 5-4: Modelo de Nodo genérico modelado en SCoPE

5.4.1 Plataforma HW

La infraestructura HW se compone de dos partes principales: los elementos de comunicación y los propios componentes HW.

5.4.1.1 Comunicaciones

Una de las decisiones de diseño más importantes a explorar al crear un sistema MpSoC es dimensionar los elementos de comunicación y comprobar que las comunicaciones disponibles según una determinada arquitectura son capaces de realizar las transacciones de manera adecuada. Por esa razón en la tesis se han desarrollado modelos de comunicación y se han integrado en la simulación de los sistemas, como parte de SCoPE. Estos modelos son de dos tipos: modelo de bus y modelo de red. Para ambos tipos de elementos se han realizado modelos abstractos con diferentes opciones de configuración para permitir la simulación y evaluación de diversas arquitecturas y configuraciones.

Estos elementos de comunicación reciben todas las transferencias de los diversos componentes del sistema y modelan las transferencias, realizando los análisis requeridos en el nivel de abstracción en el que se mueven el resto de elementos desarrollados. En los modelos de tiempo aproximado utilizados en SCoPE, se considera el modelado de contenciones en los distintos recursos. Esto significa que aunque no se requieren modelos exhaustivos, es necesario un análisis preliminar para comprobar que las peticiones pueden ser abordadas con la capacidad de los recursos previstos para el sistema. En este caso, las comunicaciones dan soporte al resto de componentes del sistema de manera compartida, por lo que un análisis de

anchos de banda requeridos y aportados es necesario para comprobar la viabilidad del sistema.

5.4.1.1.1 Modelo de Bus

El primer elemento de comunicaciones modulado es el bus. Es el elemento más común de comunicación entre el procesador y la memoria, y por extensión con el resto de periféricos. En sistemas sencillos el bus está siempre suficientemente dimensionado, ya que tan solo el procesador y algún DMA (si existe) pueden comenzar transacciones. Sin embargo, en sistemas complejos esto cambia llegando a tener una importancia crítica. Los sistemas MPSoC contienen no uno sino múltiples procesadores y múltiple HW bajo el control de DMAs. Eso significa, que si hay un único bus en el sistema al que están conectados todos los maestros el tráfico se convierte en realmente intenso, llegando incluso a saturar el sistema cuando las colisiones son excesivas.

Es en estos casos donde una correcta elección de la arquitectura del sistema, basada en los requisitos de comunicaciones es necesaria. Poner uno o varios buses, diseñar nodos de cómputo independiente, o incluso sustituir el bus por una red son algunas de las opciones que pueden tomarse cuando el número de procesadores es excesivo para poder compartir el bus. Estas decisiones deben tomarse en función de las estimaciones de requerimientos de ancho de banda de los procesadores y deben probarse mediante las simulaciones adecuadas para comprobar su corrección.

Para poder realizar ambas cosas se ha realizado un modelo abstracto de bus capaz de analizar las comunicaciones. El bus recibe las transacciones de los distintos elementos conectados a él, y teniendo en cuenta el ancho de banda del bus, el tamaño de cada transacción y el ancho de banda máximo requerido por cada transmisor y receptor realizan una simulación temporal aplicando los retardos correspondientes.

5.4.1.1.2 Red

De la misma forma que el análisis del bus es crítico en sistemas MPSoC, cuando su complejidad aumenta lo suficiente para requerir una red en el chip (NoC), también ésta ha de ser modelada. En esta ocasión el modelo puede resultar un tanto más complejo, ya que una red no es solamente unas líneas y un árbitro que gestiona las comunicaciones. En una red hay que modelar los nodos, los “switches”, tener en cuenta los protocolos de comunicación, los algoritmos de enrutado o las acciones para equilibrar el volumen de tráfico en los distintos enlaces de la red.

Todos estos elementos pueden realizarse con modelos de tiempo relajado, aplicando la misma técnica utilizada en el bus. En el modelo de red se modelan los paquetes como entes únicos, y no palabra a palabra o ciclo a ciclo. Sin embargo, la complejidad derivada de considerar todos los componentes que componen la red puede implicar una sobrecarga excesiva en el tiempo de simulación. Por esta razón se han realizado dos modelos de red.

El primer modelo es un modelo extremadamente simple. Basado en una arquitectura "mesh", se modelan las comunicaciones suponiendo condiciones ideales. Esto significa que se proporciona a las comunicaciones un retraso estimado en función de la distancia a recorrer, pero sin tener en cuenta la carga en los puntos de la red o los algoritmos de enrutado. Esto consigue no incrementar el tiempo de simulación, manteniendo la coherencia temporal de la simulación.

Sin embargo, con este modelo no se puede analizar si los anchos de banda de la red o la colocación de los elementos en los nodos son adecuados o pueden producir sobrecargas en puntos determinados de la red. Para resolver este problema se ha recurrido a un simulador de redes preexistente, llamado Sicosys. Con este simulador puede realizarse un modelado más preciso de la red integrando los resultados dinámicamente durante la simulación. Para ello se ha integrado el simulador en SystemC y se ha conectado a través de unas interfaces de red que pueden ser fácilmente conectadas al modelo de red presentado anteriormente.

5.4.1.2 Componentes HW

Aunque la variedad de componentes HW que pueden encontrarse en un sistema electrónico es prácticamente infinito, hay una serie de componentes que se encuentran habitualmente y que pueden modelarse adecuadamente con modelos abstractos permitiendo una cierta configurabilidad. En este sentido, en la tesis se han estudiado las posibilidades de modelado de elementos como GPP (general purpose processors), memorias, DMAs e interfaces de red. Para el resto de elementos que pueda querer integrar el usuario se ha desarrollado una técnica que permite adaptar semi-automáticamente los componentes HW diseñados en SystemC al modelo de bus propuesto con objeto de ser fácilmente integrados en la simulación.

5.4.1.2.1 Procesador (GPP)

El procesador es el elemento encargado de la ejecución del SW. Funcionalmente su resultado se obtiene en SCoPE directamente a partir del código fuente de las aplicaciones que van a ejecutar. Sin embargo, el modelado funcional no es suficiente cuando se intenta modelar el sistema al nivel de tiempo aproximado para análisis de rendimiento y exploración del espacio de diseño. Parámetros no funcionales, como el tiempo de simulación, el consumo o el número de accesos a memoria como efecto de fallos de cache no se pueden obtener directamente a partir de la ejecución de código fuente. Es por ello que en SCoPE se implementa un mecanismo que permite obtener estos parámetros y aplicarlos a la simulación.

En esta tesis se han desarrollado una serie de técnicas que se basan en integrar el modelado del efecto del procesador en la ejecución del código fuente. Para ello se modifica el código fuente inicial incluyendo las anotaciones necesarias para realizar ese modelado. De esta forma, durante la ejecución, la información no funcional es simulada conjuntamente con el propio código de aplicación.

Estas simulaciones presentan las ventajas de que no se necesitan modelos específicos para emular los procesadores ni es necesario transformar el código original en código binario para ser ejecutado sobre un simulador de instrucciones. El código es simulado a un nivel superior, a nivel de código fuente, en lugar de usar código ensamblador. Esta abstracción implica una simulación más rápida, al tener en cuenta menor detalle, pero por el contrario el modelado es menos preciso. En consecuencia, el mecanismo es muy adecuado para una simulación de tiempo aproximado.

A parte de este modelado, se ha desarrollado un componente adicional necesario para conectar el modelado SW con el resto del sistema HW. Este componente gestiona las transferencias requeridas por el procesador y las envía al modelo de bus. Fallos de cache o accesos a periféricos HW acceden a esta interfaz que se encarga de entregar las peticiones al modelo de bus y gestionar las respuestas. Así mismo, esta interfaz recibe las interrupciones HW y se las entrega al control de interrupciones del modelo de sistema operativo para su manejo.

5.4.1.2.2 Jerarquía de memoria

Uno de los elementos que tienen más impacto en el rendimiento del código SW es la jerarquía de memoria. La mayor parte del código binario se centra en dos grupos de instrucciones, las que se ejecutan dentro del propio procesador, y las que ejecutan accesos a memoria principal. Es por ello, que el segundo elemento de mayor importancia al modelar la ejecución de código SW después del procesador es el sistema de memoria.

Una jerarquía de memoria convencional se compone principalmente de dos tipos de elementos, las memorias caches y la memoria principal. En esta tesis se han tratado ambos tipos, aunque de diferente forma.

Las caches del sistema son la parte de la jerarquía de memoria que tiene contacto directo con el procesador. Cada vez que se requiere realizar un acceso a memoria se busca la información en las caches, y en caso de que esta no se encuentre aquí se accede a las memorias de jerarquía superior.

En un sistema convencional puede haber varios niveles de cache. La cache más cercana es la memoria de nivel 1 o L1. En SCoPE se simula el funcionamiento de las caches conjuntamente con el modelado temporal del código SW de aplicación. Los efectos de las caches se integran como extensiones de la anotación de código fuente realizada para obtener el modelado no funcional del procesador durante la ejecución. De esta forma, al igual que con el modelado del procesador, se elimina la necesidad de modelos específicos de las caches, o de código binario. Este tipo de modelado es además el que más se ajusta al nivel de abstracción propuesto.

En cuanto al modelado de la memoria principal, se ha desarrollado un modelo simple que gestiona las transferencias entre memoria y procesador. Esto es necesario para permitir un correcto modelado de las transacciones en el bus.

El modelado de otros niveles de cache, como L2 no ha sido considerado directamente en este documento. No obstante, su simulación es posible mediante la generación e integración de componentes HW que emulen el funcionamiento de la cache L2 [Real10].

5.4.1.2.3 Interfaz de red

A parte del modelado de memoria y procesador, que son los componentes principales de un sistema SW, también se han modelado otros periféricos auxiliares. El primer componente a destacar es una interfaz de red. Esta interfaz se encarga de conectar el modelo de bus con el modelo de red. De esta forma, el código SW puede acceder a la red y comunicarse con otros nodos del sistema.

La interfaz realizada se ha modelado de forma abstracta, sin describir un componente real específico. Teniendo en cuenta la gran cantidad de tipos de redes y protocolos, muchos de los sistemas actuales implementan sus propias interfaces, con lo que modelar un tipo específico no sería muy útil. Además, al nivel de abstracción elegido, el modelado detallista de los componentes no aporta información crucial, sino que solamente tiende a sobrecargar la simulación.

5.4.1.2.4 DMA

Como último elemento HW específico desarrollado esta un modelo de DMA. Este elemento puede ser utilizado para realizar transferencias entre los periféricos y la memoria sin la intervención dedicada del procesador. Con este componente por tanto, se permite un modelado más adecuado de las transferencias en el bus y se permite la ejecución de drivers y el uso de periféricos que habitualmente hacen uso de este recurso en los sistemas reales.

5.4.1.2.5 Interfaz genérica de conexión al bus

Adicionalmente se ha realizado un esfuerzo encaminado a permitir la fácil integración de periféricos desarrollados por el usuario al modelo de bus abstracto propuesto como base de la plataforma. Para ello se ha desarrollado una interfaz genérica capaz de gestionar el protocolo del bus y permitiendo al usuario una adaptación muy sencilla de las comunicaciones.

Todos estos componentes se describirán en más detalle en el capítulo 8.

5.4.2 Plataforma SW

La plataforma SW esta compuesta por todos aquellos componentes independientes de la aplicación que se ejecutan sobre un procesador. Estos componentes pueden ser divididos en elementos del sistema operativo y “middleware”.

Considerando el nivel de abstracción elegido para la herramienta de simulación, SCoPE integra modelos tanto de OS como de “middleware”, en lugar de adaptar códigos

reales. No obstante, aunque los modelos no son tan pesados como los originales, su complejidad es bastante alta, soportando la mayor parte de la funcionalidad estándar en ambos casos. Esto hace que aunque se usen modelos, estos sean lo suficientemente completos como para soportar casi cualquier aplicación desarrollada para los entornos reales.

El uso de los códigos originales no es recomendable, ya que además de la sobrecarga de simulación que pueden presentar, el esfuerzo de integración es excesivamente alto. Por otra parte, el uso de modelos muy restrictivos limita las capacidades de la infraestructura, tanto con propósito de modelado como de diseño. Dado que SCoPE se presenta como una infraestructura para permitir el refinado de sistemas, la plataforma SW integrada debe soportar la mayor parte de la funcionalidad real.

En sistemas reales, el uso de middleware ligero (ligh-weigh) y OS sencillos hace que el tiempo de la plataforma SW respecto al SW de aplicación sea considerablemente menor. Por tanto, los modelos de plataforma SW utilizados sobre SystemC han de representar una pequeña parte del tiempo de simulación global del sistema, comparado con el propio código de la aplicación. Es por ello que el uso de modelos muy simplificados del middleware o el OS proporcionan mejoras mínimas en la velocidad de simulación.

5.4.2.1 Sistema operativo

El sistema operativo es el encargado de dar soporte básico a las aplicaciones. Representa el nivel inferior de la plataforma SW. Se encarga de generar y controlar los mecanismos de concurrencia, permitiendo la ejecución de múltiples procesos e hilos sobre el mismo procesador. Además, el sistema operativo se encarga de proporcionar las utilidades básicas que necesita el código de aplicación. Dado que el lenguaje C no tiene definidas funciones auxiliares dentro del propio lenguaje, es el sistema operativo el encargado de proporcionar este soporte. Facilidades de comunicación entre tareas, funciones de gestión de tiempo, alarmas y señales, o librerías de funciones matemáticas o de manejo de cadenas de caracteres son algunos de los ejemplos de estas funciones auxiliares. Además el sistema operativo se encarga de gestionar la comunicación con el resto del sistema, tanto HW como la comunicación de bajo nivel con otros procesadores.

Para estas tareas, el modelo de sistema operativo integrado en SCoPE se divide en las siguientes partes:

API: Es la parte encargada de dar soporte a las aplicaciones de usuario. Contiene las funciones que llama el código de usuario. Estas funciones deben llamar al resto de componentes del OS para realizar su propósito. Especialmente es el núcleo del OS el encargado de responder a sus peticiones, aunque pueden requerir del uso de otros elementos, como los controladores de dispositivo.

Controladores de dispositivo (Drivers): Son los encargados de realizar las comunicaciones con el HW. Los accesos a periféricos y las operaciones a realizar ante interrupciones HW son gestionados por los drivers. En SCoPE existe tanto la posibilidad de

utilizar drivers integrados en el propio sistema como drivers proporcionados por el usuario. El ejemplo más importante de drivers proporcionados por la herramienta es el driver de red, que controla el acceso a la interfaz de red del modelo HW. La capacidad de incluir de drivers de usuario permite que el usuario pueda gestionar sus periféricos particulares dentro del modelo de sistema global. Además, la posibilidad de integrar los drivers en código fuente permite que SCoPE pueda servir como plataforma de desarrollo para drivers. Para que los drivers puedan ser ejecutados estos pueden acceder tanto a funciones del núcleo del sistema operativo como a funciones del HAL.

HAL: La interfaz de abstracción HW es la encargada de independizar el resto del sistema de la plataforma subyacente. Esta parte se encarga principalmente de la gestión de interrupciones y las funciones de acceso al bus. Es la parte de la plataforma SW que se conecta directamente con la plataforma HW y da soporte al resto de la infraestructura.

Kernel: Es el núcleo del sistema operativo. Incluye el control de concurrencia y planificación así como la implementación de algunas de las funciones de cómputo incluidas en la API. Es en general el encargado de mantener al resto de elementos del sistema funcionando cooperativamente.

Pila TCP/IP: Es la parte encargada de la gestión de red. Se encarga de manejar el tránsito de paquetes entre diferentes nodos de las redes IP, y responder automáticamente a las peticiones de gestión del propio sistema, como los paquetes ARP.

5.4.2.2 Middleware

En sistemas complejos con grandes requisitos de cómputos, la arquitectura subyacente acaba siendo muy compleja. Múltiples procesadores, con distintas arquitecturas funcionan cooperativamente para permitir una correcta ejecución del sistema. Sin embargo, en muchas ocasiones interesa que la aplicación sea lo más independiente posible de esta plataforma. El middleware es en general la capa SW que se pone bajo la aplicación para ocultar los detalles de plataforma al código de aplicación.

El middleware se encarga de proporcionar a la aplicación una interfaz estándar independientemente de los sistemas operativos subyacentes o de la arquitectura HW de la plataforma. Esto proporciona una gran portabilidad al código, que puede ser reutilizado en sistemas completamente distintos. Además permite el desarrollo del SW aun sin la disponibilidad de la plataforma final.

En SCoPE para proporcionar estas capacidades se ha integrado un modelo de middleware basado en CORBA, resultado de la adaptación del modelo CORBA sobre SystemC realizado por TIMA.

5.5 Estructura interna

SCoPE, herramienta obtenida como resultado del trabajo de tesis, esta implementada en base a una estructura de archivos y directorios que intenta reflejar cada una de las partes presentadas anteriormente. Para ello se puede dividir su estructura en tres partes.

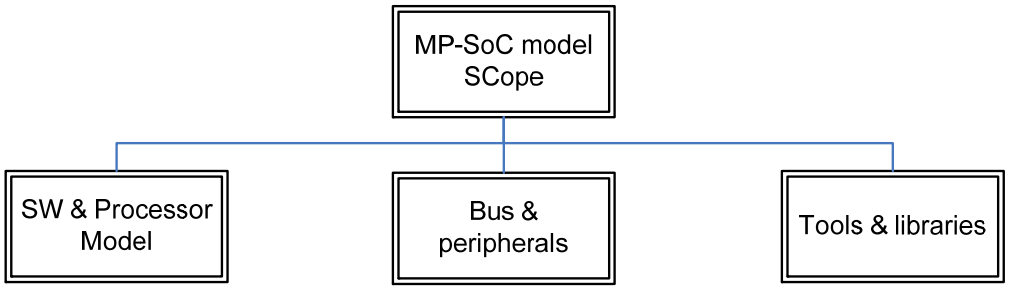


Figura 5-5: Estructura general de SCoPE

La primera parte integra todos los componentes necesarios para modelar los componentes SW, la infraestructura SW y el modelado de procesador y caches.

La segunda parte contiene la infraestructura HW del nodo. Está compuesta por el modelo de bus, los modelos de periféricos y la interfaz del modelo SW con el HW.

La tercera parte contiene las herramientas y librerías auxiliares necesarias para realizar las tareas de estimación y anotación. Además en esta parte se integran el conector para la integración junto con herramientas de exploración del espacio de diseño.

La primera y segunda parte se compilan generando una librería que contiene todos los elementos necesarios para construir los modelos. La tercera genera los ejecutables que permiten realizar las tareas de pre-procesamiento necesarias, que se presentarán en capítulos posteriores (6 y 9).

Cada una de estas partes es presentada a continuación, describiendo su objetivo y subdivisión interna, con objeto de tener una primera visión general de SCoPE antes de entrar a analizar en detalle las aportaciones concretas desarrolladas en cada uno de los aspectos de modelado de alto nivel cubiertos. En los siguientes capítulos, estas aportaciones serán descritas en detalle.

5.5.1 SW y modelado del procesador

La parte de SCoPE encargada de realizar el modelado SW se ha desarrollado como una herramienta automática, que puede funcionar sin el resto del modelo de plataforma. Esta parte se ha denominado PERFidiX, y como tal se incluye en el árbol de ficheros como un directorio propio.

En PERFidiX se integra el modelo del sistema operativo, el modelado del efecto del procesador, y otros paquetes adicionales, como el modelo de ORB y una pila TCP/IP. El modelo de sistema operativo intenta contener todos los niveles presentes en un sistema operativo estándar, y que son necesarios para el correcto modelado de la infraestructura SW de un sistema MPSoC convencional.

Para ello cuenta con un modelo de kernel, que se encarga del control de las tareas y de realizar el resto de servicios que componen el núcleo del sistema operativo. Estas operaciones se encuentran dentro de una subcarpeta llamada "PERFIDIX_Kernel" en el árbol de directorios. En ella se encuentran los archivos de estructuras de datos del núcleo (información de procesadores y sistema operativo), las funciones encargadas de gestión de tareas (creación, destrucción y sincronización de procesos e hilos), la planificación (planificador de procesos e hilos), la gestión de relojes, timers y otros elementos de tiempo o la gestión de señales.

En segundo lugar PERFidiX tiene una interfaz de usuario o API. Esta API contiene las funciones del OS que pueden ser accedidas desde el código de usuario para utilizar los servicios del sistema operativo. Estas funciones están desarrolladas siguiendo el estándar POSIX y se encuentran en una carpeta denominada "POSIX_Interface". En esta carpeta se encuentran las cabeceras y las implementaciones de las funciones POSIX, especialmente para comunicaciones, funciones de manejo de datos y aquellas que requieren el acceso a recursos del núcleo del sistema operativo.

El tercer componente fundamental de los sistemas operativos convencionales es la parte de bajo nivel, encargada de realizar las comunicaciones con la plataforma HW. Esta parte se encarga de permitir los accesos al bus y de gestionar el manejo de interrupciones. Se encuentra en una carpeta llamada "low_level".

Adicionalmente a esta carpeta se ha integrado una sección encargada de permitir la integración de drivers. Esta sección se ha introducido algunas funcionalidades en base al kernel Linux 2.6, para la gestión de drivers de carácter y de drivers de red. Incorpora una funcionalidad limitada pero suficiente para implementar drivers de complejidad moderada en SCoPE. La ampliación de la sección de drivers con una funcionalidad mucho más extensa ha sido realizada e introducida en la última versión de SCoPE por David Quijano.

Por último existe una carpeta adicional donde se agrupa la funcionalidad encargada de conectar el modelo de OS con el kernel de SystemC. En esta carpeta se encuentran las funciones que permiten el control de los SC_THREADs de SystemC y la sincronización, ejecución y parada de los mismos. Estas funciones son utilizadas por el kernel para realizar las tareas del sistema operativo adecuadamente.

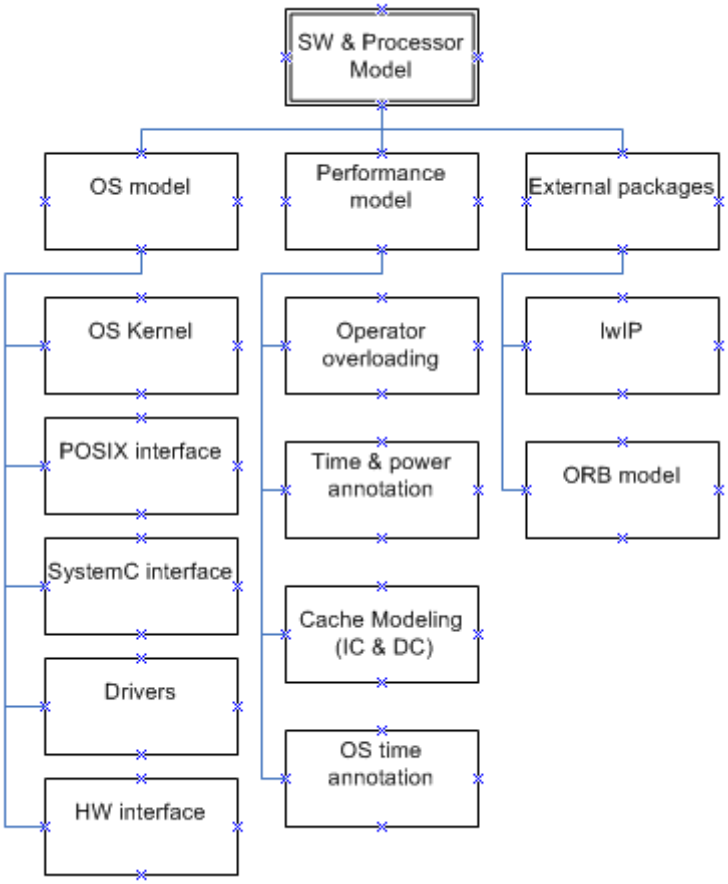


Figura 5-6: Estructura de la infraestructura SW de SCoPE

La segunda parte del modelo SW se centra en el modelado del efecto de procesador y caches sobre la ejecución de código. Las estimaciones de tiempo de ejecución o potencia, así como la transformación de la ejecución atemporal de código SW en una emulación temporal se encuentran en esta carpeta.

5.5.2 Modelado de la plataforma HW

Con objeto de organizar los elementos proporcionados para la generación de modelos de plataformas, y en pos de su mejor comprensión, se encuentran divididos en tres áreas, el modelo de bus y sus componentes, como el gestor de mapa de memoria, los componentes HW que actúan de maestros, como el procesador y el DMA, y los periféricos esclavos, como la memoria.

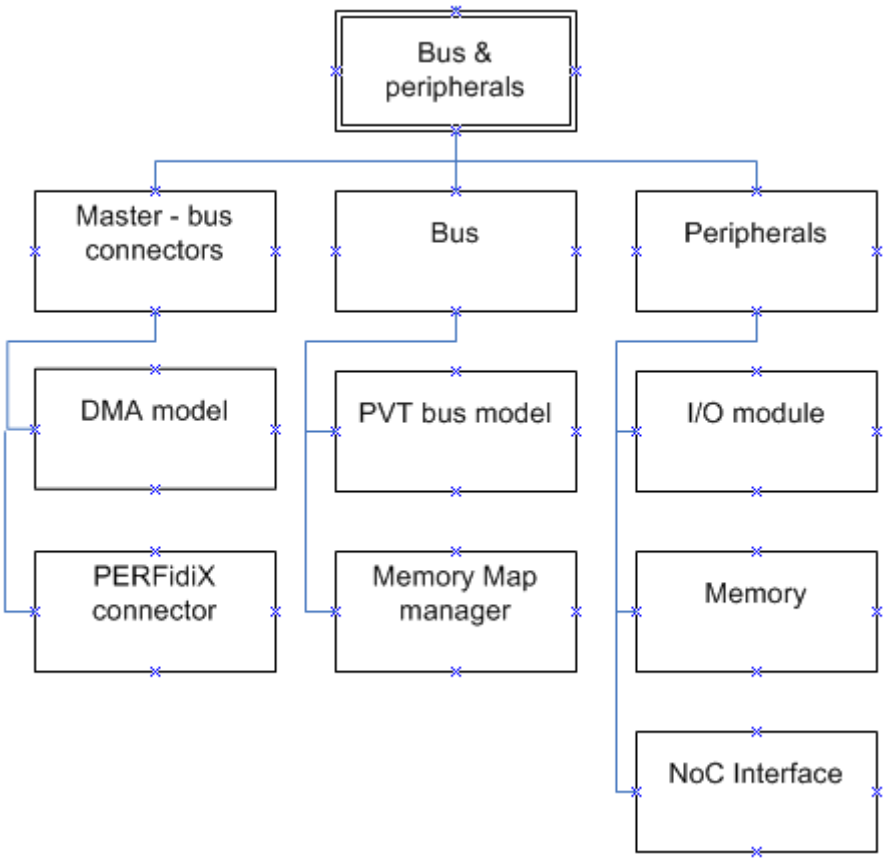


Figura 5-7: Estructura del modelo de plataforma HW de SCoPE

Además se incluyen en esta parte los modelos de red, tanto el modelo sencillo como la adaptación del simulador Sicosys.

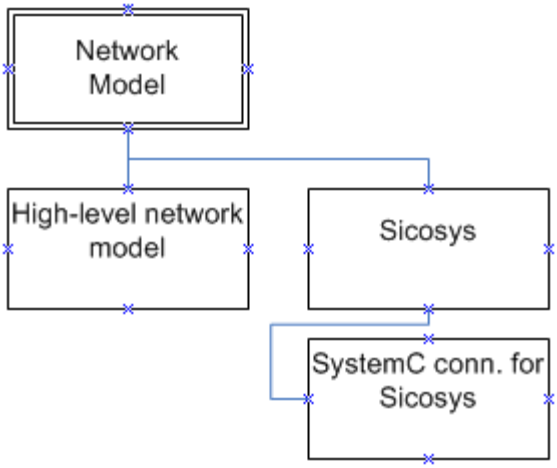


Figura 5-8: Modelos de red de SCoPE

5.6 Interfaces de entrada y salida

SCoPE ha sido desarrollada inicialmente como una librería de extensión de SystemC. Por ello el objetivo es se parte del ejecutable resultado de la compilación de la simulación SystemC. Esto significa que no es en si una herramienta al uso, basada en una interfaz gráfica, desde la cual se realizan las operaciones requeridas, que realiza cómputos internos y finalmente muestra los resultados. Esta posibilidad se ha propuesto como trabajo futuro.

No obstante, aunque el hecho de ser una librería da una flexibilidad y una capacidad de integración muy alta, puede presentar ciertas dificultades para la interacción por parte de los usuarios. Por esta razón se ha desarrollado una extensión auxiliar que facilita dicha interacción, pero manteniendo la posibilidad de su uso como librería auxiliar. Esta extensión se ha denominado M3P.

Esta extensión se centra en las interfaces de entrada y de salida, de tal forma que la información requerida y proporcionada por SCoPE pueda ser gestionada de forma más amigable. Además se ha conseguido que SCoPE pueda ser utilizada realmente como una herramienta y no solo como una librería auxiliar de SystemC. Para poder entender estos resultados vamos a analizar primeramente cuales son las entradas y salidas de SCoPE+M3P.

5.6.1 Entradas requeridas para la simulación

SCoPE es una herramienta de modelado de sistemas desde el punto de vista funcional, temporal y parcialmente de consumo de potencia. Eso significa que la herramienta necesita la información tanto de las características de los componentes que constituyen el sistema, como de la arquitectura del propio sistema a modelar.

Para el modelado de los componentes SW, se requiere de un modelado del procesador en términos de costes temporales. Para ello a cada operación C que contenga el código de aplicación SW se le asocia el número de ciclos que esta operación requerirá en el procesador objetivo (e.g. suma= 2 ciclos, asignación= 1 ciclo, llamada a función= 20 ciclos,…). Además, a cada operación de sistema operativo se le asocia un coste temporal, con objeto de tener en cuenta la sobrecarga introducida por el sistema operativo en el sistema real. Para proporcionar esta información SCoPE lee una serie de archivos con estos parámetros. Sin embargo, el formato de estos archivos no está pensado para ser leído directamente por el usuario. Por ello en la interfaz gráfica desarrollada se ha incluido una primera sección en la cual se pueden introducir todos estos parámetros de tal forma que al final se obtienen los archivos adecuados para utilizar en SCoPE.

El modelado de los componentes HW y la descripción de la arquitectura del sistema requieren un mecanismo distinto. El modelado de cada componente HW se realiza conforme a una descripción SystemC. Los componentes específicos proporcionados por el usuario deben contener internamente la información no funcional, de tal forma que por ejemplo cada respuesta se produzca con el retraso adecuado respecto de la llegada de la petición correspondiente. En cuando a los componentes genéricos proporcionados con SCoPE, pueden ser parametrizados para adecuarse en la medida de lo posible a sus equivalentes reales. Por ejemplo, el tiempo de respuesta de la memoria es un parámetro que debe ser proporcionado al modelo en el momento de su construcción.

Sin embargo, la parte más compleja de describir es la arquitectura del sistema. Para realizar esto se han considerado dos opciones que pueden ser elegidas por el usuario. La primera implica utilizar SCoPE como librería pura de SystemC. Esto implica que el sistema debe ser descrito en código C++, siguiendo las reglas de SystemC. Esta opción es la más potente, pero requiere de un conocimiento más profundo por parte del usuario. Además los ejecutables obtenidos son menos adecuados para tareas de exploración arquitectural, ya que la arquitectura, al ser descrita en código fuente es inamovible.

La segunda opción implica la utilización de SCoPE más como herramienta, ocultando al usuario el uso de SystemC y reduciendo el código fuente a proporcionar en la medida de lo posible. Al contrario que el caso anterior, el ejecutable no contiene información de la plataforma destino, sino que esta se proporciona mediante archivos XML. Esto permite que el mismo ejecutable pueda ser reutilizado para una exploración arquitectural del diseño. La interfaz XML será presentada en el capítulo 9.

5.6.2 Salidas generadas por SCoPE

SCoPE retorna información sobre las prestaciones del sistema propuesto mediante cuatro formas básicas.

En primer lugar, SCoPE reporta de forma automática una determinada información por consola. Una vez que el ejecutable finaliza, reportes sobre el tiempo de simulación,

utilización de los procesadores, tiempo por tarea o utilización de la red son presentados en pantalla.

En segundo lugar SCoPE genera un reporte con información sobre la ejecución interna. Cambios de contexto, utilización puntual de cada procesador o información de comunicaciones es guardada para permitir un análisis posterior pormenorizado. Para facilitar este análisis se han generado varias interfaces gráficas específicas para la el sistema que estaba en estudio en cada momento. La realización de una interfaz genérica es una tarea pendiente que se pretende llevar a cabo en un futuro cercano.

En tercer lugar se puede pedir a SCoPE que genere un archivo XML de salida que contenga determinada información, como tiempo de ejecución, número de instrucciones, potencia requerida, utilización de los buses, accesos a memoria, etc.

Por último, dado que la simulación en si es una simulación temporal, en cada momento el usuario puede incluir las marcas que crea necesarias para obtener la información de cuándo se ejecuta cada marca, y poder realizar así los análisis específicos que un determinado sistema requiera.

6 Modelado del SW de aplicación

6.1 Modelado del subsistema SW

Llamaremos subsistema SW a todos aquellos componentes del sistema que deben ser ejecutados sobre un procesador. Esto incluye tanto la infraestructura SW como el código de aplicación. Del análisis del estado del arte realizado en los capítulos previos, se concluye que para realizar el modelado del código SW en SCoPE es necesario extender las capacidades de modelado de SystemC en dos aspectos fundamentales:

- Diseñar un mecanismo para el modelado temporal del código SW de aplicación, de tal forma que la simulación del código SW no se limite a una ejecución funcional, sino que además permita considerar su efecto sobre el rendimiento del resto del sistema

- Generar una infraestructura (o plataforma) SW capaz de proporcionar los servicios requeridos por el código de aplicación, modelado el funcionamiento de la infraestructura del sistema real

La plataforma SW está formada por todos aquellos componentes SW genéricos que sirven como soporte al código de aplicación, específico del sistema. El elemento principal de esta plataforma es el sistema operativo (OS), que es el encargado de abstraer la plataforma HW y realizar la gestión del resto del SW.

Además del OS la plataforma SW puede contener otros elementos auxiliares. Paquetes SW, como una pila TCP-IP para la gestión de las comunicaciones por red, son comunes en muchos sistemas complejos. Así mismo, en sistemas distribuidos, especialmente si son heterogéneos y variables, desplegar una capa de abstracción adicional sobre el sistema operativo es relativamente habitual. Esta capa, llamada “middleware”, tiene como objetivo independizar al código de aplicación no solo de la infraestructura HW del nodo procesador como hace la API del OS, sino también de la arquitectura del sistema completo. Con esta capa se evita que al diseñar los componentes del SW de aplicación deban saber si el resto de componentes con los que han de interactuar están ejecutándose en el mismo procesador o en un punto remoto de la red, a qué dirección deben dirigirse para conectar con ellos o incluso cual debe ser el formato de los datos para evitar problemas de endianismo.

En el resto de este capítulo se presentarán las soluciones desarrolladas para el modelado temporal del código de aplicación. El modelado de la infraestructura será abordado en el capítulo siguiente.

6.2 Modelado del código de aplicación

La obtención de una simulación rápida implica evitar el uso de código binario cros-compilado junto con modelos ejecutables del procesador, como un ISS. No obstante, el modelado del comportamiento del código SW debe incluir todos los efectos que consideraría

un ISS en caso de estar presente en la simulación. Esto implica modelar el tiempo de ejecución y el consumo de dicho código.

Para realizar dicho modelado se han de tener en cuenta tanto el funcionamiento del núcleo del procesador como los efectos de la caché. Otros elementos, como el tiempo de acceso a memoria o a periférico no se consideran directamente en el modelado del código SW de aplicación. Los retrasos producidos por el bus, la memoria u otros periféricos se incluyen dentro del propio modelo de plataforma HW, que será presentado en el capítulo 8. El modelado SW tan solo ha de generar los accesos correspondientes a la plataforma para que se puedan considerar los efectos de dichas comunicaciones.

Para realizar un modelado óptimo del código de aplicación SW la aproximación propuesta se basa en los siguientes puntos:

- Definir un mecanismo de modelado y análisis de rendimiento del código utilizando SystemC como lenguaje de alto nivel para describir la estructura y funcionalidad del resto del sistema.
- Desarrollar técnicas rápidas de estimación de tiempo y consumo que se basen en el análisis del código fuente de forma que faciliten la portabilidad de la técnica y su adaptación a diferentes plataformas.
- Definir los mecanismos que interpreten esas estimaciones temporales y que la incluyan en la simulación temporal, teniendo en cuenta la semántica de simulación de SystemC.
- Considerar el mapeo de dichas tareas SW a recursos HW, de forma que el modelado del sistema tenga en cuenta las limitaciones en recursos que la plataforma HW impone.
- Considerar el efecto de la plataforma HW en la ejecución de las tareas SW, incluyendo paradas, expulsiones, cambios de frecuencia, …

Los puntos anteriores se centran en los elementos funcionales y temporales, ya que son los más complejos de modelar. La integración del cálculo de energías y consumos en la simulación es una tarea relativamente más sencilla, ya que no son elementos intrusivos que puedan alterar el orden de los eventos de la simulación. La estimación de energías y consumos se limita únicamente a la acumulación de la valores en variables globales, mientras que la integración de parámetros temporales en la simulación puede alterar el orden de ejecución de las tareas y las comunicaciones.

6.2.1 Modelado temporal del código de aplicación

Para poder realizar el modelado temporal del código SW de aplicación debemos ceñirnos al modelo de ejecución de SystemC, que soporta la simulación global de todo el sistema. Como se indicó en el capítulo 3, el mecanismo de simulación del lenguaje SystemC se basa en la ejecución consecutiva de ciclos de evaluación y actualización. Para realizar el modelado del código SW la solución adoptada pasa por ejecutar secuencias de código SW en la etapa de evaluación y aumentar el tiempo durante la etapa de actualización.

Para poder realizar el avance de tiempo es necesario conocer el tiempo necesario para ejecutar cada secuencia de código en la plataforma destino durante la simulación. La solución aplicada para incluir esta información en la simulación es modificar el código SW de aplicación mediante técnicas de anotación de código. De esta forma el código puede ser ejecutado en nativo a la vez que se modelan los efectos de la plataforma destino en SystemC.

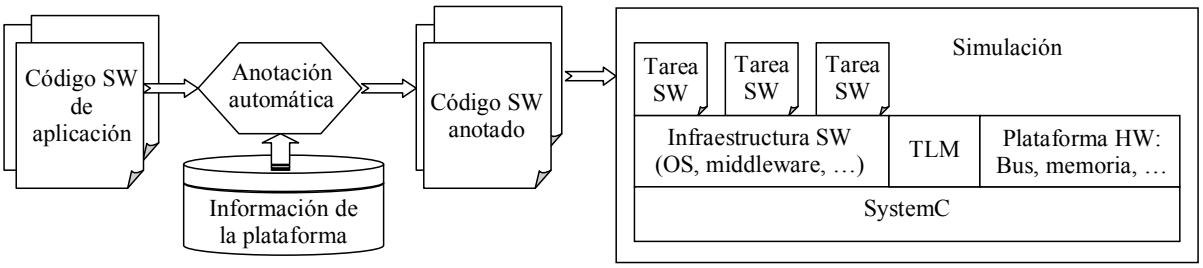


Figura 6-1: Proceso de modelado del código SW de aplicación

Para realizar la anotación se ha considerado fundamental evitar cualquier re-codificación manual del código por parte del usuario. La infraestructura desarrollada es capaz de realizar automáticamente todas las operaciones necesarias para la estimación y anotación, facilitando su uso. Además, esta solución evita la aparición discrepancias entre el código original y el simulado, fruto de la re-codificación manual del mismo. Esto reduce la posibilidad de cometer errores intermedios en el proceso de diseño.

Por último, dado que en los sistemas embebidos actuales la mayor parte de la funcionalidad está en SW, su modelado en alto nivel conlleva una gran parte del tiempo de simulación. Por ello, en SCoPE la aceleración de la simulación SW es aún más importante que la aceleración del modelado de los componentes HW. De hecho, el uso de descripciones ligeras del HW, comúnmente usadas como técnicas de modelado en alto nivel, proporcionan suficiente velocidad de simulación para dichos componentes.

El tiempo mínimo requerido para ejecutar el código de aplicación se puede definir como el tiempo que tarda ese código en ejecutar nativamente (sin anotaciones) en el computador donde se realizará la simulación. Como uno de los objetivos perseguidos en el modelado SW es la obtención de simulaciones lo más rápidas posibles, cualquier añadido que se aplique a la ejecución funcional del código SW para modelar su rendimiento tendrá un efecto indeseado. Cuanto menos intrusivos sean los añadidos necesarios para el modelado de comportamiento menos sobrecarga introducirán en la simulación. No obstante, la simplificación del modelado implica una reducción en la precisión. Por tanto, es fundamental determinar un punto que permita un compromiso adecuado entre velocidad y precisión.

Considerando el mecanismo de ejecución de SystemC, cuantos más ciclos de evaluación/actualización se necesiten para modelar el sistema, mayor será la sobrecarga en la simulación. El modelado más estricto posible de código fuente implicaría realizar una simulación donde tras cada instrucción se esperase su tiempo correspondiente. Sin embargo,

la sobrecarga introducida por este modelado sería extraordinariamente alta: un ciclo completo del núcleo de SystemC por instrucción.

Para optimizar la velocidad de simulación, el primer punto es definir la máxima cantidad de código SW que se puede ejecutar sin esperas de tiempo intermedias sin que se altere la precisión del modelo. Para ello, cada secuencia de código ejecutable en una etapa se ha denominado "segmento de código". La definición y cualidades de estos segmentos serán presentadas en la sección 6.2.1. Así, el procedimiento utilizado para modelar el SW de aplicación aplicado consiste en dividir el código de cada componente SW en segmentos. A continuación se estiman los parámetros de rendimiento de cada segmento de código sobre la plataforma destino (tiempo de ejecución y consumo) y posteriormente se anotan estos parámetros en el código y se integra el código anotado en la simulación (Figura 6-2).

Para describir este proceso en mayor detalle comenzaremos describiendo como se divide el código en segmentos. A continuación se presentará el mecanismo utilizado para realizar las estimaciones de los segmentos de los componentes software. Después se describirá la técnica empleada para obtener los resultados globales a partir de las estimaciones de cada uno de los segmentos. Finalmente se presentará la técnica de modelado de cachés.

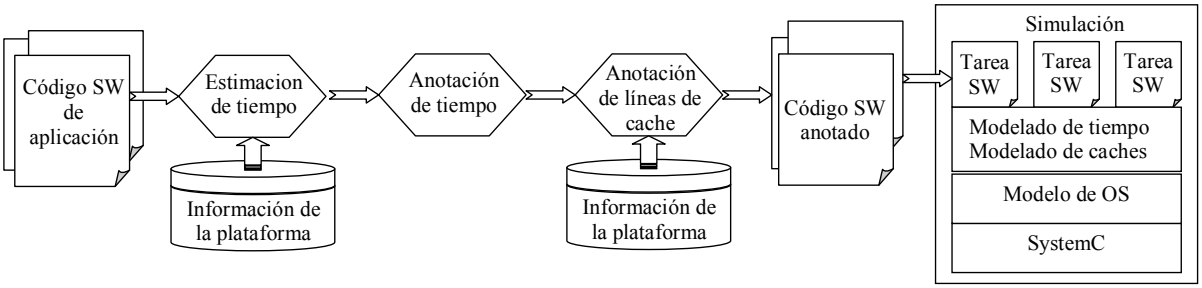


Figura 6-2: Modelado temporal de código de aplicación

6.2.2 Granularidad: Subdivisión del código en segmentos

La definición de una granularidad adecuada es clave para obtener una buena relación entre velocidad y precisión en el modelo. Los programas SW suelen contener una serie de etapas transformativas, de tal forma que se conforma un lazo de tres etapas: se adquieren datos, se computan, y se devuelve el resultado, posiblemente almacenando información para el siguiente cómputo. Esta visión, que puede parecer muy simplista, tiene gran interés en la definición de la granularidad. Significa que un código SW puede verse como una sucesión de etapas de cómputo y puntos de comunicación.

Así, la eficiencia del código SW depende de la arquitectura procesadora subyacente. El uso de procesadores, buses, caches, y dispositivos periféricos de almacenamiento, y entrada/salida implica una serie de efectos al considerar, en función de las capacidades de cada uno de estos elementos. Así como el cómputo restringido al procesador/cache suele ser muy rápido, el acceso a otros componentes implica perdidas de rendimiento. Además debemos tener en cuenta que la ejecución en un procesador se realiza de manera secuencial.

El modelado de sistemas HW/SW requiere de un manejo preciso de los puntos de comunicación. En consecuencia, al incluir las anotaciones de tiempo se debe asegurar que todos los elementos del modelo interactúen adecuadamente. Esta correcta interacción requiere que todas las comunicaciones se realicen en el momento adecuado, algo difícil de conseguir utilizando el SW original, que no tiene información temporal. Si toda ejecución de SW se realiza en tiempo cero la interacción entre componentes puede resultar poco realista. Así, por ejemplo, podríamos ejecutar todo el código en primer lugar, y una vez ejecutado ir añadiendo la información temporal que creyésemos oportuna a los resultados de la simulación. Como consecuencia, los resultados de la simulación podrían ser funcionalmente coherentes pero temporalmente absurdos.

Para asegurar que el mecanismo de simulación garantice la corrección completa de los resultados debemos asegurar la coherencia de todas las comunicaciones, tanto respecto a los datos como a los tiempos y sincronizaciones. Esto significa que cuando se realice una transacción, el dato recibido por una tarea sea coherente con el dato proporcionado por otra, teniendo en cuenta el punto temporal de la transacción. Además es necesario asegurar que el coste temporal del cómputo de la tarea productora se vea reflejado también en la tarea consumidora, o dicho de otra forma, que una tarea del modelo no pueda computar con un dato de entrada antes de que la productora haya acabado de generarlo. De esta forma los costes temporales pueden ir encadenándose adecuadamente.

Sin embargo, no es necesario fijar exactamente cada una de las operaciones de cómputo internas del programa, especialmente aquellas que no representen una comunicación con el resto de tareas del sistema. Por ejemplo, supongamos que tenemos una tarea SW que hace de servidor y se dedica a calcular la raíz cuadrada del número que se le proporcione. Para obtener un modelado adecuado es necesario en primer lugar que se tome el dato de entrada en el momento correcto. Además es necesario devolver el resultado de la raíz cuadrada en el momento oportuno. Así, el resto de operaciones de la tarea receptora dependerán de cuanto ha tardado este servidor en calcular la raíz, a fin de modelar los efectos colaterales que el tiempo de dicha raíz cuadrada tendrá en el resto del sistema. Pero para ello no es necesario conocer en que momento se realiza cada operación interna de la raíz, sino el tiempo total de la operación en su conjunto.

Para demostrar esta afirmación, hay que analizar si el resto del sistema HW/SW tiene alguna necesidad de conocer que operaciones se realizan en cada momento dentro cada módulo. Centrándonos en el modelado de SW, consideremos el conjunto procesador-cache-código SW como una unidad. Esta unidad es completamente independiente del resto del sistema durante la ejecución de un cómputo, excepto en tres casos concretos: que se produzca un fallo de cache, un acceso a periférico o una interrupción HW. En cualquier otro caso, lo único que necesita saber el sistema es que el procesador está ocupado computando, pero sin importar qué esta haciendo exactamente. Por ello comenzaremos proponiendo un mecanismo para el modelado del SW de aplicación que será válido mientras el procesador no se vea afectado por eventos externos. Posteriormente se presentarán los mecanismos necesarios para que tanto los accesos a memoria como la gestión de interrupciones sean manejados adecuadamente.

El mecanismo base de modelado de SW propuesto se centra en suponer una estructura comunicación-cómputo-comunicación general en el código SW. Siguiendo esta estructura denominaremos "segmento de código" o "segmento" a la sección de código consecutivo que se ejecuta sin que se realice ninguna interacción con el resto del sistema.

Para ver la solución adoptada para desarrollar el uso de segmentos dentro de los procesos, comencemos representando los procesos mediante grafos. En estos grafos, los arcos corresponden a segmentos de código, sin eventos de comunicación o sincronización. Estos eventos serán los nodos del grafo. Por tanto, el sistema solo interaccionará con el proceso en los nodos. Esto se puede ver en la figura 6-3:

```
N0      void process() {

                do {                    //code of segment S0-1
                        …               //common code to S0-1 and S4-1
N1                      ch1.read();
                        …               //common code to S1-2 and S1-3
                         if(condition){ //common code to S1-2 and S1-3
                                …       //code of segment S1-2
N2                              ch2.write();
                                …       //code of segment S2-3
                        }
                        …               //common code to S1-3 and S2-3
N3                      sleep(delay1);
                        …               //code of segment S3-4
N4                      ch2.read();

                } while (true);         //code of segment S4-1
        }
```

Figura 6-3: Segmentos en un código fuente.

En esta pieza de código, N0 es el nodo inicial del proceso cíclico. N1, N2 y N4 son los nodos correspondientes a los accesos a canal. N3 es una espera de tiempo. 'Si-j' representa el segmento entre el nodo "Ni", y el nodo "Nj". El gráfico resultante se puede ver en la figura siguiente.

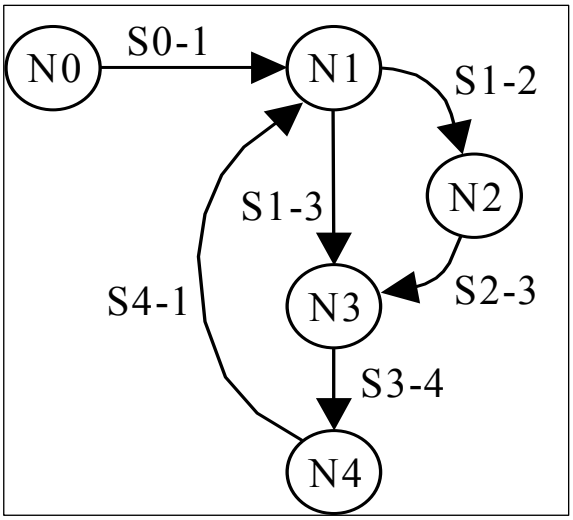


Figura 6-4: Grafo del proceso de la figura 6-4.

Una vez se ha definido la representación del proceso en virtud de los segmentos de código, se puede obtener el tiempo estimado para cada segmento, y realizar el modelado del SW. Para realizar el modelado, en la simulación se ejecuta cada segmento de código como una unidad. Una vez ejecutado el segmento, se toma la estimación del tiempo de ejecución de dicho segmento y se utiliza para realizar la espera temporal correspondiente al tiempo de procesador necesario para dicha ejecución. Como consecuencia, al final del proceso de modelado de un segmento se debe haber producido tanto la ejecución del código SW como el avance del tiempo de simulación correspondiente.

Hay que tener en cuenta, que en SystemC toda la ejecución SW se realiza sin que se avance en el eje temporal de la simulación, es decir se ejecuta en "tiempo cero". En general, en SystemC todo código C se ejecuta en tiempo cero. Únicamente se produce un avance de tiempo cuando se realiza una llamada a un estamento "wait". Por tanto, para producir el efecto de avance del tiempo de simulación conforme se ejecuta el código SW, cuando se llega a un acceso a punto de comunicación se ha de realizar una llamada a la función "wait" de SystemC con el valor de tiempo estimado. De esta forma, el modelado de la ejecución del procesador correspondiente se para hasta que el eje temporal ha avanzado el valor correspondiente a la estimación realizada. En conclusión, Al final el resultado obtenido desde que se comenzó la ejecución del segmento hasta su final es idéntico al alcanzado si cada operación del segmento hubiese hecho avanzar el eje temporal con su coste correspondiente.

El comportamiento final es equiparable al resultado obtenido si hubiésemos utilizado un ISS para ejecutar el código, avanzando un determinado número de ciclos por cada instrucción. Sin embargo, el aumento en la granularidad produce un incremento sustancial de la velocidad de simulación. Hay que tener en cuenta que cada llamada a un "wait" implica una sobrecarga de simulación, ya que requiere llamar al núcleo de simulación de SystemC para lanzar otros procesos SystemC hasta que el tiempo avance en la cantidad indicada. Por eso, aprovechando la estructura común de "comunicación-computo-comunicación" de los programas SW se puede reducir notablemente el número de llamadas al kernel de simulación

sin incurrir en ningún error de modelado. No obstante, como hemos dicho anteriormente, esto ha de ser modificado cuando se producen fallos de cache o interrupciones. El modelado de las interrupciones será tenido en cuenta, cuando se explique como se utiliza el tiempo estimado para producir el avance de tiempo mediante sentencias "wait". El modelado de caches se presentará posteriormente

Finalmente, antes de que se retorne de una comunicación hay que tener en cuenta que no haya otro componente software del mismo procesador ejecutando, en cuyo caso habrá que esperar a que el procesador quede libre para que pueda tomarlo. Esto es realizado por el modelo de sistema operativo integrado en SCoPE (Capítulo 7).

.

En todo caso, el punto principal para definir cada segmento de código es identificar todas las comunicaciones que el código de aplicación realiza. Estas comunicaciones pueden reducirse a tres casos:

- Acceso a variable global / memoria compartida
- Llamada al sistema operativo
- Acceso directo a un periférico

6.2.2.1 Acceso a variables globales

La forma más simple de comunicar tareas es el uso de variables globales. En muchas ocasiones, para compartir información entre hilos del mismo proceso, se utilizan variables globales. Estas variables, al estar compartidas, permiten que los valores escritos por un hilo puedan ser leídos por otro sin sobrecarga adicional en el tiempo de ejecución.

No obstante, esa simplicidad para la ejecución SW es a la vez un gran problema para el modelado. Dado que el acceso a esas variables globales es indistinguible del resto de variables de la tarea, no es fácil realizar la detección. Durante la tesis se han evaluado varios mecanismos para resolver este problema, tal y como se describe a continuación.

La primera solución evaluada para el modelado de acceso a variables globales es pre-procesar el código de tal forma que se detecte la declaración de todas las variables globales. En el código es suficiente con marcar cada acceso a una de esas variables, forzando la finalización del segmento de código actual y comenzando uno nuevo tras la transferencia. No obstante esto necesita el desarrollo de un programa que haga el pre-procesado, lo que requiere un cierto esfuerzo.

La segunda solución es sustituir los tipos de datos de las variables globales por tipos sobrecargados, de tal forma que cada vez que se acceda a esos tipos, tanto para leer como para escribir se fuerce un cambio de segmento. Esto se puede hacer sobrecargando los operadores de conversión automática de tipo y de asignación. La ventaja de esta técnica es que no necesita buscar en el código cada acceso a la variable, sino que se realiza la detección

automáticamente al acceder a los operadores sobrecargados. No obstante sigue siendo necesario modificar el código para cambiar los tipos de datos de las variables globales.

```
class int_global {
        int m_dato;
        // escritura
        int operador = (int dato){
                m_dato = dato;
                fin_segmento();
        }
        // lectura
        int &operator (){
                fin_segmento();
                return m_dato;
        }
};
```

Figura 6-5: Ejemplo de clase con operadores sobrecargados

Esta modificación puede no requerir ningún esfuerzo adicional dependiendo de la técnica de estimación utilizada. En la técnica de sobrecarga de operadores que se verá posteriormente, todas los tipos de datos se modifican para permitir la sobrecarga de operadores, así que la detección de variables globales no supone un coste añadido. La única operación adicional requerida es detectar si una variable es global o local para decidir si finalizar el segmento actual o no. Esto es sencillo si tenemos en cuenta que las variables globales se construyen antes del comienzo del programa, y las locales después de haber empezado la función principal (“main”).

No obstante, estas técnicas no son válidas para memoria compartida. La utilización de memoria compartida entre procesos se realiza dinámicamente en el código, solicitando área de memoria compartida y asignándosela a las variables que se crea conveniente. Esto significa que conocer que variables son asignadas a esa zona de memoria implicaría un análisis de asignación de direcciones de memoria, y por tanto, de ejecución de código. Por ello se requeriría un pre-procesador complejo.

Otra posible solución es restringir el error limitando el tamaño del segmento. El error en el acceso a una variable compartida no detectable depende de la distancia de la comunicación más cercana detectada. Esto significa que depende del tamaño en tiempo del segmento. Si limitamos el tamaño máximo del segmento a un determinado tiempo, podemos controlar ese error. De esta forma, un segmento duraría hasta la siguiente comunicación o hasta el máximo tiempo de un segmento, tras lo cual comenzaría un nuevo segmento. No obstante, esta técnica implica una mayor sobrecarga de simulación al limitar el tamaño de todos los segmentos, tengan en su interior variables globales o no.

En todo caso, garantizar el correcto modelado del acceso a las variables depende estrechamente del mecanismo de anotación con el que se apliquen los tiempos estimados por segmento a la simulación. Por ello serán analizados en más detalle en la sección 6.4.1.

6.2.2.2 Modelado de las llamadas al sistema operativo

El modelado de las llamadas al sistema, al contrario que en el caso de las variables globales, es muy simple. Comunicaciones a través de colas de mensajes, "sockets" o tuberías, requieren introducir en el código llamadas a la función del sistema operativo correspondiente. Esto mismo es aplicable para los elementos de sincronización, como "mutex", semáforos, variables condicionales, "rwlocks" o "spinlocks". Por último, operaciones como esperas a señales, tiempos de espera u otras operaciones que paran la ejecución del proceso también requieren llamadas al sistema.

Para detectar estos puntos de comunicación es suficiente con detectar las llamadas a función. Para ello basta con rodear todas estas llamadas por un "envoltorio". De esta forma se genera una nueva función que primero finaliza el segmento actual, realiza la llamada al sistema real, e indica el comienzo de un nuevo segmento (Figura 6-3).

```
int nueva_llamada_al_sistema (args) {

        Fin_segmento ();

        Llamada_al_sistema(args);

        Comienzo_segmento ();

}
```

Figura 6-3: Modelo genérico de llamada al sistema

Para sustituir la llamada original por la nueva llamada puede aplicarse una modificación por defines ("#define llamada_al_sistema nueva_llamada_al_sistema"), con lo que el pre-procesador de C++ modifica automáticamente la llamada, o utilizar un pre-procesador específico. No obstante, la solución utilizada depende de la técnica de estimación. Para técnicas basadas en "define" se ha usado otro "define", mientras que para técnicas basadas en preprocesado se ha aprovechado el pre-procesador desarrollado.

6.2.2.3 Modelado de las llamadas a periféricos

Por último, los segmentos deben ser modificados por el acceso directo a periféricos. Por ejemplo, si sabemos que un periférico esta en la dirección de memoria 0x80000000, una operación de tipo "*0x800000000=dato;" implica el envío de ese dato al periférico y por tanto una comunicación. La detección de estos casos es bastante compleja, e implica la utilización

de técnicas de comunicación HW/SW. Por ello se pospondrá su explicación al capítulo de comunicación HW/SW.

Como adelanto, baste indicar que la ejecución de la instrucción anterior en un PC que no sea la plataforma destino provocará un fallo crítico ("segmentation fault"), ya que el sistema operativo subyacente no permitirá a la simulación escribir en una dirección fuera del mapa de memoria base para el proceso. La resolución automática del fallo y la detección de la comunicación se realizarán de conjuntamente, permitiendo la división de los segmentos.

6.2.3 Técnicas de estimación temporal de segmentos de código SW

Varias técnicas ha sido analizadas para estimar el tiempo requerido por cada segmento del código en el procesador destino, tiempos que serán aplicados posteriormente a la simulación con objeto de obtener un modelo temporal del SW.

Para realizar dicha estimación se pueden listar principalmente las siguientes técnicas:

- Estimación por ajuste de tiempo nativo
- Estimación dinámica por sobrecarga de operadores
- Anotación a partir de código fuente
- Anotación a partir de código binario
- Anotación con información de ISS

Durante el desarrollo de la tesis todas estas técnicas han sido evaluadas e implementadas. No obstante, las dos primeras no están disponibles en la versión actual de SCoPE, por considerarse las menos efectivas, aunque desde el punto de vista investigación todas tienen un gran interés.

6.2.3.1 Estimación temporal mediante ajuste de tiempo nativo

La primera técnica estudiada se basa en ejecutar la simulación sobre un PC nativo obteniendo el tiempo necesario para ejecutar cada segmento de código en dicho PC. La estimación del coste temporal de los componentes en la plataforma destino se realiza en función del tiempo que requiere la simulación de cada módulo. Para ello se multiplica este valor por un factor de ajuste en función de las características del PC nativo y de la plataforma destino.

El procedimiento requerido por esta técnica presenta algunas diferencias con respecto al presentado en la figura 6-2. A diferencia del resto de técnicas cubiertas en la tesis, esta no requiere la generación de un código SW anotado. La estimación y modelado de tiempo se realiza automáticamente al realizar las llamadas al sistema (Figura 6-3). El modelo de sistema operativo obtiene el tiempo de la ejecución nativa, aplica el factor y ejecuta el "wait" de SystemC (Figura 6-6).

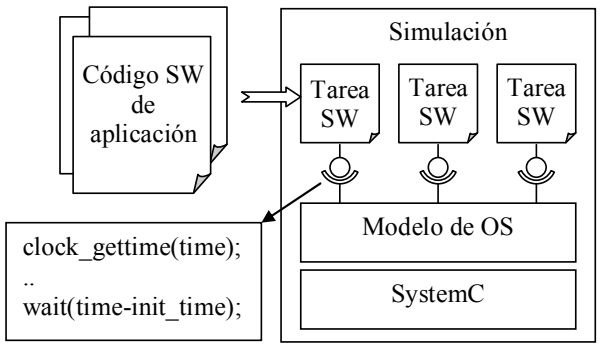


Figura 6-6: Modelado mediante ajuste de tiempo nativo

El tiempo de ejecución de cada segmento se obtiene mediante la llamada a la función "clock_gettime()" del sistema operativo nativo. Para minimizar el error producido por el resto de tareas del PC, la simulación debe ser lanzada con la mayor prioridad posible.

Esta opción presenta la ventaja de ser muy rápida, pero tiene una serie de desventajas que dificultan su utilización real. En primer lugar, hay que ser capaz de asegurar que el tiempo de simulación sea realmente causa de la ejecución del código del sistema, y no de otros procesos parásitos que estuvieran en ejecución en el propio computador nativo. Además, dada la poca información que se puede obtener de la simulación, tan solo el tiempo de simulación, la transformación posible se reduce a una transformación lineal. Sin embargo no se puede asegurar que el coste en el PC nativo y en la plataforma se ajuste a una relación lineal. Al contrario, la existencia de estructuras HW diferentes, como diferentes caches, arquitecturas de memoria o co-procesadores matemáticos, puede producir errores importantes en la estimación.

Por ejemplo, si consideramos un procesador nativo con una unidad aritmético-lógica capaz de realizar sumas, restas, multiplicaciones y divisiones, y un procesador en la plataforma final que solo posea sumador y restador podemos encontrarnos en el siguiente caso: una serie de aplicaciones que solo utilicen sumas y restas presentarán una relación lineal, pero unas aplicaciones con muchas divisiones pueden presentar una relación cuadrática. Sin embargo, con la mera simulación del código y analizando el coste final en tiempo de dicha simulación no se pueden distinguir ambas situaciones. En conclusión, el mecanismo tan solo puede producir resultados aceptables en unas condiciones limitadas.

6.2.3.2 Estimación por sobrecarga de operadores

La técnica de sobrecarga de operadores se basa en la redefinición de todos los operadores aritméticos y lógicos e instrucciones de control del código SW original. Los nuevos operadores realizan dos cometidos: incrementan el tiempo de segmento según el tipo de operador y ejecutan la operación correspondiente. Para ello los tipos de datos básicos de C/C++ son reemplazados por clases con los operadores sobrecargados.

La técnica de estimación por sobrecarga de operadores calcula el coste del código SW conforme este se va ejecutando. Por tanto, cada operación que ejecute el código debe ir acompañada por una consideración del coste temporal que representa. Acumulando el tiempo necesario para ejecutar todas las operaciones de un segmento obtendremos la estimación temporal de cada segmento de código. Esto evitará costosos algoritmos y cálculos estáticos, evitando obtener tiempos sobredimensionados, como en el caso de técnicas de estimación de peor caso (WCET), o la consideración de falsos caminos. El tiempo estimado dependerá exactamente del código que se haya ejecutado.

El mecanismo de funcionamiento de esta técnica de estimación se puede ver en la figura 6-7. En primer lugar el código original es modificado sustituyendo los tipos de datos originales por las nuevas clases sobrecargadas. Esto se realiza automáticamente utilizando el pre-procesador del compilador C. A continuación se ejecuta el código utilizando los operadores sobrecargados.

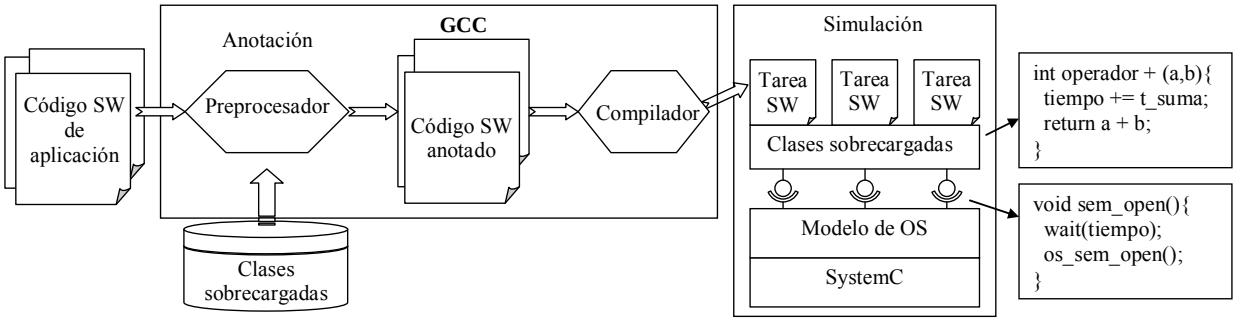


Figura 6-7: Modelado temporal por sobrecarga de operadores

El código de aplicación original es compilado sin ningún análisis o modificación previa. Por ello el modelado por sobrecarga de operadores es una técnica completamente dinámica. Todas las operaciones realizadas en el código son monitorizadas por la técnica de anotación. Esto implica que la técnica tiene una enorme potencia como técnica de análisis de código. Estudios sobre el número de operaciones, tipos de datos o seguimiento de variables pueden ser fácilmente realizados modificando mínimamente la sobrecarga de los operadores. No obstante, esto tiene un coste en velocidad de simulación, que posteriormente valoraremos.

La técnica de sobrecarga de operadores trabaja exclusivamente sobre el código fuente, a diferencia de otras técnicas que veremos posteriormente. Eso implica un alto grado de portabilidad. No es necesario disponer de un conjunto de herramientas de desarrollo completo de la plataforma destino (e.g. gnu-toolchain), de un sistema operativo adaptado o de un ISS. La única información necesaria para modelar la plataforma destino es un archivo con el tiempo requerido para ejecutar cada operador e instrucción de control de C/C++ en la plataforma destino.

No obstante, la generalidad conseguida conlleva una perdida de precisión. Para analizar las perdidas de precisión de está técnica consideremos las simplificaciones realizadas con respecto a una ejecución sobre un simulador de conjunto de instrucciones (ISS) con precisión de ciclo.

Los ISS ejecutan el código binario generado por el compilador. De manera general podemos pensar en el compilador como un traductor que va sustituyendo cada operación del código original por un número de instrucciones ensamblador. De hecho, si compilamos un código sin ninguna optimización podemos ver fácilmente la correspondencia en la traducción del código original en código ensamblador. Por tanto, podemos asociar a cada operación C un coste de tiempo sumando las instrucciones ensamblador que conlleva y cuanto tarda cada una, podremos estimar cuánto tiempo conllevaría la ejecución del segmento completo.

Sin embargo la estimación mediante la asociación de un número de instrucciones ensamblador a cada operador C no es exacta cuando aplicamos optimizaciones en la compilación. La consideración de todos los efectos de las posibles optimizaciones del compilador requeriría la implementación parcial de un compilador propio. Dada la complejidad de la tarea se ha dejado fuera del ámbito de la tesis. En su lugar, se consideran las optimizaciones como un factor de corrección aplicado en el cómputo del tiempo de cada segmento.

Para obtener el factor de corrección consideraremos la reducción experimentada en el tamaño del código entre un código compilado con optimizaciones y uno compilado sin ellas. De esta forma supondremos que la reducción en el número de instrucciones total del código compilado es extrapolable a cada segmento de código particular, suponiendo que el número de instrucción de dicho segmento se reducirá también en esa proporción. Esta suposición implicará un cierto error, pero es asumible teniendo en cuenta que nuestro objetivo es la velocidad, permitiendo un cierto error, y que al nivel de abstracción que se encuadra la tesis es más exigible garantizar la fidelidad de las estimaciones que su precisión.

Además hay que tener en cuenta que un ISS preciso tiene descritas todas las partes del procesador, incluyendo las etapas del camino de datos, mecanismos de predicción de salto o sistemas reordenamiento de instrucciones. Como consecuencia el ISS es capaz de modelar adecuadamente las paradas por fallos de cache, dependencias de datos o descarte de instrucciones por salto.

Por el contrario, el modelado genérico en código fuente propuesto no puede tener en cuenta estos detalles. Teniendo en cuenta una pequeña pérdida media podemos evitar el manejo de la información interna de la segmentación del camino de datos. Para minimizar esta pérdida de precisión consideraremos las instrucciones binarias que se ejecutan y el número de ciclos requeridos, aplicando como factor de corrección el número de ciclos medio por instrucción del procesador (CPI). Este factor suele ser proporcionado por el vendedor del procesador.

Una vez descrito el funcionamiento general de la técnica de sobrecarga de operadores, nos centraremos en qué operaciones hay en un código C, cómo asociar automáticamente un coste temporal a cada operación y en cómo obtener al final la estimación del tiempo de ejecución de cada segmento.

6.2.3.2.1 Estimación de coste de las operaciones de código fuente

Consideremos que un código C/C++ está formado por la sucesión de operaciones de dos tipos: operaciones de cómputo y operaciones de control. Las operaciones de cómputo son todas aquellas en las que se toma un valor de una o varias variables, se opera y se devuelve a otra variable. Las operaciones de control son aquellas que se encargan de controlar el flujo de datos, produciendo saltos hacia delante, hacia atrás o a zonas remotas al código, como son los saltos a función.

6.2.3.2.1.1 Estimación del coste de las operaciones de cómputo

Teniendo en cuenta la definición anterior, para asociar un coste a las operaciones de cómputo tenemos que manejar información tanto por cada tipo de operador como por cada tipo de variable implicadas. En determinados operaciones, el tiempo requerido para su ejecución depende del tipo de sus datos. Por ejemplo, en que no disponen de un operador HW, como divisiones u operaciones en punto flotante, el tipo y tamaño de los datos determinan la complejidad de las funciones ejecutadas.

Para integrar el coste de cada operación en el código fuente debemos modificar el código SW. Para facilitar el uso de la técnica de estimación esta operación ha de realizarse automáticamente. Para ello utilizaremos las facilidades del propio lenguaje.

El compilador C/C++ traduce directamente las operaciones convencionales a sus correspondientes instrucciones ensamblador automáticamente. Estos operadores no pueden ser redefinidos si no es modificando el compilador internamente, cosa que se escapa de los objetivos de la tesos. Sin embargo, esto no ocurre en C++ para las operaciones de cómputo con objetos. Cuando se realiza una operación con un objeto, normalmente de un tipo clase definido por el usuario, es el propio usuario el que debe definir qué cómputo realiza dicho operador. Esto es así porque el compilador es consciente de lo que significa sumar dos enteros o realizar una comparación de flotantes, pero si tenemos una clase que por ejemplo contenga los datos personales de un cliente, el compilador no sabe qué significa sumar o restar los datos de dos clientes.

Por tanto, si podemos convertir todas las variables de tipos básicos a clases definidas por el usuario podemos redefinir todos los operadores. Por ejemplo, si en el código “a += b;” hacemos que los tipos de “a” y “b” no sean “int” si no que definimos un nuevo tipo “class mi_int” la operación automáticamente llamará al operador “+=” de dicha clase. Esto nos permitirá resolver el problema de cómo asociar un tiempo a cada operador C/C++. Para ello haremos que el operador “+=” de la clase primero añada el tiempo correspondiente a una suma de dos enteros al tiempo acumulado del segmento actual, y a continuación haremos que realice la operación “+=” con los valores contenidos en las instancias de clase “a” y “b”. Como resultado el código habrá realizado la operación “a+=b”, actualizando “a” con el valor de “b”, y además el tiempo de segmento se habrá incrementado con el coste del operador “+=”.

Por tanto, para realizar la anotación automática deberemos reemplazar todos los tipos de datos por clases, y sobrecargar todos los operadores para que añadan el coste temporal

asociado a la vez que realizan la operación esperada del operador para los tipos de datos originales. La redefinición de todos los tipos de datos se puede conseguir utilizando las cláusulas "#define" del pre-procesador de C. Si creamos un archivo que contenga traducciones del tipo "#define int mi_int" para todos los tipos de datos, cada declaración será automáticamente reemplazada, con lo que todas las operaciones podrán tener asociado su tiempo, sin requerir un esfuerzo extra de re-codificación manual.

Sin embargo, C/C++ no está pensado para realizar estas transformaciones. Pasemos pues a ver los problemas que se presentan y cómo resolverlos.

6.2.3.2.1.2 Tipo de dato para la sobrecarga

Para crear los nuevos tipos de datos utilizaremos clases de C++. Estas clases van a sustituir a los tipos básicos de C/C++, por lo que tienen que almacenar al menos la misma información, esto es, el valor de las variables a las que representan. El uso de clases nos facilitará en primer lugar el encapsulado de los datos con objeto de minimizar el impacto de las transformaciones en el resto del código. Además, las cualidades de sobrecarga y polimorfismo nos ayudarán en la tarea del modelado temporal.

Empecemos considerando cuántas nuevas clases de tipos de datos necesitaremos y sus datos miembro. Inicialmente evaluó la posibilidad de utilizar una clase única para todos los tipos de datos. El resultado de los operadores, como por ejemplo una multiplicación, es equivalente tanto si el dato es de tipo "short", "int" o "long int". Sin embargo, la precisión de los tipos de datos no es la misma. No es igual almacenar el resultado de dividir dos números en un tipo entero que en un tipo flotante, ya que el primero no tendrá decimales. De la misma forma, si multiplicamos una variable de valor 1000 por si misma, obtendremos 1 millón si los tipos de datos son "int", pero 0 si son "short int", ya que no es posible guardar ese número en una representación de 16 bits.

Además el tiempo requerido para realizar una operación puede cambiar en función del tipo de dato. En un procesador de 32 bits no cuesta lo mismo computar con un "int" (32 bits) que con un "long long" (64 bits). En el primero una suma puede realizarse en un ciclo, mientras que en el segundo es necesario sumar las partes bajas (bits 0 - 31) primero y luego las partes altas (bits 32-63) más las llevadas, lo que en conjunto requiere más de un ciclo. Esto es extensible a procesadores con 8 o 16 bits, procesadores con varios conjuntos de instrucciones de distinto tamaño (como ARM), procesadores sin coprocesador de punto flotante, etc.

Una posible solución alternativa a la creación de una clase distinta por cada tipo de dato es el uso de una plantilla ("template"). De esta forma, la variable miembro que almacena el tipo de dato tendría el tamaño conveniente, y además sabiendo en tipo de la plantilla se podría asociar el coste correspondiente con el tipo de dato.

Sin embargo esta solución presenta varios problemas. Hay que tener en cuenta, por ejemplo, que no todos los tipos de datos aceptan todas las operaciones (e.g. no se puede

dividir un número por un booleano). Además, las conversiones automáticas de tipos utilizando plantillas son mucho más problemáticas, requiriendo re-codificación manual.

6.2.3.2.1.3 Conversiones de tipos

Uno de los principales inconvenientes de la técnica de sobrecarga de operadores está relacionado con las conversiones implícitas de tipos de datos. Para tipos de datos básicos el compilador se encarga de realizar las conversiones automáticamente siempre que sea posible. De esta forma, si queremos sumar un entero (int) con un entero corto (short int) y guardarlo en un flotante el compilador hace las transformaciones adecuadas y realiza la operación correctamente.

Sin embargo, cuando sustituimos los tipos de datos por las nuevas clases sobrecargadas el compilador no es capaz de resolver estas inconsistencias automáticamente. Considerando que el objetivo es realizar el modelado de SW sin que el usuario tenga que intervenir, la técnica de modelado ha de resolver el problema automáticamente. Para resolver esta limitación dividiremos las conversiones en dos tipos: manejo de variables y manejo de punteros.

6.2.3.2.1.4 Manejo de variables

La gestión de las conversiones implícitas de variables que debe realizar la anotación por sobrecarga de operadores puede resumirse en tres casos, en función de su lugar de uso: llamadas a función, instrucciones de control y operaciones de cómputo de datos.

Para poder realizar una llamada al sistema operativo o a una librería externa debemos incluir la declaración de dicha funciones en el código. En un código SW sin sobrecargar convencional estas llamadas no requieren ninguna consideración especial, salvo proporcionar el número de argumentos correcto y unos tipos de datos adecuados o implícitamente convertibles. Sin embargo, al aplicar el modelado por sobrecarga de tipos, nos encontramos con que mientras las variables del código modelado han sido transformadas en clases sobrecargadas, las declaraciones de las funciones tienen los tipos básicos originales. Por tanto es necesaria una conversión de los tipos sobrecargados a los tipos básicos originales para poder realizar las llamadas a función.

Una librería de tipos sobrecargados ha sido diseñada para poder realizar estas conversiones automáticamente. Cada vez que un tipo de dato sobrecargado requiera ser manejado con el tipo original, el compilador realiza la conversión automáticamente. Para ello C++ dispone un operador de conversión implícita. El operador de conversión se puede definir en una clase como solución cada vez que el compilador considere que el tipo sobrecargado no es válido para realizar una operación, pero si lo sea el “tipo_basico”. Denominado como “tipo_basico & operator ();”, este operador es llamado automáticamente por el compilador cada vez que se requiera la conversión, utilizando el resultado de retorno de la función como el dato con el que computar. Todos los tipos sobrecargados han sido programados

conteniendo este operador, devolviendo el valor con el tipo de dato original cada vez que sea necesario.

El segundo tipo de conversión implícita se encuentra en las instrucciones de control. Todas las instrucciones de control condicional requieren un valor entero (o booleano) en el que se decide si se realiza o no el salto, y, en su caso, a donde. Cuando en lugar de indicar el resultado de la condición con un booleano se proporciona un tipo de dato sobrecargado, el compilador no es capaz de detectar si se ha de saltar o no. Es por ello que se requiere una transformación automática del tipo sobrecargado al tipo original. Esta transformación se puede realizar de la misma forma que en el caso anterior para las llamadas a función. Por tanto, la resolución del caso anterior también sirve para este caso.

El tercer caso se da en operaciones de cómputo de datos. Cuando se opera con tipos de datos compatibles, pero no exactamente iguales, el compilador realiza las conversiones automáticamente. Al sumar un “char” a un “int”, el compilador automáticamente transforma el “char” a “int” para a continuación sumar dos “int”. Al utilizar tipos sobrecargados, esto se traduce en que queremos sumar un “generic_char” a un “generic_int”. Por simetría, la solución directa sería realizar una transformación del “generic_char” a “generic_int” para luego sumarlos, utilizando un constructor adecuado.

Sin embargo al compilador se le presentan varias posibilidades además de la conversión propuesta. Podría convertir el “generic_char” a “char” y luego utilizar el constructor de “generic_int” para operar, o transformar ambos tipos sobrecargados a tipos básicos y luego operar. El número de posibilidades aumenta si alguno de los tipos tiene algún declarador adicional, como “unsigned”. El resultado del caso anterior es que el compilador no sabe cual de las soluciones escoger y da un error por múltiples posibilidades. Incluso en algunos casos, aunque solo detecte una posible combinación de transformaciones, si esta requiere varios pasos el compilador falla.

La solución adoptada consiste en utilizar el polimorfismo en los operadores. En el caso anterior, además de definir en la clase “generic_int” el operador suma como “generic_int operator + (generic_int);” también habría que definir un operador “generic_int operator + (generic_char);”. De esta forma, cuando sumemos dos enteros sobrecargados se utilizará el primero, y cuando sumemos un “char” a un entero se utilizará la segunda. De esta forma evita la necesidad de conversión implícita.

Esta técnica se ha aplicado a todos los tipos sobrecargados, para evitar el problema. Para cada uno de los tipos de datos sobrecargados creados se ha realizado una implementación polimórfica de todos sus operadores binarios. Esta implementación polimórfica implica una implementación distinta para cada posible tipo de datos que se pueda recibir como argumento, a saber, todos los tipos básicos más los tipos sobrecargados.

Para hacernos una idea del número de operadores requeridos para realizar una implementación completa consideremos:

- Existen 9 tipos de datos básicos: bool, char, shor int, int, long int, long long, float, double, long double.
- Si contamos con sus equivalentes "unsigned" tenemos 18 tipos básicos.
- Para realizar una sobrecarga completa necesitaremos 18 nuevos clases para los tipos sobrecargados.
- Cada clase tiene unos 6 operadores unarios y 30 binarios (alguna clase tiene menos, pero lo despreciaremos para simplificar los cálculos)
- Cada operador binario requiere una implementación distinta en función del argumento: 18 para los tipos básicos y 18 para los tipos sobrecargados

En total el número de funciones de operadores binarios es: 18 tipos * 30 operadores * (18+18) posibles argumentos = 19440 funciones. Aunque en la tesis no se implementaron los 18 tipos, la codificación de todas estas funciones requirió del uso de "defines" para simplificar la solución. Hay que tener en cuenta que este calculo considera únicamente las funciones para contabilizar los operadores del tipo "variable operador variable" (a + b) y "variable operador inmediato" (a + 1). Si además consideramos los tipos "inmediato operador variable" (1 + a) el numero de operaciones alcanzarían las 30000.

6.2.3.2.1.5 Manejo de punteros

El manejo de punteros, tanto para el propio modelado temporal como para la conversión de tipos es el punto más limitado de la técnica de sobrecarga de tipos. Dado que los punteros no se pueden sobrecargar en C++ sus operaciones no pueden ser interceptadas mediante sobrecarga de operadores. No obstante, el número de operaciones entre punteros es habitualmente despreciable, ya que dos punteros no se pueden sumar, ni dividir, ni admiten la mayoría de las operaciones convencionales de cómputo de datos.

El acceso a un puntero, especialmente en el caso de los arrays, implica un aumento del número de instrucciones ensamblador requeridas. Para minimizar este error de estimación se ha utilizado un efecto colateral, ya que la sobrecarga directa no es aplicable a punteros. Al acceder a una posición de un array, el índice debe ser un tipo entero básico, no sobrecargado, al igual que para las operaciones de control explicadas anteriormente. Por tanto, los accesos a array llevan aparejados una ejecución del operador de conversión de tipo.

Teniendo en cuenta que el operador de conversión de tipo se utiliza casi exclusivamente en llamadas a función e instrucciones de control, podemos introducir el coste de acceso a posición de array en el operador de conversión. Para compensar el efecto en el modelado del resto de ocasiones que se usa dicho operador, en la estimación de las instrucciones de control y los saltos a función se resta el coste de más añadido por el operador de conversión al creer erróneamente que se trataba de un acceso a array. Esta solución, si bien no resuelve perfectamente el problema, reduce el índice de error introducido por los accesos a array.

Otro punto a tener en cuenta en la gestión de punteros es que el tipo sobrecargado puede no tener el mismo tamaño que el tipo base original. El sistema de sobrecarga propuesto

ha sido diseñado para soportar extensiones, de tal forma que cada objeto pueda almacenar información adicional, además del propio valor de la variable. Esta información adicional puede contener datos para la generación de estadísticas o históricos de acceso a las variables, necesario para la estimación de descripciones HW, como se verá en el siguiente capítulo.

El principal problema derivado de estas extensiones se debe a que el tamaño de un buffer de tipos básicos es habitualmente menor que el de su equivalente buffer de tipos sobrecargados. El tipo sobrecargado en si es más grande que el tipo original dado que almacena información adicional. Esto significa que al gestionar un buffer de datos, es necesario saber si los datos que contiene son datos originales o datos sobrecargados. Las operaciones de reserva de memoria (“malloc”, “calloc”) de copia (“memcpy”), de asignación (“memset”) o lectura y escritura de flujos de datos (“read”, “write”), entre otras, han de recibir como parámetro el tamaño del buffer. Por tanto, al transformar automáticamente los tipos básicos en tipos de datos sobrecargados es necesario modificar también el tamaño indicado.

El conocimiento del tipo de dato que se esta utilizando es normalmente posible mediante la operación “typeid (var).name ()”. De esta forma podemos obtener una cadena de caracteres que nos indique si el tipo está sobrecargado o no y actuar en consecuencia. Sin embargo, hemos de tener en cuenta que en muchas ocasiones los búferes se convierten a tipo “void*”, con lo que la información de la sobrecarga se pierde. De esta forma, al realizar una operación no sabemos si hemos de utilizar el tamaño original o hemos de modificarlo para un tipo sobrecargado.

Para solucionar este problema se ha añadido a cada tipo sobrecargado una nueva variable miembro, llamada número mágico. Se ha definido un número mágico distinto para cada tipo de datos. De esta forma, la primera variable de cada tipo sobrecargado contiene un valor específico de 32 bits que identifica a la variable como un tipo sobrecargado. Para detectar si un buffer esta compuesto de tipos sobrecargados, basta con leer la primera posición del buffer. Si no se corresponde con el número mágico entonces el buffer esta compuesto de elementos de tipo básico; si es un número mágico, se considera que son tipos sobrecargados, y por tanto hay que modificar el tamaño del buffer adecuadamente.

Esta solución no es completa, ya que puede darse el caso de que el valor contenido por una variable de tipo básico sea casualmente igual que un número mágico. No obstante, la posibilidad de coincidencia es en teoría de 1 entre 400 millones considerando el tamaño de un entero. En la práctica, para valores grandes esta relación es aún mayor, ya que normalmente los enteros se usan mucho más conteniendo valores relativamente pequeños que valores grandes. En caso de que se quiera minimizar esta posibilidad al máximo se puede convertir el número mágico en un “long long” (64 bits) o incluso un “long double”.

Por último, esta técnica implica una limitación en el código. Aunque en C/C++ es posible inicializar las cadenas de caracteres en su construcción, la sobrecarga de tipos anula esta posibilidad. La asignación directa se realiza en C/C++ colocando la cadena indicada a partir de la primera posición del array. Sin embargo, al sustituir los “char” por tipos sobrecargados, ya no es posible suponer que la primera posición de memoria (addr)

corresponde al primer carácter, la siguiente (addr + 1) al segundo, la siguiente (addr + 2) al tercero y así sucesivamente. Dichas posiciones dependen del tamaño del nuevo tipo. Por tanto, la asignación de una cadena al instanciar un array de tipos sobrecargados provoca un error en compilación. Así, las asignaciones deben ser modificadas manualmente en el código o mediante un pre-procesador específico.

6.2.3.2.2 Estimación del coste de las operaciones de control

Para modelar el coste temporal de las operaciones de control las dividiremos en dos grupos: instrucciones de control y saltos a función. Esta división se basa en la forma de detectar los elementos de ambos grupos. Mientras que las instrucciones de control son fácilmente detectables, ya que se corresponden con palabras reservadas, las llamadas a función no pueden ser detectadas de esta manera.

6.2.3.2.2.1 Instrucciones de control

La forma más sencilla de añadir la funcionalidad necesaria a la simulación para considerar el coste de ejecución de las instrucciones de control es preprocesar el código mediante "#define". De la misma forma que anteriormente sustituimos los tipos de datos básicos por las clases sobrecargadas ("#define int generic_int"), en esta ocasión sustituiremos las instrucciones de control. Reemplazaremos cada instrucción de control por una secuencia de instrucciones que mantenga la funcionalidad original y que además se encargue de añadir los costes de tiempo de la instrucción a la estimación de tiempo SW (for, if, else,…)

- If/else

Una instrucción "if" puede implicar un salto condicional, lo que conlleva un determinado coste temporal asociado. Por tanto hay que detectar si el salto se produce o no, para añadir el tiempo correspondiente. Que el salto se produzca o no depende del valor de la condición y de la existencia o no de la cláusula "else". Si la condición es verdadera, solo se producirá un salto si hay "else", dado que dicho bloque no debe ser ejecutado. Si la condición es falsa habrá un salto independientemente de la existencia o no de este bloque.

Para realizar la anotación se ha creado una función específica. Utilizando un "define" se reemplaza la condición del "if" inicial por una llamada a esta función. La función recibe dos argumentos, la línea de código donde se encuentra la condición y el valor de la condición. Esta función devuelve el valor de la condición que ha recibido como entrada, que de esta forma se usa como condición del nuevo "if". El resultado es el siguiente:

```
#define if(condition) if(add_branch_time(__LINE__, basic_type(condition)))
```

La condición es usada para evaluar si el salto se produce o no. Dado que determinar durante ejecución si existe el bloque "else" es muy complicado, la técnica implica un cierto error. Si la condición es verdadera se considera la existencia del "else" de forma probabilística. La función de conversión a tipo básico se utiliza como extensión a la técnica de

estimación del coste del acceso a arrays. Como dijimos anteriormente, el coste de dichos accesos se estima detectando la transformación de un tipo estándar en tipo básico. Además de en los arrays y en algunas llamadas a función, es en las instrucciones de control donde se produce ese cambio de tipo, ya que el "if" no acepta un tipo sobrecargado. Para evitar que se suponga erróneamente la existencia de un acceso a array en el if, se utiliza la función "basic_type". Esta función convierte la condición a un tipo básico sin añadir el tiempo de acceso a array.

El uso de la información de línea de código sirve para identificar la instrucción de control, y se emplea para el modelado de caches que se explicará posteriormente.

- Switch

La anotación del switch es similar a la del "if". En este caso se reemplaza la condición del switch por una función de anotación del coste del switch. Se usa además el número de línea y la función de conversión de tipo de la misma forma que antes:

```
#define switch(num) switch(add_switch_time(__LINE__, basic_type(num)))
```

- While / do-while

La anotación del "while" es también similar a la del "if". En este caso se reemplaza la condición del "while" por una función de anotación del coste del "while". Se usa además el número de línea y la función de conversión de tipo de la misma forma que anteriormente. En función del valor de la condición se sabe si se realizará el salto o no. La conversión se realiza de la forma:

```
#define while(cond) while(add_while_time(__LINE__, basic_type(cond)))
```

- For

La anotación del "for" parte de la misma idea anterior, aunque es un poco diferente. Debido a la estructura del "for" ("for(a; b; c)") el reemplazo de la condición del salto por una función de anotación no es posible. No obstante, sabemos que en todo "for" se producirán n saltos hacia atrás y uno hacia delante. Habrá tantos saltos hacia atrás como veces se ejecute el bloque "for", y uno hacia delante al acabar el "for" también similar a la del "if". Sabemos que la primera parte del "for" ("a") se ejecuta una sola vez, mientras que la tercera ("c") se ejecuta tantas veces como se ejecute el bloque "for". Por tanto vasta con añadir el coste del salto hacia atrás a la tercera parte del "for" y el salto hacia delante a la primera. Esto puede hacerse mediante:

```
#define for(...) \
        for(add_jump_time(__LINE__), __VA_ARGS__, add_for_time(__LINE__))
```

Esta solución impone una restricción al uso de los "for" en el código a analizar. Todo "for" debe tener primera y tercera cláusula. Es decir, no están permitidos los del tipo "for

(;;)". Estos han de ser reemplazados por "for (0; ;0)". De otra forma, la modificación realizada por la macro provocaría un error de compilación.

- Break / continue / return

Estas instrucciones representan un salto no condicional a una posición determinada: el comienzo de un lazo, el final de un lazo, o el final de una función. Por ello, su coste temporal es el correspondiente salto en la arquitectura destino. Dado que estos saltos ya están considerados por la anotación propuesta tanto en los lazos como en el salto a función, no es necesario sobrecargar estas instrucciones. Por ejemplo, un salto hacia atrás en un "for" tiene el mismo coste temporal si se ejecuta debido a un "continue" que por llegar al final del "for".

6.2.3.2.2.2 Llamada a función

Las llamadas a función no pueden ser detectadas por el pre-procesador de C, ya que no se corresponden con palabras reservadas. Es por ello que no se pueden interceptar mediante defines. Sin embargo, la sobrecarga que implican los saltos a función en el tiempo de ejecución de un programa puede ser muy importante, por lo que debe ser contabilizada. Para contabilizarla se ha utilizado la opción de gcc "-finstrument-function". Al compilar un código C/C++ con esta opción, el usuario puede definir dos funciones que son llamadas cada vez que se realiza un salto a función (la primera) y cada vuelta de función (la segunda). Haciendo que dentro de estas dos funciones adicionales se anote el coste de un salto y una vuelta de función, la sobrecarga introducida por los saltos a función del programa puede ser modelada.

6.2.3.2.3 Obtención del coste estimado de un segmento

Como hemos visto anteriormente, las operaciones de C pueden ser sobrecargadas para permitir la estimación del tiempo de ejecución del SW. De esta forma puede ser calculado el tiempo de cada segmento de código. Para ello deberemos tener una variable global donde guardemos el tiempo acumulado del segmento.

Cada vez que volvamos de una llamada al sistema, comenzaremos la estimación de un nuevo segmento. En este punto pondremos la variable global que contiene el coste del segmento a 0. A partir de aquí, cada operación SW sobrecargada, añadirá a dicha variable el coste de la operación correspondiente. Cuando entremos a una nueva llamada al sistema, el cambio del modo de ejecución, de modo usuario a modo kernel, indicará el fin del segmento (Ver capítulo 7 para más información). En este punto, el valor de la variable global será el resultado de sumar el coste de todas las operaciones SW ejecutadas en ese segmento.

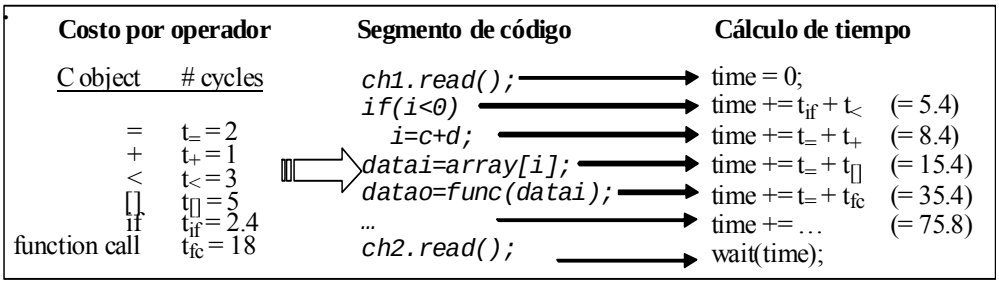


Figura 6-8: Cálculo del retraso de un código.

La asociación de un coste a cada operador es fácilmente extensible para modelar operaciones como la multiplicación, división u operación modular. Estas operaciones, que pueden tener un coste constante en algunos procesadores, en otros en los que no exista un co-procesador adecuado pueden presentar costes no constantes, sino representados por funciones lineales o cuadráticas del tamaño de los datos de entrada. Dado que la suma del coste se realiza dinámicamente y los datos de entrada a la operación son conocidos, este reajuste no presenta ningún coste extra en la simulación.

En resumen, podemos decir que la técnica de sobrecarga de operadores tiene la ventaja de que maneja cada operación del código y es capaz de acceder a toda la información posible, dándole una tremenda flexibilidad. Esto permite una gran capacidad de generación de estadísticas, manejo de caches con gran precisión, e incluso su extensión para el modelado de segmentos de código orientados a una implementación HW. Sin embargo, para un análisis rápido del código fuente realiza una sobrecarga, que si bien es menor que la de un ISS, es relativamente alta, por lo que la razón principal por la que esta técnica no está presente en la versión actual de SCoPE es un problema de velocidad de simulación.

Hay que tener en cuenta que la técnica propuesta reemplaza cada operación básica de C por una llamada a un operador sobrecargado. Esto implica una sobrecarga de simulación importante, dado que el código se convierte en una ejecución permanente de operadores sobrecargados. Por ello es muy importante optimizar al máximo el tiempo de ejecución de cada operador. Esta restricción conlleva la idoneidad de tener clases específicas por cada tipo de dato en lugar de una única plantilla parametrizable.

6.2.3.3 Anotación con análisis de código fuente

La técnica de anotación con análisis de código fuente es una técnica más rápida que la de sobrecarga de operadores. Para conseguir una baja sobrecarga en simulación es necesario realizar parte del análisis del tiempo del segmento en estático, antes de compilar el código a simular. Por ello la técnica de estimación comienza con el preprocesado del código mediante un analizador estático. De esta forma se puede obtener el coste de cada bloque básico de código. Posteriormente, durante la ejecución de cada bloque básico se toma su coste y se añade al del resto del segmento.

Al igual que en la técnica de sobrecarga de operadores, la técnica de estimación se basa en asignar un coste temporal a cada operador C y estimar el coste total de cada segmento sumando el de los operadores ejecutados en el segmento. El coste de cada operador se calcula de la misma forma que se mostró en la sección previa. Además, como antes, los efectos de las optimizaciones del compilador son difíciles de estimar desde el análisis del código fuente. Por ello también se mantiene la aplicación de un factor de ajuste para considerar la mejora introducida en el código por las optimizaciones.

Una vez obtenido el tiempo del segmento puede ser utilizado de la misma forma que con la técnica anterior para imbuir el comportamiento temporal a la simulación de SW. Por tanto, ambas técnicas pueden ser fácilmente intercambiables, obteniendo diferentes relaciones de precisión / velocidad, manteniendo el resto del sistema de modelado intacto.

Esta técnica explota la cualidad del código SW de que dentro de un mismo bloque básico se ejecutarán siempre todas sus instrucciones. Esto significa que podemos añadir por cada bloque básico el tiempo correspondiente a todas sus instrucciones internas, sin preocuparnos de cómo se ejecutará cada una. Así se modifica la granularidad del análisis, disminuyendo el detalle y aumentando la velocidad.

No obstante, este aumento de granularidad produce un aumento del error menor de lo que se podría pensar. Esto se debe a la propia definición de segmento dada anteriormente. Hemos de tener en cuenta que la técnica propuesta se basa en ejecutar un segmento de código íntegro, estimarlo, y por último introducir el tiempo correspondiente en la simulación. Por ello, el aumento de granularidad de operación a bloque básico no afecta a la integración de los tiempos de ejecución estimados en el modelado del sistema.

6.2.3.3.1 Proceso de análisis

Para realizar la estimación basada en análisis de código fuente se han de realizar una serie de tareas (Figura 6-9). La secuencia completa es:

1. Pre-procesar el código
2. Anotar el código añadiendo la información por bloque básico
3. Compilar el código anotado
4. Ejecutar

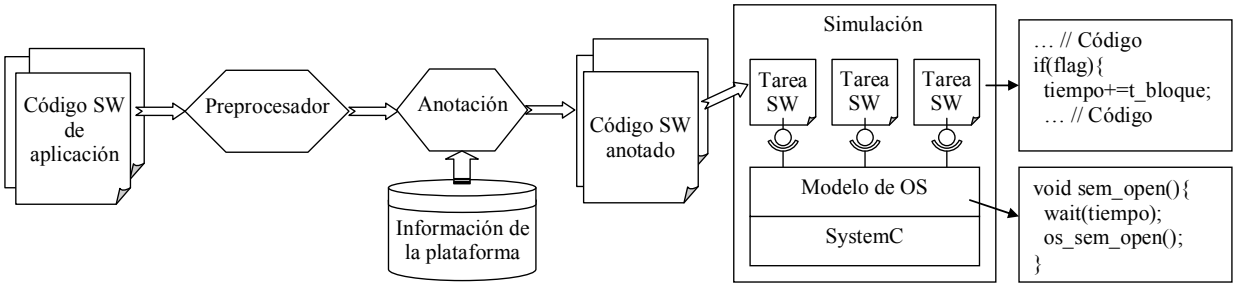


Figura 6-9: Modelado temporal con análisis de código fuente

El pre-procesado debe ser realizado en primer lugar para permitir la anotación del código. Esta anotación se realiza a través de un análisis gramatical del código. Para que este análisis sea posible y considere todos los elementos del código, es necesario trabajar con el código completo. Para ello, todas las cláusulas de pre-procesador, especialmente los "define", deben ser aplicados al código, a fin de tener un código fuente gramaticalmente completo.

Una vez tenemos el código pre-procesado se realiza el análisis y se añade la información de tiempo por cada bloque básico. Posteriormente, el código puede ser compilado y ejecutado para obtener un modelo temporal del SW.

Para realizar la anotación del código debemos tener en cuenta el código que recibimos como entrada. Dividiremos este código en tres categorías:

- Cabeceras auxiliares y de sistema
- Código SW para modelar
- Código SW que no debe ser modelado.

Al pre-procesar el código antes de realizar el análisis, no solo recibimos el código fuente original. Además, el código contiene todas las cabeceras del sistema que hayan sido directa o indirectamente incluidas. En consecuencia, el código anotado preprocesado puede tener un tamaño muy superior al original. Este incremento de líneas de código conlleva un aumento considerable del tiempo requerido para realizar el análisis gramatical de forma innecesaria. Estas cabeceras no contienen código ejecutable, sino únicamente declaraciones de funciones y variables, por lo que no han de ser analizadas ni anotadas.

Además, no todo el código SW ha de ser modelado necesariamente. En un modelo de sistema podemos tener un código SW que no queremos que influya en el análisis del sistema, ya sea para facilitar el análisis o porque dicho componente no va a estar presente en el sistema real. Además, en determinadas ocasiones nos podemos encontrar con códigos que ya han sido estimados y anotados externamente, de modo que no deben ser re-anotados.

Para identificar qué parte del código ha de ser re-anotado se han creado dos cláusulas adicionales:

"#pragma START_ANALYSIS" y "#pragma END_ANALYSIS"

Con estas cláusulas, el análisis se limita al código delimitado por dichas cláusulas, haciendo más eficiente el análisis. El uso de "#pragma" hace que puedan ser definidas en el código original, mantenidas por el pre-procesador e ignoradas por el compilador. De esta forma solo afectan a la anotación del código.

6.2.3.3.2 Estimación del tiempo de bloque básico

Para realizar la anotación se ha partido de una gramática C++. Dicha gramática está descrita para ser utilizada aplicando "flex" [Flex] y "byson" [Bison], para los análisis léxicos

y gramaticales. La gramática de la que se ha partido en este trabajo es la desarrollada por Ed Willink [Willink] como parte de su tesis. Esta gramática ha demostrado ser muy completa. Salvo alguna extensión requerida para dar soporte a elementos del tipo "__atribute__ ()" de C++, los ejemplos a los que se ha aplicado han demostrado que la gramática soporta completamente un código C++.

Aunque un análisis completo requeriría una gestión completa de los elementos de la gramática, en esta tesis nuestro objetivo es simplemente demostrar la viabilidad de la utilización de esta técnica mixta como técnica de estimación SW para exploración rápida de sistemas. Por ello, solo se ha utilizado una parte de la capacidad de la gramática. En concreto, nos hemos limitado a utilizar la gramática para detectar los bloques básicos, las operaciones de cómputo y las instrucciones de control.

El objetivo es realizar un análisis estático de cada bloque básico antes de la compilación. En este análisis se obtiene el coste estimado para cada bloque básico (Figura 6-10). Una vez analizado el código, se reconstruye, introduciendo en el propio código unas instrucciones adicionales que se encargan de añadir el tiempo del bloque al tiempo total del segmento.

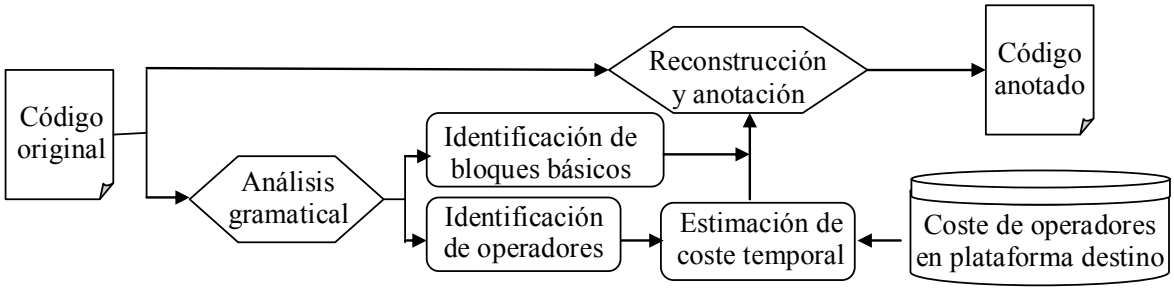


Figura 6-10: Estimación con análisis de código fuente

Dado que la técnica se basa en añadir instrucciones a cada bloque básico, lo primero que se ha añadido en la gramática es la capacidad de reconstruir el código de tal forma que todo bloque básico está encerrado entre llaves, aunque el original careciera de ellas.

A continuación, en toda regla gramatical de operación de cómputo se ha añadido una función que incrementa el coste del bloque en curso en el coste correspondiente a dicha operación. La modificación implementada no tiene en cuenta el tipo de los datos involucrados en la operación. Esta limitación puede ser resuelta mediante la generación de una tabla de variables. No obstante, para comprobar las posibilidades de la técnica de anotación con análisis de código fuente esta extensión no es estrictamente necesaria.

De la misma forma, en cada regla gramatical de control se ejecuta un incremento de la variable de coste del bloque básico. Las reglas gramaticales de control además indican el comienzo y el final de los bloques básicos, junto con las llaves de delimitación de bloque. El coste de las reglas se ha de considerar o bien en el bloque anterior, en el bloque siguiente o en ambos. En nuestro caso, las operaciones de control condicional han sido consideradas en el bloque siguiente, mientras que las de salto incondicional se consideran en el bloque anterior.

Esto se debe a que después de una operación de salto incondicional no existe un nuevo bloque. Cuando se ejecuta un "return", "break" o "continue", se salta a un bloque en una posición más o menos distante de dicha instrucción, de forma que no es posible anotar el coste del salto en el bloque destino.

6.2.3.3.3 Anotación del código fuente

Con el tiempo de bloque básico estimado, y teniendo delimitado dicho bloque debemos introducir la información temporal en el código. En este punto debemos tener en cuenta la sobrecarga temporal que implican las operaciones que requiere la técnica de modelado.

En ejecuciones grandes, la sobrecarga introducida por el análisis estático no es excesivamente costosa. Esto se debe a que cada bloque básico se analiza una sola vez mientras que durante la simulación puede ejecutarse muchas veces. Por ello es crucial limitar la sobrecarga en simulación. Cualquier código adicional que añadamos para realizar la anotación supondrá un aumento del tiempo de simulación. Esto es especialmente importante si consideramos que habitualmente los bloques que más se repiten en un código suelen ser los que se encuentran dentro de varios lazos anidados, normalmente bloques pequeños. Al Debido a su pequeño tamaño, cualquier código adicional tiene gran repercusión en la ralentización de la simulación.

Para minimizar este problema la anotación de cada bloque básico se realiza añadiendo el coste del bloque básico a una variable global que se indica se mantenga en un registro del procesador. De esta forma la sobrecarga es realmente mínima. Esto se realiza añadiendo al buffer de regeneración del código anotado la información siguiente:

```
sprintf(buffer, "uc_segment_consumed_time+=%d;", block_time);
```

6.2.3.4 Anotación con análisis de código binario

Para mejorar la precisión de las técnicas de anotación anteriores, la siguiente solución analizada e implementada se basa en sustituir el análisis de operadores de código fuente por análisis de código binario. Para considerar con precisión efectos del compilador cruzado como las optimizaciones debemos trabajar con el código compilado en lugar de usar código fuente. Sin embargo, el modelado a partir de código binario presenta en general dos limitaciones: es lento y muy dependiente del procesador.

Para evitar estas limitaciones se ha propuesto una técnica híbrida basada en la técnica anterior de anotación con análisis de código fuente. La anotación y la identificación de los bloques básicos se realizarán a partir del código fuente, como anteriormente (figura 6-9). La caracterización de los bloques básicos se realizará a partir del código binario generado por el compilador cruzado.

Esta aproximación híbrida tiene el problema de que la correlación entre el código fuente y el código compilado es en ocasiones muy compleja [Cifuentes]. Esto se debe a capacidades del compilador como el reordenamiento de instrucciones y la eliminación de código muerto. La compilación sin optimizaciones permite identificar puntos en el código binario insertando etiquetas en el código fuente. Sin embargo, las optimizaciones tienen la capacidad de mover o incluso eliminar esas etiquetas. Por ejemplo, si insertamos una etiqueta en un lazo, y aplicamos una optimización de desenrollo de lazos, la etiqueta pierde su sentido.

Para mantener la correlación entre el código fuente y el binario se ha decidido marcar el código mediante etiquetas de la forma:

asm volatile("etiqueta_xx:");

La utilización de etiquetas volátiles obliga al compilador a mantener las etiquetas en el sitio apropiado. De esta forma, insertando etiquetas al principio y al final de cada bloque básico podemos obtener fácilmente el número de instrucciones ensamblador del bloque básico. La identificación de bloques básicos se realiza mediante un análisis gramatical, igual que en la técnica de estimación de análisis por código fuente.

Considerando que la mayoría de procesadores embebidos tienen un tamaño de instrucción constante, podemos calcular el número de instrucciones mediante la siguiente ecuación:

Num_instrucciones = (etiqueta_final – etiqueta_inicio) / tamaño_de_instrucción

La obtención del valor de las etiquetas puede realizarse fácilmente mediante el comando:

*readelf –s código_binario.o | grep etiqueta_*

Para procesadores de tamaño de instrucción variable puede obtenerse el número de instrucciones a partir del código ensamblador, contando las instrucciones que hay entre las marcas.

La estimación del tiempo necesario para ejecutar cada bloque básico en la plataforma destino se obtiene multiplicando el número de instrucciones por el número de ciclos por instrucción (CPI) proporcionado por el fabricante. Aunque esta solución conlleva un pequeño error, al no considerar de efectos internos del procesador, como paradas por dependencias de datos, tiene la ventaja de ser rápida y genérica. Para evaluar el comportamiento de un programa en un procesador solo es necesario disponer de un compilador cruzado. No es necesario disponer de librerías o sistemas operativos adaptados ni de simuladores binarios como ISSs.

Una vez obtenido el tamaño de cada bloque básico solo resta anotar la información en el código inicial.

En resumen, la técnica de anotación con análisis de código binario comparte con la de análisis de código fuente los mecanismos de identificación de bloques básicos y de anotación de la información en el código. La diferencia principal está en el mecanismo de estimación del tiempo por bloque básico. Esto se puede ver en la figura 6-12.

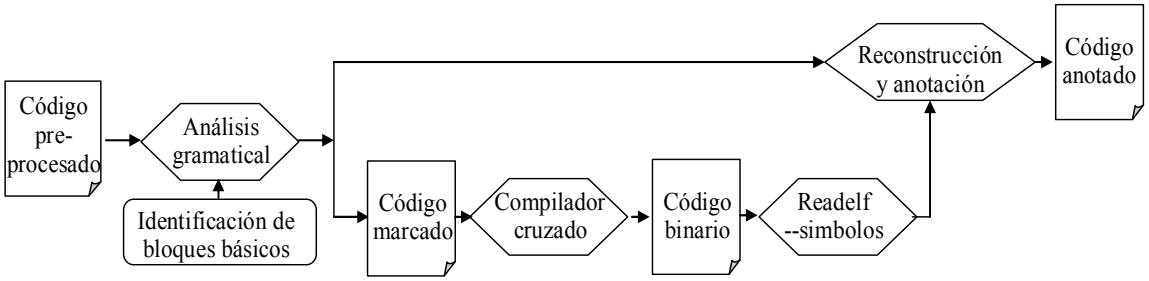


Figura 6-12: Estimación con análisis de código binario

La introducción de etiquetas volátiles permite la correlación entre código fuente y código binario a costa de impedir la aplicación de algunas optimizaciones. La mayoría de las optimizaciones, como la eliminación de accesos a memoria mediante la reutilización de registros, son correctamente modeladas. Únicamente unas pocas optimizaciones, con efectos menores, no pueden ser modeladas. El despliegue de lazos no es posible, aunque su uso para procesadores con cache no es habitual, ya que aunque reduce el número de saltos, incrementa los fallos de cache. El reordenamiento de instrucciones para evitar dependencias de datos tampoco es modelado, pero dado que los efectos internos del procesador no son modelados, esta optimización no tiene efecto en la técnica de estimación.

La técnica de anotación con análisis de código ensamblador ha sido desarrollada en colaboración con Juan Castillo, que ha desarrollado la implementación definitiva de la técnica, trabajado especialmente en la delimitación correcta de bloques básicos complejos, como por ejemplo los de la cláusula “for” (Figura 6-13).

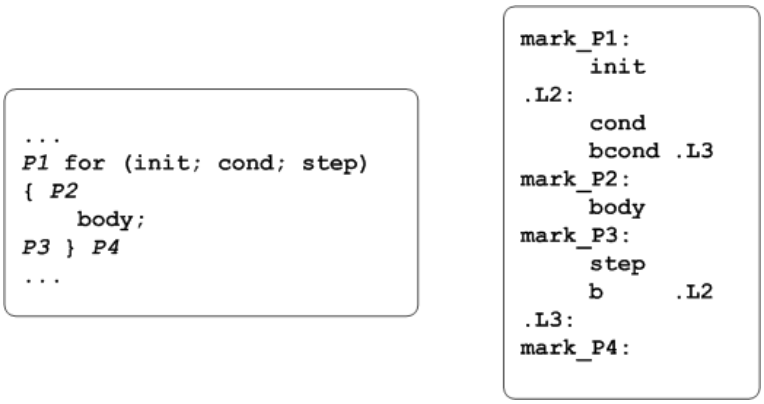


Figura 6-13: Colocación de marcas para la identificación de un lazo “for”

6.2.3.5 *Modelado con información de ISS*

Tras desarrollar las técnicas anteriores, un aumento de la precisión ha sido posible mediante la utilización de un ISS. Por tanto podríamos obtener mejores resultados si estimamos el tiempo de ejecución de cada segmento de código mediante la ejecución antes de la anotación del código binario obtenido mediante compilación cruzada. Además, deberemos aumentar la granularidad de la anotación del bloque básico, al segmento completo. Este aumento permitirá considerar el comportamiento del procesador en los pasos de un bloque básico a otro.

Evidentemente, la utilización de una simulación en ISS antes de la simulación de alto nivel neutraliza cualquier incremento de velocidad conseguido por la simulación de alto nivel. Sin embargo, en procesos de exploración del espacio de diseño se deben realizar un gran número de simulaciones del mismo código SW modificando los parámetros de la plataforma. Dado que el código SW es el mismo, se puede utilizar la información de una única ejecución en ISS para todas las simulaciones de la exploración. Tan solo será necesario multiplicar por un factor de ajuste si queremos modificar la frecuencia de funcionamiento del procesador. El resto de elementos, como la arquitectura del sistema, la velocidad de la memoria, o el ancho de banda del bus no afectan al tiempo de procesador.

En consecuencia, al realizar exploraciones de miles de puntos una única ejecución inicial del código sobre un ISS aumentará la precisión sin repercutir en exceso en el tiempo completo de la exploración.

Para implementar esta técnica se ha añadido a SCoPE la posibilidad de integrar la librería diseñada por IMEC a tal efecto, llamada ARP (Atomium Record/Playback library). Para utilizar esta técnica hay que comenzar anotando el código con las llamadas a la librería ARP en los bordes de los segmentos de código. Una vez generado el código anotado se realiza una compilación cruzada y se ejecuta sobre el ISS, indicándole a la librería que debe almacenar la información de tiempo (Figura 6-13).

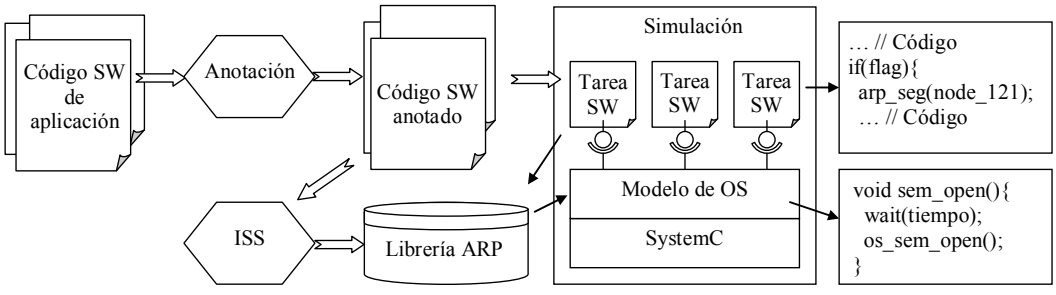


Figura 6-13: Modelado temporal con análisis de código fuente

Para realizar la exploración se integra el código anotado en SCoPE y se selecciona el modo “retornar tiempos” de la librería ARP. Estos tiempos son utilizados en SCoPE como entrada a las funciones encargadas a llamar a las funciones “wait” de SystemC.

Esta técnica es la más compleja de utilizar, ya que necesita de una infraestructura completa para poder ejecutar el código en el ISS. Debemos disponer del compilador cruzado, el ISS, un sistema operativo adaptado, y todas las librerías necesarias para la ejecución del código.

La información sobre la utilización de la librería ARP es confidencial, por lo que no presentaremos los detalles internos.

6.3 Modelado de caches

Las caches son una de las soluciones más usadas para mejorar el rendimiento de los procesadores, al reducir el número de accesos a memoria. Es por ello, que para obtener estimaciones precisas del tiempo de ejecución del código SW debemos tener en cuenta el efecto de las caches.

El modelado de las caches tiene por objetivo considerar el tiempo adicional que una cache real añade al tiempo de ejecución de un programa con respecto a la ejecución ideal. Este tiempo se debe a la existencia de fallos de cache (miss). Cada vez el procesador requiere un dato de la cache y dicho dato no esta disponible es necesario acceder a la memoria principal. Esto provoca dos efectos que han de ser integrados en la simulación: por un lado se produce una parada del procesador y por otro lado se produce un acceso al bus. La parada del procesador hasta que el dato está disponible produce un aumento del tiempo requerido para ejecutar el código de aplicación. Los accesos al bus han de ser modelados para poder analizar los retrasos y colisiones producidas.

Por tanto, el modelado de la cache debe ser capaz de estimar el número de mises de cache que se van produciendo conforme avanza el programa e inicial los accesos a memoria correspondientes en el modelo de plataforma. No obstante, este proceso no debe representar una sobrecarga excesiva en el tiempo de simulación.

Si consideramos un sistema convencional, el uso de caches hace que habitualmente el CPI del procesador se sitúe entorno a 0.6 – 1, siendo 1 el CPI de una cache infinita con todos los datos cargados. Esto significa que si aplicamos un factor de 0.8 a las estimaciones obtenidas con las técnicas anteriores, no modelar la cache produciría un error máximo del 20%.

Teniendo en cuenta que las técnicas de estimación de alto nivel propuestas en las secciones anteriores están pensadas para proporcionar estimaciones con errores del 10-20%. En conclusión, podemos decir que el modelado de las caches debe permitir una cierta reducción del error siempre y cuando no conlleve una sobrecarga excesiva en el tiempo de simulación. Es preferible un modelado de caches sencillo que reduzca el error lo justo para ser inferior al error propio de la estimación sin caches, que un modelado complejo que ralentice la simulación sin producir beneficios tangibles.

En esta tesis se han considerado procesadores con caches separadas para instrucciones y datos. En primer lugar vamos a abordar el modelado de las caches de instrucciones. Posteriormente, consideraremos también las caches de datos.

6.3.1 Modelado de Cache de Instrucciones

Para realizar el modelado de caches de instrucciones se han desarrollado dos técnicas en función del mecanismo de modelado de tiempo utilizados: una técnica para ser utilizada junto con la técnica de sobrecarga de operadores, y otra para las técnicas de anotación.

6.3.1.1 Modelado de caches con la técnica de sobrecarga

La técnica de simulación de sobrecarga de operadores es una técnica de estimación completamente dinámica. Por ello, el modelado de cache debe realizarse utilizando únicamente información disponible durante la simulación. Para ello se ha implementado una técnica basada en un cálculo probabilístico del cambio de línea de cache en función de las operaciones ejecutadas. En primer lugar se evalúa si se produce un cambio de línea en cada operador ejecutado. A continuación, se calcula la probabilidad de que la nueva línea esté o no en cache.

El procedimiento aplicado en cada operación es el siguiente:

- Estimar la posición actual de la instrucción en la línea de cache.
- Estimar una vez ejecutada la instrucción si se ha alcanzado una nueva línea, y la posición en dicha línea.
- En caso de alcanzar una nueva línea, identificar si la nueva línea está en cache o no.

La posición actual en la línea se estima la primera vez que se ejecute la línea, a partir de la posición previa y la instrucción ejecutada. Una vez ejecutada, esta información se guarda con la información de línea. Esto es especialmente útil para vueltas atrás en lazos, ya que al empezar sabemos en que posición estamos si sabemos la de la instrucción anterior. Sin embargo, al volver hacia atrás no tenemos ninguna información, ya que no sabemos el tamaño del salto.

Conociendo la posición en la línea, se estima que se alcanza una nueva línea en función del tipo de operación ejecutada:

- Durante la ejecución de instrucciones consecutivas, alcanzaremos una nueva línea cuando hayamos ejecutado tantas instrucciones como el taño de línea.
- Si cambia el proceso en ejecución, se alcanzará una nueva línea, ya que se cambiará la zona de memoria
- Si se produce un salto, habrá una nueva línea con una probabilidad “p”, que se calcula dependiendo del tipo de salto y la distancia del salto.

Para instrucciones consecutivas, la nueva instrucción está en la posición de la instrucción anterior más el tamaño de la instrucción. Si la posición final está fuera de la línea actual significa que se han producido uno o varios cambios de línea. Por tanto, se accede a la cache a comprobar si estos han producido un fallo de cache o no. Esto se aplica para:

- Operaciones de cómputo
- "if", "for", "while" con condición verdadera

Para saltos hacia delante, primero sumaremos el tamaño de la instrucción correspondiente a la posición anterior, como si fuera una instrucción consecutiva. A continuación, dado que durante la ejecución dinámica no sabemos cual es el tamaño del salto no podemos saber exactamente cual es la dirección de destino. Así, se calcula de forma probabilística, sabiendo que tenemos la misma probabilidad de ir a cualquier posición dentro de una línea. Además, se supone siempre que hemos saltado a otra línea, lo que implica una nueva comprobación en la cache. Esto se aplica para:

- "if", "for", "while" con condición falsa
- Saltos a función, retorno de función
- "break", "return"

Por último, para identificar si la nueva línea produce un fallo de cache, utilizaremos un modelo de cache. En este modelo se almacena la información de las líneas presentes en la cache en cada momento. El modelo contiene una lista de líneas cuyo tamaño es función del tamaño de la cache a modelar. Por cada línea se guarda una estructura de datos que contiene un identificador de línea de información de su último uso, que es utilizada por el mecanismo de reemplazo.

Cada línea de cache se identifica mediante una cadena de caracteres. En las operaciones de control, está cadena se obtiene directamente del código fuente. En el resto de operadores, la cadena está compuesta por la de la operación de control anterior más un identificador del número de líneas ejecutadas desde dicha instrucción de control.

En cada operación de control se obtiene el identificador de línea de cache en función del número de línea en el archivo C. Para ello se utilizan las macros presentadas en la sección 6.1.5. Por ejemplo, la macro correspondiente a la claúsula "if" es:

```
#define if(cond) if(add_branch_time(__LINE__,__FILE__, basic_type(cond)))
```

Con la información de línea y archivo ("__LINE__" y "__FILE__"), la instrucción de control es identificada. La línea correspondiente de cache se llamara "archivoX_lineaY". De esta forma, cuando volvamos a ejecutar la instrucción y, bastará comprobar si la línea "archivoX_lineaY" está en la cache o no. Las líneas consecutivas se identifican mediante un prefijo que indica el valor de la condición y el número de líneas desde la instrucción de

control. De esta forma una línea se podría llamar “archivoX_lineaY_F_5” si es la quinta línea ejecutada y la condición fue falsa.

Con esta información, cada vez que se cambia de línea se chequea el modelo de cache. Si la línea está en la lista de la cache, continuamos, y si no, se modela un fallo de cache. En el modelado del fallo se carga la nueva línea en la cache. Para ello se comprueba si hay hueco en cache, expulsando una línea de la cache, según un mecanismo de reemplazo LRU. Además, se realiza un acceso al bus del tamaño de la línea, acceso durante el cual la ejecución de código se paraliza.

6.3.2 Modelado de caches con técnicas de anotación por preprocesado

El modelado de la cache de instrucciones implementado para técnicas de anotación estática es más sencillo que su equivalente para sobrecarga de operadores. Esto se debe que las técnicas de anotación se basan e una combinación de anotación estática y modelado dinámico. El procedimiento de anotación estática permite conocer a priori el tamaño de cada bloque básico. Esto significa que es posible asociarle a cada uno una posición de memoria en tiempo de pre-procesado. Por tanto no es necesario obtener información del camino de ejecución para estimar si se ha pasado por un bloque o no. La simplicidad del modelado reduce la sobrecarga en el tiempo de simulación sin reducir la precisión del modelo.

La técnica de modelado de caches para anotación estática está dividida en dos partes: una realizada mediante un análisis estático y otra ejecutada durante la simulación. Durante el análisis estático se estima el tamaño de cada bloque básico y por tanto el número de líneas de cache que requiere. Durante la simulación se dispone de un modelo de cache que emula el comportamiento de una cache real. Este modelo ha sido codificado por Juan Castillo.

La técnica implementada ha sido diseñada como una evolución a los modelos utilizados en simuladores de instrucciones (ISS). La sobrecarga de estos modelos en una simulación nativa no es asumible. Por cada instrucción nativa ejecutada deberíamos realizar una búsqueda en cache, y provocar un fallo de cache y una expulsión en los casos necesarios. Para evitar esta sobrecarga, la técnica implementada modifica la técnica de los ISSs, aumentando la velocidad a cambio de una relativa reducción de la precisión.

En primer lugar, en vez de modelar cada línea de cache por separado se agrupan todas las líneas de cada bloque básico. Esto evita que se realice una gestión de cache por línea y lo reduce a una por bloque básico. Como consecuencia cada bloque se corresponde con una entrada de cache, que tiene su dirección y su tamaño en lugar de “n” entradas, una por línea. Esto implica que cada vez que se necesita expulsar una línea de la cache, en realidad se expulsan todas las líneas del bloque juntas. Hemos de considerar que normalmente cada salto no es automáticamente seguido de otro salto, viene seguido de unas cuantas instrucciones secuenciales antes del próximo salto. Por tanto la probabilidad de que si se expulsa una línea de cache también se vaya a expulsar a las siguientes es bastante alta. Esto limita el error introducido por la agrupación de las líneas del bloque.

En segundo lugar es crucial eliminar la búsqueda para saber si las líneas del bloque están en cache. Para ello se ha creado una estructura que contiene toda la información de cache del bloque: el tamaño, la posición y si está o no en cache. Aprovechando que cada bloque de código rodeado por llaves genera un nuevo ámbito de visibilidad, se crea una instancia de dicha clase por bloque básico. Dicha instancia se declara como estática, de forma que se crea durante la primera ejecución del bloque y se mantiene durante las demás ejecuciones. De esta forma no es necesario ir a la cache a comprobar si la línea está cargada o no. Es suficiente con comprobar la variable de la estructura que indica si la información está en cache o no. Cuando no está en cache se ejecuta el código de modelado de un fallo de cache. Este código carga la estructura en el modelo de cache y produce una transferencia por el bus. Cuando un bloque ha de ser expulsado de la cache basta con quitarlo de la lista de la cache y cambiar su variable para indicar que ya no está en la cache.

El código que introduce la anotación para el modelado de la cache es el siguiente:

```
fprintf(file,"static struct ic_cache_block_t *block_%d=init_block(%d,%d);",
      parsing_struct.block_id, parsing_struct.instr_count,
      parsing_struct.init_line);

fprintf(file,"if(!block_%d->m_in_cache){icache_insert_block(block_%d);}",
      parsing_struct.block_id, parsing_struct.block_id,
      parsing_struct.current_set);
```

De esta forma, la sobrecarga en caso de que la línea ya esté en cache se reduce a un "if" con condición falsa. En caso de que haya que modelar el miss la sobrecarga es mayor, pero dado que la sobrecarga produce una transferencia por el bus, no es preocupante. Una transmisión por el bus implica una sobrecarga bastante considerable, ya que implica una simulación del comportamiento de varios componentes HW del sistema.

Para reducir esta última sobrecarga se permite agrupar los fallos de cache de forma que en lugar de modelar una transferencia de bus por miss se realice una por cada varios mises, con el tamaño correspondiente a todos ellos.

6.3.3 Modelado de la cache de datos

6.3.3.1 Modelado de la cache de datos con la técnica de sobrecarga

Al contrario que con la cache de instrucciones, el modelado de la cache de datos es uno de los puntos más favorables de la técnica de sobrecarga respecto a la anotación estática. Dado que en cada operador ejecutado disponemos de toda la información sobre sus operandos, es posible analizar si estos están o no en cache y aplicar así las operaciones necesarias para el modelado del comportamiento de la cache de datos.

Para realizar el modelado debemos introducir en la clase sobrecargada información sobre la dirección estimada de la variable en el mapa de memoria e información sobre si está o no en cache. Además necesitamos un modelo del funcionamiento de la cache donde

almacenar las variables disponibles en la cache de forma que cada vez que una sea expulsada se modifique la información interna de la clase sobrecargada como corresponde.

Para gestionar la dirección asociada a la variable basta con asignarla adecuadamente en el constructor de la clase. Si esta no ha sido inicializada en el constructor se debe a que ha sido generada mediante un "malloc" o una función equivalente y emplazada en el "heap". En este caso la información de dirección se obtiene a partir de la dirección de la propia variable en la simulación en curso. Dado que la clase sobrecargada es más grande que la variable original, para modelar correctamente los "arrays" hay que ajustar esta dirección en función de los tamaños. Para simplificar esta operación se ha hecho que el tamaño de la clase sobrecargada sea potencia de 2. De esta forma, dividiendo la dirección de la variable en la simulación por el tamaño de la clase y multiplicando por el tamaño base de la variable se obtiene la dirección necesaria para realizar el modelado de la cache.

6.3.3.2 Modelado de la cache de datos en anotación de código fuente

Al contrario que en caso anterior, el modelado de la cache de datos con esta técnica es bastante más complicado que el modelado de la cache de datos. La técnica propuesta se basa en anotar en cada bloque básico las líneas de cache de datos que han de manejarse y utilizar un modelo de cache similar al utilizado para la cache de instrucciones.

La técnica de modelado de caches de instrucciones e basa en el hecho de que cada instrucción se usa en un único punto del código, y que la dirección de memoria donde se guarda puede ser obtenida analizando el código, sin necesidad de la etapa de simulación. Por contra, las variables con los datos se usan en múltiples sitios, y su dirección en memoria no siempre se puede conocer analizando el código, solo se conoce durante la ejecución del código, requiriendo el paso de simulación, e invalidando el uso de las técnicas previas empleadas para otras métricas. Así, es necesario obtener otro tipo de información y realizar distintas anotaciones.

Para realizar un modelado eficiente de memoria cache de datos, se deben superar dos retos: cómo obtener las direcciones de acceso a partir de ejecución nativa y cómo crear un modelo de cache rápido. Las anotaciones que se insertan en el código para permitir el modelado de la cache de datos, tienen como objetivo resolver ambos retos.

Para anotar información de caches de datos 106 se identifican las variables accedidas en el código. Esto puede realizarse de varias formas, tales como mediante un análisis gramatical o un análisis del código compilado antes de realizar dicha anotación.

Para obtener las direcciones de cada acceso de datos a memoria, se usan las direcciones del código fuente durante ejecución nativa. Para realizar la anotación se necesita un identificador único para cada variable que permita chequear durante su uso sucesivo si ésta se encuentra almacenada en cache o no. Además, se necesita un mecanismo que permita analizar la localidad espacial de las variables (si las direcciones son similares o no). En el sistema real se utiliza la dirección de la variable en la plataforma destino para ambos fines.

Sin embargo, en la simulación propuesta no se dispone de dichas direcciones (puesto que la plataforma destino tal vez ni siquiera esté implementada).

El método propuesto utiliza, en lugar de la dirección de la variable en la plataforma destino, la dirección en la plataforma donde se realiza la simulación (las direcciones de la simulación nativa). En una realización preferente, en la que la simulación se realiza en lenguaje C (o cualquiera de sus variantes, como C++), para obtener la dirección de cada variable a chequear se utiliza el operador de indirección "&" del lenguaje C (por ejemplo, la dirección de la variable "v" es "&v"). Esto también se hace para variables complejas, como arrays y punteros. De forma adicional, para accesos a puntero, se incluye un acceso adicional al propio puntero. Es decir, en caso de accesos a variables simples, un único chequeo de la cache es necesario. En caso de acceso a variables complejas, como punteros, arrays o miembros de estructuras o clases, se realiza un acceso adicional por nivel de indirección (ej. un puntero doble requiere dos accesos adicionales).

Para chequear si cada variable accedida se encuentra en cache de datos o no, se ha de acceder a la información de la cache utilizando la dirección asociada a la variable. De forma convencional, esto se realiza llamando a una función del modelo de cache, sin perjuicio de utilizar otras soluciones más optimizadas que se presentan a continuación. Como modelo de cache se puede utilizar un modelo convencional como los utilizados habitualmente en los ISS. Para modelar los accesos a cache durante la simulación, se accede a un modelo de cache de datos usando las funciones como el que se describe más adelante. Por ejemplo, considerando una línea simple (a[i]=b), el código anotado resultante es el expresado en las siguientes funciones de acceso a cache de datos:

{ a[i]=b; // original code

data_cache_read(&i); data_cache_read(&b);

data_cache_read(&a); data_cache_write(&a[i]); }

Con estas consideraciones se resuelve el problema de identificar las variables a chequear y la obtención de un identificador válido. Para cada bloque básico, durante la anotación del código SW se genera una lista de las variables leídas y escritas. Al final de cada bloque básico se anota un chequeo de existencia en cache por cada variable de la lista.

Se puede reducir el número de chequeos analizando la localidad de los accesos en el código. Si una variable se accede en un punto del código, es de esperar que en las instrucciones siguientes, si se accede otra vez, la probabilidad de que se encuentre en cache sea cercana al 100%, por lo que se pueden evitar dichas comprobaciones.

Dado que en caches las variables se agrupan por "líneas" que se manejan como una unidad, es suficiente con chequear si la línea está en cache, sin tener que chequear la dirección entera. La línea se identifica con la dirección de la variable, obviando los bits menos significativos. El número de bits depende del tamaño de línea de la cache.

Además, si dos variables o más se han declarado consecutivamente (o casi, esto es, en el mismo bloque básico o bloques adyacentes) y ambas van a ir a la misma línea de cache, es suficiente con comprobar el acceso a la primera, evitando el chequeo de la(s) otras, de la misma forma que se puede evitar el chequeo de accesos múltiples a la misma variable.

Por último, se incluyen algunas modificaciones en las direcciones que permiten transformar las direcciones nativas en direcciones de la plataforma destino para realizar la simulación. Para transformar las direcciones nativas en direcciones destino, es necesario analizar la estructura ejecutable. En un fichero ejecutable común, las variables de datos se almacenan en varias secciones. Siguiendo el estándar ELF (Executable and Linkable Format, vers 1.2. TIS committee http://pdos.csail.mit.edu/6.828/2005/readings/elf.pdf), las variables globales se almacenan principalmente en las secciones "data", "rodata" y "bss". Las variables locales se almacenan en el "stack". Finalmente, la memoria dinámica se obtiene a partir del "heap".

Todas las variables se colocan en su sección correspondiente, ordenadas según se declaran. Como se usan el mismo front-end del compilador y el mismo orden de enlazado, el orden de las variables es el mismo en la plataforma nativa y en la plataforma destino. Así, es posible obtener las direcciones de modelado de caches a partir de la ejecución nativa.

No obstante, hay que realizar algunas consideraciones adicionales. Primero, para que las direcciones de las variables en la simulación sean válidas, el tamaño de los datos ha de ser el mismo en la plataforma de simulación (por ejemplo, computador) y en la plataforma real. Con objeto de asegurar esto, en caso de que ambas tengan procesadores con el mismo ancho de palabra (ej. 32 bits) no son necesarias modificaciones. Sin embargo, dependiendo del procesador destino y del compilador, los tamaños de los distintos tipos de datos pueden cambiar. En este caso (ej. computador de 64 bits y plataforma real de 32 bits), se modifican todos los tipos de todos los datos durante el proceso de compilación para que se ajusten a tipos de tamaño equivalentes en la plataforma real (ej. modificar un tipo "long int" de tamaño 64/32 por un "int", que tendría 32 bits en ambas plataformas). Para ello, se ha extendido la herramienta de anotación de forma que es posible indicar el tamaño de cada tipo de datos de la plataforma destino y los tamaños de los datos en la ejecución nativa. Usando esta información, cuando se anota el código, se modifica el tipo de cada variable, aplicando un tipo de datos que casa el tamaño esperado en la plataforma destino. Además hay que indicar al compilador utilizado para realizar la simulación que utilice la misma granularidad que el compilador para la plataforma real (ej. mediante *#pragma pack(push,2)*).

Segundo, para obtener las direcciones exactas de cada variable, es necesario aislar las variables de la aplicación SW del resto de la simulación. Para aislar las variables, el método compila la aplicación SW como una librería dinámica en la plataforma u ordenador host (anfitrión), y lo enlaza al resto de la simulación. De esta forma, las secciones "text", "data", "rodata" y "bss" se separan del motor de simulación y de los modelos de componentes HW. El orden de las variables "stack" variables' se mantiene automáticamente, ya que las llamadas a funciones del código original no se han modificado, y las variables locales no se añaden. Finalmente, las direcciones "heap" deben modelarse. En la infraestructura presentada, la

infraestructura no realiza llamadas a funciones “malloc” durante la simulación. Por tanto, no se requieren mecanismos adicionales. Sin embargo, para otras infraestructuras, la solución recomendada es aplicar un gestor “heap” específico para las tareas de aplicaciones SW. El gestor incluido en la librería “newlib” (SystemC. IEEE-1666 estándar, www.systemc.org) puede adoptarse para este propósito fácilmente. En cualquier caso, como el “heap” se usa comúnmente para crear arrays grandes, la localidad de los datos puede modelarse con un error mínimo incluso cuando la infraestructura de simulación realiza accesos “heap” adicionales.

Finalmente, debe tenerse en cuenta que las secciones de código empiezan en direcciones distintas en la plataforma host (anfitriona) y en la plataforma destino. Por eso, es necesario aplicar una corrección final para conseguir la dirección real. Por ejemplo, en sistemas Linux, la posición de cada sección de código en la simulación nativa puede encontrarse a partir de la propia información de formato de la librería dinámica y de la información de los ficheros “/proc/self/stat” y “/proc/self/maps”. Si el diseñador también tiene las direcciones de las secciones correspondientes en las plataformas destino, se puede obtener la dirección de cada variable en la plataforma.

Sin embargo, como el rendimiento cache depende principalmente de la ubicación espacial, a menudo no es necesario obtener las direcciones de destino exactas. Por eso, como se dijo anteriormente, este tercer paso es opcional. Los bits más altos de la dirección pueden usarse como etiquetas en la cache, pero la identificación de la línea cache depende sólo de los bits más bajos. De hecho, aplicar esta simplificación del modelo produce una tasa de error baja.

El segundo problema que debe resolverse es cómo realizar un chequeo eficiente para comprobar si las variables están o no en la cache. Los modelos de memoria cache convencionales basados en ISS se basan en búsquedas de las direcciones en el modelo de cache. Estas búsquedas pueden ser demasiado lentas para el tipo de simulación propuesto para la aplicación de la técnica. La solución propuesta es simplificar el proceso de chequeo quitándolo del modelo de cache (con lo que se obtiene un modelo de cache simplificado) e introduciéndolo como parte de la anotación, lo que permite evitar la realización de búsquedas durante el chequeo.

Aunque las llamadas a funciones propuestas anteriormente en este documento representan una solución válida para el modelado de caches de datos, producen una sobrecarga adicional de recursos de simulación. En simulación nativa, cada bloque básico de código SW requiere para su modelado unas pocas instrucciones máquina. Sin embargo, se requieren muchas instrucciones de la máquina anfitriona para accede a las funciones del modelo cache de datos y realizar una búsqueda por etiquetas para averiguar si la variable está en cache o no. Por ejemplo, en un ARM920, cada dirección puede mapearse en 64 líneas. Comprobar las 64 etiquetas en el modelo requiere mucho tiempo.

Para reducir esa sobrecarga de simulación, se modifican las llamadas a cache, trasladando parte del esfuerzo de chequeo a la anotación. Para realizar esto, una posible implementación se basa en sustituir los modelos hardware de la memoria cache de datos

basados en búsquedas por un modelo basado en arrays de memoria. En este tipo de modelo, se designa una zona de memoria en la que en cada bit se indica si una línea de memoria está en cache o no. Así, se divide todo el espacio de memoria utilizado en líneas, conforme se guarda en cache. En este caso, es suficiente chequear el bit correspondiente a la línea para saber si está en cache o no, evitando el proceso de búsqueda. Más adelante se detalla un modelo simplificado de memoria cache de datos que puede usarse.

Los procesadores embebidos convencionales son procesadores de 32 bits. Así, se puede acceder a un máximo de 232 direcciones de memoria. Considerando una línea de cache convencional de 8 palabras de 4 bytes, se puede mapear en cache un máximo de 227 líneas diferentes durante toda la ejecución del código. Entonces, es posible crear un array de 16 Mbytes donde cada bit indica si la línea está o no en cache. Considerando que los ordenadores (host, anfitriones) actuales tiene del orden de GigaBytes de RAM, esta cantidad de memoria supone un requisito pequeño.

Utilizando el modelo basado en arrays de memoria se puede simplificar el chequeo de la cache. En lugar de realizar una llamada a una función del modelo de cache para comprobar cada variable, se puede integrar un código de chequeo (un "if", por ejemplo) en la anotación del código SW, evitando el coste de realizar llamadas a función.

Usando este array, es posible encontrar una variable en cache leyendo el valor del bit correspondiente en el array de memoria. Por ejemplo, este valor puede obtenerse aplicando la siguiente instrucción:

#define CACHED(addr) mem[(addr>>8)]&1<<((addr>>5)&7)

Entonces la función "data_cache_read" puede sustituirse por ejemplo por una expresión "if", que compruebe el bit correspondiente, y solo en caso de que no esté se llame al modelo simplificado de cache:

"If(CACHED(&variable)) dcache_insert_line(&addr);"

Consecuentemente, se evita la búsqueda de etiquetas (tag search) y la llamada a función para acceder al modelo de cache de datos se realiza solo en caso de fallos (misses). Como en operaciones normales la mayoría de los accesos a cache resultan en un éxito (hit), esta solución minimiza la sobrecarga de modelado de cache, haciendo viable su aplicación para co-simulación nativa.

Para caches de escritura retardada (write-back), se necesita otro array de memoria para almacenar los bits sucios (dirty bits). Para pasar los bits a verdaderos (true) cuando se almacena un dato, se puede usar, por ejemplo, la siguiente macro:

#define DIRTY(addr) dirty[(addr>>8)] | 1<<((addr>>5)&0x7)

La figura 6-6 ilustra un posible modelo de cache de datos para usar durante la simulación. Internamente, este modelo de cache es similar a la infraestructura real de cache. Principalmente comprende uno o varios arrays, en los que se almacenan las direcciones de las líneas que se encuentran en un momento dado en cache. Cuando se requiere una nueva línea, tras comprobar si está o no en cache por los mecanismos explicados anteriormente de verificación del array, se debe confirmar la existencia de una línea vacía, verificando los arrays 201. Si no existe, se elimina una línea de la cache siguiendo una política o táctica aleatoria o de eliminación (todos contra todos, round robin). Una vez que la línea se ha eliminado, se chequea el bit de sucio correspondiente y se limpia el bit en el array de memoria. En caso de estar el bit de sucio activo, se realiza un acceso a la memoria principal o a caches de orden superior, directamente o a través de un buffer intermedio. Por último guarda la información de la nueva línea en los arrays de direcciones.

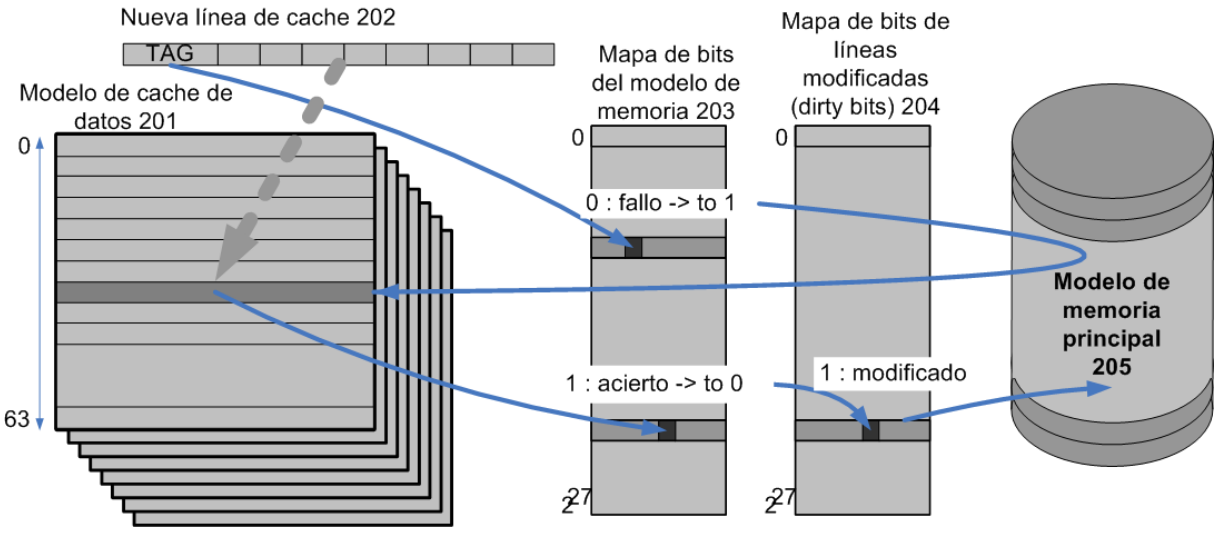


Figura 6-6: Modelo de la cache de datos.

El modelo desarrollado permite configurar el tamaño de cache de datos, el grado de asociatividad, el tamaño de línea y el método de sustitución.

Esta técnica ha sido desarrollada en colaboración con Luis Díaz, y está en proceso de protección mediante una patente internacional.

6.4 Modelado del avance de tiempo de la ejecución SW

Una vez estimado el coste temporal de cada segmento de código hay que proceder a introducir esa información en la simulación, transformando la ejecución atemporal del código SW en un modelado temporal. Esta transformación añade dos nuevas cualidades a la verificación funcional propia de la ejecución SW. Por un lado, la transformación de la ejecución en un modelo temporal permite la verificación de las restricciones funcionales del sistema. Además, la introducción de tiempos puede alterar los resultados funcionales del código.

La consideración del efecto del tiempo puede modificar el orden de ejecución de las tareas, el acceso a variables globales, colas de mensajes o cualquier otro canal de comunicación, alterar el orden de entrada en secciones críticas, etc. En sistemas SW, habitualmente no completamente deterministas, estas modificaciones pueden alterar la ejecución funcional del código. Suponiendo que las estimaciones temporales son correctas, es razonable pensar que la simulación temporal resultante está más cercana a la ejecución real. Por ello la propia verificación funcional considerando efectos temporales es más completa que la ejecución atemporal.

No obstante, conseguir que la ejecución temporal sea lo más cercana posible a la ejecución real no solo depende de la precisión de las estimaciones. La asociación de tiempo a segmentos completos de código implica una simplificación, y como tal puede dar lugar a discrepancias en la simulación que han de ser adecuadamente contempladas. Eventos como las interrupciones o las expulsiones producen que la ejecución de un segmento de código pueda ser dividida en dos, en lugar de ser ejecutado de una sola vez. Esto significa que la anotación del tiempo en un solo paso produciría un modelado erróneo. A continuación veremos como afrontar estos problemas.

El modelado del tiempo de ejecución del SW ha de considerar tanto el tiempo necesario para ejecutar el código de aplicación como la infraestructura SW, especialmente el sistema operativo. En este capítulo consideraremos el modelado del código de aplicación, dejando el modelado del sistema operativo para el capítulo siguiente.

6.4.1 Modelado temporal básico

Para poder realizar la simulación de sistemas completos teniendo en cuenta el tiempo de ejecución del código SW, los costes temporales de los segmentos han de ser introducidos en la simulación en los puntos adecuados. Para ello la técnica de modelado propuesta se basa en ejecutar los segmentos de código, estimar su coste temporal durante la ejecución y dormir el proceso al final de dicho segmento durante el tiempo estimado. Durante el tiempo que el proceso esta dormido esperando este tiempo, el modelo considera que el recurso que representa al procesador está ocupado, de tal forma que otro proceso no puede ejecutar simultáneamente. De esta forma la aplicación del tiempo de segmento se realiza directamente sobre el reloj de la simulación SystemC y el resultado final es, en principio, equivalente al que tendríamos si hubiéramos realizado una anotación por cada operación realizada en el código.

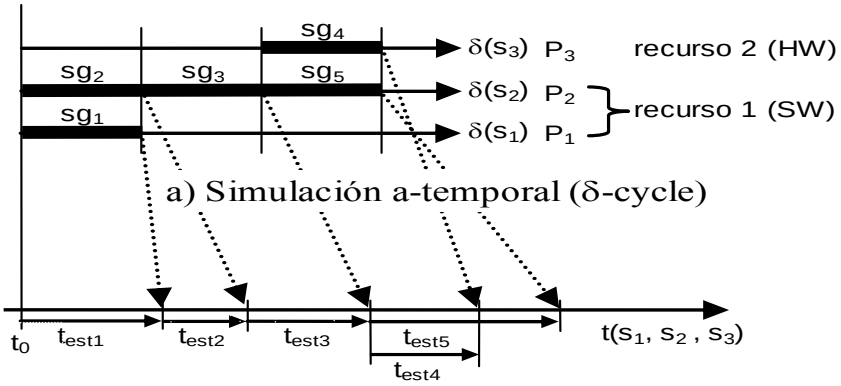
Como hemos dicho, para que esto se lleve a cabo, es necesario realizar una espera del tiempo correspondiente, bloqueando el procesador. Teniendo en cuenta que cada comienzo y fin de segmento se corresponde por definición con una llamada al sistema operativo, la anotación puede ser automatizada fácilmente. Como se explicará en el capítulo siguiente, en el modelo de sistema operativo desarrollado, cada vez que se realiza una llamada al sistema que puede provocar una interacción con el resto del sistema, el proceso pasa de ejecutar en modo Usuario a ejecutar en modo Súper-usuario, de la misma que cualquier sistema Unix o Linux.

La mejor forma de automatizar la introducción del tiempo de segmento en la simulación es hacerlo en el cambio de modo de ejecución. Cuando un proceso pasa de modo Usuario a Super-usuario, se toma el tiempo del segmento de código y se realiza la espera, poniendo inicializando el tiempo de segmento para poder calcular el del segmento siguiente.

De esta forma, se emula el comportamiento de la plataforma. Únicamente añadiendo la librería de estimación y el sistema operativo el tiempo de ejecución SW es automáticamente estimado e introducido en la simulación sin ningún trabajo adicional por parte del usuario. Considerando que los modelos que componen la plataforma HW tienen información temporal asociada, cada evento producido en el sistema, ya sea debido a funcionalidad SW o HW, tiene un tiempo de simulación asociado.

Esto permite el fácil encadenamiento de los eventos, de tal forma que cada evento considera adecuadamente todos los puntos temporales de los eventos previos, y todos los efectos colaterales que el tiempo de un segmento pueda producir en los eventos siguientes puede ser modelado. Además, al tener un comportamiento temporal, toma sentido la aparición de modelos de sistema operativo. Cuando cada tarea se ejecuta en tiempo cero, no hay colisión de los procesos en los recursos, y un gestor de ejecución no es necesario. Sin embargo, al asociar tiempo a la ejecución SW nos encontramos con que los recursos procesadores se pueden encontrar ocupados cuando una nueva tarea quiere ejecutar, dando lugar a la necesidad de planificación y por extensión de un sistema operativo.

Esto puede verse fácilmente en la figura siguiente. En la figura, las señales s1, s2 y s3 se corresponden los eventos de tres procesos concurrentes llamados P1, P2 y P3 respectivamente. Los procesos 1 y 2 comparten el procesador 1, mientras que el proceso 3 dispone de un procesador exclusivo. Además el proceso 3 necesita que P2 ejecute el segmento 3 para comenzar a ejecutar. La parte a) de la figura representa el resultado de la ejecución sin consideraciones temporales, y la parte b) el resultado una vez aplicado el modelado de tiempo de ejecución.



b) Simulación temporal

Figura 6-7: Efectos de la anotación de tiempos.

Consideremos que los tres procesos son arrancados por un dominio externo en t0. Dado que P3 necesita un evento de P2, tan solo P1 y P2 pueden comenzar la ejecución. En el caso atemporal, la ejecución se realiza instantáneamente, de forma que no hay limitaciones en el uso del recurso. De esta forma, en t0, tanto P1 como P2 pueden ejecutar simultáneamente. P2 ejecuta hasta requerir un evento y P1 acaba. En el ciclo siguiente, P2 continúa ejecutando hasta que requiere un nuevo evento. En el tercer ciclo, P3 puede empezar a la vez que P2 acaba. Como resultado, la ejecución de P3 es completamente independiente de P1.

En el caso temporal, en t0 no pueden empezar P1 y P2, dado que están en el mismo procesador. Por ello P2 espera a que P1 acabe. A continuación P2 ejecuta sus dos segmentos, hasta que despierta P3. En ese momento P2 y P3 pueden ejecutar en paralelo, ya que están en procesadores distintos. En esta ocasión, el tiempo en el que ejecuta P3 no solo depende de P2. Para que P2 comenzara tuvo que esperar a que el procesador estuviera disponible. De esta forma, el efecto colateral que tiene P1 en P3 puede ser simulado. Este sistema es capaz de detectar los efectos que unos procesos tienen en el resto del sistema, permitiendo comprobar, por ejemplo, si como consecuencia de que P1 requiera demasiado tiempo en ejecutar, P3 puede no cumplir una restricción temporal.

6.4.1.1 Problemática del modelado temporal básico

En la técnica de modelado temporal básico propuesta, el estado final del proceso al concluir un segmento es equivalente a una ejecución ciclo a ciclo. Sin embargo, la técnica no es capaz de modelar el efecto de eventos intermedios. Por tanto, esta técnica es bastante precisa siempre y cuando no se produzcan dos circunstancias: que se acceda a variables compartidas o que la ejecución del segmento sea interrumpida por un evento externo. Desgraciadamente, en código SW, el uso de variables globales compartidas por los hilos de un proceso y la aparición de interrupciones, son elementos muy comunes, que violan estas restricciones. Veamos un ejemplo.

Supongamos que tenemos un ejemplo con dos tareas y un procesador. La tarea 1 tiene prioridad baja, como corresponde a una tarea de cómputo, mientras que la tarea 2, encargada de responder a eventos externos, tiene una prioridad mayor. Al comienzo del ejemplo, la tarea 1 tiene que ejecutar un determinado código hasta que acaba. La tarea 2 comienza esperando a un evento externo o a la finalización de un tiempo de espera. El tiempo de espera expira a los 20us. Una vez expirado el tiempo de espera se ejecuta un código y se queda esperando un nuevo evento HW. Dicho evento llega en T=55 us, con lo que ejecuta un código para responder a la interrupción y termina.

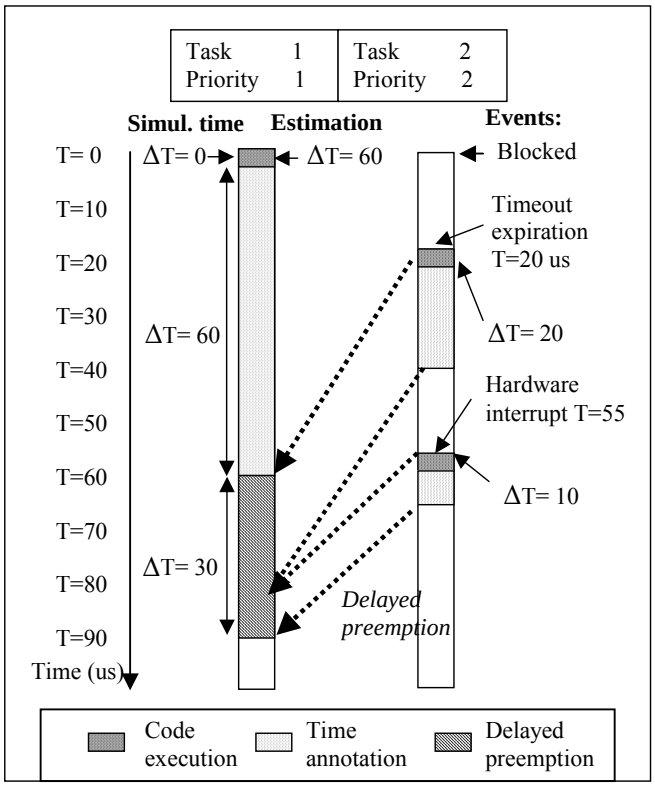


Figura 6-8: Ejemplo de simulación temporal.

Aplicando la técnica anterior, la tarea 1 comienza, mientras que la anterior esta bloqueada. Esto significa que ejecuta su código hasta la finalización del segmento. Dado que no tiene llamadas al sistema intermedias, esto significa que ejecuta completamente su código de una vez. Una vez ejecutado en código en tiempo 0 se bloquea durante el tiempo estimado (60us), bloqueando el procesador.

A los 20 us, el tiempo de espera de la tarea 2 finaliza, con lo que ha de ejecutar. Sin embargo, el procesador está ocupado hasta T=60us. Dado que la tarea 2 tiene mayor prioridad que T1, esto implica un modelado erróneo del sistema. Para complicar aun más las cosas, supongamos que la tarea 1 lee una variable que se modifica en la tarea 2. Si la variable se lee después de T=20us, el valor con el que ha computado la tarea 1 es erróneo.

Lo mismo ocurre en T=55. La tarea 2 debe ejecutar, pero el procesador sigue ocupado. Es esta caso la tarea 1 ha vuelto a leer la variable antes de ser modificada por la tarea 2. Sin embargo, si consideramos la expulsión que debería haber producido la primera ejecución de la tarea 2, la modificación de la variable debería haber sido anterior a su uso en la tarea 1. La situación es incluso más preocupante, porque la espera de la tarea 1 ha finalizado antes de que la tarea 2 modifique la variable. Esto implica no solo un error, sino que además ni con un análisis de modificación de variables hay forma de saber que se produce dicho error. Además, no hay forma de simular que la tarea T1 recibe dos valores distintos al leer la variable

compartida. Dado que se ejecuta toda la tarea en tiempo 0, no es posible modelar dicha modificación de ninguna forma.

Para resolver el problema, hemos de considerar que elementos pueden causar estos problemas. Como veremos en el grafo de ejecución de las tareas en el sistema operativo, tan solo una interrupción externa puede forzar una expulsión, requiriendo una modificación tanto del orden de ejecución como del valor de variables compartidas. En nuestro ejemplo, tanto el tiempo de espera, que se produce como consecuencia de un evento del temporizador HW del sistema, como el evento HW se deben a interrupciones.

En sistemas multiprocesador, el cambio de valor de las variables compartidas se puede producir en cualquier momento. En resumen, podemos decir que hay que gestionar tanto los cambios en variables compartidas como el efecto de las interrupciones en la gestión del orden de ejecución de las tareas.

6.4.2 Modelado temporal basado en rodajas temporales

La primera solución que vamos a considerar es definir un tiempo máximo de ejecución para cada segmento. Esto significa que cuando el tiempo acumulado de ejecución del segmento supera un valor predefinido, el segmento es finalizado automáticamente. De esta forma el proceso se bloquea para esperar el tiempo máximo, y una vez pasa ese tiempo continúa la ejecución como si se tratase de un segmento nuevo.

Esta técnica permite limitar el error producido por la anotación completa de segmentos. Nunca el error es mayor que el tiempo máximo permitido. Esto permite una gran flexibilidad, ya que es posible manejar fácilmente el binomio precisión – sobrecarga de simulación. Si definimos un tiempo límite grande, el efecto en el tiempo de simulación será mínimo, pero resolverá pocos problemas. Si definimos un tiempo máximo pequeño, la precisión será alta, pero aumentaremos el tiempo de simulación. Hay que tener en cuenta que el mayor coste de tiempo de simulación se produce cuando se bloquea un proceso, ya que entra en ejecución el kernel de simulación de SystemC analizando todas las tareas del sistema. Por tanto, si limitamos el tiempo máximo, dividiremos todos los segmentos del código en múltiples fragmentos, generando muchos bloqueos temporales, y por tanto mucha sobrecarga en la simulación.

Para incrementar la precisión sin aumentar la sobrecarga se ha propuesto una optimización de esta técnica. Recordemos que las limitaciones de la técnica de modelado dependen fuertemente de las interrupciones. Analizando los generadores de interrupciones HW podemos ver que uno de los elementos que está siempre presente en el sistema y que más interrupciones genera es el temporizador del sistema. Este temporizador tiene la cualidad de generar interrupciones periódicas. Cada cierto tiempo se genera una interrupción.

Como el periodo del temporizador es conocido, podemos decir que sus interrupciones son predecibles. Si ajustamos los tiempos máximos para que coincidan con las ejecuciones del temporizador, podemos minimizar los errores, eliminando todos los que dependen de elementos temporales: esperas, alarmas, temporizadores… Para ello, cuando un segmento empieza se comprueba cuanto tiempo resta para la siguiente interrupción del temporizador, y si este es menor que el tiempo máximo, se pone como límite el evento del temporizador. Teniendo en cuenta que en un sistema operativo convencional, tipo Linux, el temporizador del sistema tiene un periodo de 10ms, podemos decir que el aumento de segmentos que produce es limitado. Si además consideramos que la activación del temporizador va a provocar la ejecución del kernel de simulación de SystemC por si mismo, la finalización de un segmento en ese momento no implica una ejecución extra del kernel de simulación, ya que se ejecutaría de todas formas. Por tanto la sobrecarga que implica sobre la simulación es aceptable.

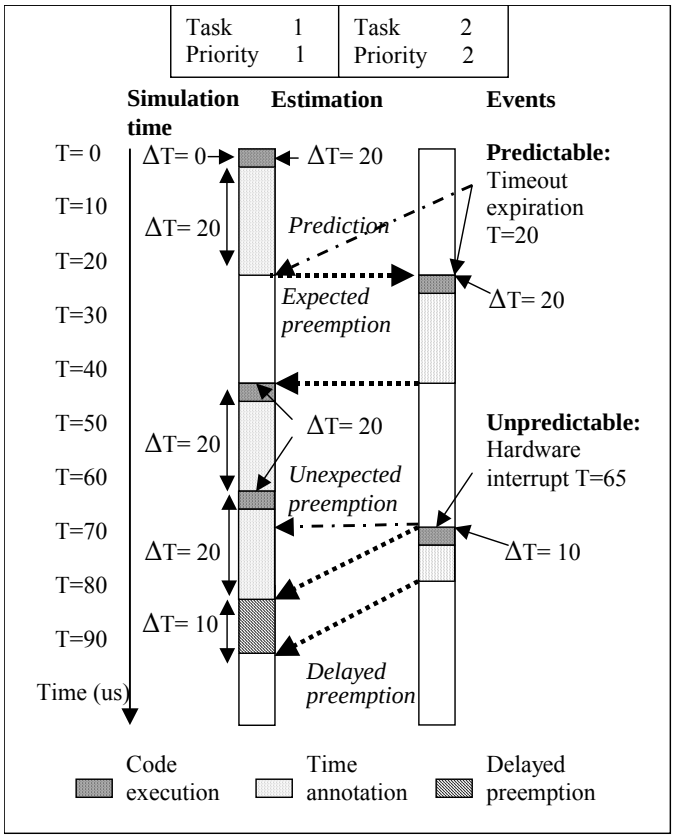


Figura 6-9: Ejemplo considerando Tic-Temporizador.

Si aplicamos esta técnica al ejemplo anterior con un tiempo máximo de 20, vemos que la ejecución de la tarea 2 debida a la finalización del tiempo de espera es correcta. De esta forma tanto el orden de ejecución de las tareas, como el valor de la variable compartida implicada son correctos. Sin embargo, la ejecución de la respuesta a la interrupción HW sigue siendo errónea. Tanto la planificación de la ejecución de las tareas, como el valor de la variable son erróneos.

Por tanto podemos decir que esta técnica, aunque es capaz de minimizar los errores de modelado, no garantiza un correcto funcionamiento de la simulación. Además implica una cierta sobrecarga en la simulación que ha de ser tenida en cuenta. Esto es especialmente importante cuando se aplica con la técnica de estimación mixta explicada anteriormente. En este caso, dado que la técnica de estimación es extremadamente rápida, el aumento del tiempo de simulación puede ser considerable en cuanto se empiece a reducir el tiempo máximo de ejecución de los segmentos.

6.4.3 Transformación de variables en canales

Tras considerar una posible solución para minimizar todos los problemas del modelado temporal básico, vamos a considerar técnicas que resuelven los problemas por separado.

La segunda solución que se ha considerado en la tesis es la transformación de las variables compartidas en canales. Para ello la técnica consiste en detectar las variables globales del código y transformarlas en una clase con los operadores de acceso, asignación y conversión sobrecargados. Con ello se consigue que cada vez que se accede a una variable global se ejecuten las funciones de dichos operadores.

Una vez podemos detectar esos puntos de comunicación con el exterior, pasamos a aplicar la definición de segmento propuesta al principio de la sección de manera estricta. Dado que un segmento es el código ejecutado consecutivamente entre dos interacciones con el exterior, los accesos a variable global se han de convertir en delimitadores de segmento. Cuando se acceda a una variable global se realizará la espera correspondiente al tiempo de segmento ejecutado anteriormente y se comenzará un nuevo segmento.

Esta técnica no resuelve el problema de re-ordenación en el orden de ejecución de las tareas, pero evita problemas con las variables compartidas.

La técnica es recomendable para su aplicación con la técnica de sobrecarga. De hecho puede ser fácilmente integrada con el uso de tipos sobrecargados para realizar la estimación de tiempo. La detección de si una variable es local o global puede detectarse fácilmente. Las variables globales se generan en estático, antes del comienzo de la simulación, mientras que las locales se crean al entrar en las funciones, durante la simulación. Por tanto, mirando si la variable se construye antes o después del comienzo de la simulación podemos detectar si es global o no, y si se debe forzar la terminación de los segmentos o no.

Con la técnica mixta también es aplicable, ya que el análisis estático puede detectar si las variables son globales o locales y hacer las transformaciones necesarias. Sin embargo, considerando que algunos programadores SW tienen tendencia al uso de variables globales, y que la transformación de una variable en un objeto sobrecargado implica una carga para la simulación, el aumento del tiempo de simulación puede hacer no recomendable su uso en determinados casos. Un análisis que compruebe si cada variable global se comparte realmente o no, puede limitar la sobrecarga, pero implica un análisis estático más complejo.

6.4.4 Modelado basado en esperas subdivisibles

La última técnica considerada resuelve el problema de inconsistencia en el orden de ejecución de las tareas, sin considerar el problema de las variables compartidas. Por tanto puede ser utilizada en combinación con la anterior para obtener un resultado completo.

La técnica consiste en permitir la división de las esperas temporales en tramos. De esta forma una vez finalizado el segmento se espera el tiempo estimado. Sin embargo, si se produce una interrupción externa, la tarea es automáticamente despertada, permitiendo la entrada de otra tarea.

Para resolver el problema se han de realizar dos modificaciones sobre la técnica básica. En primer lugar la espera temporal con la que se modela el consumo de tiempo durante la ejecución se transforma en una espera de evento con tiempo de espera. Si es recibe una interrupción se salta el evento, y si no la espera continua hasta agotar el tiempo. Una vez finalizada la espera, se chequea si se ha producido un evento o no. En caso afirmativo, se calcula el tiempo que queda por esperar y se indica a la tarea que cuando vuelva a ser re-planificada continúe esperando el tiempo restante antes de pasar a un nuevo segmento.

Como consecuencia el modelado temporal resultante es correcto, pero no así el funcional. Hemos de considerar que aunque la espera se paraliza, el segmento ya ha sido ejecutado completamente en tiempo 0, antes de empezar la espera. Por tanto la modificación de variables compartidas durante el segmento no es adecuadamente modificado.

Esta técnica tiene una sobrecarga prácticamente nula sobre la simulación, ya que no genera eventos adicionales en el sistema ni llamadas adicionales al kernel de simulación. Además, no modifica la ejecución del código de aplicación.

6.4.5 Modelado temporal recomendado

Considerando las soluciones anteriores se ha concluido que la técnica basada en esperas divisibles presenta una solución eficaz para el correcto modelado del orden de ejecución de las tareas. No obstante esta técnica no resuelve el problema de las variables globales.

La solución final adoptada se basa en el convencimiento final de que las variables globales no deben ser puntos de finalización de segmento, aunque sean puntos de comunicación. Para explicarlo recuperemos las razones por las cuales hemos dicho que hay que colocar en tiempo los puntos de comunicación. Hemos dicho que dichos puntos del código deben ser colocados adecuadamente en tiempo para garantizar que se coge el dato correcto y para que el tiempo que tarde en realizarse en esa comunicación por el productor se vea considerada en los consumidores.

Sin embargo, si analizamos una variable global vemos que eso no se cumple. Si queremos garantizar que el dato leído es correcto, el código deberá incluir un elemento de

sincronización. Si no, no solo no se garantizará el modelado correcto, sino que la ejecución real no será segura. Además si no hay elemento de sincronización, el tiempo en el que el productor genera el dato es irrelevante, ya que la ejecución de los consumidores será independiente de que el productor haya acabado o no su cómputo correspondiente. Por tanto, dado que los elementos de sincronización son llamadas al sistema, el modelado adecuado de tiempo de comunicación por variable compartida se resuelve modelando adecuadamente las llamadas al sistema.

En resumen, para obtener una simulación que optimice el balance precisión/velocidad de simulación, se han tomado las siguientes acciones:

- En general, el modelado exacto de dichas variables no es necesario. Los errores producidos por este efecto se deben a modificaciones puntuales en el orden de ejecución del código. Hemos de considerar que el diseño de un sistema fiable no se pueden utilizar variables compartidas no sincronizadas de tal forma que la corrección del resultado final dependa de la ejecución temporal de cada segmento de código.

- En casos excepcionales que queramos modelar estos efectos se podrá elegir entre las otras dos técnicas teniendo en cuenta el número de variables globales a sobrecargar y el error que estamos dispuestos a asumir.

6.5 Estimación de consumos de SW de aplicación

La ejecución del código de aplicación supone un consumo importante dentro del sistema. Por ello, parte del esfuerzo realizado se ha dedicado a estimar el consumo del software en ejecución. La estimación se realiza en términos de energía, obteniendo su equivalente en potencia dividiendo entre el tiempo de ejecución.

El consumo del sistema por ejecución de software se reparte principalmente entre procesador, caché, bus y memoria, siendo el consumo del procesador el principal objetivo de este apartado. El consumo de procesador y cache de nivel 1 se estima junto con el modelado de tiempo del SW. El efecto del resto de componentes de la plataforma se considera en el modelado de los propios componentes (capítulo 8).

La estimación del consumo del procesador se ha realizado en base a otros trabajos previos realizados en el campo de los microprocesadores embebidos [Sinha01]. Como se indico en el estado del arte, estos trabajos demuestran que el consumo de potencia en dichos dispositivos se mantiene estable en torno a un valor medio. Si la secuencia de instrucciones ejecutadas es suficientemente variada, la energía por instrucción puede considerarse constante y puede ser calculada multiplicando la potencia media por el tiempo medio de instrucción (obtenido a partir del CPI del procesador).

La estimación de consumo se realiza durante la simulación del código de aplicación. De la misma forma que para el modelado temporal, el código fuente SW se modela directamente, sin necesidad de traducción a código máquina. Esto reduce considerablemente

el tiempo de simulación respecto a las estimaciones realizadas sobre ISS, a costa de un ligero aumento del error de estimación. Para realizar la estimación de consumo se asigna un peso a cada operador fuente (entendiendo por peso la energía necesaria para ejecutarlo), de forma que la energía total consumida puede estimarse sumando los pesos de todos los operadores ejecutados.

Para la técnica de sobrecarga de operadores, los pesos se obtienen a partir del número de instrucciones necesario para ejecutar cada operador fuente (asumiendo que el consumo por instrucción es constante). Para ello se analiza la traducción llevada a cabo por un compilador, tomando como referencia el compilador gcc de GNU. Mediante una batería de pruebas que emplea los diversos operadores fuente en diferentes situaciones, podemos extraer un peso medio para cada uno.

Debido a que no requiere el mismo número de instrucciones (y por tanto, energía) operar con variables que con inmediatos, el método de simulación ha sido adaptado para que diferencie ambos casos. La razón es que un inmediato puede almacenarse dentro de la propia instrucción máquina, mientras que con una variable es necesario emplear instrucciones de acceso a memoria para leer su valor.

En caso de ejecución de código anotado, conocido el número de instrucciones ensamblador ejecutadas, la energía consumida puede calcularse multiplicando dicho número por la energía media por instrucción.

En ambas clases de técnicas, el consumo de las caches se obtiene multiplicando el número de fallos de cache por el consumo por fallo de cache. A esto se añade un consumo medio durante ejecución del código SW.

7 Modelado de la plataforma SW

Para realizar el modelado temporal del subsistema SW, la estimación y anotación del tiempo de ejecución de cada componente SW debe ser complementada con el modelado de la interacción de los diferentes componentes entre sí, y con la infraestructura de ejecución. El modelado temporal de la ejecución del subsistema SW ha de considerar el tiempo que cada programa SW necesita para ejecutar, junto con el efecto resultante de tener un número limitado de recursos procesadores del sistema. Dado que normalmente el número de tareas concurrentes es mayor que el número de procesadores disponibles, es necesario disponer de una infraestructura SW capaz de gestionar el uso de los recursos de cómputo.

Sistemas operativos, infraestructuras de comunicación, de gestión de recursos distribuidos ("middleware") y componentes SW dependientes de plataforma ("HW-dependent SW") son los componentes principales de las plataformas SW. Por consiguiente, para modelar adecuadamente el comportamiento del sistema es necesario que el entorno de simulación proporcione un modelo de plataforma SW.

Dicho modelo debe cumplir tres tareas fundamentales: proporcionar los servicios requeridos por las aplicaciones SW, controlar la interacción entre aplicaciones y gestionar la relación con la plataforma HW.

En primer lugar, el modelo debe ser capaz de proporcionar al SW de aplicación los mismos servicios que estos esperan obtener de una plataforma real. Este modelo de plataforma debe ser por lo tanto lo suficientemente completo como para permitir que las aplicaciones SW sean simuladas sin necesidad de modificar su código original. En segundo lugar, el modelo debe comportarse de la misma forma que la plataforma real. Tanto los retrasos asociados a los servicios de la plataforma, como el efecto que producen dichos servicios en la ejecución de las tareas del sistema deben ser emulados. Planificaciones, expulsiones, esperas y sincronizaciones deben ser modeladas. Por último, la ausencia de un modelo HW que represente al procesador hace que las comunicaciones entre el SW y la plataforma HW tengan que realizarse directamente. De esta forma la plataforma SW es la encargada de leer y escribir en el modelo de bus, así como de recibir las peticiones de interrupción.

Para proporcionar todos los servicios necesarios debemos tener en cuenta los componentes habituales de una plataforma SW. Esto implica considerar y modelar el núcleo del sistema operativo, encargado del control de la concurrencia del SW, la interfaz de código de usuario y elementos como la infraestructura "middleware" o los sistemas de acceso a los periféricos HW.

Para generar un modelado lo más realista posible del funcionamiento del subsistema SW el modelo desarrollado en la tesis reproduce la arquitectura convencional de una plataforma SW. Este modelo proporciona los siguientes servicios y facilidades:

- Generación de concurrencia y planificación

- Comunicaciones y sincronización entre tareas
- Funciones de ayuda al programador: matemáticas, cadenas de caracteres,…
- Gestión de tiempo: temporizadores, esperas,…
- Comunicaciones externas: TCP/IP …
- I/O y Comunicación con periféricos
- Manejo de interrupciones
- Soporte de “middleware”

Con estos elementos, el SW de aplicación que vayamos a introducir en nuestros diseños podrá ser ejecutado en el modelo de alto nivel propuesto. La integración de código real nos permitirá obtener unas estimaciones adecuadas del rendimiento del SW del sistema de forma eficiente. Además podremos utilizar el modelo como un entorno de verificación donde comprobar el comportamiento del SW final.

Para realizar estas tareas adecuadamente la plataforma SW contiene un modelo de sistema operativo, y puntualmente una infraestructura “middleware”. Para acceder a cada servicio de la plataforma, el código de aplicación debe realizar las llamadas al sistema necesarias, de la misma forma que en un sistema real. Si una aplicación requiere una comunicación con otro componente del sistema, esta debe saber en qué punto del sistema esta ese componente y como llegar a el, ya sea SW o HW. A grandes rasgos, el modelo de plataforma SW desarrollado puede verse en la figura 7.1.

El sistema operativo es, de por sí, capaz de realizar las tres tareas citadas anteriormente, si bien la interacción entre aplicaciones puede ser optimizada utilizando la infraestructura “middleware”. El “middleware” se utiliza para evitar que la aplicación requiera ningún conocimiento del resto del sistema, facilitando la portabilidad, a diferencia del uso directo de los servicios del sistema operativo, donde información como las direcciones IP de los componentes remotos han de ser indicadas explícitamente. El “middleware” está diseñador para resolver ese problema de forma autónoma. El propio “middleware” tiene la inteligencia necesaria para saber dónde están los componentes y cómo comunicarse adecuadamente con ellos.

Así pues, ambos elementos son muy importantes en el diseño y modelado de sistemas HW/SW. Sin embargo, en la versión actual de SystemC no se proporciona ninguno de ellos. Ni la aportación de servicios SW, ni la comunicación HW/SW, ni el control de la interacción entre tareas están adecuadamente resueltos en SystemC. Es por ello que durante la tesis se ha desarrollado un completo modelo de sistema operativo y una infraestructura “middleware” que permita introducir los componentes SW en los modelos del sistema.

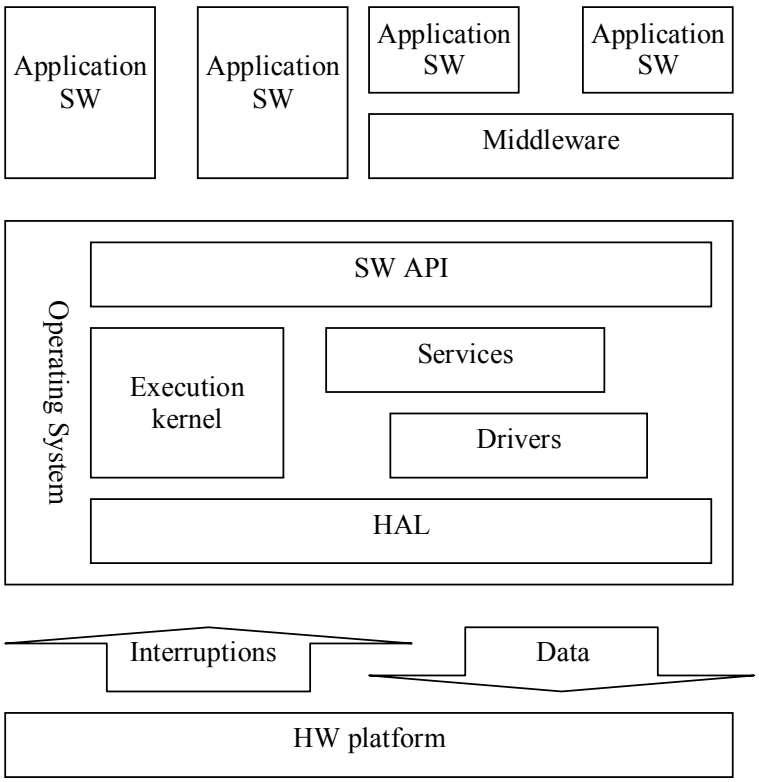


Figura 7-1: Ubicación del modelo de sistema operativo en SCoPE

El modelo de sistema operativo considera todas las partes de la figura anterior: API SW para interacción con la aplicación, HAL para la comunicación con la plataforma SW, un kernel para la gestión de tareas y una serie de servicios y controladores de dispositivo para proporcionar las funcionalidades adicionales requeridas por los componentes SW.

Para desarrollar el modelo de sistema operativo se ha partido de una especificación real, a diferencia de otros trabajos similares, que proponen modelos de sistemas mínimos y sin correlación directa con ningún sistema real (ver capítulo 4). Se ha elegido como especificación del sistema operativo uno de los estándares más comunes (si no el más importante): POSIX. Este estándar define los servicios e interfaces que un sistema operativo debe proporcionar para proporcionar un completo soporte a las aplicaciones. Esto contempla servicios para planificación, comunicación, sincronización, manejo de tiempos, etc.

En cuanto al "middleware", se ha partido de otro estándar comúnmente aceptado: CORBA. Se ha adaptado un modelo de ORB desarrollado en los laboratorios de TIMA (Grenoble) basado en una implementación ligera de CORBA desarrollada por Thales (OpenCCM).

Si bien la necesidad de un modelo de sistema operativo está muy ligada a la asociación de tiempos con la ejecución del SW de aplicación, se ha mantenido una cierta separación entre la estimación temporal y el modelado del sistema operativo. Esta separación tiene por

objeto permitir la utilización del modelo de sistema operativo sin estimación temporal del SW de aplicación o con cualquiera de los mecanismos de estimación presentados en el capítulo anterior u otros proporcionados por el usuario. También por esta razón se presentan en este documento en capítulos separados.

Esta separación es posible ya que la simulación del SW se realiza mediante la alternancia de ejecución de código de usuario y ejecución del modelo de sistema operativo. Durante la ejecución de cada segmento de código de usuario no se producen interferencias por parte de la infrastructura SW; este solo opera al final de cada segmento de código. De hecho, esta es una de las razones por la que se ha definido un "segmento" o "segmento de código" como una secuencia de instrucciones de código de usuario que se ejecuta entre dos llamadas al sistema operativo. Utilizando la técnica de modelado temporal basado en esperas subdivisibles (sección 6.4.4) es posible evitar la intromisión del sistema operativo durante la ejecución del código de usuario, a la vez que se modelan adecuadamente interrupciones y expulsiones (el código de usuario se ejecuta sin intromisiones y en tiempo cero; es la función "wait" de SystemC, con la que se anota el tiempo, la que es interrumpida por el modelo de sistema operativo).

La anotación de tiempo por esperas subdivisibles se realiza al cambiar el modo de ejecución de Usuario a Super-Usuario en cada llamada al sistema, punto en el que acaba el segmento de código según la definición anterior, como veremos a continuación. En esta transición, inicialmente se obtiene del estimador el coste temporal del segmento ejecutado y se realiza la espera, bloqueando el planificador durante el tiempo correspondiente. A continuación se espera el tiempo correspondiente a la llamada a la función del sistema operativo en ejecución o la interrupción en curso y finalmente se ejecuta la llamada y se llama al planificador cambiando de contexto si es necesario. Tanto los costes de las llamadas al planificador como los cambios de contexto y otras operaciones del sistema operativo también tienen un tiempo asociado, que es introducido en la simulación. En consecuencia se consigue realizar un modelo abstracto completo del sistema operativo, considerando tanto su funcionalidad como su sobrecarga en el sistema.

7.1 Modelado Abstracto de Sistemas Operativos

El modelado de un sistema operativo implica el desarrollo de una larga lista de funcionalidades y servicios que permitan la ejecución del SW de aplicación y del "middleware". Para simplificar su desarrollo, los modelos generados se centran en modelar "que hace" el sistema en lugar de "cómo lo hace". Este modelo abstracto, o de caja negra resulta también más eficiente, ya que al ser más simple reduce la sobrecarga en la simulación. No obstante, se ha encontrado que no es necesario el desarrollo de todos los elementos internos del sistema operativo desde cero: es posible aprovechar diversos servicios disponibles en el entorno de simulación para reducir dicho esfuerzo.

Para identificar que servicios pueden ser reutilizados hay que considerar que el modelo de sistema operativo desarrollando se ejecuta sobre una simulación SystemC. SystemC proporciona diversos recursos que se pueden utilizar como base para el modelo de sistema

operativo, como la gestión de procesos (bloqueos, cambios de contexto), el manejo de eventos y el modelado temporal mediante un doble eje tiempo/deltas.

Además, la simulación SystemC se ejecuta sobre un computador convencional, que llamaremos computador nativo o "host". Esta simulación corre sobre el sistema operativo nativo del computador. En nuestro caso consideramos que este sistema operativo es Linux (o cualquier otro sistema operativo similar que esté basado en las ideas de la especificación POSIX). Así nos encontramos que la mayoría de los servicios que queremos modelar están disponibles en el propio computador nativo. De estos servicios, algunos podrán ser reutilizados en el modelo SW y otros no. Para identificar cuales son reutilizables se ha dividido la funcionalidad a implementar en: concurrencia y planificación, comunicación y sincronización, tiempo, interacción con el HW y otros servicios adicionales.

Para proporcionar concurrencia y planificación, un sistema operativo Linux tiene diversos servicios para la creación de hilos ("threads") y de procesos, así como para la planificación de los mismos. No obstante, su utilización directa en el modelo de sistema operativo no es viable por varias causas.

Una primera causa es que si utilizamos la función de creación de procesos del computador nativo ("fork"), se realiza una nueva copia del proceso duplicando todos los datos. Esto significa que utilizando este servicio no obtenemos un nuevo proceso dentro de la simulación SystemC; lo que tenemos es dos simulaciones SystemC independientes y concurrentes. La aparición de dos núcleos de simulación con informaciones independientes hace que su uso directo en para la generación del modelo no sea recomendable, además de implicar una sobrecarga adicional en el tiempo de simulación.

En cuanto a las funciones de planificación, su uso directo tampoco es posible. Linux permite generar nuevos hilos durante la simulación, hilos que pueden recibir características de política y prioridad. Sin embargo, estas características no tienen ningún efecto en la simulación SystemC. El núcleo de ejecución de SystemC hace que, en cada momento, solo uno de sus hilos pueda estar activo, permaneciendo los demás bloqueados. Por tanto no es el sistema operativo subyacente el que elige que proceso se ejecuta, sino el núcleo de SystemC, ignorando las características de planificación de las tareas en el Linux subyacente.

Para comunicar y sincronizar las tareas, Linux proporciona elementos como semáforos, "mutexes", "fifos", ficheros o colas de mensajes. Todos estos elementos implican cierto control de planificación. Cuando el canal está ocupado, la tarea en curso ha de bloquearse, planificándose otra. En este caso, el problema de utilizar dichos canales directamente en el modelo es que, al considerar que la ejecución SystemC solo tiene una tarea activa en cada momento, cuando se produce un bloqueo en un canal del sistema operativo subyacente no hay ninguna otra tarea de la simulación que pueda ser desperada por el sistema operativo. Por tanto, no se produce únicamente el bloqueo de la tarea en ejecución, sino que la simulación completa es bloqueada ("deadlock"), impidiendo que otras tareas de la simulación puedan continuar para desbloquear el sistema.

En cuanto a las funciones de tiempo del sistema operativo nativo, como esperas temporales ("sleep"), temporizadores o alarmas, estas tampoco pueden ser utilizadas, ya que modelarían esperas en función del tiempo que tarda la simulación en ejecutar en el computador nativo, y no del tiempo estimado que tardaría el código en ejecutar en la plataforma destino.

Un razonamiento similar podemos utilizar con la comunicación HW/SW. Si intentamos realizar una comunicación HW a través del sistema operativo nativo, realizaremos un acceso a la plataforma HW del computador, y no al modelo de plataforma HW en SystemC.

El resto de operaciones de ayuda a la programación, como las operaciones matemáticas, de manejo de cadenas de caracteres o incluso de gestión de memoria, sí que pueden ser reutilizadas en el modelo, ya que no tienen efectos sobre el orden de ejecución de las tareas, lo que nos reducirá la lista de servicios a implementar.

Por su parte, la infraestructura SystemC proporciona servicios para el manejo de hilos, tiempo y eventos, aunque no proporciona servicios de planificación (el planificador de SystemC es de tipo FIFO sin prioridades). Para realizar la planificación de tareas se ha alterado externamente la ejecución de SystemC para que modele políticas y prioridades (sin cambiar el propio código de SystemC, manteniendo la portabilidad de la solución). Recordemos que en SystemC se ejecutan en cada instante temporal todas las tareas que no estén bloqueadas de manera secuencial. Cuando una tarea bloqueada recibe un evento de sincronización o de tiempo, ésta se despierta y es puesta en la lista de ejecución, bajo control del núcleo de simulación. Cuando alcanza la siguiente llamada a la función "wait" se bloquea en espera del siguiente evento, saliendo de la lista de ejecución.

La solución aplicada para el modelado de planificación es hacer que en cada momento solo haya una tarea SystemC desbloqueada por procesador del sistema a modelar. Cuando una tarea bloqueada recibe un evento de sincronización o tiempo, no se desbloquea en SystemC, sino que pasa a la lista de planificación del modelo de sistema operativo. No será hasta que el sistema operativo decida que ha de ser planificada, cuando se desbloqueará en SystemC. De esta forma se asegura que no haya varias tareas SystemC desbloqueadas a la vez. El desbloqueo y la consiguiente ejecución dependen así únicamente del uso de las prioridades y políticas de planificación asignadas a los hilos y procesos SW en la simulación.

Por último, para la gestión del tiempo utilizamos el eje de tiempo de SystemC. De esta forma, cuando una operación del modelo requiera un determinado tiempo, modelaremos ese tiempo esperando a que la simulación avance el tiempo correspondiente.

7.1.1 Servicios modelados del sistema operativo

Como se dijo anteriormente, el modelo de sistema operativo desarrollado basa en el estándar POSIX. La especificación POSIX.1 es la versión POSIX básica para uso sobre lenguaje C/C++. Se define en el estándar IEEE Std 1003.1-2001. En el se describen la interfaz

de sistema operativo y el entorno, incluyendo el interprete de comandos ("Shell") así como algunos programas y utilidades comunes. En la tesis se han implementado únicamente las funciones y servicios utilizables desde el código de aplicación. La Shell y otras utilidades no han sido consideradas.

El estándar POSIX, cubre en amplio espectro de servicios. Para concurrencia propone el uso de hilos y procesos. Define una política de planificación convencional y tres políticas de tiempo real: "Fifo", "Round Robin" y "Sporadic Server". Para cada una se definen un rango de prioridades.

El estándar define así mismo un gran número de mecanismos de planificación y sincronización. "Mutex", variables condicionales, semáforos, colas de mensajes, "sockets" o señales son algunos de los elementos de comunicación que han sido considerados en el modelo de sistema operativo. Esperas, temporizadores, alarmas, relojes de tiempo real, sistema de archivos o manejo de memoria son algunos de los elementos del estándar que también han sido considerados en el modelo.

Dado el tipo de sistemas que se pretende modelar, las capacidades de tiempo real merecen mención especial. POSIX describe múltiples funcionalidades de tiempo real, aunque su implementación exacta depende del sistema operativo. Así por ejemplo, Linux, que sigue el estándar en su mayor parte, no se considera un sistema operativo de tiempo real. Algunas de las funcionalidades de tiempo real de POSIX incluyen:

- Políticas de planificación
- Control de inversión de prioridades
- Temporizadores
- Señales de tiempo real
- Entrada/salida sincronizada
- Funciones de gestión de memoria
- Sincronización con espera active

No obstante, POSIX no define un conjunto de funciones para el desarrollo de controladores de dispositivos ("drivers") y la comunicación con el HW. Por ello, para realizar esta parte del modelo de sistema operativo la tesis se ha basado en algunas de las funciones de "drivers" de Linux. Por último, también se han considerado los detalles de implementación de algunas de las extensiones más comunes de Linux para tiempo real.

7.1.2 Capas del sistema operativo

Para presentar los detalles de implementación de sistema operativo, empezaremos dividiéndolo en capas. A grandes rasgos consideraremos la existencia de tres capas: API, HAL y Kernel.

La API es la capa accesible directamente desde el código de aplicación. En el modelo, esta capa está directamente definida por el estándar POSIX. Proporciona algunos servicios de ayuda a la programación, y funciones de manejo y acceso a las funcionalidades internas del sistema operativo.

El HAL es la capa encargada de la abstracción y comunicación con el HW. Esta es la capa que se encarga del control del procesador, cache, etc. También se encarga de la comunicación del SW con los periféricos del sistema. Además gestiona el control de interrupciones. Para explicar la integración del modelo de sistema operativo junto con el resto de la simulación, consideraremos en la descripción de esta capa las funciones de acceso directo a periféricos, como los controladores de dispositivo y los manejadores de interrupción.

El núcleo de ejecución del sistema operativo (kernel) es la parte que se encarga de la gestión de concurrencia, planificación, comunicación, sincronización y gestión de memoria. Es por tanto la encargada de proporcionar los servicios principales del sistema operativo. Además esta capa implementa la funcionalidad requerida por las otras dos capas, especialmente la API. En general, el núcleo del modelo de sistema operativo desarrollado se ha realizado basado también en el estándar POSIX.

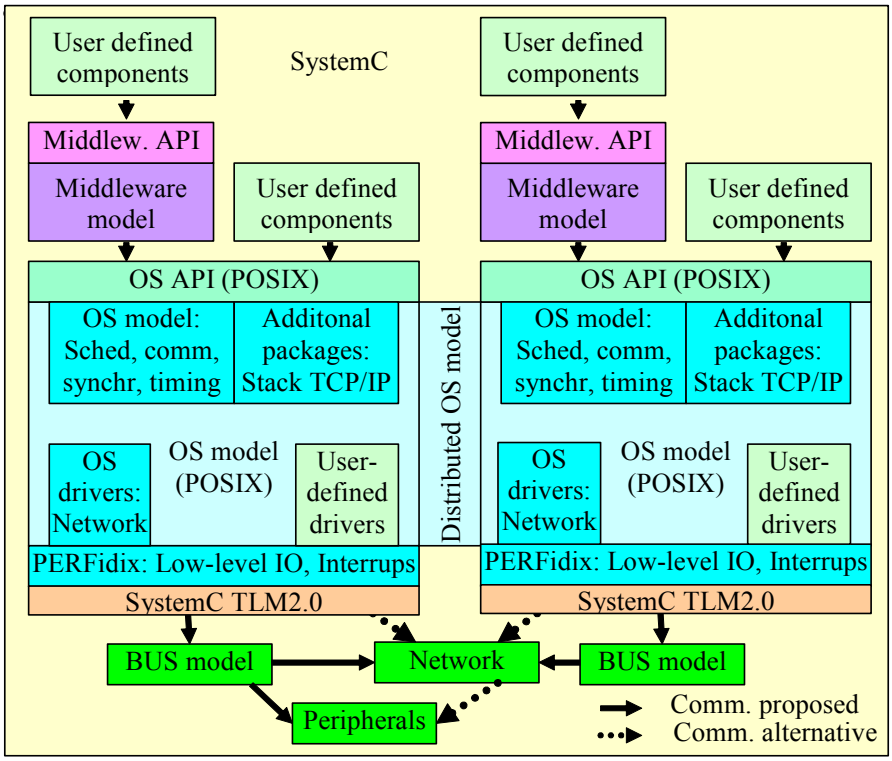


Figura 7-2: Modelo completo de sistema en SCoPE con todos los componentes SW

7.1.3 Modelo de Sistema Operativo basado en POSIX

El objetivo del desarrollo de la plataforma SW es proporcionar un entorno de simulación donde el código SW de aplicación real, preparado para ejecutar sobre un sistema operativo convencional, pueda ser simulado directamente, sin recodificar. Para que esto sea posible la plataforma proporciona un modelo de sistema operativo lo suficientemente completo para proporcionar los servicios requeridos por una aplicación estándar.

En general, si analizamos las cabeceras principales presentes en el estándar POSIX, veremos que pueden ser agrupadas en alguno de los tres casos. Algunas cabeceras que no se han considerado críticas no han sido aún resueltas, y han sido relegadas a extensiones futuras. La lista de cabeceras se presenta en la figura 7.3.

Library	IMPL	Library	IMPL	Library	IMPL	Library	IMPL	Library	IMPL
aio.h	Study	iconv.h	Host	pthread.h	SC	stropts.h	Study	utsname.h	Study
arpa/inet.h	Net	inttypes.h	Host	pwd.h	Study	sys/ipc.h	Study	sys/wait.h	SC
assert.h	Model	iso646.h	Host	regex.h	Host	sys/mman.g	Study	syslog.h	Host
complex.h	Host	langinfo.h	Host	sched.h	SC	sys/msg.h	Study	tar.h	Host
cpio.h	Model	libgen.h	Host	search.h	Host	sys/resource.h	Study	termios.h	Model
ctype.h	Host	limits.h	Model	semaphore.h	SC	sys/select.h	SC	tgmath.h	Host
dirent.h	Model	locate.h	Host	setjmp.h	Study	sys/sem.h	Study	time.h	SC
dlfcn.h	Model	math.h	Host	signal.h	SC	sys/shm.h	Study	trace.h	SC
errno.h	Host	monetary.h	Host	spawn.h	Study	sys/socket.h	Net	ucontest.h	Host
fcntl.h	Model	mqueue.h	SC	stdarg.h	Host	sys/stat.h	Study	ulimit.h	Model
fenv.h	Model	ndbm.h	Host	stdbool.h	Host	sys/statvfs.h	Study	unistd.h	SC
float.h	Host	net/if.h	Net	stddef.h	Host	sys/time.h	SC	utime.h	Model
fmtmsg.h	Model	netdb.h	Model	stdint.h	Host	sys/timeb.h	Study	utmpx.h	Study
fnmatch.h	Host	netinet/in.h	Net	stdio.h	Model	sys/times.h	SC	wchar.h	Host
ftw.h	Model	netinet/tcp.h	Net	stdlib.h	Host	sys/types.h	Host	wctype.h	Host
glob.h	Model	nl_types.h	Host	string.h	Host	sys/uio.h	Study	wordexp.h	Host
grp.h	Host	poll.h	Study	strings.h	Host	sys/in.h	Study		

SC: Functions completely implemented in SCoPE
Host: Functions that will use the host OS implementation
Model: Hardware platform specific functions. SCoPE will model them but specific implementations can be needed.
Study: Libraries currently not implemented, and candidates for future extensions
Net: Functions implemented as part of the TCP/IP stack included

Figura 7-3: Lista de las cabeceras POSIX modeladas.

Utilizando las funcionalidades subyacentes, el modelo de cada uno de las librerías del sistema operativo desarrollado se compone de funcionalidades aportadas de tres formas distintas. Una primera parte de las funcionalidades del sistema se han realizado reutilizando la funcionalidad del sistema operativo subyacente del propio PC donde se realiza la simulación ("Host", en la figura 7-3). Una segunda parte se ha desarrollado completamente, especialmente la relacionada con el control de ejecución de tareas y tiempos ("SC", "Net"). Por último, otras funciones se proporcionan adaptando la funcionalidad del sistema operativo subyacente para los propósitos requeridos en el modelo de sistema operativo ("Model").

Una vez listados los elementos de POSIX que serán considerados se procede a describir la implementación de los mismos tanto en el núcleo del sistema operativo como en las capas API y HAL.

7.1.3.1 Modelado del núcleo del sistema operativo

El modelo del núcleo del sistema operativo se ha centrado en la implementación de los elementos de concurrencia, planificación, comunicación, sincronización y tiempo.

7.1.3.1.1 Concurrencia

El control de concurrencia es la parte del sistema operativo que proporciona el paralelismo entre las distintas tareas SW. El estándar POSIX define dos niveles de paralelismo: a nivel de proceso y a nivel de hilo. Una definición rápida sugiere que el proceso es el elemento que controla los recursos que asigna el sistema para la correcta ejecución del código, mientras que el hilo es el elemento encargado de ejecutar dicho código. Por esta razón, todo proceso debe tener al menos un hilo, y cada hilo debe estar asociado a un proceso. El proceso controla el manejo de espacios de memoria, ficheros, señales y la comunicación y sincronización entre hilos. No obstante, cuando un proceso tiene un solo hilo suele utilizarse el término proceso para identificar tanto al proceso como al hilo. En lo sucesivo, para evitar problemas con esta dualidad, utilizaremos el término tarea para referirnos a un elemento de concurrencia junto con sus recursos independientemente de que nos refiramos un hilo o un proceso mono-hilo.

La ejecución de una hilo SW en bajo el control del sistema operativo se basa en la asignación de estados. Los estados en que se puede encontrar un hilo en SCoPE son 7:

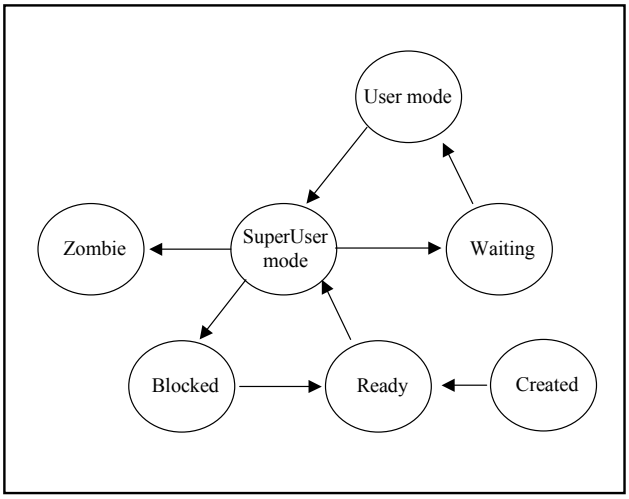


Figura 7-4: Estados de un hilo y transiciones posibles

- User (U): Esta ejecutando el código de usuario.

- SuperUser (SU): Esta ejecutando en modo kernel, por estar en una función de sistema operativo o manejando una interrupción

- Waiting (W): Esta listo para ejecutar en modo "User", esperando hasta que se le asigne un procesador libre

- Blocked (B): Esta bloqueado como consecuencia de una llamada al sistema operativo

- Ready (R): Esta listo para retomar la ejecución de una rutina de sistema operativo, esperando hasta que se le asigne un procesador libre

- Zombie (Z): El proceso ha muerto, pero sus datos aún no han sido tomados por su proceso padre

- Created (C): Acaba de ser creado y pasa a estado "Ready"

Cuando un hilo sale de estado SU y solo en ese caso, el procesador queda libre y el planificador decide que hilo debe ser el ejecutado. Para decidir cual debe ser ejecutado escoge el hilo de mayor prioridad en estado "Ready" y si no hay ninguno, el de mayor prioridad en estado "Waiting".

Esto significa que cuando un hilo que pasa de estado SU a U, (e.g. la vuelta de una llamada al sistema o una interrupción) no tiene garantizada la posesión del procesador. Esto se debe a que durante la ejecución en modo kernel otro hilo con mayor prioridad ha podido ser despertado. Sin embargo, ningún hilo que pasa de estado U a SU pierde el procesador, dado que durante la ejecución en modo U no se puede ejecutar ninguna operación que despierte un hilo, dado que la llegada de una interrupción produciría un paso automático a modo SU antes de ser tratada.

Una vez que cada tarea ha sido asociada a uno o varios procesadores, en cada instante de tiempo un planificador decide que tarea debe ejecutar en cada procesador, quedando el resto dormidas. La planificación se realiza en función del estado de la tarea en ese momento y de los procesadores reales que se hayan definido para el sistema el desarrollo (no puede haber mas hilos en ejecución en cada instante de tiempo que procesadores reales tenga el sistema). Esto se hace basado en las políticas y prioridades asociadas a cada tarea.

En SCoPE, una tarea puede ejecutar inicialmente en cualquier procesador bajo el control de un sistema operativo. No obstante esto se puede cambiar usando la función "sched_setaffinity", limitando los procesadores donde válidos.

7.1.3.1.2 Planificación: políticas y prioridades

Como se ha dicho, el planificador de SystemC no tiene capacidad de ordenar las tareas por importancia. De hecho, el planificador no tiene ni si quiera la capacidad de evitar que dos tareas asignadas al mismo procesador ejecuten simultáneamente, ya que el núcleo de ejecución no maneja información de recursos HW. Por ello es necesario desarrollar un control que gestione la ordenación y ejecución de las tareas SW. Para ello se aplicarán las políticas y prioridades descritas en el estándar POSIX.

Cada tarea que ejecuta bajo la supervisión del sistema operativo tiene asociada una política de planificación y una prioridad. Las políticas de planificación afectan a la ordenación de las tareas en cuatro supuestos principales:

- Cuando una tarea esta en ejecución y se bloquea
- Cuando una tarea esta bloqueada y pasa a ser ejecutable
- Cuando la tarea en ejecución llama a una función del sistema para cambiar la política o prioridad de otra tarea, o de si misma.
- Cuando una tarea en ejecución es expulsada por otra de mayor prioridad.

Los dos primeros supuestos sen dan habitualmente en los accesos a canales de sincronización, como mutexes o semáforos. Estos canales tienen la capacidad de bloquear y desbloquear tareas. Además, las esperas temporales, interrupciones y las señales también tienen la capacidad de bloquear y desbloquear tareas. Además, si una tarea de alta prioridad pasa a ser ejecutable o salta una interrupción, la tarea en curso es expulsada, produciéndose el cuarto supuesto.

Según el estándar POSIX, en un sistema multiproceso, cada proceso compite con los demás para hacerse con el control del procesador. Además, si un proceso contiene más de un hilo, hay dos niveles de posible planificación, por lo que la asignación del procesador al hilo puede hacerse de dos maneras.

Una opción es que una vez que el proceso se haga con el procesador los hilos de ese proceso compitan entre si. Para ello se ha de asignar el “scope” a nivel de proceso en los atributos del hilo. La otra opción es que el hilo compita directamente con el resto de procesos del sistema por el control del procesador. Para ello deberá tener el “scope” a nivel de sistema. En nuestro caso, dado que nuestra intención es aportar las capacidades de un sistema genérico ambas posibilidades han sido modeladas.

Una vez definido a qué nivel compiten procesos e hilos para tomar el procesador, debemos determinar cómo se toma esta decisión. Esto se define conforme a las políticas de planificación POSIX que han sido implementadas en el modelo de sistema operativo. El estándar POSIX define varias políticas de planificación, aunque permite la definición de políticas adicionales por parte del usuario. En el modelo de sistema operativo se consideran las cuatro habituales, que son:

- SCHED_FIFO Política de planificación “First in, first out“ (FIFO).
- SCHED_RR Política de planificación “Round Robin”.

- SCHED_SPORADIC Política de planificación de servidor esporádico.
- SCHED_OTHER Otra política de planificación

Además de la política se asigna un valor que identifica la prioridad de la tarea. Al planificar una tarea, se comprueba primero la prioridad de las tareas listas para ejecutar. En caso de que haya varias tareas con la misma prioridad se planifica en función de la política. Por ello, el modelo conceptual del planificador puede resumirse en una serie de listas, cada una para una prioridad. Dentro de cada lista, la ordenación depende del tipo de política asociada.

La política de planificación FIFO se basa en el mantenimiento de una lista ordenada de procesos o hilos. Los elementos a ejecutar se eligen en función el orden de la lista. Dado que la lista siempre se llena por el final y se vacía por el principio, los elementos del principio son los más antiguos, los que llevan más tiempo esperando. Esto se debe a que no se permiten reordenaciones internas en la lista.

Bajo la política FIFO, la modificación de los elementos de la lista se regula por las siguientes reglas:

- Cuando una tarea en ejecución es expulsada, se coloca en la cabeza de la lista.
- Cuando una tarea bloqueada se convierte en ejecutable, se coloca a la cola de la lista.
- Si se modifica la política o prioridad de una tarea mediante pthread_setschedprio() el efecto de este cambio en relación con la posición en la lista depende del tipo de modificación:
    - Si se aumenta la prioridad, la tarea se coloca en la cola de la lista.
    - Si no se modifica la prioridad, no se cambia la posición en la lista.
    - Si la prioridad disminuye, pasa al principio de la nueva lista.
- Si se modifica la política o prioridad de cualquier otra forma, la tarea pasa al final de la lista.
- Cuando una tarea cede voluntariamente el procesador mediante una función “yield”, la tarea pasa automáticamente al final de la lista.

La política de planificación Round-Robin es similar a la FIFO. En esta también se gestionan las tareas mediante listas ordenadas. Las modificaciones de la lista se realizan siguiendo las normas anteriores. Además, la política Round-Robin define un periodo máximo de tiempo durante el cual la tarea puede permanecer ocupando el procesador. Si la tarea excede ese periodo de tiempo, la tarea es expulsada y se planifica otra tarea de la misma prioridad. En caso de que no haya otra tarea de estas características, la tarea no es expulsada y sigue ejecutando durante un nuevo periodo de tiempo. Cuando la tarea es expulsada pasa al final de la cola, permitiendo que se ejecuten el resto de tareas de esa prioridad. Si una tarea que es expulsada no ha acabado su periodo de tiempo, se guarda esa información para que la próxima vez que sea planificada solo ejecute durante el tiempo restante.

La política de planificación de Servidor Esporádico se basa en la asignación de dos parámetros de tiempo, no solamente uno, como en el caso anterior. Además se le asignan dos prioridades. La ejecución en modo servidor esporádico divide la ejecución en periodos de tiempo. Estos periodos de tiempo se calculan conforme al reloj global del sistema, independientemente de que la tarea este en ejecución o no. La duración de cada periodo es indicada por uno de los parámetros. Durante cada periodo, la tarea puede estar ejecutando en alta prioridad durante un tiempo, y en baja durante el tiempo restante. El tiempo de alta prioridad se indica con el segundo parámetro temporal. Mientras una tarea se mantiene con una determinada prioridad, se comporta en la cola correspondiente como si tuviera política FIFO.

Por último, la política SCHED_OTHER no está especificada en el estándar. En general, el estándar define varios tipos de planificación, pero dejando mucha libertad para realizar las implementaciones de algunos de ellos. Por ello es necesario centrarnos en una implementación lo más extendida posible. Para ello nos hemos centrado en el kernel 2.6 de Linux. Si bien esta implementación no cumple con todas las características deseables en otros aspectos, como la libertad de decisión del ámbito de cada hilo, si que es válida para la definición de las políticas de planificación.

Siguiendo este modelo definiremos las políticas de planificación, tanto para planificación entre procesos (“System scope”) como entre hilos (“Process Scope”). Las políticas SCHED_FIFO y SCHED_RR (round-robin) se definen como de tiempo real, mientras que la restante, SCHED_OTHER, no tiene características de tiempo real. En modo SCHED_OTHER, la tarea se ejecuta mientras no se haya consumido su tiempo asociado (“time slice”), pero una vez finalizado no puede volver a ejecutar hasta que todas las tareas hayan consumido su "time slice" correspondiente y este se re-inicialice. De hecho, en esta política, la prioridad no fuerza el orden de ejecución, sino que únicamente indica el porcentaje de tiempo que la tarea puede ejecutar respecto a las otras tareas con esa política, por lo que no es considerada en la planificación.

En el modelo desarrollado cada política no permite cualquier prioridad. Los valores válidos de prioridad depende de la política que tenga asociada la tarea. Cada política tiene al menos 32 valores de prioridad dedicados. Las prioridades se pueden convertir entre políticas o no, dependiendo de la implementación seleccionada. En el modelo de plataforma, se ha optado por modelar ambos casos. Por tanto habrá prioridades exclusivas para una política y prioridades compartidas por varias políticas. Se han definido 96 valores de prioridad distintos, de tal forma que:

- 63-0 pertenecen a política FIFO
- 79-15 son de tipo “Round-Robin”,
- 95-31 son “Sporadic Server”.

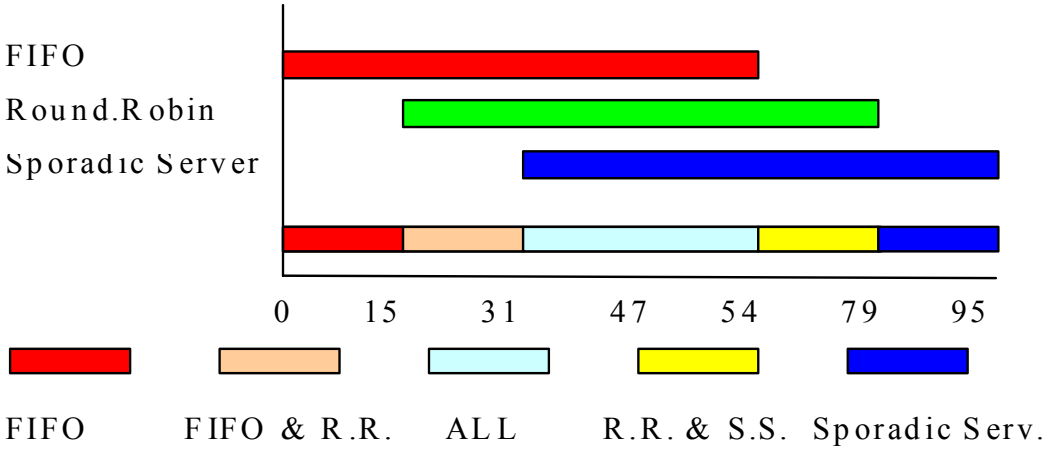


Figura 7-5: Prioridad de valores de prioridad válidos a las políticas de planificación.

De esta forma, las prioridades 0-15 son exclusivas para FIFO, las 79-95 son exclusivas para Sporadic-Server, y del 31 al 54 son compartidas por todos.

El problema que se presenta a la hora de implementar este sistema, es que la prioridad no identifica la política de cada proceso e hilo. Elementos con la misma prioridad pueden tener diferentes políticas. Esto implica que se debe realiza un planificador uniforme que sea capaz de gestionar políticas de todos los tipos conjuntamente. Por tanto debemos analizar cuidadosamente como funcionan las diferentes políticas para poder mezclarlas. Por ello, la idea de listas separadas para cada prioridad, a la par de ser costosa en recursos, no es una solución demasiado flexible.

Si analizamos el funcionamiento de las políticas presentadas anteriormente veremos que son muy similares. Todas se basan en una lista ordenada, de forma que se ejecuta antes el que primero está en la lista, que es normalmente el que más tiempo lleva esperando. Las diferencias se limitan a ver cuando el proceso es expulsado: si lo es por un bloqueo o decisión voluntaria, o si tiene un tiempo máximo de ejecución. Esto significa que se pueden mezclar tareas de distintas políticas sin mayores problemas en las listas. Dado que las diferencias entre políticas dependen de la existencia o no de máximos temporales de ejecución, solo se tendrá en cuenta en la gestión de las interrupciones del temporizador HW del sistema, que es el encargado de la gestión del tiempo en el modelo de sistema operativo.

La implementación del planificador se ha realizado en base a dos niveles de listas ordenadas, una global para los procesos y otra para los hilos dentro de cada proceso. En la lista se encuentran todas las tareas independientemente del procesador en que pueden ejecutar, de su política y de su prioridad. Cuando una tarea bloqueada pasa a ser ejecutable, esta se añade al final de la lista y espera a ser planificada. De la misma forma cuando una tarea que está en ejecución es expulsada pasa también al final de la lista y espera.

Para mantener dormido cada hilo se utiliza el tipo “sc_event” y las primitivas "notify" y "wait" de SystemC. Cuando un hilo desea tomar un procesador, se añade a la lista de hilos

en espera y se bloquea en una llamada a “wait”. Cuando un hilo va a liberar un procesador llama al planificador, que decide que hilo debe tomar a continuación dicho procesador y ejecuta la primitiva “notify” para despertarlo.

La realización del planificador soporta la definición de plataformas multiprocesador así como los distintos niveles de "scope", políticas de planificación y prioridades descritas en el apartado anterior.

Todo esto requiere la utilización de tres estructuras para contener la información de las tareas. La primera estructura contiene la información de recursos del proceso. En ella se encuentran el identificador de procesos, de grupo, los procesadores en que debe ejecutar, el procesador en el que se esta ejecutando, el porcentaje de uso o "nice", los manejadores de señal y la lista de señales recibidas, la lista de hijos del proceso, la lista de hilos del proceso, la lista de canales de comunicación/sincronización entre esos hilos, los temporizadores que dependen del reloj del proceso o la información estadística del proceso, como tiempo de usuario y tiempo de sistema. Dado que nuestra intención es modelar solo las características de planificación, comunicación y sincronización del sistema operativo, otra información como el control de la memoria, tablas de página, “stack”, descriptores de ficheros, tablas de “i-nodos”, o permisos de usuario no necesitan ser almacenadas.

La segunda estructura contiene la identificación de la tarea para la planificación en el nivel de sistema. Contiene la lista de hilos que compiten entre si dentro ese ámbito, y las características de prioridad y política de planificación con las que ese proceso compite con los otros procesos del sistema.

Por último, la tercera estructura contiene las características del hilo. Contiene el identificador de hilo, el estado del hilo, el canal de dormido del hilo, las características de prioridad y política de planificación con las que ese proceso compite con los otros hilos del proceso, la lista de señales enviadas específicamente al hilo, la lista de temporizadores dependientes del tiempo de ejecución del hilo y la información estadísticas del hilo.

Además de estas estructuras, existen otras dos que contienen la información de cada procesador y la información general del sistema operativo. Por cada procesador se guarda información sobre el proceso en ejecución mientras que en la información general están la lista de información de cada procesador, la lista de procesos, de hilos, y de elementos a competir en el "system scope", así como los canales de comunicación/sincronización entre procesos y los temporizadores dependientes de los relojes de sistema.

Para aprovechar las características de sistema multiprocesador se ha definido un planificador global por OS. El planificador intenta tener siempre un hilo en ejecución en cada procesador. Para ello, al ser global, debe decidir entre todos los elementos que compiten en el "system scope" cual es el más indicado para ejecutar. Por cada elemento que tenga al menos un hilo listo para ejecutar, comprueba si algún hilo de ese proceso esta en ejecución en otro procesador, y, en caso negativo, si puede ejecutar en el procesador que ha quedado libre. Si es

así, calcula su prioridad. Al final, el elemento con mayor prioridad es elegido, y se pasa a tomar la decisión de qué hilo ha de ejecutar.

7.1.3.1.3 Comunicación y sincronización

En un sistema multitarea, se deben implementar diferentes mecanismos que posibiliten la comunicación del núcleo con los procesos y la comunicación de los procesos entre si. En general podemos dividir los elementos relacionados con comunicación y sincronización de tareas en tres grupos: Comunicación dentro del proceso, comunicación entre procesos (IPC) y señales.

El primer grupo contiene las facilidades utilizadas para gestionar la interacción entre los hilos del mismo proceso. “Mutex”, variables condicionales, colas de mensajes y semáforos son los elementos básicos proporcionados para cubrir estas necesidades de comunicación. Estos elementos afectan a la ejecución de los hilos del proceso, ya que pueden bloquearlos o reanudar su ejecución según convenga. Estos también pueden modificar las prioridades de los hilos, modificando su planificación. Estas modificaciones se realizan para evitar incoherencias en la planificación, como inversiones de prioridad.

El segundo grupo de elementos contiene los servicios de comunicación entre procesos y de comunicación con elementos del exterior. Semáforos, colas de mensajes, memoria compartida, “spinlocks”, “sockets” y ficheros pueden ser utilizados para este propósito. En el modelo de sistema operativo se han modelado todos estos elementos, si bien el modelado de los “sockets” TCP/IP merece una sección aparte, y será tratado posteriormente.

Los semáforos y colas de mensajes utilizados para la comunicación entre procesos, son similares a los utilizados para la comunicación dentro del ámbito del proceso. De hecho se han desarrollado conjuntamente.

En todos los canales de comunicación, el funcionamiento en cuanto a control de concurrencia es bastante similar, independientemente de las características específicas derivadas de la funcionalidad de cada uno. Todos estos canales se han diseñado como una extensión de la clase "canal_base", que se usa para guardar en la información de hilo el elemento en el que esta bloqueado.

En todos los casos, la llamada a una función de acceso al canal implica el cambio de estado de “User” a “SuperUser”. A continuación se realiza la acción requerida, y finalmente se vuelve a modo “Waiting” para retomar la ejecución en modo usuario. En caso de que la función internamente bloquee el proceso, este pasará a estado “Blocked” y un nuevo hilo será asignado al procesador.

Cuando se produce un desbloqueo del canal (por ejemplo una llamada a "pthread_mutex_unlock"), existen dos posibles métodos de implementación: despertar al que tenga más prioridad o despertar todos los procesos que estén bloqueados. Si bien la primera opción sobrecarga menos al sistema, ya que no despierta hilos que no sean necesarios, y que

entorpecerán el cálculo del planificador, no es recomendable, ya que no permite una ejecución correcta en determinados casos.

Dado que el canal no es tomado en el mismo momento que el hilo es desbloqueado, sino que debe esperar a ser planificado, puede darse el caso de que durante este intervalo de espera la prioridad de los hilos cambie, o que el estado del hilo sea modificado por razones externas, con lo que el canal sería tomado por un hilo erróneo o quedaría bloqueado indefinidamente. Por esta razón se debe despertar a todos los hilos en espera de dicho evento y se deja que el planificador decida cual debe tomarlo y que hilos deberán volver a bloquearse en el canal.

En el desarrollo de los canales se han considerado los elementos de tiempo real del estándar, como las prioridades en las colas de mensajes o extensiones de tiempo real de los mutexes (e.g. las orientadas a prevenir inversiones de prioridad).

Los descriptores de fichero utilizados como elemento de comunicación se basan en archivos que pueden ser escritos y/o leídos. Principalmente para comunicaciones se han desarrollado los tipos “fifo” y “pipe”. Una fifo es un tipo de archivos con la propiedad de que los datos que se escriben en el, son leídos de tal forma que los primeros datos en entrar son los primeros en salir. Un pipe es un objeto accedido por un par de descriptores de fichero. El funcionamiento es similar al de la fifo, pero al tener descriptores dobles está especialmente indicado para comunicaciones bidireccionales.

Mediante los accesos a ficheros de comunicación los procesos también pueden ser bloqueados, dormidos o reanudados. El modo de acceso que determina si el archivo es bloqueante o no se define mediante el indicador “O_NONBLOCK”. Este indicador esta inicialmente desactivado cuando un descriptor de fichero es creado. Sin embargo puede ser modificado mediante el uso de la función “fcntl”, que ha sido implementada en el modelo para este propósito. Esto puede ser igualmente utilizado para la comunicación con los controladores de dispositivo que se realiza a través de descriptores de ficheros. La comunicación con el HW será tratada más adelante.

Para el modelado del sistema de ficheros se ha utilizado el sistema de ficheros del sistema operativo subyacente a la simulación (i.e. Linux). Si bien la gestión de los bloqueos al leer o escribir ficheros se modela directamente dentro del sistema operativo la gestión de los datos se delega en el sistema de archivos del PC donde se realiza la simulación. Para ello se genera un árbol de directorios a partir de un punto del sistema de ficheros del PC. Esto significa que a cada archivo que va a abrir se le añade un prefijo antes de realiza la apertura equivalente en el PC. La gestión del directorio “/dev/” se realiza de manera distinta. Ya que esta suele estar asociada a la plataforma HW se explicará su funcionamiento mas abajo, en la parte referente al modelado del HAL.

Además de los canales también se han implementado las señales, tanto para su uso como elementos de sincronización como de comunicación, transfiriendo unidades de información mínimas, conforme a lo recogido en el estándar. Una señal permite avisar a una

tarea de que un determinado evento se ha producido en el sistema. Las señales se pueden generar de diversas maneras:

- Como resultado de un evento bajo control del núcleo, como puede ser la finalización de un temporizador o un evento en una cola de mensajes.

- Por comunicación entre procesos, mediante el uso de las funciones del sistema "kill", "pthread_kill" y "siginfo".

- Por generación de un evento en entrada / salida (SIGIO, SIGTTIN,...)

Toda señal puede ser enviada a un hilo o a un proceso. Si es enviada a un hilo solo podrá ser tratada por dicho hilo, por lo que se añade a la lista de señales recibidas en la estructura de datos correspondiente a dicho hilo. En caso contrario, puede ser tratada por cualquiera de los hilos del proceso, por lo que se almacenará en le estructura de datos del proceso.

Cuando una señal es enviada a un hilo, se comprueba si el hilo esta esperando esa señal en una función "sigwait", en cuyo caso desbloquea la llamada a dicha función para que prosiga normalmente. Si no se esta esperando esa señal, y la señal no está enmascarada en ese hilo, se marca el hilo como "en interrupción", y se le despierta en caso de estar en estado “Blocked” y tener una prioridad que le permita ser despertado. A continuación, el proceso irá al estado de usuario tan rápido como el planificador se lo permita. Una vez en este punto, si aún está pendiente alguna interrupción, realizará las tareas definidas para manejar todas las interrupciones en espera no enmascaradas, y finalmente volverá al estado donde se encontraba inicialmente. Las señales podrán ser ignoradas, tratadas con el manejador por defecto definido en POSIX o con el manejador definido por el usuario utilizando las funciones "signal" o "sigaction".

Cuando la señal es enviada a un proceso, el modelo de sistema operativo comprueba si algún hilo está esperando la señal, en cuyo caso se le despierta para que continúe la ejecución. Si ningún hilo está esperando la señal, todos los hilos que no tengan enmascarada la señal se marcan como "en interrupción" y realizan el mismo proceso que en el caso de que la señal fuera dirigida a ellos.

Por tanto, las señales generadas no se reciben en el proceso de una manera síncrona y solo son tratadas cuando el proceso cambia de modo kernel a modo usuario. Esto implica que existe un retardo entre la generación de la señal y su tratamiento. Además, las señales no tienen una cola de llegadas, con lo que si se reciben varias señales de un tipo antes de que sean atendidas las llegadas anteriores se sobre-escribe la información, perdiendo el número de señales que han llegado. Para evitar esto el estándar POSIX define unas señales especiales, denominadas "de tiempo real" que sí mantienen una lista de todas las señales recibidas. Estas señales también han sido implementadas dentro del modelo de sistema operativo.

7.1.3.1.4 Gestión del tiempo

El modelo de sistema operativo lleva un control interno del tiempo basado en relojes. El reloj principal, que identifica el tiempo del sistema es el "CLOCK_MONOTONIC". El reloj de tiempo real "CLOCK_REALTIME" y relojes por cada proceso y hilo también han sido implementados en el modelo de sistema. Los elementos de manejo de tiempo, como temporizadores y alarmas han sido convenientemente modelados, de tal forma que se activan o desactivan conforme cualquiera de los relojes anteriores, a decisión de usuario.

Para modelar todos estos elementos es necesario simular el funcionamiento del temporizador HW del sistema. Para ello se ha utilizado un hilo SystemC ("SC_THREAD") extra que permanece dormido normalmente y que salta cada vez que deba despertar el temporizador HW. Al despertar, este hilo lanza una interrupción HW que es gestionada por el manejador correspondiente. Cada vez que se ejecuta este manejador, este actualiza todos los contadores de tiempo, tanto en los procesos en ejecución como en los temporizadores y alarmas del sistema y realiza la acción correspondiente en aquellos que hayan finalizado su cuenta de tiempo. Por último, el temporizador HW realiza una interrupción en todos los procesadores lo que provoca el cambio de estado en el proceso que estaba en ejecución. Esto implica que el planificador debe decidir el próximo hilo que debe ejecutar.

El temporizador HW debe ser usado incluso para el manejo de bloqueos con tiempos límites ("timeouts"). Aunque se podría pensar en realizar los tiempos límites usando las primitivas "wait" de SystemC esto no es recomendable. En primer lugar, un bloqueo con tiempo límite solo es despertado en la realidad durante las interrupciones del tic-timer, con lo que poniendo el valor del tiempo límite directamente en la primitiva "wait" el resultado no sería adecuado. En segundo lugar el cálculo del tiempo límite puede no estar referido al reloj del sistema, sino a relojes de hilos o procesos, con lo que basarlo en el tiempo de simulación SystemC no es adecuado.

7.1.3.2 API: Soporte POSIX

Una vez definidos los servicios que el modelo de sistema operativo desarrollado es capaz de proporcionar al SW, la siguiente tarea es proponer una interfaz que permita al SW de aplicación acceder a esos servicios. Para definir esta interfaz para la programación de aplicaciones (API) se ha utilizado el estándar POSIX.

El objetivo de esta tesis, no es generar un modelo completo y comercial con conformidad completa con el estándar. El objetivo es solamente proporcionar una serie de funcionalidades que sirvan de soporte a la ejecución del código SW de aplicación del sistema en diseño. Por ello, no se ha trabajado en la tesis en analizar los subconjuntos estrictamente definidos en el estándar y estudiar si cada componente puede ser modelado en una simulación SystemC. El trabajo se ha limitado a proporcionar todos aquellos servicios que se han considerado interesantes para el diseño de sistemas embebidos.

La utilización de las cabeceras POSIX dentro de una simulación ejecutando sobre un PC en Linux o UNIX presenta un problema a tener en cuenta. Estos sistemas operativos subyacentes tienen sus propias cabeceras y la simulación SystemC ha de ser compilada con

ellas. Por tanto no se pueden replicar los nombres de las funciones. En consecuencia se han implementado nuevas cabeceras en las que todas las funciones llevan añadido el prefijo "uc_". Además, junto a cada declaración existe un "#define" que reemplaza el nombre estándar por el nuevo nombre con el prefijo. Por ejemplo nos encontramos un código como el siguiente para cada función:

pid_t uc_getpid (); // reemplaza la función estándar "getpid ()"

#define getpid () uc_getpid () // Redirige todas las llamadas en el código SW modelado

A continuación pasaremos a describir la implementación de la API POSIX desarrollada. Para ello se describirán las principales cabeceras implementadas completamente en el modelo de sistema operativo. Las cabeceras con modificaciones menores (como la "stdio.h" o "stdlib.h") no son listadas para no extender en exceso la explicación.

7.1.3.2.1 Threads: <phtread.h>

La librería pthread.h es la más importante de las cabeceras implementadas. Esta librería define las funciones para el manejo de threads y sus comunicaciones. En ella se definen las funciones para la creación y destrucción de threads, funciones para definir los atributos de los threads y varios canales de sincronización entre threads: mutex, variables condicionales, rwlocks, spinlocks y barreras. Las extensiones de tiempo real como el control de inversión de prioridad en los mutex también han sido consideradas.

Las funciones que han sido implementadas son:

```
int pthread_atfork(void (*)(void), void (*)(void),void(*)(void));
int pthread_attr_destroy(pthread_attr_t *);
int pthread_attr_getdetachstate(const pthread_attr_t *, int *);
int pthread_attr_getinheritsched(const pthread_attr_t *restrict,int
*restrict);
int pthread_attr_getschedparam(const pthread_attr_t *restrict,struct
sched_param *restrict);
int pthread_attr_getschedpolicy(const pthread_attr_t *restrict,int
*restrict);
int pthread_attr_getscope(const pthread_attr_t *restrict, int
*restrict);
int pthread_attr_getstack(const pthread_attr_t *restrict, void
**restrict, size_t *restrict);
int pthread_attr_getstackaddr(const pthread_attr_t *restrict, void
**restrict);
int pthread_attr_getstacksize(const pthread_attr_t *restrict, size_t
*restrict);
int pthread_attr_init(pthread_attr_t *);
int pthread_attr_setdetachstate(pthread_attr_t *, int);
int pthread_attr_setinheritsched(pthread_attr_t *, int);
int pthread_attr_setschedparam(pthread_attr_t *restrict, const struct
sched_param *restrict);
```

```
int pthread_attr_setschedpolicy(pthread_attr_t *, int);
int pthread_attr_setscope(pthread_attr_t *, int);
int pthread_attr_setstack(pthread_attr_t *, void *, size_t);
int pthread_attr_setstackaddr(pthread_attr_t *, void *);
int pthread_attr_setstacksize(pthread_attr_t *, size_t);
int pthread_cancel(pthread_t);
void pthread_cleanup_push(void (*)(void *), void *);
void pthread_cleanup_pop(int);
int pthread_cond_broadcast(pthread_cond_t *);
int pthread_cond_destroy(pthread_cond_t *);
int pthread_cond_init(pthread_cond_t *restrict, const
pthread_condattr_t *restrict);
int pthread_cond_signal(pthread_cond_t *);
int pthread_cond_timedwait(pthread_cond_t *restrict, pthread_mutex_t
*restrict, const struct timespec *restrict);
int pthread_cond_wait(pthread_cond_t *restrict, pthread_mutex_t
*restrict);
int pthread_condattr_destroy(pthread_condattr_t *);
int pthread_condattr_getclock(const pthread_condattr_t *restrict,
clockid_t *restrict);
int pthread_condattr_getpshared(const pthread_condattr_t *restrict,
int *restrict);
int pthread_condattr_init(pthread_condattr_t *);
int pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
int pthread_condattr_setpshared(pthread_condattr_t *, int);
int pthread_create(pthread_t *restrict, const pthread_attr_t
*restrict, void *(*)(void *), void *restrict);
int pthread_detach(pthread_t);
int pthread_equal(pthread_t, pthread_t);
void pthread_exit(void *);
int pthread_getcpuclockid(pthread_t, clockid_t *);
int pthread_getschedparam(pthread_t, int *restrict,struct sched_param
*restrict);
void *pthread_getspecific(pthread_key_t);
int pthread_join(pthread_t, void **);
int pthread_key_create(pthread_key_t *, void (*)(void *));
int pthread_key_delete(pthread_key_t);
int pthread_mutex_destroy(pthread_mutex_t *);
int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict, int
*restrict);
int pthread_mutex_init(pthread_mutex_t *restrict, const
pthread_mutexattr_t *restrict);
int pthread_mutex_lock(pthread_mutex_t *);
int pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int, int
*restrict);
int pthread_mutex_timedlock(pthread_mutex_t *, const struct timespec
*);
int pthread_mutex_trylock(pthread_mutex_t *);
int pthread_mutex_unlock(pthread_mutex_t *);
int pthread_mutexattr_destroy(pthread_mutexattr_t *);
int pthread_mutexattr_getprioceiling( const pthread_mutexattr_t
*restrict, int *restrict);
int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
*restrict, int *restrict);
```

```
int pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
int *restrict);
int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
int *restrict);
int pthread_mutexattr_init(pthread_mutexattr_t *);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
int pthread_mutexattr_settype(pthread_mutexattr_t *, int);
int pthread_once(pthread_once_t *, void (*)(void));
int pthread_rwlock_destroy(pthread_rwlock_t *);
int pthread_rwlock_init(pthread_rwlock_t *restrict, const
pthread_rwlockattr_t *restrict);
int pthread_rwlock_rdlock(pthread_rwlock_t *);
int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict, const
struct timespec *restrict);
int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict, const
struct timespec *restrict);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *);
int pthread_rwlock_trywrlock(pthread_rwlock_t *);
int pthread_rwlock_unlock(pthread_rwlock_t *);
int pthread_rwlock_wrlock(pthread_rwlock_t *);
int pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
int pthread_rwlockattr_getpshared( const pthread_rwlockattr_t
*restrict, int *restrict);
int pthread_rwlockattr_init(pthread_rwlockattr_t *);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
pthread_t pthread_self(void);
int pthread_setcancelstate(int, int *);
int pthread_setcanceltype(int, int *);
int pthread_setschedparam(pthread_t, int, const struct sched_param
*);
int pthread_setschedprio(pthread_t, int);
int pthread_setspecific(pthread_key_t, const void *);
void pthread_testcancel(void);
int uc_pthread_spin_lock(pthread_spinlock_t *lock);
int uc_pthread_spin_trylock(pthread_spinlock_t *lock);
int uc_pthread_spin_destroy(pthread_spinlock_t *lock);
int uc_pthread_spin_init(pthread_spinlock_t *lock, int pshared);
int uc_pthread_spin_unlock(pthread_spinlock_t *lock);
```

Todas estas funciones se han diseñado para funcionar con los identificadores descritos en el estándar POSIX:

```
PTHREAD_CANCEL_ASYNCHRONOUS
PTHREAD_CANCEL_ENABLE
PTHREAD_CANCEL_DEFERRED
PTHREAD_CANCEL_DISABLE
PTHREAD_CANCELED
PTHREAD_COND_INITIALIZER
PTHREAD_CREATE_DETACHED
PTHREAD_CREATE_JOINABLE
PTHREAD_EXPLICIT_SCHED
PTHREAD_INHERIT_SCHED
PTHREAD_MUTEX_INITIALIZER
```

```
PTHREAD_ONCE_INIT
PTHREAD_PRIO_INHERIT
PTHREAD_PRIO_NONE
PTHREAD_PRIO_PROTECT
PTHREAD_PROCESS_SHARED
PTHREAD_PROCESS_PRIVATE
PTHREAD_SCOPE_PROCESS
PTHREAD_SCOPE_SYSTEM
```

Se han definido los tipos correspondientes indicados en el estándar:

```
pthread_attr_t
pthread_cond_t
pthread_condattr_t
pthread_key_t
pthread_mutex_t
pthread_mutexattr_t
pthread_once_t
pthread_rwlock_t
pthread_rwlockattr_t
pthread_t
pthread_barrier_t
pthread_barrierattr_t
pthread_spinlock_t.
```

7.1.3.2.2 Planificación: <sched.h>

Las funciones de interfaz dedicadas al control de planificación se encuentran en la cabecera sched.h. Aquí se definen las funciones para cambiar la prioridad y política de los procesos. Las funciones principales son las siguientes:

```
int sched_get_priority_max(int);
int sched_get_priority_min(int);
int sched_getparam(pid_t, struct sched_param *);
int sched_getscheduler(pid_t);
int sched_rr_get_interval(pid_t, struct timespec *);
int sched_setparam(pid_t, const struct sched_param *);
int sched_setscheduler(pid_t, int, const struct sched_param *);
int sched_yield(void);
int uc_sched_setaffinity(pid_t pid, unsigned int len, unsigned long
*mask);
int uc_sched_getaffinity(pid_t pid, unsigned int len, unsigned long
*mask);
```

Para su funcionamiento se ha generado la estructura sched_param_t, mediante la cual se proporciona toda la información de planificación a las funciones anteriores.

Por ultimo se han implementado las políticas de planificación recogidas en el estándar:

SCHED_FIFO, SCHED_RR, SCHED_SS, SCHED_OTHER

7.1.3.2.3 Semaforos: <semaphore.h>

Para la sincronización tanto de hilos como de procesos se ha implementado la cabecera de semáforos:

```
int sem_close(sem_t *);
int sem_destroy(sem_t *);
int sem_getvalue(sem_t *restrict, int *restrict);
int sem_init(sem_t *, int, unsigned);
sem_t *sem_open(const char *, int, ...);
int sem_post(sem_t *);
int sem_timedwait(sem_t *restrict, const struct timespec *restrict);
int sem_trywait(sem_t *);
int sem_unlink(const char *);
int sem_wait(sem_t *);
```

7.1.3.2.4 Message Queues: <mqueue.h>

Las colas de mensajes son los elementos de POSIX propuestos para transferencia de datos sincronizada entre threads o procesos. Las transacciones llevan asociada una prioridad que sirve para reordenar los mensajes dentro de la cola. Las funciones implementadas son:

```
int mq_close(mqd_t);
int mq_getattr(mqd_t, struct mq_attr *);
int mq_notify(mqd_t, const struct sigevent *);
mqd_t mq_open(const char *, int, ...);
ssize_t mq_receive(mqd_t, char *, size_t, unsigned *);
int mq_send(mqd_t, const char *, size_t, unsigned );
int mq_setattr(mqd_t, const struct mq_attr *restrict, struct mq_attr
*restrict);
ssize_t mq_timedreceive(mqd_t, char *restrict, size_t, unsigned
*restrict, const struct timespec *restrict);
int mq_timedsend(mqd_t, const char *, size_t, unsigned ,
const struct timespec *);
int mq_unlink(const char *);
```

Se proporciona así mismo la estructura mq_attr conteniendo los campos indicados en el estándar:

```
long mq_flags Message queue flags.
long mq_maxmsg Maximum number of messages.
long mq_msgsize Maximum message size.
long mq_curmsgs Number of messages currently queued.
```

7.1.3.2.5 Signals: <signal.h>

Se ha implementado la cabecera asociada a la generación y gestión de señales enviadas a threads o procesos. Se han implementado tanto las señales normales como las de tiempo real, con sus consiguientes características propias. Las funciones implementadas son:

```
void (*bsd_signal(int, void (*)(int)))(int);
int kill(pid_t, int);
int killpg(pid_t, int);
int pthread_kill(pthread_t, int);
int pthread_sigmask(int, const sigset_t *, sigset_t *);
int raise(int);
int sigaction(int, const struct sigaction *restrict,
struct sigaction *restrict);
int sigaddset(sigset_t *, int);
int sigdelset(sigset_t *, int);
int sigemptyset(sigset_t *);
int sigfillset(sigset_t *);
int sighold(int);
int sigignore(int);
int siginterrupt(int, int);
int sigismember(const sigset_t *, int);
void (*signal(int, void (*)(int)))(int);
int sigpause(int);
int sigpending(sigset_t *);
int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
int sigqueue(pid_t, int, const union sigval);
int sigrelse(int);
void (*sigset(int, void (*)(int)))(int);
int sigsuspend(const sigset_t *);
int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
const struct timespec *restrict);
int sigwait(const sigset_t *restrict, int *restrict);
int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
```

Además se han definido los identificadores de los 32 tipos de señales así como las macros y tipos de datos indicados por el estándar:

[1] SIG_DFL, SIG_ERR, SIG_HOLD, SIG_IGN
[2] SIG_BLOCK, SIG_UNBLOCK, SIG_SETMASK
[3] SA_NOCLDSTOP, SA_NOCLDWAIT, SA_RESTART, SA_SIGINFO, ...
[4] SIGEV_NONE, SIGEV_SIGNAL, SIGEV_THREAD
[5] sig_atomic_t, sigset_t, stack_t, sigstack, siginfo_t, sigaction, sigval and sigeventstack_t

7.1.3.2.6 Relojes y temporizadores: <time.h>

La cabecera time.h contiene las funciones para la gestión de los relojes del sistema y la generación de temporizadores periódicos:

```
int clock_getcpuclockid(pid_t, clockid_t *);
int clock_getres(clockid_t, struct timespec *);
int clock_gettime(clockid_t, struct timespec *);
int clock_nanosleep(clockid_t, int, const struct timespec *, struct
timespec *);
int clock_settime(clockid_t, const struct timespec *);
double difftime(time_t, time_t);
int nanosleep(const struct timespec *, struct timespec *);
```

```
time_t time(time_t *);
int timer_create(clockid_t, struct sigevent *restrict, timer_t
*restrict);
int timer_delete(timer_t);
int timer_gettime(timer_t, struct itimerspec *);
int timer_getoverrun(timer_t);
int timer_settime(timer_t, int, const struct itimerspec *restrict,
struct itimerspec *restrict);
```

Se han definido así mismo las macros junto con su funcionalidad asociada, siguiendo el estándar:

```
CLOCKS_PER_SEC, CLOCK_PROCESS_CPUTIME_ID, CLOCK_THREAD_CPUTIME_ID
CLOCK_REALTIME, TIMER_ABSTIME, CLOCK_MONOTONIC
```

7.1.3.2.7 Funciones estandar unix: <unistd.h>

```
pid_t getpid();
gid_t getgid();
pid_t getppid(void);
pid_t getpgid (pid_t pid);
int   setpgid (pid_t pid,pid_t pgid) ;
pid_t getpgrp (void);
int   setpgrp (void);
long gethostid(void);
int open(const char *camino, int flags);
int open(const char *camino, int flags, mode_t modo);
int creat(const char *camino, mode_t modo);
int pipe(int filedes[2]);
int close(int fd);
int write(int fd,const void* buf,size_t len);
int read(int fd,void* buf,size_t len);
int pread(int fd, void *buf, size_t count, off_t offset);
int pwrite(int fd, const void *buf, size_t count, off_t offset);
int nice(int inc);
int usleep(unsigned long int useconds);
unsigned int sleep(unsigned int seconds);
unsigned int alarm(unsigned int seconds);
int pause(void);
void _exit(int status);
void exit(int status);
```

La función “fork” también ha sido implementada, aunque con ciertas limitaciones. Se recomienda usarla únicamente al comienzo de los procesos. La función está preparada para soportar la creación de los canales de comunicación necesarios para la conexión entre padre e hijo, como semáforos o colas de mensajes. Sin embargo, su uso posterior a haber realizado accesos a dichos canales puede dar lugar a errores. Esto se debe a que la simulación de espacios de memoria distintos dentro de una simulación, que es un único proceso en el PC presenta ciertos inconvenientes.

7.1.3.2.8 Otras librerias: <sys/>

Dentro de la carpeta "sys" recogida en el estándar POSIX se han implementado algunas librerías, que no vamos a detallar para no extender más la presentación de la implementación. Estas librerías son:

- sys/wait.h
- sys/types.h
- sys/time.h
- sys/select.h
- sys/resource.h
- sys/ioctl.h
- sys/do_select.h

7.1.3.2.9 Funciones de red

El soporte de red TCP/IP no se ha implementado de cero, sino que se ha integrado una pila IP externa. Por consiguiente las funciones POSIX de red serán presentadas junto con la integración de dicho paquete externo, en una sección posterior.

7.1.3.2.10 Funciones específicas para modelado por sobrecarga de tipos

Como se explicó en el capítulo anterior, la estimación del tiempo de ejecución SW mediante la técnica de sobrecarga implicaba la sustitución de los tipos de datos convencionales por otros tipos sobrecargados. Como consecuencia, si bien el acceso desde estos nuevos tipos a funciones externas estaba resuelto mediante los operadores de conversión implícita, el manejo de "arrays" necesita un tratamiento especial. Los "arrays" de tipos sobrecargados han de ser transformados en "arrays" de tipos básicos antes de ser recibidos por el modelo de sistema operativo, ya que dicho modelo es independiente de la técnica de estimación utilizada.

Por esta razón se han implementado funciones adicionales para manejo de "arrays". Esto tiene especial relevancia en las funciones "write", "read" y similares de la "unistd.h" y en el manejo de Springs ("string.h").

7.1.3.3 Conformidad con el estándar POSIX

Como se dijo anteriormente, el objetivo de este trabajo no es desarrollar una implementación del estándar POSIX completa. Sin embargo, para comprobar la cercanía del modelo al estándar, se ha analizado el comportamiento del mismo mediante unos test de conformidad con el estándar.

La comprobación de conformidad se ha realizado utilizando la batería de test Open POSIX Test Suite 1.5.0 [OPTS]. Este es un proyecto de código abierto que pretende

proporcionar un conjunto de test que permitan verificar la conformidad funcional de una implementación respecto al estándar IEEE Std 1003.1-2001.

Los tests han tenido que ser adaptados para compilar con C++ y han sido integrados en SCoPE. Se ha generado un sistema de verificación automática de tal forma que se comportan como una batería de test de regresión. Una vez arrancada la batería, se van ejecutando automáticamente todos los test y se genera un reporte de qué tests han pasado correctamente y cuales han presentado algún problema.

A continuación se listan los problemas detectados mediante el test:

- **Librerías no implementadas:**
  - El principal problema encontrado en los test es que no toda la funcionalidad chequeada ha sido implementada. La lista de elementos no implementados en la siguiente:
    - Aio
    - sys_mman
    - sigaltstack
    - Pthread_stack
    - Escritura con almacenamiento intermedio como “fsync”, etc…
    - Lio_listio

- **Limitaciones en ejecución:**
  - Se han detectado errores en algunos test en la gestión de identificadores de usuario y modificación de permisos. El modelo de sistema operativo no está diseñado para emular la ejecución de múltiples usuario y la gestión de permisos. Esto se debe a que no se ha considerado una característica crítica en sistemas embebidos.

- **Test de sobrecarga de recursos:**
  - Se han eliminado aquellos test que comprueban los fallos producidos por el uso excesivo de recursos. La comprobación de fallos en funciones como “malloc” en una simulación SystemC tiende a bloquear el PC, por lo que han sido sacados de la lista de test de regresión
  - Se han eliminado así mismo aquellos test que realizan comprobaciones de tiempo mediante esperas muy largas. Dado que la simulación de los test es más lenta que una ejecución directa en un PC, estos test de tiempo lastran demasiado la comprobación, y también han sido eliminados de los test de regresión.

Los resultados obtenidos una vez ejecutados los test se presentan en la siguiente tabla. En dicha ejecución se han quitando aquellos test comprometidos por los problemas citados en los puntos 1 y 3. Los test fallados se centran en las limitaciones del punto 2. Los ejemplos no resueltos, no testados o no soportados se deben a funcionalidades POSIX no implementadas.

RESULTADOS	NUMERO DE TESTS	PORCENTAJE (%)
PASSED	1066	96,47 (passed+failed)
FAILED	39	3,52 (passed+failed)
UNRESOLVED	71	-
UNTESTED	35	-
UNSUPPORTED	20	-
Total	**1231**	-

Tabla 7-1: Comprobación POSIX.

7.1.3.4 Modelado de la comunicación con la plataforma HW: HAL

Uno de las elementos más importantes en los sistemas embebidos HW/SW es la comunicación entre los elementos SW y la plataforma HW. Para que el SW de aplicación pueda interactuar adecuadamente con los componentes HW del sistema se necesita que la infraestructura de simulación proporcione las funcionalidades de comunicación necesarias. Por ello el modelo sistema operativo desarrollado también contiene una parte orientada a la interacción del SW con los periféricos.

El modelado de la comunicación del SW con los componentes de la plataforma HW realizada en SCoPE se basa en una relaciónes maestro-esclavo. El procesador, y por tanto la parte SW, toma el rol de maestro, mientras que los periféricos actúan como esclavos respondiendo a las peticiones del SW. No obstante, en determinados casos es necesario que los periféricos inicien la comunicación. Como consecuencia la comunicación entre SW y HW se realiza de dos maneras:

- el SW realiza peticiones al HW mediante accesos al bus del sistema
- el HW envía eventos al SW por medio de interrupciones

En el primer caso el SW envía peticiones de lectura o escritura. Para ello el SW envía por el bus la dirección del registro al que quiere acceder y espera que el periférico actúe en consecuencia y responda. En el caso de las interrupciones, estas son recibidas por el SW. El SW encargado de la gestión de las interrupciones ha de contener la capacidad para averiguar la razón que provocó la petición de atención y actuar en consecuencia.

Para independizar el SW de aplicación en la medida posible de la plataforma HW facilitando su desarrollo y portabilidad, la funcionalidad requerida para realizar la comunicación con el HW debe ser separada y aislada del resto del programa. Esto permite desarrollar el SW de aplicación de forma independiente y que esta funcionalidad pueda ser reutilizada por el resto de programas del sistema. Ambas posibilidades permiten optimizar el desarrollo de SW tanto desde el punto de vista de confiabilidad como de esfuerzo. A la funcionalidad específica encargada de la gestión directa de un periférico se le conoce habitualmente como conformando los controladores de dispositivo o “device drivers”.

El acceso al HW desde el SW se puede realizar en sistemas reales mediante dos procedimientos: escribiendo directamente sobre los punteros que representan las direcciones de memoria de los periféricos, o utilizando las funciones proporcionadas por el sistema operativo a tal efecto. En sistemas sin gestión de memoria virtual, las direcciones del mapa de memoria y las indicadas en el código son idénticas, con lo que ambas opciones son aplicables. De hecho, estos sistemas suelen llevar asociados sistemas operativos sencillos, como eCos o uC-OS, que no proporcionan servicios para acceso al bus. En sistemas con memoria virtual, el sistema operativo ha de realizar la transformación de direcciones, con lo que el acceso mediante los servicios de dicho sistema es obligatorio. En consecuencia, en la tesis se han desarrollado mecanismos de soporte para ambas opciones.

En cuanto a las interrupciones, en ambos tipos de sistemas han de ser manejadas por el sistema operativo, con lo que se ha desarrollado un único mecanismo de modelado

7.1.3.4.1 Acceso mediante punteros

La ejecución de código SW que contiene accesos a periféricos mediante punteros en la simulación SystemC produce errores críticos de fallos de segmento al ejecutar sobre el computador nativo. Esto se debe a que se pretende acceder a direcciones de memoria que no han sido habilitadas por el sistema de gestión de memoria del sistema operativo nativo. Para resolver estos errores, el mecanismo de simulación desarrollado realiza dos procedimientos: evita los fallos del sistema y redirecciona las peticiones al modelo de bus.

Para evitar los fallos provocados al acceder a direcciones de memoria inválidas el sistema habilita dichas direcciones, forzando un mapeo previo, mediante la función “mmap” del sistema operativo nativo (i.e. Linux). Esta función hace que dichos accesos se realicen sobre un archivo asociado, en lugar de provocar un error. Por tanto, para realizar la comunicación en la simulación basta con leer o escribir la posición correspondiente del archivo y enviar dicha información al bus.

El problema se transforma entonces en detectar cuando hay un acceso, si es lectura o escritura, y la dirección en la que leer o escribir. Para realizar la detección de los accesos la solución aplicada es retrasar el mapeo de memoria. Al no realizar el mapeo de memoria al comenzar la simulación, el primer acceso provoca un evento de violación de segmento. Dicho error provoca la ejecución del manejador asociado a la señal SIGSEGV.

El manejador desarrollado recibe como argumento la dirección que provocó el error, o lo que es lo mismo, la dirección de la comunicación HW/SW. Internamente, el manejador mapea la memoria llamando a “mmap”, permitiendo que se pueda reejecutar el código correctamente. No obstante queda por resolver la determinación del tipo de acceso: lectura/escritura. Además este procedimiento solo es válido para el primer acceso, ya que una vez que la dirección es mapeada los siguientes accesos no producen error.

Para hacer que extender el procedimiento a todos los accesos de la simulación, la solución implementada es inyectar código justo después del acceso que provocó el error. Este

código inyectado deshace el mapeo de memoria, restaura y ejecuta el código inicial. La inyección de código se realiza en el manejador de la señal.

La detección del tipo de comunicación se ha resuelto utilizando otro tipo de señal. Para provocar señales distintas para lectura y escritura se realiza un nuevo mapeo. El mapeo durante los accesos se mantiene como se explico anteriormente, pero además, en lugar de dejar la dirección sin mapeada el resto del tiempo, se realiza un mapeo de las direcciones sobre archivos vacíos con permiso de solo lectura. No tener permiso de escritura el mapeo falla, y se provoca un SIGSEGV como anteriormente. Sin embargo, al tener el archivo tamaño cero, en lugar del tamaño necesario para mapear la zona indicada, los accesos de lectura no producen in SIGSEGV sino un SIGBUS. De esta forma el manejador de SIGBUS identifica accesos de lectura y el SEGSEGV accesos de escritura. En lectura se realiza el acceso dentro del manejador, poniendo el valor a leer en el archivo, mientras que en la escritura se realiza la obtención del dato y el acceso al bus dentro del código inyectado, ya que el dato no está disponible hasta que se realiza el acceso correcto.

El procedimiento completo puede verse en la figura 7.6.

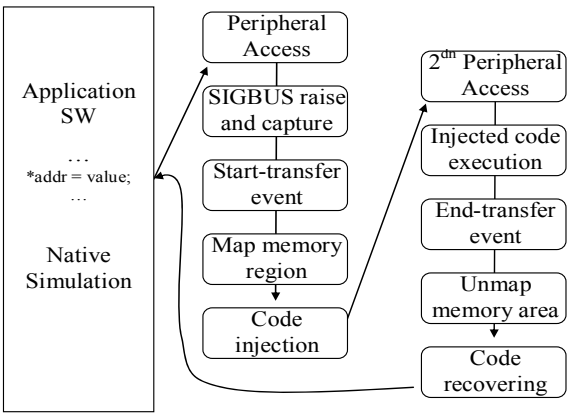


Figura 7.6: Process for complete handling of HW accesses directly using pointers

Los códigos requeridos para cada etapa, así como otros detalles adicionales pueden encontrarse en el artículo “Automatic HW/SW interface modeling for scratch-pad & memory mapped HW components in native source-code co-simulation”, presentado en la conferencia IESS’09.

7.1.3.4.2 Funciones de sistema operativo para acceso HW

De la misma forma que se ha utilizado el estándar POSIX para implementar una interfaz de sistema operativo conocida y fácil de usar por las aplicaciones, para la implementación de las funciones de manejo de controladores de dispositivo se han desarrollado algunas de las funciones más importantes de Linux. En el estándar POSIX no se definen directamente funciones que permitan al SW de aplicación acceder a los periféricos del sistema, por lo que se ha utilizado Linux como estándar “de facto” en su lugar.

La solución adoptada en los sistemas operativos de este tipo para el manejo de los controladores de dispositivos es mapear los controladores de dispositivo como nodos del sistema de ficheros. Para ello, en el modelo de sistema operativo se han implementado las funcionalidades necesarias para permitir tanto que los controladores sean instalados adecuadamente en el modelo de sistema como que el SW pueda acceder a los controladores mediante los accesos adecuados en el sistema de archivos.

Funciones de instalación de controladores

Para la carga del controlador en el modelo de sistema operativo es necesario registrar y arrancar dicho controlador. Estas operaciones implican la aparición de un nodo correspondiente al dispositivo en el directorio "/dev/" del modelo de sistema operativo. Además, el registro del controlador carga las funciones encargadas de responder a peticiones de apertura, cierre, lectura, escritura, espera y control desde el código de aplicación. Fundamentalmente, estas operaciones se basan en las siguientes funciones implementadas:

```
void module_init(module);
void module_exit(module);
int register_chrdev(unsigned int major, const char *name, struct
file_operations *fops);
int unregister_chrdev(unsigned int major, const char *name);
```

Desde el punto de vista del funcionamiento interno del controlador se requiere disponer de listas, bloqueos y sincronizaciones. En determinados puntos de la ejecución el controlador ha de esperar una respuesta, debe esperar a que otro elemento libere el recurso, o recibe un evento indicando que ha de despertarse para realizar alguna operación. Para gestionar estos casos el sistema operativo y el controlador han de tener los elementos de interconexión y sincronización adecuados. En el modelo se han implementado a tal efecto varias funciones como las siguientes:

```
Funciones para gestión de waiy_queue;

void wake_up();
void schedule_timeout(long);
Funciones para gestión de spin_lock del núcleo
struct device *dev_get_by_name(const char *name);
```

Funciones de acceso al bus

Para permitir el acceso del SW a los registros de los periféricos conectados al bus se han desarrollado las funciones necesarias siguiendo el modelo Linux. El acceso a los periféricos se realiza seleccionando primero una región de memoria. Posteriormente es posible leer o escribir en ella según sea necesario.

Las funciones desarrolladas para seleccionar la zona de memoria a utilizar son las siguientes:

```
int check_region(unsigned long from, unsigned long extent);
struct resource *request_region(unsigned long from, unsigned long
extent, const char *name);
void release_region(unsigned long from, unsigned long extent);
int check_mem_region (unsigned long start, unsigned long extent);
struct resource *request_mem_region (unsigned long start, unsigned
long extent, const char *name);
void release_mem_region (unsigned long start, unsigned long extent);
```

Una vez reservada la zona de memoria, las funciones para leer o escribir son las siguientes:

```
char uc_ioread8(void *addr);
void uc_iowrite8(unsigned char value, void *addr);
long uc_ioread32(void *addr);
void uc_iowrite32(unsigned long value, void *addr);
int uc_ioreadBurst(void **value,int size, void *addr);
int uc_iowriteBurst(void *value, int size, void *addr);
```

Estas funciones se conectan automáticamente al componente encargado de conectar el procesador desde donde se estén ejecutando estas funciones con el modelo de bus correspondiente.

7.1.3.4.3 Manejo de interrupciones HW

Con todo lo dicho, las comunicaciones HW/SW se pueden implementar como relaciones maestro-esclavo. Sin embargo, en determinadas ocasiones es el HW periférico el que informa al procesador de la aparición de un nuevo evento. Cuando un periférico recibe una comunicación del exterior o finaliza una operación solicitada por el SW, este debe informar al procesador de que eso ha ocurrido. Una vez recibida la información el SW debe responder a la petición realizando las operaciones necesarias para que el evento sea adecuadamente atendido.

Para que este proceso pueda llevarse a cabo el modelo de sistema operativo debe proporcionar dos tipos de servicios. Por una parte el modelo ha de permitir a los controladores que indiquen qué manejadores de interrupción han de ejecutarse y puedan bloquear y desbloquear las interrupciones. Por otro lado es necesario que el sistema operativo proporcione los servicios necesarios para que las peticiones de interrupciones recibidas desde el modelo de bus produzcan la ejecución de dichos manejadores.

Para la carga de interrupciones el modelo incorpora las siguientes funciones:

```
int request_irq(unsigned int irq, irqreturn_t (*)(int, void *, struct
pt_regs *), unsigned long irqflags, const char * devname, void
*dev_id);

void free_irq(unsigned int irq, void *dev_id);
```

Para que las interrupciones generadas por los periféricos lancen la ejecución del manejador correspondiente el modelo proporciona la función "send_irq" que es accedida desde la interfaz procesador-bus.

Una vez esta función es ejecutada el sistema operativo decide si ha de lanzarse el manejador, en función de los manejadores cargados y las máscaras de ejecución. Para ejecutar el manejador el modelo arranca un hilo POSIX de muy alta prioridad, mayor que la de cualquier tarea que pueda ser lanzada desde la aplicación. Las propias tareas de manejadores mantienen prioridades entre ellas, de forma que unas interrupciones tienen prioridad sobre otras.

7.1.4 Integración en SystemC

Para integrar la ejecución del código SW en una simulación SystemC, el modelo de sistema operativo debe utilizar ciertos elementos de SystemC. Estos elementos se deben encargar fundamentalmente de:

- Asignar un flujo de ejecución los distintos procesos e hilos.
- Bloquear y desbloquear los flujos de ejecución
- Comunicar el SW con el resto del sistema

7.1.4.1 Flujos de ejecución en SystemC

El núcleo de ejecución de SystemC se basa en un modelo similar al empleado en VHDL y otros simuladores orientados a descripciones RTL. Al igual que en estos lenguajes la funcionalidad a ejecutar se asocial a procesos SystemC. Estos procesos se ejecutan de forma completamente concurrente. Además las instrucciones del proceso se ejecutan secuencialmente sin que se produzca ningún avance en el tiempo de simulación mientras no se encuentre un elemento de SystemC que bloquee la tarea.

Existen varios tipos de procesos en SystemC de entre los que se ha seleccinado el tipo SC_THREAD para proporcionar la concurrencia al modelo de sistema. Esta elección se ha debido a que son los que tienen mayor parecido con los hilos SW que han de modelar. Estos tipos de procesos son iniciados una única vez al comienzo de la simulación. A partir de aquí se continuarán ejecutado hasta que se salga de la función principal o se produzca un bloqueo del que no se despierte nunca. Los SC_THREAD permiten el uso dentro código de funciones que bloquean la ejecución del proceso. Estas funciones de bloqueo se denominan "wait". Dichas funciones pueden bloquearse durante un tiempo y/o hasta que llegue un evento. Estos eventos se envían mediante funciones "notify" desde otros SC_THREADs.

Por definición, los procesos SystemC solo pueden ser declarados en módulos SystemC. Un módulo es una clase C++ que hereda una serie de métodos necesarios para el control de SystemC. Por ello, cuando un usuario quiere dotar de flujo de ejecución a una tarea SW debe generar un módulo y un SC_THREAD correspondiente. El SC_THREAD recibe como argumento la función a ejecutar. Esto se puede ver en la figura siguiente:

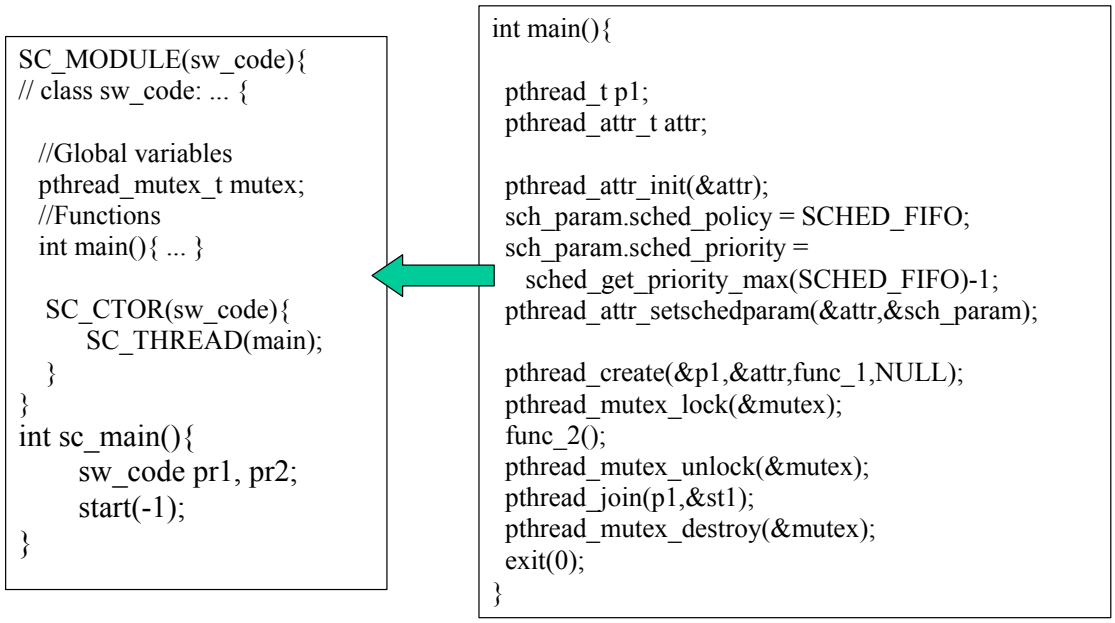


Figura 7-7: Carga de una tarea en SystemC.

La librería permite la integración automática de las tareas SW en la simulación SystemC. A través de la función miembro "create_process" de la clase "rtos", se puede proporcionar el puntero a la función principal de la tarea SW, dejando que la librería se encargue de la creación del módulo y del SC_THREAD.

Para modelar el sistema, antes de la función "sc_start", que comienza la simulación se deben declarar todos los procesos que tendrá el sistema. Estos procesos se llamarán procesos estáticos, y representan las aplicaciones que contiene el sistema. El sistema de modelado no dispone de una Shell, con lo que no es posible lanzar nuevas aplicaciones una vez que ha empezado la simulación.

Para simular los procesos estáticos, se usa una función que actúa de envoltorio, y extiende la funcionalidad de los SC_THREAD. Los procesos SystemC arrancan automáticamente al comienzo de la simulación. Esto significa que todas las aplicaciones se disponen a ejecutan a la vez, aunque no haya procesadores suficientes en el modelo de plataforma para ello.

Para evitar este problema y hacer que el proceso pase a estar bajo el control del planificador del sistema operativo se utiliza esta nueva función. La función es ejecutada por el SC_THREAD en lugar de la función proporcionada por el usuario. Esta función comienza cambiando el estado del proceso a "Create" y "Ready", con lo que fuerza a que la tarea espere a ser planificada. Una vez planificada comienza la función SW indicada por el usuario. Una vez el proceso finalice se han de realizar también todas las tareas asociadas a su destrucción. La función encargada de ejecutar los procesos SW se puede ver en la siguiente figura:

```
void sc_process_static_function(){
        change_state_Crated_to_Ready();
        //Waits for the scheduler
        change_state_SuperUser_to_Waiting();
        //Waits for the scheduler
        return_value=main_process_function();  //SW component main function
        exit(return_value);
}
```

Figura 7-8: Mecanismo de ejecución de un proceso de arranque estático.

Para la creación de hilos dinámicos mediante "pthread_create", se han de generar nuevos SC_THREADs. Sin embargo, dependiendo de las versiones de SystemC, la generación de estos threads no es siempre posible. Para evitar este problema se ha implementado una "piscina de SC_THREADs". Esta piscina contiene un número de SC_THREADS que han sido creados automáticamente al principio de la simulación por el modelo de sistema operativo y están bloqueados antes de pasar a estado "Create". Los procesos no tienen inicialmente una función SW asociada.

Cuando se crea un hilo dinámico se asocia la función y se despierta el proceso. Al finalizar el hilo dinámico se liberan los recursos asociados y se coloca al SC_THREAD en posición de poder comenzar la ejecución de un nuevo hilo dinámico. Hay que tener en cuenta que un hilo no solo finaliza al acabar la función asociada. Funciones como "exit" o "pthread_cancel" pueden hacer que la ejecución se finalice en mitad de la función. Para que el hilo finalice y el SC_THREAD sea reutilizable, se utiliza un sistema de excepciones. Cuando un hilo es finalizado abruptamente se lanza una excepción que es capturada en la función del SC_THREAD. Esto puede verse en la figura:

```
void sc_thread_dynamic_function(){
        while(true){
                wait(start_event);
                change_state_Crated_to_Ready();
                //Waits for the scheduler
                change_state_SuperUser_to_Waiting();
                //Waits for the scheduler
                try{
                        return_value=dynamic_task_function(args);
                        exit(return_value);
                }catch(End_thread_exception exc){;}
        }
}
```

Figura 7-9: Código de la ejecución de una tarea de creación dinámica

En resumen, la concurrencia de las distintas tareas en la simulación sobre SystemC se basa en la idea de procesadores virtuales. Cada tarea tiene asociado un hilo de SystemC ("SC_THREAD") que se encarga de proporcionar un flujo de ejecución, gestionar cambios de contexto, En SystemC, como heredero de los lenguajes HW, los hilos están diseñados para ejecutar de manera paralela en la simulación. Como resultado, este método realiza una simulación en la que todos los hilos SW se ejecutan en paralelo. Así pues se puede considerar que hay tantos procesadores en la plataforma como hilos declarados.

La labor de SCoPE es hacer que la simulación modele el funcionamiento de las plataformas reales, considerando que el número de procesadores sea menor que el de tareas. Para ello se definirá, en primer lugar, en qué procesador puede ejecutar cada tarea, y posteriormente el planificador comprobará dichas definiciones a la hora de cambiar los estados de las tareas y asignar procesadores.

7.1.4.2 Bloqueo y continuación de tareas

El planificador del modelo de sistema operativo se encarga de organizar la ejecución de las tareas SW del sistema. Para que esto se cumpla en la simulación SystemC el modelo se encarga de dejar bloqueadas todas las tareas SW que no estén planificadas en ese momento. De esta forma el núcleo de ejecución de SystemC solo arrancará las tareas adecuadas. Para realizar esto se utilizan las funciones de SystemC "wait" y "notify".

El uso de "wait" y "notify" depende del estado en el que se encuentre la tarea en el sistema operativo:

-Create: Cuando una tarea es creada, el hilo de SystemC empieza a ejecutar automáticamente. En el modelo de sistema operativo la tarea pasa automáticamente a Ready una vez inicializados los recursos necesarios, y sin que se produzca un avance de tiempo en la simulación.

-Ready: Cuando una tarea pasa a ser ejecutable se pone en modo “Ready” y se bloquea en un “wait”, esperando a que el planificador la despierte.

- Waiting: De la misma forma que en el caso anterior, una vez una tarea entra en este estado se bloquea en un “wait” y espera a ser planificada.

- Bloqued: Cuando una tarea está bloqueada en un canal o por una espera temporal, la tarea está bloqueada por un “wait” de SystemC. El modo de salir del bloqueo depende del elemento en el que la tarea esté bloqueada. En canales, se requerirá un “notify” para despertar a la tarea. En esperas de tiempo se despertará automáticamente en el momento adecuado.

- User: En este estado la tarea está en ejecución y por tanto no está bloqueada.

- Super-User: De la misma forma que en “User” la tarea no está bloqueada. En el paso de “User” a “SuperUser” se realiza la anotación temporal explicada en el capítulo anterior. Para ello se utiliza un nuevo “wait” de SystemC. Además, cuando una tarea sale de estado “SuperUser” se ejecuta el planificador de forma que despierta a la siguiente tarea a ejecutar mediante un “notify”.

Para que el mecanismo del planificador funcione correctamente es necesario un uso adecuado de ciclos delta durante de la simulación. Los ciclos delta de SystemC permiten la ejecución de múltiples ciclos de simulación en el mismo punto temporal. Su iso es necesario para garantizar que cada vez que el planificador es ejecutado todos los hilos que modelan las tareas SW estén en el estado correcto. Para ello se utilizan tres ciclos delta. En el primer ciclo se realizan las operaciones en modo kernel que llevarán al abandono del procesador por parte del proceso en ejecución, ya sea para pasar al estado bloqueado, “Waiting” o “Zombie”. Estas operaciones pueden modificar el estado de otros hilos. Por ejemplo la ejecución de una llamada a “sem_post()” modificaría el estado de un hilo esperando en un “sem_wait()” de “Blocked” a “Ready”.

En el caso de que otro hilo pueda ser afectado por las operaciones realizadas, el hilo en ejecución despierta al hilo afectado para que este pueda realizar las acciones y comprobaciones adecuadas para realizar el cambio de estado. Para despertar al hilo en SystemC se usa la función “notify()”. Dado que el primer hilo llama a la primitiva “notify()” el cambio de estado no puede ser garantizado hasta el segundo ciclo delta. Esto significa que hasta el final del segundo ciclo delta no se puede garantizar que todos los hilos estén en el estado correcto. Como consecuencia, el planificador no puede ser ejecutado hasta el tercer ciclo delta. En general, para que el mecanismo del planificador funcione correctamente es necesario un uso adecuado todos los mecanismos de simulación de SystemC, como los ciclos delta, los eventos o las esperas temporales. Esto será analizado con más detalle posteriormente (sección 7.1.4).

7.1.4.3 Conexión con la plataforma HW: bus-SW

Para comunicar el SW con la plataforma HW se utilizan las funciones de envío de información a través del bus explicadas en la sección de HAL. Para que estas funciones puedan acceder adecuadamente a los modelos de bus y enviar esta información se ha desarrollado un componente encargado de reemplazar el procesador. Este componente también se encarga de recibir las interrupciones del bus y lanzar las funciones del modelo de sistema operativo encargadas de su gestión.

El componente de interconexión se basa en una interfaz de enlace con el bus. Esta interfaz contiene los puertos necesarios para conectarse con el modelo de bus que se presentará en la sección siguiente. La interfaz está formada por dos puertos. El primer puerto ("sc_port") requiere del bus las funciones para enviar y recibir datos. El segundo puerto ("sc_export") proporciona una función para la recepción de interrupciones.

El componente internamente no contiene funcionalidad propia, y se limita a conectar las funciones del sistema operativo y de la interfaz con el bus. Una vez el SW llama a una función de envío por el bus, se revisa en qué procesador está ejecutando la tarea llamante y se realiza la petición al componente adecuado. De la misma forma, cuando se recibe una interrupción, el componente realiza una llamada al sistema operativo adecuado, indicando la interrupción recibida y el identificador del procesador donde se ha recibido.

7.1.5 Soporte para sistemas multiprocesador

Los sistemas multiprocesador en chip (MpSoC) son sistemas con múltiples procesadores integrados en el mismo chip que trabajan cooperativamente para resolver un problema. El modo en que estos procesadores se interrelacionan es crítico en el rendimiento y capacidad del sistema. Un factor fundamental en este ámbito es el grado de acoplamiento entre los procesadores. Habitualmente se consideran dos grados de acoplamiento: sistemas fuertemente acoplados (tightly-coupled) y sistemas débilmente acoplados (loosely-coupled). Dado que SCoPE tiene como objetivo proporcionar la mayor flexibilidad posible al diseñador, ambos tipos han sido considerados en su desarrollo.

Los sistemas fuertemente acoplados contienen múltiples procesadores conectados directamente al mismo canal de comunicación, normalmente un bus. Estos procesadores pueden acceder directamente a los mismos periféricos y una memoria central compartida, de forma que todos se comportan de forma simétrica (SMP). También pueden tener una jerarquía de memoria que combine memorias locales y globales, de forma que el acceso no es completamente uniforme (NUMA). Los sistemas débilmente acoplados se basan en múltiples nodos computadores que funcionan de forma autónoma y se conectan entre si para realizar transferencias de información mediante una red de comunicaciones.

El modelo propuesto en SCoPE tiene como objetivo permitir el modelado tanto de sistemas SMP como de sistemas débilmente acoplados. El modelado de sistemas SMP requiere que el sistema operativo sea capaz de gestionar la ejecución de tareas sobre varios

procesadores simultáneamente. El modelado de sistemas débilmente acoplados no añade requerimientos al modelo de sistema operativo, aparte de requerir métodos de comunicación entre nodos distintos.

Los sistemas SMP permiten a cualquier procesador trabajar en cualquier tarea, sin importar los recursos que esta necesita, dada la simetría en los accesos. Por esta razón, los sistemas SMP permiten mover tareas entre procesadores automáticamente para balancear la carga de trabajo de la manera más eficiente posible.

El modelo de sistema operativo desarrollado ha sido diseñado convenientemente para realizar esta gestión. Se han incluido en el modelo servicios para controlar la asignación de tareas a procesadores y comunicar y sincronizar estas tareas adecuadamente. El planificador realiza la asignación de tareas teniendo en cuenta los procesadores disponibles. La gestión de interrupciones se encarga de entregar cada interrupción al procesador correspondiente. Además, tener varias tareas accediendo simultáneamente a los recursos del sistema operativo hace que este deba ser seguro, para evitar conflictos en los accesos.

Para permitir el control de la ejecución de las tareas en los distintos procesadores se han implementado las siguientes funciones:

- int sched_setaffinity(pid_t pid, unsigned int len, unsigned long *mask);
- int sched_getaffinity(pid_t pid, unsigned int len, unsigned long *mask);

Estas funciones permiten asignar una máscara a cada proceso, de forma que se define en que procesadores puede ser ejecutado el proceso y en cuales no. De esta forma, los threads del proceso pueden ser ejecutados simultáneamente si la mascará permite la ejecución en más de un procesador.

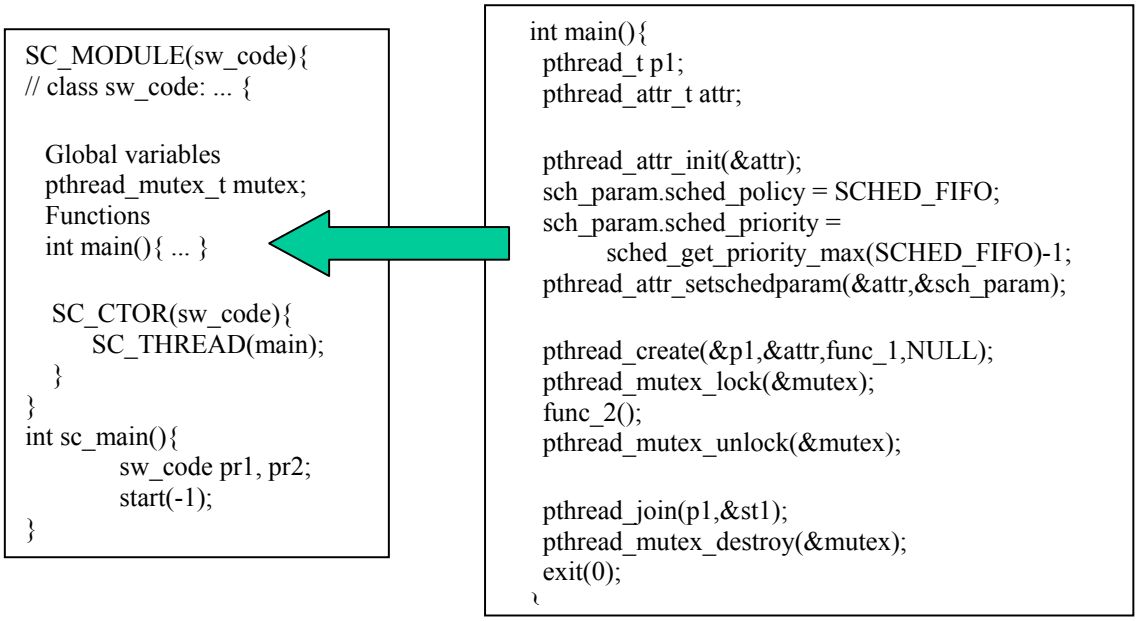
Para permitir la sincronización entre tareas de distintos procesadores se han implementado los bloqueos de espera activa (spinlocks). Estos canales son especialmente útiles para proteger el acceso de variables globales sin forzar cambios de contexto, cuando el tiempo de espera previsto es suficientemente bajo.

7.1.6 Ejemplo de aplicación POSIX

Para facilitar la comprensión del uso de la librería presentada se muestra a continuación un ejemplo simple. El código del ejemplo no tiene ninguna interpretación especial más allá de la muestra del uso de funciones POSIX descritas anteriormente.

Como se puede ver en la figura 7-10, el código comienza definiendo el recurso en el será ejecutado inicialmente el módulo. Para ello se asocia a RESOURCE el valor PROCESSOR_1. De esta forma sabemos que el módulo siguiente será software. Una vez se ha hecho esto, se puede comenzar con la definición del modulo. En primer lugar se define un puerto que tiene acceso a una fifo mediante la cual se podrá realizar la comunicación con otros módulos descritos en SystemC, especialmente con módulos hardware.

Seguidamente se declara un mutex que se usará para la sincronización de los threads que aparecen en el ejemplo. A continuación se declaran las funciones "func_1" e "init". La primera será la función que ejecutará el thread auxiliar que generaremos en el ejemplo y la segunda, la función principal del proceso que simularemos en el ejemplo. El código de la función "func_1" no es necesario para explicar el ejemplo y será descrito en otro lugar del código.



```
SC_MODULE(sw_code){
// class sw_code: ... {

   Global variables
   pthread_mutex_t mutex;
   Functions
   int main(){ ... }

    SC_CTOR(sw_code){
        SC_THREAD(main);
    }
}
int sc_main(){
            sw_code pr1, pr2;
            start(-1);
}
```

```
int main(){
  pthread_t p1;
  pthread_attr_t attr;

  pthread_attr_init(&attr);
  sch_param.sched_policy = SCHED_FIFO;
  sch_param.sched_priority =
        sched_get_priority_max(SCHED_FIFO)-1;
  pthread_attr_setschedparam(&attr,&sch_param);

  pthread_create(&p1,&attr,func_1,NULL);
  pthread_mutex_lock(&mutex);
  func_2();
  pthread_mutex_unlock(&mutex);

  pthread_join(p1,&st1);
  pthread_mutex_destroy(&mutex);
  exit(0);
}
```

Figura 7-10: Ejemplo de uso de la interfaz POSIX en SystemC

Por último, se coloca el constructor del módulo, en el cual se realiza la creación de un SC_THREAD, que servirá como procesador virtual para el proceso que vamos a simular. En la declaración se define que el proceso ejecutará la función "init".

La ejecución del proceso comienza declarando las estructuras que serán necesarias posteriormente. A continuación comienza redefiniendo los procesadores donde podrá ejecutar el proceso. En este caso se asigna el valor de "mask", que ha sido inicializado para permitir la ejecución en los procesadores 1 y 2.

A continuación se realiza la creación del hilo que correrá en paralelo con el proceso inicial. Para ello se inicializan los atributos del hilo y se modifican la política de planificación y la prioridad. Finalmente se crea el hilo que ejecutará la función "func_1" con los nuevos atributos.

Después se toma el "mutex", se escribe un valor en la "fifo" y se libera el "mutex". Para que esto funcione la "fifo" que esta conectada al puerto de este módulo ha de estar modificada conforme a lo mostrado en la figura 2.

Por último el proceso espera a que acabe el hilo que creo anteriormente y destruye el "mutex".

Como se puede ver el código simulable es muy similar al código final que se podrá introducir en el sistema final. Tan solo será necesario sustituir el acceso al módulo hardware a través de la "fifo". De esta forma se puede refinar en profundidad el diseño mediante la simulación conjunta de los componentes software y hardware del sistema.

7.2 Paquetes integrados en el modelo de sistema operativo

7.2.1 Integración de una pila TCP/IP

Para permitir el uso de las funciones de comunicación por red incluidas en el estándar POSIX se ha integrado en el modelo de sistema operativo una pila TCP/IP. Esta pila puede ser utilizada para comunicar nodos independientes en sistemas débilmente acoplados, o para modelar comunicaciones con el exterior del sistema simulado.

Para la integración se ha escogido el paquete lwIP http://savannah.nongnu.org/projects/lwip/). Esta pila es una implementación reducida e independiente de sistema operativo del protocolo TCP/IP. Ha sido desarrollado por el Swedish Institute of Computer Science (SICS).

El objetivo de la pila es proporcionar soporte TCP utilizando unos requerimientos de recursos mínimos. El lwIP está por ello especialmente indicado para sistemas embebidos, con decenas de kB de RAM y entorno a 40KB de ROM.

La pila lwIP proporcional fundamentalmente los siguientes servicios:

- IP (Internet Protocol)
- ICMP (Internet Control Message Protocol)
- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)
- DHCP (Dynamic Host Configuration Protocol)
- PPP (Point-to-Point Protocol)
- ARP (Address Resolution Protocol)

7.2.1.1 Requerimientos para integrar el lwIP

La pila lwIP se distribuye con una adaptación a Linux, con lo que es adecuada para ser integrada en este modelo de sistema operativo. Para su funcionamiento la pila necesita las siguientes funciones POSIX:

- ‘exit'
- ‘free'
- ‘gettimeofday'
- ‘ioctl'
- ‘malloc'
- ‘memcpy'
- ‘memset'
- ‘open'
- ‘perror'
- ‘printf'
- ‘pthread_cond_broadcast
- ‘pthread_cond_destroy'
- ‘pthread_cond_init'
- ‘pthread_cond_timedwait
- ‘pthread_cond_wait'
- ‘pthread_create'
- ‘pthread_equal'
- ‘pthread_mutex_destroy'
- ‘pthread_mutex_init'
- ‘pthread_mutex_lock'
- ‘pthread_mutex_unlock'
- ‘pthread_self'
- ‘read'
- ‘select'
- ‘snprintf'
- ‘system'
- ‘write’

Todas estas funciones están soportadas por la API POSIX integrada en el modelo de sistema operativo. Únicamente la función “system” no se encuentra disponible. Esta función se usa en la pila para obtener la dirección IP del dispositivo. Para resolver este problema se ha realizado una pequeña modificación en la pila.

7.2.1.2 Modificaciones al lwIP

Aunque la pila está preparada para se integrada en un sistema Linux, hay algunas peculiaridades del modelo que han requerido realizar modificaciones en la propia pila.

En primer lugar, la pila está diseñada para mantener una única copia activa en un sistema. El código se basa en el uso de variables globales y funciones que no permiten la integración de más de una pila. Sin embargo, la infraestructura de simulación que se ha desarrollado contempla la posibilidad de tener varios sistemas operativos conjuntamente. En los sistemas débilmente acoplados, cada nodo tiene su propio sistema operativo. En consecuencia, se necesita integrar una pila TCP por cada nodo.

Para resolver el problema se han modificado todos los accesos a variables globales, de forma que las variables son accedidas dependiendo del sistema operativo desde el que se acceda.

En segundo lugar, la adaptación de la pila a Linux no está preparada para soportar llamadas al propio nodo. Para resolver el problema las funciones de envío y recepción de bajo nivel han sido modificadas.

Además, la adaptación de la pila no obtiene adecuadamente las direcciones MAC ni IP del sistema. Para resolver el problema se ha modificado la secuencia de inicialización, de forma que obtiene estos parámetros de la forma correcta.

7.2.1.3 Modelo de controlador de red

Para que la pila pueda acceder al dispositivo HW de red se ha implementado un controlador de red que se ha integrado en el modelo de sistema operativo. Este controlador contiene las funciones para la inicialización y cierre y proporciona servicios para la apertura, cierre, lectura y escritura del dispositivo. Se ha implementado la función "ioctl" para permitir la configuración del controlador y del dispositivo. También se ha implementado la función "poll" para permitir el acceso mediante la función POSIX "select". Por último se ha integrado un manejador de interrupción que se encarga de gestionar las peticiones de atención de la tarjeta de red.

7.2.1.4 Inicialización de la pila lwIP

La inicialización de la pilas en cada nodo se realiza dentro de la secuencia de arranque del nodo. En dicha secuencia se inicializan las variables, se arrancan los hilos encargados de la gestión de los paquetes TCP y se asignan los valores MAC e IP.

La dirección IP asignada a cada pila depende del número de identificación del nodo correspondiente. Sigue la siguiente estructura:

IP Address: 192.168.0.node_id

La dirección MAC es generada durante la instanciación de los componentes HW. Se carga en la pila durante la inicialización y se asigna a la tarjeta de red mediante la función "ioctl". La dirección MAC se genera siguiendo el patrón:

MAC Address: 01:02:03:04:05:netcard_id

Dado que la estructura de la red de comunicaciones depende del sistema a modelar, la tabla ARP no está inicializada por defecto. De esta forma, antes de enviar cualquier paquete la pila realiza las peticiones necesarias para conocer las direcciones MAC del resto de componentes del sistema.

7.2.2 Extensión para Linux de tiempo real

7.2.2.1 Introducción

Como se presento en los capítulos 3 y 4, los sistemas Linux, convencionales son RTOS validos para sistemas de tiempo real relajado. Sin embargo, no proporcionan los servicios necesarios para gestionar sistemas de tiempo real estricto garantizando el cumplimiento de todos los plazos temporales. Como consecuencia, en la literatura se han propuesto múltiples extensiones de tiempo real para completar los sistemas Linux para su uso en estos sistemas de tiempo real estricto.

Considerando que los sistemas de tiempo real estricto conforman un importante grupo dentro de los sistemas embebidos, se ha considerado necesario extender el modelo de sistema operativo POSIX/Linux con una de estas extensiones. De esta forma SCoPE puede aplicarse para la co-simulación de sistemas de tiempo real estricto.

Para ello, aprovechando las posibilidades de colaboración que proporcionan los proyectos europeos, se ha realizado una extensión de SCoPE que modela el sistema operativo de tiempo real diseñado bajo dirección de Mandrakesoft, Intel y Thales en el proyecto Hyades.

7.2.2.2 Elementos de Hyades a modelar en SystemC

Aunque algún caso de los considerados en el desarrollo de Hyades ya ha sido integrado en SCoPE al integrar gran parte del API posix y algunas extensiones del núcleo Linux 2.6, como la ejecución del planificador en los accesos a los “spinlocks“, el resto de características no están cubiertas.

Para cubrir dichas modificaciones hay que considerar dos partes distintas: la integración de las funciones de interfaz en la API del modelo SCoPE y el modelado de las modificaciones internas en el propio kernel. Dado que el SCoPE no contiene un linux original, sino un modelo, la integración de las modificaciones de Hyades no puede hacerse utilizando la propia implementación, sino traduciendo las ideas subyacentes. Esto requerirá un esfuerzo mayor, y aumenta el riesgo de que el modelo no se ajuste correctamente a todos los requerimientos.

7.2.2.2.1 DIC API

La API proporcionada por el DIC para acceder a la funcionalidad de Hyades se compone de las siguientes funciones:

Creación y arranque de hilos:

```
pthread_init_rt
pthread_create_rt
pthread_start_rt
pthread_sync_rt
pthread_barrier_rt
```

Gestión de hilos en el DIC:

```
pthread_migrate_rt
pthread_inquire_rt
```

Gestión de hilos periodicos:

```
pthread_set_periodic_rt
pthread_wait_period_rt
```

Semáforos:

```
pthread_create_sem_rt
pthread_delete_sem_rt
pthread_wait_sem_rt
pthrad_signal_sem_rt
```

Temporizadores y gestión de tiempo:

```
pthread_start_timer_rt
pthread_stop_rt
pthread_time_r
pthread_time_rt
pthread_ns2ticks_rt
pthread_ticks2ns_rt
pthread_sleep_rt
```

Estas funciones están basadas en funciones similares de la API POSIX original, pero enlazadas con diversas partes del DIC. En principio no son muchas funciones y pueden ser implementadas en SCoPE. No obstante, algunas de ellas, especialmente las de creación y arranque de hilos no son fácilmente comprensibles a partir de la documentación aportada, y su adecuado modelado puede ser problemático.

7.2.2.2.2 Interfaz del núcleo de Hyades

Estas funciones no son parte de la API, con lo que en teoría no son accesibles desde el código de usuario. Sin embargo, pueden ser necesarias para la generación de controladores y módulos de kernel. Por esta razón han de ser implementadas, al menos en parte. El resto, en vez de ser implementado estrictamente conforme a la interfaz propuesta, han sido modeladas de la forma mas adecuada posible para adaptarse al OS existente en SCoPE.

Las funciones de la interfaz del núcleo se agrupan de la siguiente forma:

- Dynamic Memory Allocation
- Thread Synchronization
- HAL
- Timers
- Interrupt Management
- Real-time pod
- Real-time shadow

De todas estas, se han implementado directamente las funcionalidades de manejo de interrupciones y HAL. El resto son necesarias para la implementación interna de la DIC API y

no son accesibles desde código de usuario. Por esta razón, en su lugar de implementarlas directamente, se realizará el modelado necesario para integrar la nueva funcionalidad en el modelo de RTOS POSIX/Linux existente.

7.2.2.3 Modelado de las extensiones de tiempo real

En la implementación Hyades, dos sistemas operativos, un Linux convencional y una extensión de tiempo real son integrados sobre la base de la infraestructura "Adeos". Ambos sistemas operativos son independientes, con distintas zonas de memoria, infraestructuras de gestión de procesos, comunicaciones, etc. Por esta razón el sistema Hyades tiene una duplicidad en la gestión de los procesos, manteniendo una copia en cada uno de los dos sistemas.

En el modelo realizado en SCoPE, esta separación no es tal. Por un lado, los modelos de sistema operativo son más sencillos y han sido diseñados para ejecutar de forma compatible. En segundo lugar, es posible controlar la ejecución de las tareas sobre el núcleo de simulación de SystemC de formas que no es posible hacer sobre el procesador real. Por último, la simulación SystemC no permite esa separación de espacios de memoria, al estar integrada en un único proceso en el ordenador nativo donde se realiza la simulación. En consecuencia, es posible modelar el comportamiento del sistema Hyades sin necesidad de implementar un modelo tan complejo como la implementación real.

El funcionamiento del modelo ha sido diseñado para emular el comportamiento del sistema Hyades en tres puntos básicos: La gestión de los dos entornos de ejecución, la gestión de interrupciones y la implementación de servicios de tiempo real adicionales.

7.2.2.3.1 Modelado de los dominios del sistema operativo

La implementación de dos sistemas, uno convencional y uno de tiempo real, en dos dominios distintos en el modelo Hyades permite gestionar de forma independiente las tareas convencionales y las tareas de tiempo real. Esta gestión implica principalmente la preferencia en las prioridades de las tareas y las interacciones e intromisiones que pueden sufrir de otras tareas y del resto del sistema.

Esto implica controlar tres elementos básicos: los algoritmos de planificación, la acción de las interrupciones y los tiempos de sobrecarga y respuesta del sistema operativo. La gestión de interrupciones se ha resuelto con un sistema de gestión adicional, descrito en la siguiente sección.

Los tiempos de respuesta del sistema operativo convencional, tanto para llamadas al sistema como para operaciones internas, no son considerados un elemento crítico. Aunque se desea que sean lo más reducidos posibles, en muchas ocasiones se prefiere una funcionalidad más completa que una respuesta un poco más rápida. Sin embargo, en sistemas de tiempo real estricto, el tiempo es crítico, y su reducción es un elemento básico. La consideración de los tiempos de sobrecarga del sistema operativo en SCoPE se limita a modificar el valor asociado

a cada función del sistema. Por tanto, manteniendo dos listas de tiempo, una para cada dominio, se resuelve el modelado de esta parte.

Por tanto, la parte fundamental del modelado de los dominios se centra en el modelado de los sistemas de planificación. En el sistema real, el flujo de las tareas SW es el siguiente: Toda tarea comienza en el entorno Linux convencional. Cuando necesita tener cualidades de tiempo real, la tarea migra, abandonando ese dominio y pasando a ser controlada por el dominio de tiempo real. Cuando acaba la sección de tiempo real la tarea retorna al dominio convencional. Para realizar el cambio de dominio es necesario llamar a la función: "pthread_migrate_rt (domain)".

Las tareas del dominio de tiempo real son siempre planificadas antes que las convencionales. De esta forma mientras haya tareas de tiempo real listas para ejecutar las otras permanecen bloqueadas. Para implementar esto, se ha añadido una segunda lista de tareas listas para ejecutar. La nueva lista contiene las tareas en el dominio de tiempo real. Cuando una tarea cambia de dominio, se cambia la lista en la que se almacena dicha tarea. De esta forma, cuando es necesario seleccionar una nueva tarea para ejecutar, se busca en la lista de tiempo real, y si no hay ninguna tarea disponible se busca en la segunda lista. De esta forma se garantiza el orden de ejecución de las tareas, dando máxima prioridad a las tareas de tiempo real.

```
…
if( (next_task = RT_scheduler()) == NULL){ //New code
        next_task = nonRT_scheduler();
}// New code
resume ( next_task );
…
```

Figure 7-12: Código introducido para utilizar ambos planificadores

Para controlar el dominio en el que se encuentras las tareas se han añadido tres estados más al grafo de estados de cada tarea. Estos estados indican cuando la tarea está ejecutando, bloqueada o lista para ejecutar en el dominio de tiempo real.

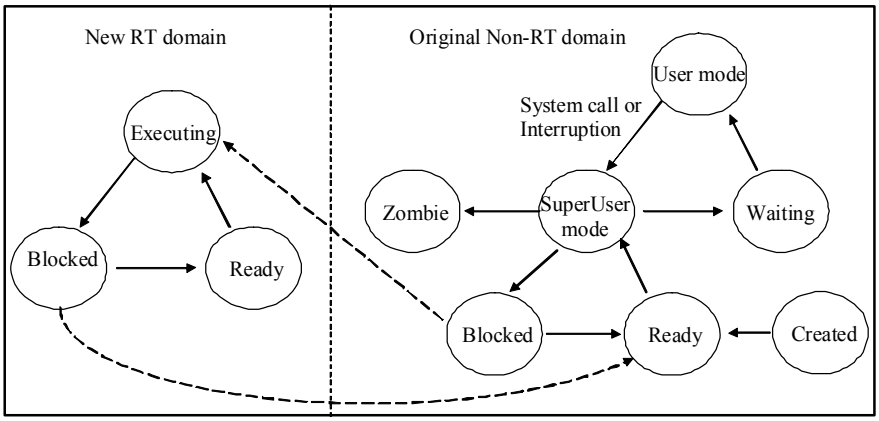


Figure 7-13: Grafo de estados considerando ambos dominios

De esta forma, la función de migración de tareas entre dominios, únicamente ha de modificar el estado en el que se encuentra la tarea.

7.2.2.3.2 Soporte de interrupciones

Las interrupciones producen expulsiones imprevistas en las tareas, introduciendo retrasos que en tareas de tiempo real pueden implicar la violación de los requisitos temporales del sistema. En un sistema convencional, toda interrupción expulsa una tarea en funcionamiento normal. Sin embargo, en sistemas de tiempo real estricto se considera que puede haber tareas de más prioridad que determinadas interrupciones. Por esa razón se realiza una asignación de prioridades a las interrupciones de tiempo real, de forma que pueden competir con las tareas de tiempo real. Las interrupciones que no son gestionadas en el sistema de tiempo real se consideran de menor prioridad que todas las tareas de tiempo real, y son entregadas al dominio de no tiempo real.

Para modelar esto se ha implementado un doble sistema de tratamiento de interrupciones. En SCoPE, el tratamiento convencional de una interrupción implica generar una tarea de máxima prioridad cada vez que se recibe una interrupción. De esa forma es ejecutada expulsando cualquier tarea SW. Para modelar la gestión de tiempo real se ha realizado un sistema de gestión de dos etapas, similar al planificador presentado en la sección anterior.

En primer lugar se realiza la gestión de interrupciones de tiempo real. Si la interrupción ha sido declarada como de tiempo real se genera una tarea de tiempo real con la prioridad asociada, se incluye en la lista de tiempo real, y se fuerza una re-planificación del sistema. De esta forma la gestión de la interrupción se realiza conforme a la prioridad indicada y siempre en el dominio de tiempo real.

Si la interrupción no ha sido declarada como “de tiempo real” se entrega al sistema convencional de gestión de interrupciones de SCoPE. El sistema convencional gestiona la interrupción creando una tarea de máxima prioridad y añadiéndola a la lista de tareas

convencionales, como haría si no estuviese activo el dominio de tiempo real. En consecuencia, al estar introducida en la lista de tareas convencionales, tiene siempre menos prioridad que las tareas de tiempo real, ya que el planificador solo accede a esa lista si la de tiempo real esta vacía.

```
…
if( manage_rt_irq ( irq_number ) == 0){ //New code
        manage_irq ( irq_number );
}// New code
…
```

Figura 7-14: New code use to execute the new interrupt control

7.2.2.3.3 Servicios de tiempo real adicionales

Además de la planificación y la gestión de interrupciones, el sistema de tiempo real estricto proporciona algunos servicios adicionales. Los servicios novedosos son principalmente: gestión de tiempo de alta precisión, adaptación del funcionamiento de algunas llamadas POSIX en función del dominio de ejecución, y nuevas funciones para la gestión de los dominios.

El temporizador HW convencional de un sistema responde en el orden de decenas de milisegundos. En entornos de tiempo real estricto este tiempo es habitualmente excesivo. Para modelar los servicios de tiempo de alta precisión se ha añadido un temporizador HW adicional, con un periodo menor que el convencional. Temporizadores, esperas, alarmas y "timeouts" en el dominio de tiempo real pueden hacer uso de este temporizador de alta precisión para obtener precisiones más altas.

Por otro lado, hay que tener en cuenta que las funciones de la API POSIX pueden ser llamadas desde el dominio de tiempo real. Dado que muchas de estas funciones producen cambios en el estado de ejecución de la tarea, al aumentar el número de estados para modelar el dominio de tiempo real es necesario modificar estas funciones. Para ello se ha extendido la función encargada de realizar los cambios de contexto. Cuando la tarea se encuentra en el dominio de tiempo real se llama a una función especial que gestiona los nuevos estados, en lugar de ejecutar a la función original.

Además, las funciones de tiempo deben detectar el dominio en el que son llamadas para decidir por que temporizador del sistema van a funcionar, ajustando automáticamente su precisión.

Por último, las funciones de la API DIC han sido implementadas, proporcionando las llamadas necesarias para utilizar toda la funcionalidad de tiempo real descrita.

7.2.2.4 Activación de la extensión de tiempo real

La extensión de tiempo real puede ser activada o no en función de los requerimientos del usuario. Las funciones adicionales añadidas para planificación, gestión de interrupciones y gestión de estados adicionales han sido introducidas como llamadas a punteros a función. De esta forma, cuando se activa la extensión de tiempo real, los punteros a las funciones son cargados adecuadamente y llamados en el lugar adecuado. En caso contrario, los punteros son inválidos y no son llamados por el sistema.

Este sistema de punteros a función permite además la integración de otras extensiones al modelo de sistema operativo de forma sencilla. Por ejemplo, la implementación de otros algoritmos de planificación, como un EDF puede ser introducida añadiendo el puntero a función adecuado como función de planificación adicional.

7.3 Modelado de “middleware”: CORBA

Desde el punto de vista de la ingeniería SW, en muchos entornos se desea que las aplicaciones SW puedan ser desarrolladas de manera completamente independiente de la plataforma subyacente. Esto permite por un lado una reducción del esfuerzo del programador, ya que ha de considerar menos elementos en la generación del código. Por otro lado, el desarrollo de código independiente de plataforma dota de gran flexibilidad y portabilidad al código.

El sistema operativo es el elemento encargado de proporcionar los servicios básicos requeridos para la ejecución del SW de aplicación. Además se encarga de gestionar la comunicación con el HW, proporcionando al SW una interfaz más simple de acceso.

Sin embargo, el desarrollo de código sobre el sistema operativo sigue requiriendo que el programador SW conozca los componentes HW de la plataforma, la arquitectura de la plataforma, como número de procesadores y nodos o formas de interconexión entre los mismos o el mapeo de tareas a procesadores. Además es completamente dependiente del sistema operativo, de tal forma que los servicios utilizados del mismo han de ser habitualmente adaptados y extendidos con módulos adicionales. Si tenemos en cuenta que distintas plataformas tendrán distintas arquitecturas, componentes HW e incluso distinto sistema operativo, la portabilidad del código generado es reducida.

Para solucionar este problema se ha propuesto la utilización de una capa adicional entre el SW de aplicación y el sistema operativo, denominada “middleware”. Esta capa se encarga de aislar completamente el SW de la plataforma subyacente, proporcionando a cada módulo SW los servicios requeridos independientemente de quienes sean o donde estén los proveedores de dichos servicios. El “middleware” se encarga de encontrar y adaptar los

servicios proporcionados por la plataforma o por otros módulos SW y proporcionárselo a cada módulo en ejecución.

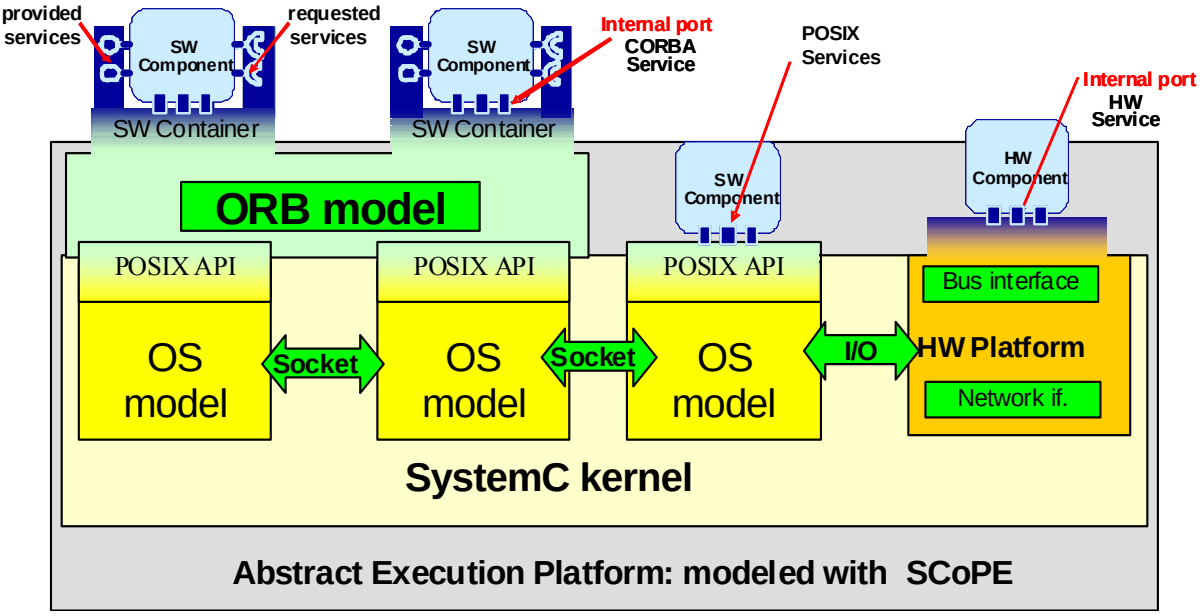


Figura 7-15: Arquitectura del modelo de plataforma de ejecución CORBA.

Uno de los "middleware" más utilizados es CORBA (Common Object Request Broker Architecture). CORBA proporciona una gran infraestructura que se encarga de aislar completamente los componentes SW de la plataforma subyacente. Por cada componente SW se ha de describir una interfaz que genera un esqueleto CORBA donde el código SW puede ser integrado recibiendo todos los servicios necesarios, y proporcionando servicios para otros componentes del sistema.

Toda esta potencia hace que CORBA se convierta en un "middleware" potente pero pesado. En sistemas embebidos la funcionalidad está más localizada que en sistemas distribuidos, por lo que no es necesaria tanta potencia para conseguir independizar el SW de la plataforma. Además, dichos sistemas embebidos requieren componentes ligeros, con bajo costo computacional y mínimos requerimientos de recursos. Dada la complejidad de CORBA, para solventar estos problemas existe una especificación más ligera desarrollada por el OMG denominada Lighweith CCM (CORBA Component Model).

Con objeto de permitir el modelado de sistemas embebidos HW/SW con componentes CORBA se ha integrado un modelo CORBA sobre el modelo de sistema operativo presentado anteriormente. El modelo CORBA integrado ha sido generado mediante la aportación de esfuerzos de varias compañías. El modelo parte de la implementación CORBA llamada MICO (http://www.mico.org/). Esta implementación es de código abierto, y se distribuye bajo licencia GPL.

Para adaptar esta implementación a los sistemas embebidos, se ha generado una implementación modificando la distribución MICO que sigue la especificación “Lighweigh CCM”. Esta implementación se denomina microCCM y ha sido desarrollada por Vincent Seignole, Jean-Louis Gilbert para Thales Communications.

Por último, con objeto de modelar componentes CCM en modelos SystemC, TIMA ha adaptado el código de microCCM para ser ejecutado directamente sobre el núcleo de ejecución de SystemC.

La capacidad de integrar modelos CCM en SystemC proporciona grandes capacidades para el modelado funcional de sistemas embebidos complejos. Sin embargo, en el proceso de diseño de un sistema embebido es crítica la gestión de los recursos y la optimización de parámetros de rendimiento los sistemas.

Por último, se ha adaptado el modelo de CORBA desarrollado por TIMA para su ejecución sobre el sistema de modelado desarrollado en la presente tesis. Esto implica que elementos como la asignación de tareas a recursos, la planificación de tareas o la sobrecarga producida por las comunicaciones pueden ser adecuadamente modeladas. Así la simulación de los sistemas CORBA proporciona al diseñador información sobre el rendimiento del sistema que es útil en el diseño y refinado de los sistemas.

Para generar dicho modelo de CORBA, se ha modificado el modelo de TIMA, para ser ejecutado sobre la interfaz POSIX del modelo de sistema operativo. Para ello se han recuperado algunas de los elementos de acceso a POSIX eliminados al adaptar microCCM a la ejecución en SystemC. La implementación microCCM original está diseñada para ser ejecutada sobre un sistema operativo Linux, que proporciona funciones contenidas en POSIX. Por esta razón, en determinados casos, la mejor forma de adaptar el modelo CORBA a POSIX es recuperar código de la implementación original.

Para el modelado de las comunicaciones, el modelo CORBA desarrollado hace uso de la pila TCP/IP desarrollada, que conectada al controlador de red correspondiente hace uso de los modelos HW de bus, interfaz de red y red que serán descritos en el capítulo siguiente. De esta forma el modelado de sistemas CORBA considera todos los elementos que afectan al rendimiento de los sistemas reales, proporcionando estimaciones de rendimiento consistentes.

8 Modelado de la plataforma HW

8.1 Introducción

En general, el funcionamiento de los sistemas electrónicos, y especialmente de los embebidos, depende fuertemente de la plataforma HW subyacente. Esta plataforma HW interviene tanto en el completo desarrollo de la funcionalidad del sistema como en su rendimiento. Es por ello que la simulación de los componentes HW del sistema es necesaria para un correcto modelado del mismo. Por ello junto con la infraestructura de modelado SW presentada en los capítulos anteriores, en la tesis también se ha desarrollado una infraestructura de modelado de plataformas HW.

El trabajo realizado para el modelado de la infraestructura HW se ha centrado en plataformas compuestas por procesadores rodeados de una serie de componentes HW que dan soporte a la ejecución de la aplicación. El modelado de los procesadores y sus efectos se resolvió en los dos capítulos anteriores. Para realizar el modelado del resto de los componentes HW, estos se han dividido en dos grupos: componentes genéricos y componentes específicos de la aplicación.

Los componentes genéricos son aquellos que están presentes en la mayoría de los sistemas, independientemente de la funcionalidad a la que esté orientado el sistema. Componen este grupo elementos como el bus, la memoria o las interfaces de red. Estos componentes tienen la particularidad de que no proporcionan funcionalidad para la aplicación sino que se limitan a tareas de almacenamiento de la información y de comunicaciones. Los componentes HW particulares de la aplicación son todos aquellos componentes que proporcionan una funcionalidad requerida por la aplicación.

Basándonos en esta división, la infraestructura de modelado de la plataforma HW desarrollada en la tesis proporciona modelos para los componentes genéricos. De esta forma un diseñador puede construir una arquitectura de propósito general capaz de realizar la ejecución SW a partir de los modelos proporcionados sin necesidad de desarrollar modelos propios. Sin embargo, deberá ser el diseñador el encargado de proporcionar los modelos para los componentes específicos del sistema. Para simplificar esta tarea, se han desarrollado modelos de interfaces genéricas que permiten integrar fácilmente los modelos de comportamiento del usuario en el modelo de plataforma, permitiendo un conexionado automático con el modelo resto del sistema.

8.1.1 Nivel de abstracción del modelo de plataforma HW

Las técnicas de modelado y estimación de todo este trabajo han sido diseñadas para proporcionar unas estimaciones consistentes basadas en simulaciones lo más rápidas posibles.

Para conseguir una relación adecuada entre velocidad y precisión el modelado del SW, presentado en el capítulo 6, se basa en la ejecución consecutiva de secuencias de instrucciones SW. Así se consigue que la sobrecarga introducida en la simulación sea muy reducida. De la

misma forma, la plataforma HW debe ser modelada manteniendo esta filosofía. No tendría sentido, por ejemplo, modelar el código SW en base a grandes secuencias temporales y modelar el comportamiento del bus ciclo a ciclo.

Tanto los modelos desarrollados de los periféricos como los elementos de comunicación han sido descritos en alto nivel. Todas las comunicaciones realizadas en el modelo de plataforma están basadas en el nivel de tiempos difusos ("loosely timed") de TLM. Los modelos de los componentes son descritos en base a su comportamiento reduciendo en la medida de lo posible los detalles de funcionamiento ciclo a ciclo para pasar a realizar modelados más generales de su comportamiento.

Siguiendo este modelo, las transacciones en el bus son modeladas en base a grandes trasferencias de información. Por ejemplo, para mantener la coherencia con el modelo SW, los fallos de cache de cada secuencia de código son agrupados en bloques, y transferidos en conjunto. De esta forma, en lugar de producirse una transferencia por palabra, o incluso por fallo de cache, se minimiza considerablemente el número de transferencias. Otro ejemplo pueden ser los envíos de paquetes generados por la pila TCP/IP. Cada paquete se envía en una única transferencia, minimizando la sobrecarga por simulación de las transferencias de red en el sistema.

Por otra parte, la ejecución de los modelos de comportamiento de los componentes HW es, en general, mucho más pesada que la simulación SW. La simulación SW, basada en ejecución nativa es más ligera que la simulación basada en eventos de los modelos HW, puramente SystemC. Por esta razón la reducción del número de ejecuciones de los componentes HW tiene un gran peso en la optimización del tiempo de simulación de los modelos de sistema.

8.1.2 Arquitectura HW modelada

Como se presentó en el capítulo 5, el modelo de plataforma HW desarrollado se basa una arquitectura intermedia entre los sistemas multiprocesador fuertemente acoplados y los sistemas multi-computador, con un acoplamiento más débil pero menos flexibles. Esto permite optimizara la reacción entre las capacidades de balanceo de carga computacional y los requisitos de comunicaciones en el chip.

La arquitectura está compuesta por nodos computadores interconectados formando un sistema global. En cada nodo computador un número variable de procesadores operando conjuntamente con buses, memorias y otros periféricos. El modelado de las interconexiones entre los nodos se realiza mediante modelos de red. En las siguientes secciones se presentarán tanto las técnicas y componentes desarrollados para el modelado de cada nodo como los modelos de red generados para posibilitar la comunicación de los nodos entre sí.

Para conseguir que estas arquitecturas puedan ser modeladas de forma sencilla y flexible sobre TLM, la implementación propuesta se basa en la combinación de

configuraciones en estrella. Estas configuraciones contienen un elemento de comunicación en el centro, y múltiples componentes HW, maestros o esclavos alrededor.

Supongamos por ejemplo un sistema con una red que comunica cuatro nodos con procesadores y memorias locales. En este caso la implementación se compone de cinco configuraciones en estrella. Cuatro configuraciones tienen como elemento central un bus, y como elementos periféricos los procesadores, las memorias y las interfaces de red. La quinta configuración tienen como elemento central la red y como elementos periféricos las interfaces de red. Así pues cada interfaz de red está conectada forma parte de dos estrellas.

La ventaja de esta técnica es que, aunque los componentes que se usen en cada configuración, tanto en el núcleo como en los periféricos, sean distintos, la estructura y la forma de interconexión es común. Dicho de otra forma, se pueden generar interfaces comunes que permitan conectar elementos centrales o periféricos a cualquier tipo de configuración estrella, maximizando la flexibilidad en la construcción de arquitecturas e incluso permitiendo la construcción automática de modelos, como se verá en el capítulo 9.

Por el momento se han desarrollado dos tipos de configuraciones estrella, dependiendo del elemento central, que son: nodos locales de cómputo y redes. Esto no significa que únicamente se puedan modelar sistemas con redes en configuración estrella. Una cosa es el mecanismo de construcción del modelo y otro lo que se modele. Al construir una configuración en estrella de una red, se conectan múltiples interfaces de red a un elemento central, que es el modelo de red. Desde este punto de vista, es una configuración muchos a uno, construible como estrella. Adicionalmente, el modelo de red al que se conectan las interfaces puede modelar internamente cualquier configuración deseada, incluyendo los protocolos de enrutamiento, retrasos y sobrecargas que correspondan.

8.2 Modelado HW de los nodos de cómputo

Los componentes HW de un nodo pueden ser divididos en tres grupos: los maestros, los esclavos y los elementos de comunicación. Ciertos elementos, como un DMA, pueden considerarse simultáneamente tanto maestros como esclavos, requiriendo que sean conectados dos veces al elemento central de comunicación, una como maestro y otra como esclavo. De forma simular, otros elementos diseñados para la comunicación entre nodos, como un “bridge”, pueden ser conectados simultáneamente a dos nodos, usualmente en uno como maestro y en otro como esclavo.

Para alcanzar los balances entre precisión y velocidad de simulación requeridos, el modelado de los elementos de comunicación representa la parte más importante de la infraestructura. Para realizar el modelado de las comunicaciones en el nodo se ha desarrollado un modelo de bus genérico capaz de transferir información entre los procesadores y los periféricos del sistema. Además, el modelo de bus es capaz de entregar a los procesadores las interrupciones generadas por los periféricos. El modelo de bus, aunque desarrollado para manejar grandes transferencias de información, es capaz de manejar aquellos detalles de funcionamiento interno con gran efecto en el rendimiento del sistema, como la gestión del

ancho de banda disponible y la gestión y arbitrio de colisiones. El modelo ha sido diseñado para que sea posible la generación de arquitecturas con buses encadenados. Todo esto ha sido realizado siguiendo la especificación TLM2.

Para modelar la interconexión entre el bus y los procesadores y realizar la integración entre la infraestructura de simulación SW y la infraestructura HW se ha desarrollado un componente encargado de proporcionar la interfaz al bus de los procesadores del sistema.

Para el modelado de los periféricos, considerados esclavos, se han desarrollado modelos de los componentes genéricos: memoria, interfaz de red. Por último, se describirán las interfaces genéricas que permiten una fácil integración de los modelos de comportamiento de los componentes HW desarrollados por el usuario.

8.2.1 Modelo de bus

El desarrollo adecuado de un modelo del bus de comunicaciones es un elemento crítico a la hora de evaluar sistemas HW/SW. En cada nodo del sistema, todas las comunicaciones del procesador con los periféricos e incluso de algunos periféricos entre sí tienen que atravesar el bus. Además, en sistemas distribuidos, las comunicaciones entre distintos nodos procesadores también implican múltiples accesos al bus.

Por esta razón es necesario que la herramienta de modelado ofrezca al diseñador elementos para poder modelar adecuadamente los buses del sistema, considerando tres características principales: precisión, rapidez y generalidad.

Como se ha dicho, el modelo de bus desarrollado se ha basado en la versión 2 del estándar TLM, creada por los desarrolladores de SystemC. TLM2 surge con la intención de generar interfaces de comunicación que permitan obtener un cierto grado de portabilidad entre sistemas generados por diseñadores distintos sobre dicho estándar. Para ello, TLM2 propone una serie de interfaces y de estructuras para la transferencia de información mucho mas estrictamente definidas que en la versión inicial de TLM. De esta forma, todos los diseñadores que utilicen estas interfaces y estructuras podrán entender y adaptar más fácilmente otros diseños. Esto es un avance muy importante en el intento de ofrecer generalidad al modelo de bus.

Más en concreto, la prueba de concepto desarrollada como parte de la tesis ha utilizado la implementación TLM2-draft1. La adaptación a la versión definitiva de TLM2 no ha sido realizada como parte de la propia tesis. Esta adaptación ha sido introducida en versión de SCoPE por David Quijano como modificación de la versión de investigación presentada aquí. Por esta razón, en lo sucesivo, el término TLM2 se referirá a TLM2-draft1.

Además, tras analizar el funcionamiento del bus en su primera versión, se han desarrollado nuevos planteamientos para los mecanismos de comunicación utilizados. Estas nuevas soluciones permiten un modelado más rápido e incluso más preciso, sin perder la generalidad buscada.

8.2.1.1 Diseño del bus

El modelo del bus esta dividido en dos partes. Por una parte, el modelo se encarga de realizar la transferencia de datos entre los módulos maestros y los esclavos. Por otra parte, el bus entrega las señales de interrupción de los periféricos a los procesadores.

Mientras que las interrupciones sólo tienen que transferir el número de la interrupción enviada, la cantidad de información manejada para la transferencia de datos es mucho más alta. Esta información da lugar a mayores transferencias, tanto en el tiempo requerido para la propia transferencia, como en la cantidad de información transferida cada vez. Cuando empieza una transferencia de datos, se indica la dirección de la transferencia, el tipo (lectura / escritura), la prioridad, el tamaño del dato, etc. En cuanto a los datos a transferir, la información es agrupada en bloques que se transfieren en conjunto. Esta peculiaridad posibilita la obtención de velocidades de simulación mayores sin perder capacidad de modelado. Para optimizar la transferencia de toda la información enviada en un mismo instante, la librería TLM2 proporciona estructuras adecuadas donde esa información puede ser almacenada de forma estándar.

Para aprovechar la transferencia de bloques en ráfagas, el bus se ha modelado de tal forma que sus funciones de comunicación permiten tanto la transferencia de bloques, indicando el tamaño de los datos, como la identificación del ancho de banda necesario. En lugar de modelar todas las señales individuales que se utilizan cada ciclo en el bus, el sistema modela la transferencia de datos en una única operación. Así, la reducción del número de elementos modificados maximiza la reducción del tiempo de simulación.

No obstante, el mayor avance se obtiene de la propia gestión de los bloques de datos como elementos únicos. En lugar de modelar la transferencia ejecutando una operación de bus por cada palabra del bloque, se ejecuta de una sola vez. Esto implica que el número de transferencias a modelar es mucho más pequeño, reduciendo el número de llamadas a función por Kb transferido.

Otro detalle tenido en cuenta en el modelo es que varias transferencias de bloques de información puede realizarse simultáneamente. Si dos procesadores quieren realizar transferencias a la vez (por ejemplo volcar los datos de sus caches en memoria), el bus será usado alternativamente por los procesadores hasta que todas las transferencias terminen. Para simular esto se ha incluido en el modelo de bus un sistema de gestión de ancho de banda.

La gestión de anchos de banda permite calcular los tiempos de transferencia, modelando las colisiones en el bus y la utilización de los recursos. Esto es especialmente importante cuando se modele jerarquía de buses. En este caso, el ancho de banda requerido por una transferencia en un bus dependerá de la velocidad de transferencia en el resto de buses. De esta forma, el bus puede modelar la multiplexación producida cuando las transferencias no pueden utilizar toda la velocidad disponible.

En la implementación desarrollada, el bus reparte en cada momento el ancho de banda disponible entre todas las transferencias en curso, en función de la prioridad del maestro que realiza la comunicación. Cada vez que una transferencia empieza o termina, el resto de transferencias en curso son reajustadas con objeto de repartir de nuevo el ancho de banda conforme a la nueva situación.

8.2.1.2 Interfaces de TLM2 usadas

Para implementar los dos tipos de conexiones necesarios para conectar tanto maestros como esclavos al bus se han incluido dos tipos de interfaces TLM2 distintas.

La interfaz de envío de interrupciones está basada en la interfaz "tlm_nonblocking_put_if". Esta interfaz esta compuesta por tres funciones:

- "nb_can_put": Retorna siempre "true"; siempre se puede mandar la interrupción
- "ok_to_put": Retorna un sc_event donde el proceso pueda esperar a que la interrupción pueda ser enviada. Dado que siempre puede ser mandada, esta función no tiene utilidad.
- "nb_put": Envía la interrupción. La función es ejecutada y retorna automáticamente: no se espera a que el procesador acepte la interrupción.

La función "nb_put" de la interfaz procesador-bus llama a la función de bajo nivel del sistema operativo para la recepción de interrupciones automáticamente. Esta función interrumpe la tarea en ejecución en ese procesador y lanza el manejador de interrupción correspondiente.

En cuanto a la interfaz para la transferencia de datos se ha usado la interfaz "tlm_annotated_transport_if". Esta interfaz proporciona dos funciones:

- "response & transport (request);"
- "void transport (request, transport, time);"

Para los miembros "request" y "response" se utilizan las estructuras estándar de TLM2. La estructura "request" esta formada por varios campos, entre los que se encuentra la dirección, los datos, la prioridad, el tipo de paquete (datos, control, degub), tipo de transferencia (lectura/escritura), identificador de transferencia, identificador de dispositivo, etc. La estructura "response" contiene un subconjunto de estos campos. La utilización de algunos de estos campos depende en buena medida del periférico.

El parámetro de tiempo sirve para indicar el ancho de banda requerido. Sabiendo el tiempo máximo que se desea tardar en la transmisión y el tamaño de la información a transferir, se puede calcular el máximo ancho de banda requerido.

El bus también contiene un módulo encargado de la decodificación de direcciones. Este módulo sirve para relacionar las direcciones de las peticiones de transferencia de datos con los periféricos correspondientes.

8.2.1.3 Extensiones a las interfaces TLM2

Además de las funciones propuestas por el estándar TLM2 se han extendido dos nuevas funciones a la interfaz de transferencia de datos. Estas funciones son “stop” y “abort”.

La función “stop” se usa para indicar que transferencia debe ser paralizada. Esta función indica al periférico que la función “transport” que esta realizando la correspondiente transferencia debe terminar, aunque no haya pasado el tiempo suficiente para aceptar todos los datos. Al retornar, se informa al transmisor de la cantidad de información que ha sido aceptada, para que la siguiente transmisión continúe desde ese punto. Además el periférico debe almacenar la información aceptada y esperar a que se reinicie la comunicación.

La función “abort” se usa para indicarle al periférico que la transferencia en curso ha sido anulada. De esta forma, la función de transferencia retorna con valor de error, y la información recibida en el periférico se elimina.

Estas funciones tienen dos utilidades básicas. En primer lugar, dado que cada transferencia implica el envío de bloques de datos durante un determinado tiempo, puede ocurrir que durante ese tiempo el sistema cambie de estado. Por ejemplo, si la tarea SW que ha pedido la transferencia es expulsada por otra de mayor prioridad, la transferencia debe ser paralizada hasta que vuelva a tener el procesador. Si el proceso es matado por otro, la transferencia debería ser abortada. Además la función “stop” sirve para implementar el reparto dinámico de ancho de banda explicado en la sección anterior.

8.2.2 Modelos de componentes HW

8.2.2.1 Interfaz bus-periférico genérica

Para facilitar el diseño y la integración de componentes HW con el modelo de bus proporcionado se han desarrollado interfaces genéricas que resuelven la gestión de los protocolos de comunicación de forma automática. Se han desarrollado dos tipos de interfaces: interfaces tipo maestro e interfaces tipo esclavo.

Las interfaces maestro proporcionan funcionalidades para comenzar comunicaciones en el bus. Estas interfaces proporcionan elementos SystemC tipo “export”, que representan peticiones de servicio. Estas interfaces deben ser conectadas al modelo de bus de tal forma que estas peticiones de servicio sean atendidas por los servicios del bus. Además las interfaces maestro tiene la capacidad de aceptar peticiones de interrupción. Las interfaces maestro están diseñadas para ser utilizadas desde interfaces de procesador y otros maestros como DMAs.

Las interfaces esclavo proporcionan servicios de comunicaciones. Estas interfaces están diseñadas para servir de adaptador entre los modelos de comportamiento de los componentes HW del sistema y el bus de comunicaciones. Están diseñadas para recibir peticiones de lectura y escritura del bus sobre determinados registros del mapa de memoria. Estas interfaces se pueden utilizar de dos formas: o como elementos de almacenamiento para intercambio o como adaptadores de peticiones.

Las interfaces tienen capacidad de modelar conjuntos de registros. De esta forma las descripciones HW pueden escribir y leer estos registros proporcionando o leyendo los datos que han de ser transferidos por el bus. La interfaz, por su parte, utiliza estos registros para recibir y almacenar los datos recibidos mediante peticiones de escritura y para proporcionar los datos requeridos por el bus durante las peticiones de lectura.

Las interfaces tienen también la capacidad de transferir las peticiones de acceso desde el bus al modelo HW encargándose de los protocolos de comunicación conforme al estándar TLM2. De esta forma cuando se recibe una petición de lectura o escritura en el bus la interfaz realiza una llamada a una función que debe ser proporcionada por el modelo HW para gestionar dicho acceso. El uso de esta interfaz evita que el modelo HW tenga que encargarse de decodificar las estructuras de datos TLM, gestionar los protocolos o encargarse del modelado del tiempo requerido por las comunicaciones.

Estas interfaces implementan las funciones “transport”, manejando las estructuras “request” y “response” adecuadamente. Una vez decodificadas las estructuras, la interfaz llama a las funciones “read_data”, “write_data” y sus equivalentes para control del periférico. Estas nuevas funciones deben ser desarrolladas específicamente en cada periférico, para implementar su funcionalidad específica.

Las funciones “transport”, también gestionan la respuesta a las funciones “abort” y “stop”, ocultando las paradas en el transporte de la información a través del bus al resto del periférico. Este solo es informado cuando ha sido recibido un paquete entero correctamente.

Por último, la interfaz genérica también proporciona al periférico funciones para el envío de interrupciones a través del bus. Además incluye la posibilidad de definir cuatro tipos de interrupciones y es capaz de gestionar la mascara de interrupciones automáticamente. Las cuatro interrupciones están pensadas para identificar las situaciones en las que buffer de salida este lleno o vacío, que el buffer de entrada este lleno, o que se reciba un nuevo paquete. Se ha considerado que estos son los casos principales en los que un periférico genérico puede requerir el envío de una interrupción.

No obstante, todas estas funciones pueden ser redefinidas al ser heredadas en la generación de nuevos modelos de periférico. De esta forma se puede modificar el funcionamiento tanto de las funciones de envío y enmascaramiento de interrupciones como de las de transferencia de datos por el bus para adaptarse a las necesidades específicas del periférico.

8.2.2.1.1 Integración de las comunicaciones en los modelos HW

Uno de los problemas más importantes en el diseño de los sistemas HW/SW es la integración y refinado de los protocolos de comunicaciones durante las distintas etapas de desarrollo. Si analizamos las interfaces de los componentes que pueden ser conectados a un mismo bus, podemos apreciar que dichas interfaces son bastante similares, al estar encaminadas a gestionar un mismo protocolo de comunicaciones a un mismo nivel de abstracción. Es por ello que se pueden generar elementos reutilizables que permitan una fácil conexión de los componentes HW al modelo de bus.

En modelos de alto nivel, la adaptación de los modelos de comportamiento HW a los modelos de bus suelen ser resueltos con el uso de transactores. Estos transactores son componentes que existen solo en el modelo. Son colocados entre el componente HW y el bus y que se encargan de adaptar las comunicaciones.

Sin embargo el uso de transactores no siempre es deseable, ya que implican introducir nuevos componentes en el diseño, lo que puede complicar el modelo. Además sacan la gestión de los protocolos de comunicaciones del modelo HW, con lo que se plantea el problema de cuando integrarlo en el diseño.

Otros intentos de resolver este problema han dado lugar a la integración de los protocolos de comunicaciones en los puertos de los componentes. Aunque esto evita la aparición de nuevos elementos en el modelo, la integración de cualquier tipo de funcionalidad en el puerto, aunque se limite al control de las comunicaciones, va en contra de la idea de puerto. Un puerto debe ser solo una interfaz que proporciona la definición de una serie de funciones y a través de cual se hace el enlace entre el periférico y el bus.

La solución propuesta en este trabajo es integrar la funcionalidad de comunicaciones de forma automática como una parte más del modulo HW. Para realizar esto se han utilizado las cualidades de herencia y polimorfismo de C++. De esta forma, un módulo HW, en lugar de heredar directamente de un SC_MODULE, que no proporciona ninguna funcionalidad adicional, se hereda la clase “tlm_xxx_bus_module”, donde xxx indica el nivel de abstracción utilizado: pv, pvt o bca. Esta herencia, hace que el modulo HW contenga automáticamente los puertos necesarios para su conexión al bus, así como las funciones necesarias para la gestión de dichas comunicaciones.

La utilización de la herencia hace que sea en la propia funcionalidad del módulo donde estén integradas las comunicaciones. Esto permite adaptar el módulo a distintos niveles de abstracción de una forma muy sencilla, ya que solo es necesario cambiar la clase heredada. Con este mínimo cambio se reemplazan automáticamente los puertos y los protocolos de acceso, permitiendo una fácil conexión a otros tipos de bus.

Además, la aplicación de polimorfismo permite que cualquiera de las funciones de comunicaciones sea refinada desde el modelo HW. Así puede ir refinándose la comunicación HW/SW paso a paso y función a función.

Para realizar esta herencia, la macro SC_MODULE ha sido redefinida, de tal forma que el nuevo módulo SystemC hereda automáticamente la clase correspondiente al nivel de abstracción. Además, la macro SC_CTOR también ha sido redefinida para que el constructor del nuevo módulo se adapte a la nueva clase heredada. La redefinición de estas macros se ha llevado a cabo proponiendo nuevas macros BUS_MODULE y BUS_CTOR, de forma que los módulos SystemC que no necesiten incluir las interfaces de comunicación con el bus sigan permaneciendo conforme al estándar SystemC, mientras que aquellos que si vayan a ir conectados directamente al bus integren automáticamente la funcionalidad requerida.

TABLA 8.1- MODIFICACIONES REALIZADAS EN LAS MACROS SYSTEMC

#define BUS_MODULE(name) struct name: tlm_pvt_bus_module
#define BUS_CTOR(name) … name(…): tlm_pvt_bus_module(...)

Para implementar interfaces a distintos niveles de abstracción se han desarrollado modelos a los niveles PV (atemporal), PVT (con tiempo aproximado), y BCA (con precisión de ciclos de bus). Esto puede ser fácilmente extendido a niveles inferiores como conexiones a nivel de pin.

Para conectar la interfaz con el modelo HW asociado se proporcionan una serie de funciones de lectura y escritura. Las funciones “read (addr, data, size)” y “write (addr, data, size)” proporcionan y retornan los datos transferidos por el bus a las direcciones indicadas. Así mismo se proporcionan funciones “wait_read” y “wait_write” que implementan bloqueos. Estas funciones paran la tarea hasta que se realicen los accesos requeridos por el bus. Además las interfaces proporcionan funciones para enviar interrupciones por el bus mediante “send_interrupt”.

8.2.2.2 Modelos de componentes HW genéricos

Utilizando las interfaces anteriores se han desarrollado algunos modelos para componentes genéricos. Estos modelos son principalmente una interfaz procesador-bus, un modelo de memoria, y un modelo de interfaz de red.

8.2.2.2.1 Interfaz bus-procesador

Para comunicar el bus con los procesadores se ha desarrollado una interfaz que comunica las funciones de bajo nivel del modelo de sistema operativo POSIX con el bus. Para ello la interfaz aporta funciones para enviar peticiones de lectura y escritura de control y datos, tanto con información de tiempo como sin el, para indicar el ancho de banda requerido.

uc_bus_read_data
uc_bus_annotate_read_data
uc_bus_write_data
uc_bus_annotate_write_data

Además la interfaz proporciona la funcionalidad necesaria para la recepción desde el sistema operativo de las interrupciones generadas por el HW. Para ello se utiliza la función del sistema operativo 'uc_interrupt_received'.

8.2.2.2.2 Memoria

El segundo modelo HW genérico implementado es un modelo de memoria. El modelo utilice la interfaz genérica de conexión al bus de tipo esclavo. El modelo implementa las funciones "read" y "write", para gestionar lecturas y escrituras. El modelo de memoria da respuesta a las transferencias producidas por los fallos de cache y a los accesos desde otros maestros, como los DMAs.

Como se explicó en el capítulo de modelado de código SW de usuario, el modelado de SW de aplicación tiene únicamente propósitos de estimación de rendimiento, no funcionales. Esto significa que la ejecución SW no accede al modelo de caché para obtener cada instrucción o dato que se va a ejecutar. Al ser una ejecución nativa, el código se ejecuta realmente sobre la infraestructura del PC donde se realiza la simulación, cache incluida, no sobre la cache del modelo. El modelo de cache tiene como únicos propósitos modelar los retrasos producidos por los fallos de cache, las transferencias por el bus y los accesos a memoria en el modelo de plataforma.

Por esta razón los accesos a memoria desde la cache no transfieren datos válidos. Lo importante en las transferencias es la cantidad de información transferida, pero no los valores. De hecho estos no son guardados en el modelo de cache. Esto evita copias de datos innecesarias y acelera la simulación.

Para accesos desde otros periféricos, la memoria salva o entrega al bus los datos correspondientes a las posiciones de memoria en la simulación en curso.

Por ultimo, indicar que el modelo de memoria no genera interrupciones.

8.2.2.2.3 Interfaz de red

Para sistemas basados en múltiples nodos conectados por una red es necesario disponer de interfaces de red. Estas interfaces se conectan al bus de cada nodo y permiten la comunicación entre nodos. La interfaz de red generada hereda la interfaz genérica de conexión al bus para esclavos, al igual que la memoria. Las operaciones de lectura y escritura son redireccionadas para acceder al simulador de red correspondiente. Dependiendo de la interfaz de red que realiza la transferencia el simulador sabe el nodo desde el que se realiza la comunicación. De la misma forma cuando un paquete alcanza el destino, el modelo de red ha de entregárselo a la interfaz de red adecuada.

Cuando un paquete es recibido se genera una interrupción que indica al sistema operativo que hay un dato disponible.

La información sobre los modelos de red se presentará en la sección 8.3.

8.2.2.3 Integración de módulos de propósito específico

Además de los módulos básicos necesarios para construir un modelo de sistema, el usuario puede integrar los modelos HW que considere oportunos. Para integrar dichos modelos ha de utilizar las interfaces presentadas anteriormente, que permiten una fácil conexión al modelo de bus.

Para el modelado correcto de la plataforma, es necesario que los modelos HW no solo tengan capacidad para modelar la respuesta funcional de los componentes. Además han de contener la información temporal necesaria para que su efecto en el rendimiento del sistema pueda ser modelado.

En algunos casos, cuando el diseñador desea integrar modelos HW de alto nivel, no se dispone de información necesaria para realizar un modelo temporal correcto del componente. Para resolver este problema se ha realizado una técnica capaz de realizar estimaciones del coste temporal de la ejecución de los componentes HW. El trabajo realizado sobre estimación HW tiene como objetivo obtener una primera estimación aproximada sobre los tiempos de respuesta. El desarrollo de una técnica realmente precisa para la estimación de los mismos queda fuera del ámbito de la tesis, y se ha considerado como una posible línea de investigación futura.

8.2.2.3.1 Estimación de módulos hardware

La técnica utilizada para la estimación del tiempo de ejecución de los componentes HW se basa en la técnica de sobrecarga de tipos presentada para el modelado SW. De esta forma los tipos de datos se sustituyen automáticamente por otros tipos sobrecargados, de tal forma que cada operación ejecutada es interceptada y se le asocia un costo temporal.

La estimación de componentes hardware requiere un método más complejo que parar componentes software, dado que mientras la ejecución de un código software solo puede ser realizada ejecutando secuencialmente todas las instrucciones máquina, las instrucciones de una descripción hardware pueden realizarse tanto secuencialmente como en paralelo, en función de los recursos que se destinen a su implementación.

El mecanismo utilizado se basa en la definición de un límite superior y un límite inferior tanto para el número de puertas utilizadas (área) como para el tiempo requerido para la ejecución de un segmento de código. El límite de mayor tiempo y menor área se puede definir de forma muy similar a la ejecución de código software. Este punto se da cuando utilizamos el mínimo número de recursos, de tal forma que no se puedan resolver operaciones en paralelo.

El límite de menor tiempo, y que por tanto define un límite superior para el número de recursos requeridos, se puede medir utilizando un algoritmo ASAP (tan rápido como sea

posible). En este método, cada operación empieza en el momento en que está disponible el último dato necesario para realizar la operación, independientemente de las operaciones que se estén realizando en paralelo con ella. El área máxima se define como el número máximo de operadores de cada tipo que se utilizan a la vez en algún momento de la ejecución.

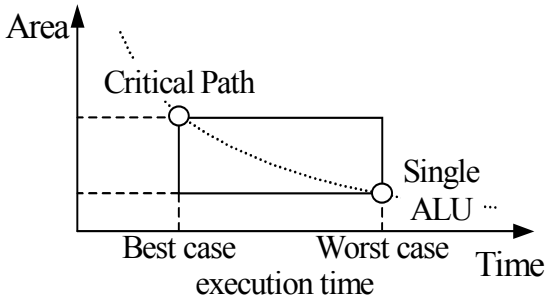


Figura 8-1: Area posible de diseño para un módulo HW

Para calcular el tiempo en que se realiza cada operación en el método ASAP, cada variable sobrecargada contiene un miembro que indica el momento en que se cargó el dato actual. Cada vez que se realiza una operación se calcula el tiempo en que se realiza esa operación como el mayor de los tiempos de las variables a computar más el coste temporal de la propia operación. Además se dispone de una lista global donde se almacenan los tiempos de las operaciones de control (if, switch,…). Con esta información se considera que el dato es válido cuando las operaciones de control han sido resuelta y cuando la operación ha acabado. Para ello se coge el mayor de los tiempos. Por último, este tiempo se asocia como tiempo de asignación a la variable que recibe el resultado de la operación, para que pueda ser utilizado en las operaciones futuras que utilicen dicha variable.

Una vez definidos los valores máximos y mínimos para tiempo y área, hay que elegir un valor estimado de tiempo para introducirlo en la simulación, dado que el manejo de pares máximo-mínimo puede convertir la simulación en extremadamente compleja y no directamente manejable por SystemC. Este tiempo se obtiene declarando una constante “C” de tal forma que el tiempo estimado finalmente es:

$$T = Tmin + C \cdot (Tmax-Tmin),\ 0 <= C <= 1.$$

La anotación de este tiempo estimado se realiza de la misma forma que en el caso del software, lo que implica que operaciones de distintos segmentos no se pueden ejecutar en paralelo ni se considera la posibilidad de optimizar el sistema desplazando operaciones entre segmentos consecutivos.

La técnica propuesta considera únicamente los tiempos y área referentes a la unidad de datos del HW. Los efectos derivados del funcionamiento de la unidad de control asociada al HW no son considerados. Este modelado ha sido dejado como trabajo futuro a efectos de la tesis.

8.3 Modelado de red

Para permitir el modelado de sistemas débilmente acoplados, es necesario un modelo de red. Para permitir flexibilidad en el modelo de sistemas complejos se proporcionan dos modelos de red: uno rápido y poco preciso, y otro más preciso pero más lento. Para el primer modelo se ha implementado un componente capaz de modelar una red de tipo mesh de forma muy ligera. Para el segundo se ha utilizado un simulador de redes desarrollado por el grupo de Arquitectura de Computadores de la Universidad de Cantabria, llamado Sicosys.

8.3.1 Interfaces para la integración del modelo de red

Para facilitar la integración y el intercambio de los modelos de red se ha desarrollado un componente que hace de enlace entre la el modelo de plataforma HW y el modelo de red. Este componente es común para ambos modelos de red de tal forma que el cambio de modelo se puede realizar de forma automática. Para ello se ha definido un número mínimo de funciones que ha de proporcionar el modelo de red para que pueda ser integrado. Además se ha definido un conjunto reducido de funciones que puede utilizar el modelo de red para acceder al núcleo de la simulación SystemC.

Como resultado es posible modificar el modelo de red ejecutado modificando únicamente la etapa de lincado que da lugar al ejecutable SystemC. Los dos modelos de red están diseñados para ser compilados como librerías. Dado que las interfaces son equivalentes, únicamente es necesario modificar la librería enlazada para cargar un modelo de red u otro.

8.3.2 Modelado de red con el modelo simple de SCoPE

El modelo de red simple desarrollado está orientado a la gestión funcional de las transferencias de paquetes, minimizando el modelado de los efectos de funcionamiento interno de la red. El modelo se encarga de recoger los paquetes en el nodo origen y entregarlos en el destino, pero no considera elementos como colisiones, protocolos de enrutado o colas internas en los nodos.

La estimación temporal se realiza considerando el número de saltos que ha de realizar el paquete para llegar al destino. Para conocer el número de saltos el modelo de red mesh en tres dimensiones se organiza en base a tres ejes: x, y, z. Cada nodo se nombra conociendo su posición en el eje x, y, z de la red. Sabiendo la diferencia en x, y, z entre el nodo origen y el destino se calcula el número de saltos de un camino mínimo. Considerando el tiempo necesario para cada salto se obtiene el tiempo mínimo total. Además se aplica un factor de corrección en función de la carga de la red.

Dado que el modelo de red no está desarrollado como una agrupación de modelos de switch, y que no se consideran colas ni protocolos de enrutado, el tiempo de simulación de la red es mínimo. A cambio se pierde precisión los tiempos de transferencia estimados.

8.3.3 Modelado de red con Sicosys

Para permitir el modelado de sistemas multiprocesadores sobre "Network on Chip" (MpSoC with NoC) se ha integrado un simulador de redes externo en SCoPE. El simulador seleccionado es el desarrollado por el grupo de Arquitectura de Computadores de la Universidad de Cantabria, llamado Sicosys.

Para integrar dicho simulador se han desarrollado tres elementos diferenciados: un periférico que modela la interfaz de red conectada al bus, un módulo que gestiona el funcionamiento del simulador, integrándolo en el kernel de simulación de SystemC, y los elementos necesarios para modificar internamente el simulador de tal forma que acepte y devuelva los paquetes de la librería de simulación adecuadamente.

8.3.3.1 Introducción a Sicosys

Sicosys es un simulador de interconexión de redes de propósito general para sistemas multiprocesador que permite el modelado de una gran variedad de routers de forma precisa. Sicosys incluye una colección de componentes, como multiplexores, o búferes, que modelan el hardware con gran precisión. Esto permite construir modelos muy próximos a las redes reales. El simulador imita la estructura hardware del router en lugar de simplemente implementar su funcionalidad. Para construir rutes más precisos, Sicosys también proporciona distintas versiones de sus elementos internos. Por tanto, los routers pueden ser construidos conectando adecuadamente estos elementos.

Tanto la estructura interna de los routers, como la forma de la red se definen mediante archivos SGML. Esto permite a Sicosys manejar descripciones jerárquicas de los router fácilmente, incrementado así mismo su reusabilidad.

Con el objetivo de hacer la herramienta fácilmente comprensible, extensible y reutilizable, su diseño está orientado a objetos e implementado en C++. La implementación del simulador está basada en una tecnología íntimamente ligada al diseño orientado a objetos: los patrones de diseño. En particular, han sido necesarias cerca de 110 clases, distribuidas a lo largo de aproximadamente 50,000 líneas de código. La portabilidad es muy alta y prácticamente puede ser ejecutado en cualquier plataforma UNIX con un compilador C++ estándar.

SICOSYS ha sido desarrollado teniendo en cuenta la modularidad, versatilidad y conectividad con otros sistemas. De hecho, es posible ejecutar Sicosys con simuladores de sistemas completos como RSIM y SIMOS. Los modelos empleados pretender ser similares en algunos aspectos a las implementaciones hardware manteniendo la complejidad lo más baja posible. Por esta razón se ha considerado adecuado para su interconexión con SystemC.

Los resultados de rendimiento de las redes modeladas están muy próximos a los obtenidos empleando simuladores de hardware, pero con un coste de computación menor.

Con el objetivo de evaluar routers y redes respecto a otras alternativas y hacer el sistema capaz de integrar nuevas extensiones manteniendo una interfaz de usuario homogénea, se ha prestado gran atención en el diseño del simulador a su extensibilidad y la simplicidad de la interfaz con el usuario.

8.3.3.2 Integración de Sicosys en SystemC

La integración de Sicosys en SystemC se ha realizado a distintos niveles. Para ello se ha desarrollado un módulo SystemC que gestiona la ejecución de Sicosys y realiza las tareas de interfaz de integración entre ambos entornos.

La correcta interconexión requiere de la correlación del simulador Sicosys con el kernel de SystemC, especialmente en cuanto a la gestión del tiempo. Para realizar adecuadamente el modelado del sistema, es necesario que los paquetes generados por el SW sean introducidos en el simulador de red en el momento adecuado, para obtener datos correctos de densidad de tráfico y colisiones. Además, la llegada de paquetes a cada nodo debe ser, así mismo, precisa, para poder modelar adecuadamente los efectos de las interrupciones y la información entregada por la red.

Para ello, cada avance de tiempo del Sicosys, debe estar acompañado por un "wait" que bloquee su ejecución hasta que el kernel de SystemC avance el mismo tiempo. Tras analizar Sicosys se ha detectado, que en cada unidad de tiempo, el simulador llama al generador de paquetes (en nuestro caso la interfaz con SystemC) para preguntarle si hay paquetes nuevos. Aprovechando esto, cada vez que el simulador realiza esta pregunta a su interfaz en SystemC, la interfaz realiza el "wait" correspondiente.

Además, Sicosys realiza la evaluación completa del sistema en cada ciclo. Esto significa que ejecuta gran cantidad de cálculos aunque no haya ningún paquete a transferir. Para evitar eso, la interfaz también es capaz de bloquear el simulador durante el tiempo necesario hasta que un nuevo paquete vaya a ser transferido.

El otro trabajo principal de la interfaz de integración es permitir el intercambio de paquetes entre ambos entornos. Para la comunicación con SystemC, la interfaz tiene funciones bloqueantes y no bloqueantes para el envío y la recepción de paquetes. Cuando se quiere enviar un paquete, la interfaz pregunta a Sicosys si hay espacio en el buffer de ese nodo, rechazando el paquete en casa de que la respuesta sea negativa.

Los paquetes a enviar o recibidos se almacenan en la interfaz de integración hasta que Sicosys o la interfaz de red del bus correspondiente los solicitan.

Además la interfaz se encarga de transformar los paquetes SystemC en paquetes Sicosys y viceversa, así como de traducir las direcciones de los nodos. Mientras que en SystemC cada nodo se identifica con un número de 1 a 'n', en Sycosis se nombran conforme a sus coordenadas x, y, z, en función de la estructura de la red.

8.3.3.3 Modificaciones en Sicosys

Para conectar Sicosys con la infraestructura desarrollada sobre SystemC han sido introducidas unas modificaciones mínimas en el propio simulador. Se ha generado un módulo de gestión de interfaz y un generador de paquetes. El generador de paquetes accede al gestor de interfaz cada vez que un nuevo paquete es requerido. El gestor a su vez accede a la interfaz de integración (7.3) para recibir la información de SystemC.

Para integrar estos dos módulos en Sicosys se han introducido dos líneas de código en las fuentes originales. La primera se encuentra en el selector de generadores de paquetes. Este selector elige el generador en función de los archivos de configuración. Por ello se ha añadido una nueva opción de configuración que activa el generador de enlace con SystemC. De esta forma, modificando los archivos de configuración, se consigue que el Sicosys reciba los paquetes de la simulación SystemC.

Además se ha modificado la función de "paquete recibido" de Sicosys, para que cada vez que un paquete llegue a su destino se informe a la interfaz de SystemC y esta active la funcionalidad correspondiente en la interfaz de bus del nodo en cuestión.

9 Exploración del Espacio de Diseño

9.1 Introducción

Como se indicó en la introducción, el proceso de explorar las posibilidades de configuración de un sistema evaluando las distintas posibilidades para encontrar la mejor se denomina exploración del espacio de diseño (Design Space Exploration, DSE). En esta exploración se define el sistema a explorar, y los parámetros configurables. Las técnicas de DSE se basan habitualmente en estrategias de dos pasos.

En primer lugar se selecciona un subconjunto de todas las posibles combinaciones de los parámetros configurables. El subconjunto puede abarcar desde unos pocos puntos escogidos estratégicamente hasta todas las combinaciones. La decisión dependerá del grado de detalle que se desee en la exploración. Este subconjunto se simula, ejecutando una simulación por combinación. De cada simulación se obtienen las métricas de tiempo, potencia, "throughput", etc., que se utilizarán para decidir que configuración es la óptima. Esta etapa se llama diseño de experimentos ("Design of Experiments", DoE). Una vez conocidas las métricas resultantes de unos puntos del espacio de diseño, se aplican técnicas matemáticas para predecir el comportamiento del sistema en el resto de configuraciones no simuladas.

Considerando la gran cantidad de posibles configuraciones de un sistema complejo, es necesario que las herramientas de DSE sean lo más eficientes posible. Para conseguir esa eficiencia, es muy importante que la ejecución de todas las simulaciones requeridas por el DoE sea lo más rápida posible. Dadas la relación velocidad de simulación / precisión obtenida mediante los mecanismos de simulación y estimación presentados anteriormente, estos son una elección ideal para la realización de dichas simulaciones. Por ello se ha preparado la herramienta especialmente para su integración en dicho flujos de DSE.

Para poder ejecutar todas estas simulaciones con la mayor eficiencia posible la herramienta de exploración ha de ser capaz de realizar la tarea automáticamente. Esto significa que la herramienta de exploración ha de ser capaz de configurar automáticamente la simulación. Además, con objeto de reducir el tiempo, es recomendable que no sea necesario realizar tareas adicionales como compilaciones extra. Si el sistema es completamente automático se reduce no solo el tiempo total de exploración, sino sobre todo el tiempo de ingeniero requerido para realizar dicha exploración.

Para poder obtener este automatismo es necesario considerar los parámetros a configurar en dos grupos. En un primer caso, en una plataforma configurable, determinadas características de los componentes pueden ser ajustadas para obtener un resultado óptimo. Parámetros comúnmente configurable son los tamaños de memoria, tamaños de cache, prioridades en los accesos a recursos compartidos o frecuencia de funcionamiento de los componentes. La exploración de estos parámetros es la que se realiza habitualmente en procesos de DSE. La configuración de una simulación para alterar alguno de los parámetros de funcionamiento de los componentes es una tarea ampliamente resuelta.

El segundo nivel de configuración se refiere a la propia estructura del sistema. Añadir o quitar componentes, modificar sus interconexiones o cambiar la asignación de las tareas a recursos son también opciones de configuración a explorar. Sin embargo, estos parámetros no son explorados por las técnicas existentes. Esto se debe a dos causas. En primer lugar se necesitan herramientas capaces de construir dinámicamente (durante ejecución) el modelo de sistema. En segundo lugar se necesita definir mecanismos para que una herramienta de exploración sea capaz de configurar la arquitectura del sistema.

El trabajo propuesto se basa en proporcionar técnicas de modelado y simulación que permitan resolver estos problemas. Las técnicas de selección de los experimentos a simular y de interpolación y otras operaciones matemáticas quedan fuera del ámbito trabajo.

En la trabajo realizado durante la tesis se han desarrollado las técnicas que permite que herramientas de exploración convencionales sea capaz de explorar ambos niveles de configurabilidad. Las técnicas hacen uso de las soluciones presentadas en los capítulos anteriores para generar la infraestructura de simulación necesaria. El resultado se ha integrado con éxito con los exploradores m3explorer (Politécnico de Milán) y modeFrontier (ESTECO©).

9.2 Integración del simulador y la herramienta de exploración

Las herramientas convencionales utilizadas para la exploración del diseño están pensadas para barrer las combinaciones de parámetros válidos y seleccionar las que presentan los mejores resultados. En general estas herramientas manejan los parámetros como valores puramente numéricos, sin aplicar informaciones adicionales del efecto producido en el sistema por la modificación de un parámetro, que ayuden a guiar la búsqueda. Se utilizan únicamente simulaciones y extrapolaciones para realizar las selecciones.

Por esta razón, la utilización de estas herramientas para la exploración de arquitecturas y configuraciones complejas debe considerar que la herramienta de exploración no posee ningún tipo de conocimiento electrónico o inteligencia artificial en ese sentido. Así, estas herramientas no pueden encargarse de construir un sistema eligiendo componentes y construyendo comunicaciones.

La solución desarrollada en la tesis permite a las herramientas de exploración analicen posibilidades arquitecturales únicamente mediante la exploración de parámetros numéricos o cadenas de caracteres. Para ello se han desarrollado dos elementos complementarios. En primer lugar se ha definido una serie de archivos XML que permiten la descripción de los sistemas de forma que sean fácilmente explorables por las herramientas de exploración. Además se ha desarrollado una extensión a la infraestructura de modelado presentada anteriormente que permite la construcción en tiempo de ejecución de los modelos de sistema a simular.

Para la integración del simulador con otras herramientas de exploración se necesitan formas de comunicación entre ellos. Para esta tarea se ha utilizado la definición de archivos

XML realizada en el proyecto europeo MULTICUBE. Estos archivos constan de un archivo que define todas las posibles combinaciones de los parámetros a explorar, un archivo para definir una combinación de parámetros para cada exploración y un archivo para entregar los resultados de rendimiento de cada simulación a la herramienta de exploración. En definitiva se utilizan cuatro tipos de archivos XML en la exploración. Esto se puede ver en la figura siguiente.

Las herramientas de exploración modeFRONTIER y M3Explorer han sido adaptadas por sus proveedores (Esteco y el Politécnico de Milán respectivamente) para utilizar los archivos XML de interconexión, de tal forma que el simulador y la técnica de definición de arquitecturas modificables son directamente compatibles con ambas herramientas.

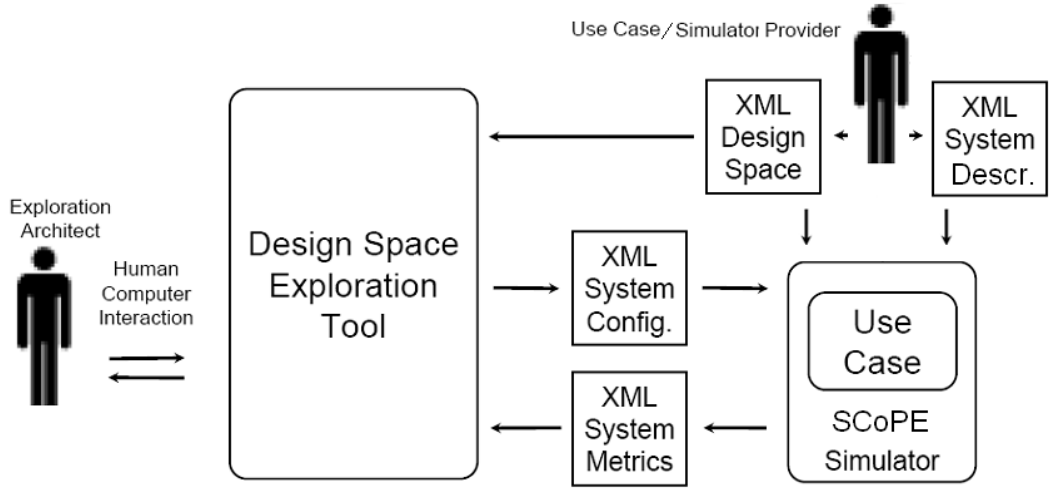


Figura 9-1: Integración de la herramienta de exploración y el simulador

9.3 Archivos XML

9.3.1 Archivo de descripción del sistema a simular

Para permitir la exploración automática en busca de las soluciones más adecuadas es necesario definir un medio de expresar dichas arquitecturas de manera configurable. Con el fin de realizar una propuesta que pueda ser aplicable a otros entornos y herramientas, se ha definido una solución descriptiva lo más potente y sencilla posible.

Para ello se han definido una serie de etiquetas XML que permiten describir completamente un sistema. Además se han definido tres tipos de reglas que permiten parametrizar dichas descripciones de tal forma que el diseñador pueda describir todas las opciones arquitecturales del diseño en curso. Empezaremos describiendo las opciones normales de descripción de los sistemas para finalmente presentar las opciones de configurabilidad.

9.3.1.1 Cláusulas XML para descripción de sistemas fijos

Aunque existen algunos mecanismos XML estándar para describir algunas partes del sistema, estos no están preparados para la descripción de todos los niveles del sistema de una forma modificable. Por ejemplo IP-XACT es uno de los estándares XML más utilizados en sistemas electrónicos. Sin embargo este lenguaje esta desarrollado para la descripción de los componentes HW del sistema y sus interconexiones.

La descripción de elementos SW como tareas de usuario, características de sistemas operativos, middleware o controladores de dispositivo no están contemplados. Además no esta diseñado para permitir una parameterización completa de los sistemas. Únicamente algunas características de los componentes del sistema pueden ser parametrizables. Por esa razón se ha definido toda una estructura XML para la descripción de sistemas electrónicos.

Cada una de las descripciones de las cláusulas XML presentadas a continuación contiene los siguientes elementos:

- Nombre: Nombre de la cláusula
- Descripción: Descripción de la utilidad de la cláusula
- Atributos: Lista de posibles atributos que pueden ser utilizados dentro de la cláusula. Estos atributos se dividen en dos grupos:
  - Atributos generales: Son atributos que pueden ser asociados a cualquier clausula de ese tipo
  - Atributos específicos: Son atributos que solo pueden ser definidos en combinación con determinados valores de otros atributos
- Estructura: Ejemplo de la clausula XML en cuestión describiendo las cláusulas que pueden ser incluidas dentro de la misma.

9.3.1.1.1 Estructura general del archivo

El archivo XML de descripción del sistema se divide en cuatro apartados:

- Cláusulas encargadas de describir la arquitectura HW
- Cláusulas encargadas de describir la infraestructura SW
- Cláusulas encargadas de describir el SW de usuario
- Cláusulas con parámetros de simulación

Como consecuencia, la estructura general del archivo XML es la siguiente:

```
< Description >
        < HW_Platform >
                < HW_Components >
                < HW_Architecture >
                < Computing_Groups >
        < /HW_Platform >
```

```
        < SW_Platform >
                < SW_Components >
                < SW_Architecture >
        < /SW_Platform >

        < Application >
                < Functionality >
                < Allocation >
        < /Application >

        < Simulation >
                < Implementation >
        < /Simulation >
< /Description >
```

9.3.2 Descripción de la plataforma HW: <HW Platform>

Descripción: Esta cláusula contiene toda la descripción de una plataforma HW, tanto sus componentes como sus interconexiones e interrelaciones.

Atributos:

- name [Opcional]: Nombre que identifica la plataforma o sub-plataforma HW descrita (Ver <Implementación> para más detalles)

Estructura:

```
< HW_Platform name="name" >
        < HW_Components >
        < HW_Architecture >
        < Computing_Groups >
< /HW_Platform >
```

9.3.2.1 HW_Components

Agrupa la descripción general de todos los tipos de componentes que pueden ser utilizados en la descripción de la plataforma HW.

Atributos:

- name [Opcional]: Nombre que identifica la plataforma HW

Estructura:

```
< HW_Components name="name" >
        < HW_Component >
        < HW_Component >
        ...
< /HW_Components >
```

9.3.2.2 HW_Component

Descripción de un tipo de componente HW. Contiene la descripción de todos los valores por defecto que serán aplicados a cada instancia de componente de ese tipo que se cree al describir la plataforma HW.

Atributos Generales:

- name [Obligatorio]: Nombre del HW_Component
- category [Obligatorio]: Tipo de componente.

  - Valores posibles: processor, memory, icache, dcache, dma, bus, network, net_if, bridge and as_hw

- mem_size [Opcional]: Tamaño de memoria asociado al componente HW. Es la memoria que se reserve en el mapa de memoria y que puede ser accedida desde el canal de comunicaciones al que este enlazado
- frequency [Opcional]: Frecuencia del componente.
- width [Opcional]: Anchura de las líneas de comunicación del componente. En elementos de comunicación como buses indica el número de líneas de datos que contiene el canal. En componentes HW indica el tamaño de palabra que el componente es capaz de aceptar en una transferencia.
- area [Opcional]: Área HW que se estima requeriría el componente en caso de ser instanciado.
- static_power [Opcional]: Potencia media consumida por el componente mientras no reciba eventos especiales. En procesadores indica la potencia media por instrucción en funcionamiento estándar. En periféricos la potencia media mientras no se reciban accesos desde el bus o la red. En caches el parámetro representa la energía consumida por cada acierto.
- read_energy [Opcional]: Energía consumida por el componente cuando se realiza un evento de lectura. En caches indica la energía de miss. Es independiente del tamaño de la lectura.
- write_energy [Opcional]: Energía consumida por el componente cuando se recibe un evento de escritura. En caches indica el consumo por la modificación de un dato desde el procesador. Incluye el consumo del HW de control de coherencia si existe. Es independiente del tamaño de la escritura
- read_size_energy [Opcional]: Energía consumida en un acceso de lectura por cada unidad de información. La energía total depende del tamaño de la información leída.
- write_size_energy [Opcional]: Energía consumida en un acceso de escritura por cada unidad de información. La energía total depende del tamaño de la información escrita.

Atributos Específicos :

`Categoría: processor`

- proc_type [Opcional]: Tipo de procesador

`Categoría: memory`

- type [Opcional]: Tipo de memoria ('RAM', 'ROM')

`Categoría: network`

- x_size [Opcional]: Número de nodos en el eje x de una red tipo mesh
- y_size [Opcional]: Número de nodos en el eje y de una red tipo mesh

Categoría: as_hw

- activation_type [Opcional]: Indica si el componente se comporta como maestro o esclavo

Estructura:

```
< HW_Component name="proc_1" category="processor" proc_type="arm926t" ... >
      < HW_Component>
      ...
</HW_Component>
```

9.3.2.3 HW_Architecture

Contiene la descripción de una plataforma o sub-plataforma HW, con sus componentes e interconexiones. Contiene las cláusulas de instanciación de componentes y conexiones.

Atributos:

- name [Opcional]: Nombre que identifica la descripción de plataforma (Ver Implementation)

Estructura:

```
< HW_Architecture name="name" >
        < HW_Instance >
        < HW_Connection >
        ...
< /HW_Architecture >
```

9.3.2.4 HW_Instance

Instancia de un componente HW que forma parte de la plataforma HW. Cuando una instancia está incluida en otra se considera que la interior esta conectada a la exterior. En canales de comunicación como buses se recomienda describir el bus como instancia englobante, y los elementos conectados al bus como instancias incluidas. Esto facilita la visión de plataforma.

Atributos Generales:

- name [Obligatorio]: Nombre de la instancia HW
- component [Obligatorio]: Nombre del componente HW instanciado y que se usa para tomar los valores por defecto. (Ver <HW_Component>)
- name [Obligatorio]: Nombre del HW_Component
- category [Obligatorio]: Tipo de componente.

  - Valores posibles: processor, memory, icache, dcache, dma, bus, network, net_if, bridge and as_hw

- mem_size [Opcional]: Tamaño de memoria asociado al componente HW. Es la memoria que se reserve en el mapa de memoria y que puede ser accedida desde el canal de comunicaciones al que este enlazado
- frequency [Opcional]: Frecuencia del componente.
- width [Opcional]: Anchura de las líneas de comunicación del componente. En elementos de comunicación como buses indica el número de líneas de datos que contiene el canal. En componentes HW indica el tamaño de palabra que el componente es capaz de aceptar en una transferencia.
- area [Opcional]: Area HW que se estima requeriría el componente en caso de ser instanciado.
- static_power [Opcional]: Potencia media consumida por el componente mientras no reciba eventos especiales. En procesadores indica la potencia media por instrucción en funcionamiento estándar. En periféricos la potencia media mientras no se reciban accesos desde el bus o la red. En caches representa la energía consumida por acierto.
- read_energy [Opcional]: Energía consumida por el componente cuando se realiza un evento de lectura. En caches indica la energía de miss. Es independiente del tamaño de la lectura.
- write_energy [Opcional]: Energía consumida por el componente cuando se recibe una evento de escritura. En caches indica el consumo por la modificación de un dato desde el procesador. Incluye el consumo del HW de control de coherencia si existe. Es independiente del tamaño de la escritura
- read_size_energy [Opcional]: Energía consumida en un acceso de lectura por cada unidad de información. La energía total depende del tamaño de la información leida.
- write_size_energy [Opcional]: Energía consumida en un acceso de escritura por cada unidad de información. La energía total depende del tamaño de la información escrita.
- start_addr [Opcional]: Dirección del mapa de memoria asignada a la instancia del componente HW. Indica el comienzo de la zona de memoria.
- irq [Opcional]: Número de interrución que genera el componente.
- local_id [Opcional]: Identificador numérico del componente. En interfaces de red indica la dirección mac.

Atributos Específicos :

`Categoría: network`

- x_size [Opcional]: Número de nodos en el eje x de una red tipo mesh
- y_size [Opcional]: Número de nodos en el eje y de una red tipo mesh

Estructura:

```
< HW_Instance name="my_net_if" component="net_if_1" start_addr="0x8000000"
irq="5"  port="1" >
        < HW_Instance >
        < HW_Connection >
        ...
</ HW_Instance>
```

9.3.2.5 HW_Connection

Debe aparecer siempre dentro de una instancia HW. Indica que la instancia HW referida en la cláusula de conexión debe conectarse con la instancia HW donde la cláusula está incluida.

Atributos Generales :

- name [Obligatorio]: Nombre de la conexión
- instance [Obligatorio]: Nombre de la instancia HW a conectar

Estructura:

```
< HW_Connection name="net_if_1_connection" instance="my_net_if" / >
```

9.3.2.6 Computing_Groups

Los grupos de cómputo se usan para identifica componentes que trabajan de manera cooperativa. Es especialmente importante al indicar los procesadores que funcionan cooperativamente bajo el control de un mismo sistema operativo. La cláusula incluye todos los grupos de cómputo definidos en el sistema.

Atributos:

- name [Opcional]: Nombre de la lista de grupos de cómputo (Ver <Implementation>)

Estructura:

```
< Computing_Groups name="name" >
        < Computing_Group >
        < Computing_Group >
        ...
< /Computing_Groups >
```

9.3.2.7 Computing_Group

Identificación de un grupo de componentes HW que trabajan cooperativamente

Atributos:

- name [Opcional]: Nombre del grupo de cómputo (Ver <Implementation>)

Estructura:

```
< Computing_Group name="name" >
```

```
        < Computing_Element >
        < Computing_Element >
        ...
< /Computing_Group >
```

9.3.2.8 Computing_Element

Componente HW que se encuentra englobado en un grupo de cómputo

Atributos:

- name [Opcional]: Nombre de la instancia HW incluida en el grupo de cómputo (Ver <Implementation>)

Estructura:

```
< Computing_ Element name="name" />
```

9.3.3 Descripción de la plataforma SW: <SW Plataform>

Contiene la descripción de la plataforma SW. Engloba a los sistemas operativos, middleware y otros componentes de la infraestructura SW.

Atributos:

- name [Opcional]: Name to identify the SW platform (See Implementation)

Estructura:

```
< SW_Platform name="name" >
        < SW_Components >
        < SW_Architecture >

< /SW_Platform >
```

9.3.3.1 SW_Components

Lista de componentes que componen la plataforma.

Atributos:

- name [Opcional]: Nombre de la lista de componentes SW (See <Implementation>)

Estructura:

```
< SW_Components name="name" >
        < SW_Component >
```

```
        < SW_Component >
        ...
< /SW_Components >
```

9.3.3.2 SW_Component

Descripción de un tipo de componente de la plataforma SW. Proporciona los parámetros por defecto que se aplicarán a cualquier instancia de este componente.

Atributos Generales:

- name [Obligatorio]: Nombre del componente SW
- type [Obligatorio]: Tipo de componente SW.

  - Valores posibles: OS, middleware

Estructura:

```
< SW_Component name="OS_1" type="OS"/ >
```

9.3.3.3 SW_Architecture

Conjunto de instancias de componentes que componen una plataforma SW.

Atributos:

- name [Opcional]: Nombre de la plataforma SW (Ver <Implementation>)

Estructura:

```
< SW_Architecture name="name" >
        < SW_Instance >
        < SW_Instance >
        ...
< /SW_Architecture >
```

9.3.3.4 SW_Instance

Instancia de componente SW que forma parte de la plataforma SW.

Atributos Generales:

- name [Obligatorio]: Nombre de la instancia de componente SW
- component [Obligatorio]: Nombre del componente SW genérico que es instanciado
- hw_resource [Obligatorio]: Nombre de la instancia SW donde el recurso HW va a ser ejecutado. Puede

ser una HW_Instance para un solo componente o un Computing_Group en caso de que el elemento SW vaya a ser ejecutado por más de un componente HW.

Estructura:

```
< SW_Instance name="OS_1" type="OS"/ >
```

9.3.4 Descripción del SW de aplicación : <Application>

Contiene la descripción de la aplicación SW que ha de ejecutar en el sistema.

Atributos:

- name [Opcional]: Nombre que identifica la aplicación descrita (Ver <Implementation>)

Estructura:

```
< Application name="name" >
        < Functionality >
        < Allocation >
        ...
< /Application >
```

9.3.4.1 Functionality

Contiene la descripción de todos los componentes SW que construyen la aplicación SW completa.

Atributos:

- name [Opcional]: Nombre que identifica el componente SW de aplicación (Ver Implementation)

Estructura:

```
< Functionality name="name" >
        < Exec_Component >
        < Exec_Component >
        ...
< /Functionality >
```

9.3.4.2 Exec_Component

Descripción de un componente SW que formará parte de la aplicación SW al ser instanciado. Contiene los valores por defecto para todas las instancias de ese tipo.

Atributos Generales:

- name [Obligatorio]: Nombre del componente SW
- category [Obligatorio]: Tipo de componente.

  - Valores posibles: SW, driver and taskload

Atributos Específicos:

`Categoría: SW and driver`

- function [Obligatorio]: Nombre de la función SW principal que arranca la tarea. Normalmente se denomina "main", pero para permitir múltiples tareas en el mismo sistema se ha de asociar un nombre distinto a cada una.
- file [Opcional]: Archivo donde se encuentra dicha función. Si es una librería dinámica el simulador intentará cargarla de dicha librería

`Categoría: taskload`

- compute_time [Obligatorio]: Tiempo que la tarea requiere para ser ejecutada. En tareas periódicas indica el tiempo de cada ejecución
- period [Opcional]: Periodo de la tarea
- data_size [Opcional]: Cantidad de datos transferidos por la tarea. En tareas periódicas es el tamaño de transferencias por cada ejecución.

Estructura:

```
< Exec_Component name="task_1" category="SW" function="my_function"/ >
```

9.3.4.3 Allocation

Contiene todas las instancias de tareas SW del sistema.

Atributos:

- name [Opcional]: Nombre de la lista de tareas SW (Ver <Implementation>)

Estructura:

```
< Allocation name="name" >
        < Exec_Instance >
        < Exec_Instance >
        ...
< /Allocation >
```

9.3.4.4 Exec_Instance

Instancia de una tarea SW del sistema

Atributos Generales:

- name [Obligatorio]: Nombre de la instancia de tarea SW
- component [Obligatorio]: Nombre del componente SW instanciado
- os [Obligatorio]: Nombre de la instancia de componente de plataforma SW sobre el que la tarea va a ser ejecutada.
- arguments [Opcional]: Argumentos que la función principal de la tarea va a recibir al arrancar
- policy [Opcional]: Política de la tarea
- priority [Opcional]: Prioridad de la tarea

Estructura:

```
< Exec_Instance name="my_task" component="task_1" resource="node0"
arguments="-s -t -f file"/ >
```

9.3.5 Descripción de los parámetros de la simulación: < Simulation>

Contiene los parámetros generales de la simulación

Atributos:

- time [Opcional]: Tiempo máximo de la simulación
- backtrace [Opcional]: Activación de las facilidades para el depurado del sistema
- end_as_sw [Opcional]: Indica que la simulación finalice cuando todas las tareas SW hayan terminado

Estructura:

```
< Simulation time="200 ms" end_as_sw="1" >
        < Implementation >
< /Simulation >
```

9.3.6 Cláusulas XML para descripción de sistemas configurables

9.3.6.1 Implementation

Indica cuales de las descripciones anteriores del archivo XML deben ser utilizadas. En la cláusula se ha de indicar la arquitectura HW, la plataforma SW y las tareas SW seleccionadas para construir el sistema.

Si no se ha especificado ninguna cláusula para alguna de las tres partes anteriores, se utilizarán todas las descripciones de ese tipo disponibles en el archivo.

Atributos:

- HW_Platform [Opcional]: Nombre de la cláusula HW_Platform seleccionada para la generación del modelo de sistema
- HW_Components [Opcional]: Nombre de la cláusula HW_Components seleccionada para la generación del modelo de sistema
- HW_Architecture [Opcional]: Nombre de la cláusula HW_Architecture seleccionada para la generación del modelo de sistema
- Computing_Groups [Opcional]: Nombre de la cláusula Computing_Groups seleccionada para la generación del modelo de sistema
- SW_Platform [Opcional]: Nombre de la cláusula SW_Platform seleccionada para la generación del modelo de sistema
- SW_Components [Opcional]: Nombre de la cláusula SW_Components seleccionada para la generación del modelo de sistema
- SW_Architecture [Opcional]: Nombre de la cláusula SW_Architecture seleccionada para la generación del modelo de sistema
- Application [Opcional]: Nombre de la cláusula Application seleccionada para la generación del modelo de sistema
- Functionality [Opcional]: Nombre de la cláusula Functionality seleccionada para la generación del modelo de sistema
- Allocation [Opcional]: Nombre de la cláusula Allocation seleccionada para la generación del modelo de sistema

Estructura:

```
< Implementation HW_Platform="HW_Plat_1" SW_Platform="SW_Plat_1"
Allocation="Alloc_2"/ >
```

9.3.6.2 Repeat

Permite indicar que una o varias cláusulas se consideren un número "n" de veces. El resultado es equivalente a considerar que esas cláusulas estén repetidas en el archivo XML ese número de veces. Para identificar el número de repetición y generar nombres distintos para los elementos repetidos se define un índice. El índice se identifica con una letra de la 'a' a la 'z'. Cuando se utiliza la combinación "%i", siendo "i" la letra correspondiente al índice, la combinación se reemplaza por el valor del índice en esa repetición. Por ejemplo, en la tercera repetición de una cláusula de índice j, la cadena "obj_%j", se reemplaza por "obj_3".

Las cláusulas "repeat" se pueden encadenar, de forma que son visibles los índices de todas las cláusulas anidadas, siempre y cuando no se repita la letra del índice en ninguna de ellas.

Atributos:

- number [Obligatorio]: Numero de veces que se han de repetir las cláusulas internas
- index [Opcional]: Letra que identifica el índice, y que es sustituida por el número de repetición cuando va precedida de '%'
- init [Opcional]: Número que indica el valor del índice en la primera repetición. Por defecto su valor es '0'.

Estructura:

```
< Repeat number="3" index="i" init="1">
        < Component_type name="name%i" >
        < Repeat number="_repeat_times" index="j" >
                < Component_type name="name_%i_%j" >
        < /Repeat >
< /Repeat >
```

```
<HW_Platform>
    <HW_Components>
        <HW_Component category="bus" name="AMBA" frequency="200" />
        <HW_Component category="processor" name="arm926t" frequency="200"/>
        <HW_Component category="memory" name="Memory"
      mem_size="500000K"  frequency="200"  mem_type="RAM" />
    </HW_Components>
    <HW_Architecture>
        <HW_Instance component="AMBA" name="my_bus" >
            <HW_Instance component="arm926t" name="my_proc" />
             <HW_Instance component="Memory" name="my_memory"
                 start_addr="0x80000000" />
        </HW_Instance>
    </HW_Architecture>
</HW_Platform>
<Application>
    <Functionality>
        <Exec_Component name="hello" category="SW" function="hello_main" />
    </Functionality>
     <Allocation>
         <Exec_Instance name="Hello_world" component="hello"
             processor="my_proc"/>
    </Allocation>
 </Application>
```

Figura 9-2: Descripción XML de un sistema utilizando las reglas anteriores.

9.3.6.3 Mecanismos de descripción de arquitecturas modificables

Para permitir el uso de la herramienta en procesos de exploración del espacio de diseño el mecanismo de descripción de los sistemas mediante archivos XML permite una gran

capacidad de configurabilidad. Sistemas altamente configurables pueden ser descritos mediante estos archivos.

La base para permitir la configurabilidad es la posibilidad de asignar a cualquier atributo de las cláusulas XML un valor variable en lugar de un dato fijo. Para asignar valores variables se asigna al atributo un identificador en lugar de un valor. Estos identificadores son resueltos en otro archivo XML que proporciona los pares identificador-valor para cada simulación. De esta forma se separan los parámetros de configuración de la descripción de la plataforma.

Aplicando estos identificadores a las cláusulas presentadas anteriormente se obtienen tres posibilidades de descripción de sistemas configurables.

9.3.6.3.1 Descripción de componentes configurables

La primera opción es definir componentes configurables en el sistema. De esta forma cualesquiera de las características de todos componentes pueden ser definidas como variables en lugar de asignarles valores fijos. Características como la frecuencia, la latencia, o el tamaño pueden ser configurables en los componentes del sistema. Ejemplos de esto pueden ser la frecuencia de procesador o el bus, la latencia de la memoria o el tamaño de la memoria o la cache.

9.3.6.3.2 Componentes opcionales

La segunda cualidad que posee la técnica de descripción por XML propuesta es la posibilidad de definir componentes opcionales en el sistema. Estos son componentes que pueden formar parte del sistema o no. Cuando existen dos componentes distintos que pueden realizar tareas equivalentes o se quiere estudiar la utilización de varios elementos iguales para aumentar la concurrencia en el sistema es necesario poder añadir o quitar componentes en cada simulación. Esto sucede cuando queremos estudiar varias opciones de bus o el número óptimo de procesadores que debemos poner en paralelo.

Esta es una cualidad que no se proporciona en los sistemas de modelado convencionales, ya que los simuladores requieren que el modelo de plataforma esté fijo para poder realizar la simulación. La utilización de componentes opcionales en el modelo de plataforma es posible ya que el modelo se construye en tiempo de ejecución, una vez el archivo de configuración XML se ha leído. Las técnicas de modelado del sistema presentadas en los capítulos anteriores hacen esto posible.

Para definir componentes o subsistemas opcionales se debe utilizar la cláusula “Repeat”. Si definimos el atributo “Number” como un atributo variable, el número de veces que se instancian los componentes afectados será variable. Con “Number=0” podemos hacer que el componente no forme parte del sistema. Esto evita su instanciación en la plataforma virtual, con lo que se evitan comportamientos indeseados a la vez que se elimina la sobrecarga asociada a su modelado. Si queremos estudiar el número óptimo de procesadores o co-

procesadores para nuestro sistema bastará con hacer variar Number del mínimo al máximo previsto. Esta posibilidad se muestra en la figura siguiente:

```
<HW_Platform>
    <HW_Components>
        ...
    </HW_Components>
    <HW_Architecture>
        <HW_Instance component="AMBA" name="my_bus" >
            <repeat numer=”CPUS” index=”i” init=”1”>
              <HW_Instance component="arm926t" name="my_proc[%i]" />
            </repeat>
          <HW_Instance component="Memory" name="my_memory"/>
        </HW_Instance>
    </HW_Architecture>
</HW_Platform>
<Application>
    <Functionality>
        <Exec_Component name="hello" category="SW" function="hello_main" />
    </Functionality>
    <Allocation>
        <repeat numer=”CPUS” index=”j” init=”1”>
          <Exec_Instance name="task[%i]" component="hello"
              processor="my_proc[%j]"/>
        </repeat>
    </Allocation>
</Application>
```

Figura 9-3: Descripción XML de plataforma con múltiples procesadores

9.3.6.3.3 Definición y selección de subsistemas completos

El tercer mecanismo de descripción de sistemas configurables se basa en la posibilidad de describir múltiples configuraciones para los sistemas o subsistemas y posteriormente elegir cual de ellas queremos probar en cada simulación. Se pueden definir múltiples plataformas HW y múltiples sub-sistemas SW. En cada simulación la plataforma HW y el subsistema SW se construyen instanciando los subsistemas SW y HW seleccionados. Varios subsistemas HW o SW pueden ser elegidos simultáneamente.

Los subsistemas HW y SW se definen asignando nombres a las cláusulas “HW_Architecture” y “Allocation”. Posteriormente en la cláusula “Implementation” se indica que arquitecturas HW y SW han de ser cargadas. Los nombres de las arquitecturas a cargar pueden definirse mediante identificadores que toman valor en el archivo auxiliar de pares identificador-valor, como el resto de parámetros configurables.

```
<HW_Platform>
        <HW_Components>
            ...
        </HW_Components>
        <HW_Architecture name=”arch1”>
            <HW_Instance component="AMBA" name="my_bus" />
        ...
        </HW_Architecture>
        <HW_Architecture name=”arch2”>
            <HW_Instance component="NoC" name="my_noc" />
        ...
        </HW_Architecture>
    </HW_Platform>
    <Application>
        ...
    </Application>
    <Simulation>
        < Implementation HW_Architecture="arch1" / >
    </Simulation>
```

Figura 9-4: Descripción XML de diferentes arquitecturas HW

9.3.6.4 Construcción automática de plataformas virtuales

Para que sea posible la utilización de las reglas de descripción de sistemas configurables es necesario que el sistema pueda ser generado automáticamente. De esta forma es posible una eficiente exploración de todas las combinaciones posibles para detectar las soluciones óptimas.

Además, para minimizar los tiempos de simulación la construcción de la plataforma se realiza completamente durante la ejecución de la simulación. De esta forma se evitan tiempos adicionales de re-compilación por cada simulación.

La instanciación de componentes HW es posible ya que todos los modelos de componentes HW en SCoPE derivan de una clase base común. De esta forma la herramienta puede realizar las operaciones necesarias sin necesidad de saber en cada momento el tipo específico de componente que se está instanciando.

La utilización de esta clase base también permite la conexión automática de los modelos HW para construir la plataforma virtual. Esta clase base proporciona una interfaz de comunicación basada en las funciones “transport” de la especificación oficial TLM en SystemC. Dado que la implementación de dichas funciones está integrada en la clase base, todas las comunicaciones de componentes son compatibles y directamente interconectables. Tan solo es necesario asegurarse de que cada enlace tenga un iniciador y un receptor.

La clase base también se encarga de realizar las llamadas necesarias para construir los mapas de memoria y las tablas de decodificación de comunicaciones de los elementos de comunicación, como buses y redes.

Para la generación de la infraestructura SW SCoPE proporciona los modelos de sistema operativo necesarios. Estos modelos contienen una capa de abstracción HW con las funciones necesarias para conectar la simulación nativa con la interfaz de procesador correspondiente, como se explicó en el capítulo anterior.

Para la carga dinámica de tareas se utilizan las funciones de manejo de librerías dinámicas. A través de las funciones dlopen y dlsym se obtienen los punteros a función correspondientes con los nombres de las funciones de entrada a cada tarea SW del sistema. Una vez obtenidos los punteros a función solamente es necesario llamar a la función encargada de generar un proceso en modelo de sistema operativo correspondiente.

9.3.7 Archivo de asignación de los parámetros configurables

En las descripciones de sistemas configurables, las opciones de configuración se fijan para cada simulación, asignando identificadores a los parámetros configurables en lugar de valores fijos. En cada simulación de sistema es necesario dar un valor a esos parámetros. El archivo XML de configuración del sistema se encarga de proporcionar esta información. El archivo es una sucesión de pares identificador – valor, que debe contener todos los parámetros del sistema a simular.

El nombre del archivo de configuración se especifica al lanzar la simulación mediante las reglas –xsc o --xml-system-configuration

La estructura del archivo es la siguiente:

```
< simulator_input_interface  xmlns="http://www.multicube.eu/"
version="1.3">
        < parameter name="mem_size" value="256" />
        < parameter name="num_proc" value="3"   />
        ...
< /simulator_input_interface >
```

9.3.8 Generación de métricas

9.3.8.1 Métricas reportadas por SCoPE

La interfaz XML de SCoPE tiene una serie de métricas predefinidas que pueden ser reportadas al final de la simulación. Las métricas consideradas son las siguientes:

- Latencia (seg): Tiempo necesario para ejecutar la aplicación en el sistema.
- Ciclos ejecutados: Número de ciclos requeridos para ejecutar la aplicación. (Solo en sistemas con un único reloj global)

- Numero de instrucciones (inst): Cuenta del número total de instrucciones máquina ejecutadas sumando todos los procesadores. Se consideran únicamente las instrucciones del código de aplicación. El sistema operativo y las instrucciones de las tareas "idle" no se añaden a la cuenta.
- Ciclos por instrucción (c/inst): Número medio de ciclos de reloj requeridos pos instrucción durante la ejecución.
- Instrucciones por ciclo (Inst/c): Número medio de instrucciones ejecutadas en cada ciclo de reloj por el sistema.
- Millones de instrucciones por segundo (MIPS): Numero de instrucciones ejecutadas por cada segundo en el sistema destino, en millones.
- Area del sistema (mm^2): Area total ocupada por el sistema simulado.
- Tasa de aciertos (%): Porcentaje de aciertos en cache respecto al número de accesos totales.
- Ciclos de parada en memoria (c): Número de ciclos que los procesadores están parados debido al tiempo asociado a los fallos de cache.
- Tiempo de acceso medio a memoria, AMAT (seg): Tiempo medio que tarda un procesador en acceder a memoria. Da una medida de la sobrecarga del bus.
- Ancho de banda del bus (bits/seg): Tasa media de transferencia de datos a través de los buses del sistema
- Ancho de banda de la red (bits/seg): Tasa media de transferencia de datos a través de la red del sistema.
- Latencia de transporte (seg): Tiempo medio necesario para transferir un paquete completo por la red.
- Energía (Julios): Energía consumida por el sistema para finalizar la aplicación.
- Potencia (Watios): Potencia media consumida por el sistema durante la ejecución de la aplicación.

9.3.8.2 Archivo de definición de métricas

El archivo de definición de métricas indica a SCoPE que métricas deben ser analizadas durante la simulación. Estas métricas son reportadas al final de la misma en el archivo de salida de la simulación.

El nombre del archivo de configuración se especifica al lanzar la simulación mediante las reglas -xmd file_name or --xml-metric_definition file_name

La estructura del archivo es la siguiente:

```
< system_metrics >
        < metric name="Execution_cycles" type="integer" unit="cycle"/>
        < metric name="Power_consumption" type="float"  unit="W"    />
        ...
< /system_metrics >
```

9.3.8.3 Archivo XML de salida

El archivo de salida contiene los valores de las métricas obtenidas por SCoPE durante la simulación.

El nombre del archivo de configuración se especifica al lanzar la simulación mediante las reglas -xof file_name or --xml-system-metrics file_name

La estructura del archivo es la siguiente:

```
< simulator_output_interface xmlns="http://www.multicube.eu/"
version="1.3">
      <system_metric name="latency" value="value"/>
      <system_metric name="instruction_count" value="value"/>
      ...
< /simulator_output_interface>
```

9.4 Ejemplo de aplicación

Para una mejor comprensión de la utilización de la interfaz XML se muestra a continuación un ejemplo de aplicación sencillo. Para ello se generará un sistema que ejecute un código SW que imprima un "hello World". En la distribución de la herramienta y en el manual se pueden encontrar más ejemplos.

Para simular el sistema hemos de tener:

- El código SW
- La descripción del sistema
- El archivo indicando las métricas a reportar

Adicionalmente es necesario disponer de los modelos del HW de aplicación específica. En este ejemplo no hay ningún HW especial, por lo que no es necesario.

El código SW generador del "Hello World" es el siguiente. Como se puede comprobar es un código POSIX convencional.

```
#include "stdio.h"
int hello_main(int argc, char **argv){
        printf("Hello world\n");
        return 0;
}
```

Para generar el archivo de descripción del sistema detallaremos la plataforma HW, la infraestructura SW y el código de aplicación.

La plataforma HW del ejemplo tendrá tres componentes: un procesador, un bus y una memoria. El XML de descripción de estos tres tipos de componentes es:

```
<HW_Components>
        <HW_Component category="bus" name="AMBA" frequency="200" />
        <HW_Component category="processor" name="arm926t" frequency="200" />
        <HW_Component category="memory" name="Memory" mem_size="500000K" frequency="200"
        mem_type="RAM" />
</HW_Components>
```

Para generar la plataforma HW debemos instanciar los componentes, indicando que el procesador y la memoria estarán conectados al bus. Para eso los instanciaremos dentro del ámbito de la cláusula de instanciación del bus:

```
<HW_Architecture>
        <HW_Instance component="AMBA" name="my_bus" >
                <HW_Instance component="arm926t" name="my_proc" />
                <HW_Instance component="Memory" name="mem" start_addr=”0x80000000” />
        </HW_Instance>
</HW_Architecture>
```

Una vez creada la plataforma HW debemos generar la infraestructura SW. Para ello declararemos un sistema operativo y le diremos que ejecute sobre el procesador de la plataforma HW:

```
<SW_Platform>
        <SW_Components>
                <SW_Component name="SO" type="OS" api=”POSIX” />
        </SW_Components>
        <SW_Architecture>
                <SW_Instance name="my_OS" component="SO" HW_Resource="my_proc" />
        </SW_Architecture>
</SW_Platform>
```

Para acabar de describir el sistema solo nos queda indicar que una instancia del programa hello_world ha de ser ejecutada sobre el sistema operativo desfinido anteriormente:

```
<Application>
        <Functionality>
                <Exec_Component name="hello" category="SW" function="hello_main" />
        </Functionality>
        <Allocation>
                <Exec_Instance name="Hello_world" component="hello"     os="my_OS" />
        </Allocation>
</Application>
```

Finalmente solo nos queda indicar los detalles de la simulación. Le diremos que ejecute un máximo de 1 segundo, y que la simulación finalice cuando acabe la tarea SW.

```
<Simulation time="1 s" en_with_sw=”1” />
```

El archivo completo de descripción del sistema sera el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>

<Description xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" name="Hello_world">
        <HW_Platform>
                <HW_Components>
                        <HW_Component category="bus" name="AMBA" frequency="200" />
                        <HW_Component category="processor" name="arm926t" frequency="200"/>
                        <HW_Component category="memory" name="Memory"
                        mem_size="500000K"  frequency="200" mem_type="RAM" />
                </HW_Components>
                <HW_Architecture>
                        <HW_Instance component="AMBA" name="my_bus" >
                                <HW_Instance component="arm926t" name="my_proc" />
                                <HW_Instance component="Memory" name="my_memory"
                        start_addr="0x80000000" />
                        </HW_Instance>
                </HW_Architecture>
        </HW_Platform>
        <SW_Platform>
                <SW_Components>
                        <SW_Component name="SO" type="OS" />
                </SW_Components>
                <SW_Architecture>
                        <SW_Instance name="my_OS" component="SO" SW_Resource="my_proc"/>
                </SW_Architecture>
        </SW_Platform>
        <Application>
                <Functionality>
                        <Exec_Component name="hello" category="SW" function="hello_main" />
                </Functionality>
                <Allocation>
                        <Exec_Instance name="Hello_world" component="hello" os="my_OS" />
                </Allocation>
        </Application>
        <Simulation time="1 s" />
</Description>
```

Además vamos a requerir de la simulación que nos retorne en tiempo que tarde el programa en ejecutar y la potencia que consume. Para ello debemos generar el archivo de definición de métricas:

```
        <?xml version="1.0" encoding="UTF-8"?>
         <system_metrics>
                <system_metric name="Latency" type="float" unit="Second"/>
                <system_metric name="Power_Consumption" type="float" unit="Watts"/>
          </system_metrics>
```

Con todo esto podemos lanzar la simulación. Para ello debemos compiar el archivo "hello_world.cpp" y lincarlo con SCoPE y SystemC. Posterirmente lanzaremos el ejecutable indicando los nombres de los archivos de descripción del sistema y de metricas.

9.5 Uso de SCoPE mediante la interfaz XML

En general, el funcionamiento de SCoPE está basado en una librería diseñada para extender SystemC. Esto significa que está pensada para ser utilizada como apoyo a un programa principal. Es necesario generar un archivo principal en SystemC donde se describe la plataforma y se arranca la simulación SystemC. Una vez compilado este archivo principal y todos los demás archivos necesarios para describir la funcionalidad del sistema se enlazan las librerías SCoPE y SystemC.

Sin embargo, cuando se usa SCoPE con la interfaz XML, no es necesario generar un archivo principal. SCoPE se convierte en el centro de la simulación. Esto se debe a que no es necesario generar el archivo con la descripción del sistema en SystemC. La interfaz genera automáticamente dicha descripción a partir de las descripciones XML proporcionadas.

Para poder utilizar SCoPE de este modo, se deben generar librerías auxiliares con la funcionalidad SW y con los modelos de HW específico. El ejecutable de la simulación se genera enlazando estas librerías auxiliares más la librería SystemC a SCoPE.

Dado que SCoPE se comporta como programa principal, los argumentos de entrada del ejecutable son definidos por la herramienta, y no por el código de usuario.

9.5.1 Opciones de línea de comandos

Dado que SCoPE se comporta como programa principal, los argumentos de entrada del ejecutable son definidos por la herramienta, y no por el código de usuario.

Para ejecutar la simulación se debe utilizar el siguiente comando

```
Usage: scope -xml file [option(s)]
```

Las opciones disponibles son las siguientes:

`-xml file_name ó --xml-system-description file_name :`

Indican el archive con la descripción XML del sistema a similar. Su aparición en la línea de comando es obligatoria.

`-xsc file_name ó --xml-system-configuration file_name :`

Indica el archive donde se indican los valores de los parámetros de configuración del sistema. Su utilización es opcional, y solo es requerida si la descripción XML del sistema contiene parámetros.

`-xmd file_name ó --xml-metric_definition file_name :`

Indica las métricas que se el simulador debe retornar en el archive de salida al final de la simulación. Estas métricas se utilizarán para analizar el rendimiento del sistema. Su utilización es opcional. Si no se indica, no se generá el archivo de salida tras la simulación

`-xof file_name ó  --xml-system-metrics file_name :`

Indica el nombre que deberá tener el archive de salida que contendrá los resultados de la simulación. . Su utilización es opcional. Si no se indica, no se generá el archivo de salida tras la simulación

`-h ó --help`

Muestra las opciones del comando.

SECCIÓN IV: APLICACIÓN, RESULTADOS Y CONCLUSIONES

10 Plataformas sobre las que se ha probado la metodología

Para desarrollar y optimizar las tecnologías de modelado de los diversos componentes del sistema presentados en los capítulos anteriores, los resultados obtenidos han sido comparados con varias herramientas y plataformas.

Dos tipos de plataformas han servido como base para el desarrollo de la tesis: una plataforma basada en el procesador OpenRISC y otra basada en ARM. En cada una se han utilizado tanto distintas herramientas que han permitido el análisis del sistema y la comparación entre los resultados obtenidos por los modelos de sistema y los sistemas reales. Plataformas físicas, simuladores de instrucciones y simuladores de precisión de ciclo han sido utilizados para tal propósito.

10.1 Plataforma OpenRISC 1500

Basado en la especificación de procesador OpenRISC desarrollada por OpenCores, se ha desarrollado en el grupo de microelectrónica de la universidad de Cantabria una plataforma física completa a nivel RTL. Esta implementación se ha denominado oficialmente OpenRISC 1500. Además, se han desarrollado un conjunto de elementos necesarios para el funcionamiento del procesador, como buses, controladores de memoria y dispositivos de entrada/salida. Conjuntamente se ha desarrollado una infraestructura SW incluyendo compiladores, utilidades binarias, depuradores, así como una adaptación del sistema operativo eCos.

Dada la disponibilidad del código RTL de la plataforma completa y los conocimientos sobre los detalles internos, dicha plataforma ha sido tomada como referencia inicial en el desarrollo de los modelos de los distintos componentes del sistema presentados en los capítulos anteriores de la tesis. Dentro de la tesis se ha desarrollado un simulador dual, capaz de comportarse tanto exclusivamente como un emulador de instrucciones como un simulador con precisión de ciclo. Estos simuladores han sido muy útiles para obtener gran cantidad de información con la que comparar y optimizar las estimaciones obtenidas por el modelo de alto nivel con datos reales.

Además se ha extendido la plataforma HW añadiendo contadores que reportaban información del número de ciclos requerido por cada tarea SW, el sistema operativo y las comunicaciones HW/SW en la plataforma real. Este módulo ha permitido la medida de prestaciones del SW en la propia plataforma real, optimizando aún más los datos utilizados para comparar los resultados obtenidos de las técnicas de estimación propuestas durante la tesis.

10.1.1 Descripción de la plataforma

La plataforma OpenRISC se compone de un número mínimo de elementos imprescindibles para la ejecución de código SW junto con algunos periféricos adicionales de entrada/salida. Aunque el desarrollo de la plataforma en sí no se ha considerado parte de la presente tesis, y el doctorando únicamente se ha encargado del desarrollo de algunos de los componentes de la plataforma, la plataforma se presenta de forma detallada a continuación dado que se utilizó de base para la realización de parte de los experimentos con los que se comprobaron y optimizaron las técnicas de estimación y modelado presentadas en los capítulos anteriores de la tesis.

Los componentes mínimos para la ejecución de código son los siguientes:

- Procesador OpenRISC 1500
- Bus y controlador de bus
- Bus "watchdog"
- RAM y controlador de RAM
- ROM y controlador de ROM

Los periféricos adicionales de la plataforma OpenRISC se encargan de la comunicación de la plataforma con el exterior. Los periféricos integrados en la plataforma son:

- USART
- Controlador VGA
- Controlador de Teclado

Adicionalmente, se ha desarrollado otro periférico encargado de realizar las estimaciones de rendimiento del sistema. Este periférico se encarga de llevar la cuenta del número de ciclos en ejecución de cada tarea SW del sistema, del sistema operativo, el tiempo de comunicaciones, accesos a memoria y otras estadísticas internas del procesador. Este periférico ha permitido obtener un análisis completo del funcionamiento del sistema, permitiendo la comparación de los tiempos estimados por los distintos mecanismos de estimación presentados durante esta tesis.

El control del periférico es llevado directamente por el sistema operativo. Para ello se realizaron ciertas modificaciones en el sistema operativo eCos original. Estas modificaciones han permitido enlazar el periférico con el sistema SW de tal forma que se puede medir con precisión el rendimiento de los componentes del sistema, como los tiempos de cada tarea, al ser controlados los contadores de los periféricos durante los cambios de contexto del sistema.

La plataforma OpenRISC física puede verse en la figura 10.1. Esta consta de una placa con 4 FPGA Virtex E, una placa para la integración del VGA y el teclado, un monitor y un teclado. Además hay un PC adicional encargado de realizar la programación de las FPGAs, y gestionar los datos de rendimiento recibidos del sistema.

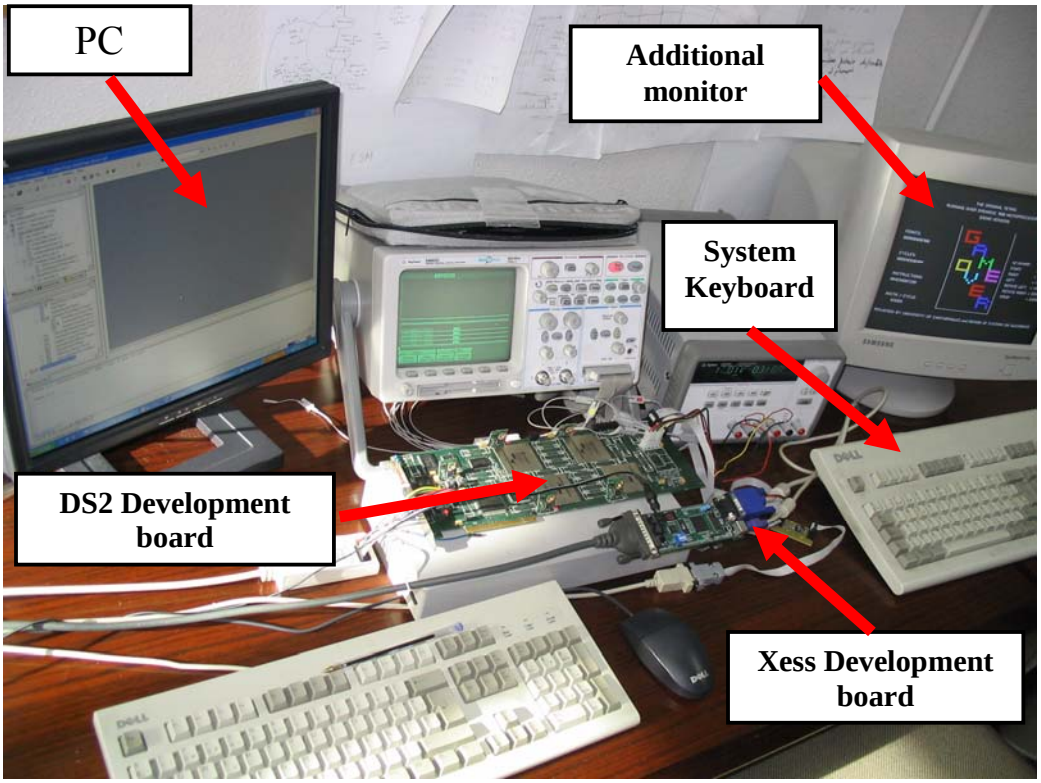


Figure 10.1: Entorno de evaluación

10.1.2 Placa de desarrollo principal

La principal parte física del sistema ha sido implementada en una placa de prototipado compuesta por varias FPGAs. En estas FPGAs se integran el procesador, el bus y los controladores de memoria y el control de las primitivas gráficas del controlador VGA. En la figura siguiente se puede ver un esquema básico con los principales componentes de la placa. La placa ha sido proporcionada por la empresa DS2 para su uso en proyectos de investigación que han tenido lugar en paralelo al desarrollo de la tesis.

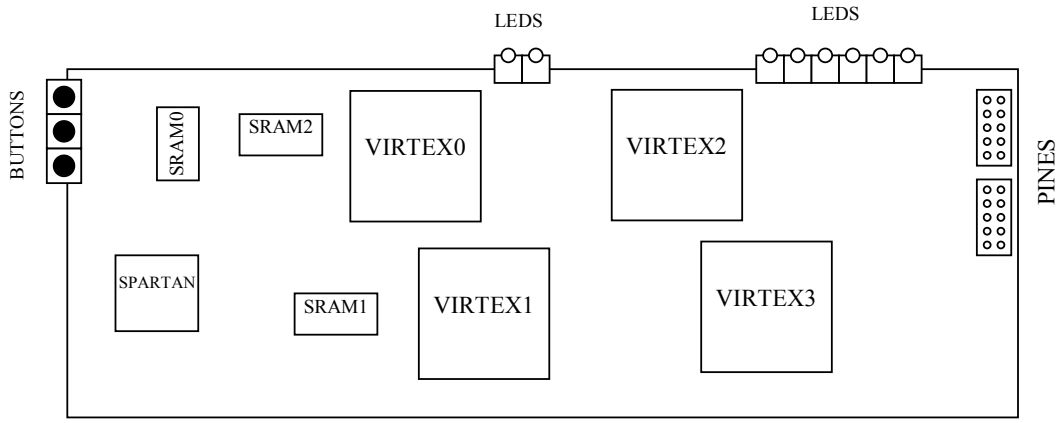


Figure 10.2: Placa de desarrollo proporcionada por DS2

Los componentes son:

FPGAs

La placa contiene cuatro FPGAs VIRTEX-E, modelo XCV2000E-6-BG560. Cada una de las FPGA tiene la siguiente funcionalidad:

- FGPA 0: En esta FPGA se encuentra el procesador OpenRISC 1500

- FPGA 1: Se utiliza como memoria ROM. De esta forma se puede descargar fácilmente el código SW a ejecutar en procesador por medio del protocolo JTAG utilizado para la programación del resto de FPGAs

- FPGA 2: Se encuentra el bus, el controlador de RAM y el control de la USART

- FPGA 3: Se encuentra el control de frecuencia para reducción de consumo, el controlador de teclado y el controlador VGA, que proporciona primitivas de dibujo de punto, línea, texto y rectángulo.

La placa también contiene una SPARTAN, modelo XCS10XL, que controla las otras FPGAs.

SRAM:

La placa contiene dos módulos de 512 KBytes de SRAM, Cy7c1351, de Cypress y un módulo de 1 MByte de SRAM, K7M163625M, de Samsung.

10.1.2.1 CPU

La CPU central de sistema es un OpenRISC 1500. El procesador sigue la arquitectura OpenRISC 1000 definida en Opencores.org. (Los detalles arquitecturales del procesador son propiedad de la empresa DS2 y no están incluidos en esta memoria)

10.1.2.2 Cache de instrucciones

El procesador cuenta con una cache de instrucciones parametrizable. La cache es de dos vías. Las posibilidades de parametrización son las siguientes:

- Tamaño de cache size: de 2KB a 8KB
- Posibilidad de bloqueo de instrucciones en cache
- Palabras por línea: 4, 8 ó 16
- Posibilidad de modificación del orden de acceso a palabra en cada ráfaga de bus.
- Precarga de líneas

10.1.2.3 Cache de datos

El procesador cuenta con una cache de datos parametrizable. La cache es de dos vías. Las posibilidades de configuración son las siguientes:

- Tamaño de cache: de 2KB a 8KB
- Palabras por línea: 4, 8 ó 16
- Posibilidad de modificación del orden de acceso a palabra en cada ráfaga de bus.
- Política de escritura en memoria principal: Write-back o write-through
- Buffer de escritura parametrizable entre 2 y 6 líneas
- Límites de zonas no cacheables variables

10.1.2.4 HW Timer

El temporizador HW se encarga de generar las interrupciones periódicas requeridas por el sistema operativo para llevar el control del tiempo. El periodo entre interrupciones en dinámicamente reconfigurable.

10.1.2.5 Watchdog

Módulo opcional encargado de evitar el bloqueo del sistema. Cuando se detecta un largo tiempo sin actividad aparente en el sistema, se considera que se ha producido una disfunción en su comportamiento, y se provoca un reinicio HW del mismo.

10.1.2.6 Bus

La arquitectura del procesador OpenRISC esta diseñada para su conexión a un bus tipo WISHBONE, definido en Opencores.org.

Además se ha desarrollado un puente que permite conectar componentes basados en AMBA-APB al sistema.

10.1.2.7 USART

Un módulo USART 16550 compatible ha sido integrado para permitir la comunicación serie entre el sistema y el exterior. La interfaz ha sido ampliamente usada para realizar la conexión entre la plataforma y el PC externo. Esto permite tanto la gestión de la monitorización del rendimiento del sistema como la conexión del depurador a la plataforma. Para su implementación se utilizó el código Verilog disponible en la Web OpenCores. Este módulo USART esta generado con una interfaz WISHBONE bus compatible.

10.1.2.8 VGA

El sistema dispone de un controlador VGA para mostrar gráficos en un monitor externo. El control VGA está diseñado con una resolución de 640 x 480 y 16 colores (bits). Para su implementación se ha utilizado una combinación de memoria de bloque y memoria distribuida en la FPGA.

El controlador proporciona una serie de primitivas para poder realizar los dibujos deseados en el monitor. Para ello proporciona primitivas de dibujo de punto, línea y rectángulo. Además permite la impresión en pantalla de mensajes de texto. Para ello se proporciona un único tipo de letra.

10.1.2.9 Controlador de teclado

Como mecanismo de entrada de información se ha desarrollado un controlador de teclado. El controlador implementa el protocolo de teclado PS/2.

10.1.3 Herramientas de desarrollo

10.1.3.1 Sistema Operativo: eCos

Como sistema operativo de la plataforma OpenRISC se desarrollo por parte de la Universidad de Cantabria una adaptación del sistema operativo eCos (embedded Configurable operating system) de RedHat. Este sistema operativo es una distribución de código abierto orientada a sistemas embebidos reducidos. La alta configurabilidad del sistema operativo permite seleccionar las funcionalidades necesarias para la aplicación en desarrollo. De esta forma se consigue que el sistema operativo sea muy ligero, minimizando los requisitos de memoria del sistema.

eCos esta pensado como un sistema operativo mono-proceso y multi-hilo, para ser usado en sistemas reducidos sin MMU. Como capacidades principales del sistema operativo eCos pueden citarse las siguientes:

- Gestión de memoria.
- Planificación de tareas.
- Canales de comunicación y sincronización

- Temporizadores, contadores y alarmas
- Manejo de interrupciones y excepciones
- Gestión de caches y watchdog.

Para permitir la utilización de las facilidades del sistema operativo eCos proporciona dos APIs, una propietaria de eCos y una conforme al estándar POSIX.

El sistema operativo proporciona así mismo una larga batería de tests con la que comprobar la correcta adaptación del sistema operativo a la plataforma física subyacente. Los tests fueron probados tanto en la plataforma real como en los simuladores. Además algunos de estos tests se utilizaron para comprobar la precisión de las estimaciones de rendimiento obtenidas utilizando las técnicas presentadas durante esta tesis.

10.1.3.2 Simulador de plataforma

Además de utilizar la plataforma real para comprobar las técnicas de estimación y modelado de sistemas desarrollados en la tesis, se utilizó el simulador de precisión de ciclo para tal efecto. Aunque el simulador no proporciona una precisión del 100% en las métricas de rendimiento, su utilización es recomendable ya que permite una mayor flexibilidad en el manejo del sistema. La realización de pruebas, evaluación de estados intermedios y análisis del estado del sistema en cada punto fue más sencilla con el simulador que con la plataforma real, ya que aunque se disponía de capacidades de monitorización del estado de la plataforma, siempre se dispone de más flexibilidad en un simulador que en una plataforma física externa.

El simulador fue desarrollado en lenguaje C, con objeto de optimizar la velocidad de simulación. Además, el hecho de estar desarrollado en este lenguaje permitió la integración del simulador juntamente con la simulación SystemC-RTL.

La posibilidad de integrar el simulador con el código RTL permitió obtener una gran precisión en el funcionamiento temporal del simulador. Para ello se integró el simulador como un módulo SystemC y se procedió a ejecutar ambos en paralelo. Tras desarrollar unos monitores adecuados se optimizó el simulador de tal forma que en cada ciclo de reloj las operaciones del código RTL se viesen fielmente reflejadas en el simulador.

El simulador cuenta con modelos del procesador y las caches, del bus, memorias y sus controladores y el puerto serie. Para el modelado de la VGA y del teclado se desarrollo un pequeño programa en Java, que conectado mediante “sockets” al simulador de plataforma permitía la entrada de datos y la visualización de los resultados en el monitor.

10.1.4 Ejemplos de aplicación

Para la verificación de las estimaciones y modelos desarrollados se usaron dos tipos de ejemplos SW. Inicialmente se aplicaron una serie de programas SW sencillos. Posteriormente se utilizó el código de un codificador GSM, como sistema complejo sobre el que obtener datos más representativos.

10.1.4.1 Benchmarks

Inicialmente, para optimizar y evaluar la herramienta SCoPE con la técnica de estimación por sobrecarga de operadores se utilizó una batería de tests cortos. Estos incluyen un filtro FIR, un algoritmo de Fibonacci, de ordenación por burbuja, de ordenación rápida (quick short), un test de trabajo masivo sobre arrays y un compresor. Los resultados obtenidos se pueden ver en la siguiente tabla. En ella se muestran también los tiempos necesarios para ejecutar las simulaciones sobre SystemC sin estimación temporal, con SCoPE y con el simulador de instrucciones del OpenRISC 1500.

	Plataforma destino Tiempo estimado (µs)			Tiempo de simulación nativo (milisegundos)		
Benchmark	SCoPE	ISS	Error (%)	SCoPE	Sobrecarga de la estimación	Ganancia con ISS
FIR	28131	29288	4	325.2	51.6	x142
Compress	167175	168744	0.9	1.61	27.28	x236
Quick sort	5197	5202	0.1	6.77	67.7	x237
Bubble	28947	28121	2.9	2.58	122.8	x178
Fibonacci	730461	741590	1.5	38.98	10.9	x278
Array	19142	18602	2.9	1.09	72.6	x247

Tabla 10.1: Tiempos estimados y tiempos de simulación para los tests cortos.

Como se puede ver, todas las estimaciones tenían un error suficientemente bajo, inferior al 5%, con respecto a código compilado sin optimizaciones. En cuanto al tiempo de ejecución, es entorno a 50 veces más lento que la simulación normal en SystemC pero más de 100 veces más rápido que el simulador de instrucciones.

10.1.4.1.1 Vocoder GSM

Posteriormente, para obtener resultados más consistentes sobre la calidad de la estimación propuesta se utilizó un modelo en SystemC de un codificador GSM según el estándar EN 301 245 del ETSI. El modelo original del estándar, con estructura secuencial tiene unas 13.500 líneas de código. Dicho código se ha dividido en nueve tareas concurrentes, que se sincronizan y planifican aplicando las funciones de un sistema operativo con interfaz POSIX.

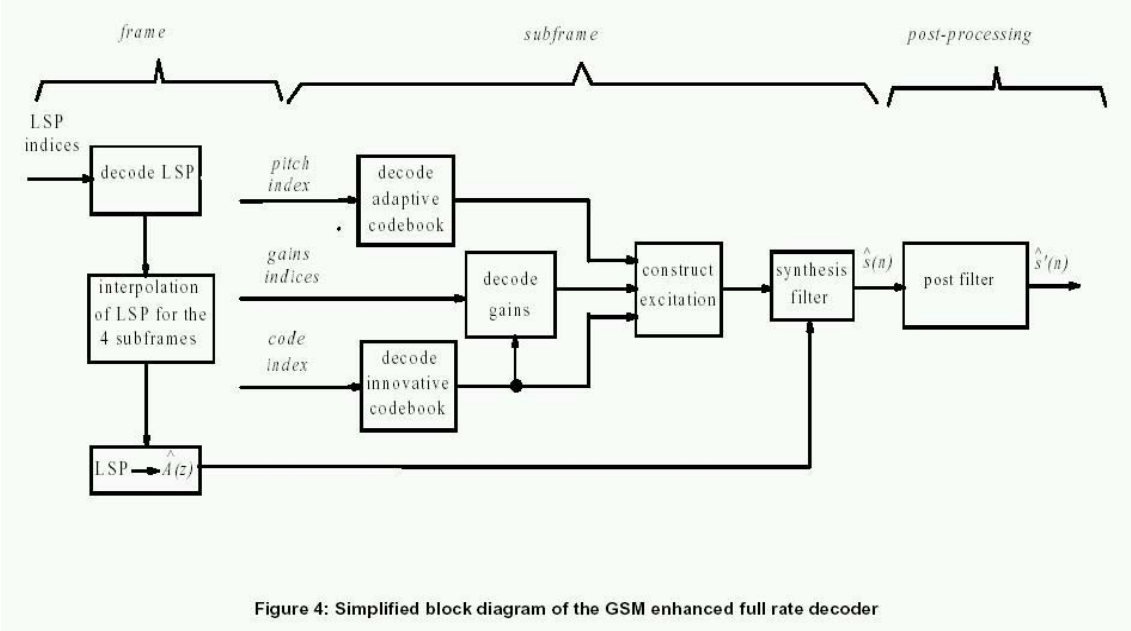


Figure 4: Simplified block diagram of the GSM enhanced full rate decoder

Figura 10-3: Diagrama de bloques del codificador GSM

En primer lugar se presentan los resultados de las estimaciones obtenidos con SCoPE sobre las tareas que componen el codificador. Se han obtenido estimaciones considerando políticas de planificación SCHED_FIFO y SCHED_RR.

Como se puede observar todos los errores de estimación de tiempo y caches de instrucciones de la técnica de estimación por sobrecarga se mantienen por debajo del 10% con respecto a una compilación sin optimizaciones, lo que implica que el método es razonablemente bueno.

Tarea	FIFO simul.(us)	RR simul.(us)	FIFO execut.(us)	RR execut.(us)	FIFO error	RR error
pre_filtering	1056.85	1056.64	1061.16	1048.88	5.4%	3.6%
homing_frame	71.27	70.84	73.12	74.45	0.4%	0.7%
frame_lsp	12430.9	12428.5	11901.9	12223.6	2.5%	4.8%
frame_int_tol	12247.7	12245.2	11246.6	12288.5	4.4%	1.6%
subframe_coder	39024.1	39018.0	37498.3	38822.7	8.9%	0.3%
serializer	184.64	184.34	173.41	169.64	4.0%	0.5%
vad_comp	12.19	12.15	11.44	12.01	6.4%	8.6%
CN_encoder	59.22	58.94	58.72	59.14	6.5%	1.1%
sid_encoding	1056.85	1056.64	1061.16	1048.88	0.8%	0.3%
RTOS	17.91e6	17.60e6	18.95e6	18.27e6	5.5%	3.7%

Tabla 10-2: Tiempos estimados y reales de los componentes del codificador GSM

	Reales	SCoPE	Abs. Error (%)
Number of Misses	66504	63449	4,59

Tabla 10-3. Estimación de mises de cache

Por último también resulta muy importante la comparación de tiempos de simulación del código sobre el kernel SystemC original y la nueva versión con SCoPE, así como una comparación con el tiempo de ejecución requerido utilizando el simulador de instrucciones desarrollado con la plataforma OR1500.

Tipo de simulación	Original SystemC	SystemC +POSIX	Sobrecarga
Tiempo(msec)	30	510	18 x
Tipo de simulación	Simul. instrucciones	SCoPE	Ganancia
Tiempo (msec)	124	1,060	124 x

Tabla 10-4: Tiempos de simulación con la estimación temporal y el modelado POSIX.

10.2 Plataforma ARM

Además de la plataforma OpenRISC también se ha procedido a probar las técnicas de estimación y modelado sobre una plataforma ARM. La comprobación de las estimaciones de tiempo, así como las técnicas necesarias para obtener los parámetros necesarios para modelar la plataforma adecuadamente se realizaron mediante un proyecto fin de carrera y otros trabajos posteriores, realizados todos ellos por Juan Castillo.

La plataforma utilizada para dicho análisis esta basada en un procesador ARM926t, integrado en una plataforma junto con una FPGA que permite el co-diseño de sistemas HW/SW. Dicha plataforma es capaz de ejecutar un Linux embebido que controla la ejecución de los componentes SW del sistema.

Además de la plataforma física los análisis de ejecución sobre ARM se realizaron utilizado el simulador Skyeye, un ISS capaz de ejecutar ensamblador del ARM9 sobre un PC.

Para la obtención de los consumos asociados al procesador se procedió a realizar una colaboración con la empresa NXP.

Con todos estos elementos se han realizado múltiples experimentos para verificar las técnicas de estimación de anotación de código. Como era de esperar la técnica de estimación mediante código binario ha demostrado ser más precisa que la de código fuente, dado que esta última no es capaz de analizar adecuadamente los efectos de las optimizaciones del compilador para el procesador destino.

Como resumen de los resultados se han generado una serie de tablas que analizan el comportamiento de las técnicas de modelado tanto en ejemplos pequeños (burbuja, factorial y Hanoy) como en ejemplos más complejos (Vocoder GSM y Códificador H264, del que sólo se han obtenido resultados sin optimizaciones, debido a un error en el compilador cruzado utilizado). En todos los casos se presentan resultados obtenidos en la estimación del número de instrucciones ejecutadas, en el modelado de caches de instrucciones y en el modelado de caches de datos. Dado que el simulador utilizado no es de precisión de ciclo, no ha sido posible realizar comparativas de número de ciclos, en lugar de número de instrucciones. En cualquier caso, los resultados demuestran la viabilidad de las técnicas de estimación.

	Número de instrucciones					
	Sin optimizaciones (-o0)			Con optimizaciones (-o2)		
	Skyeye	SCoPE	Error (%)	Skyeye	SCoPE	Error (%)
Burbuja 1000	30504511	30504511	0	4010006	4510501	12,4812
Burbuja 10000	5200180007	5200180007	0	400120008	400130013	0,0025
Vocoder	13466069	14066581	4,45945	6599330	8338713	26,357
Factorial	2747041	2996535	9,08228	1498521	1498518	0,0002
Hanoi	18481575	17695142	4,25523	13107284	11141209	14,9999
H264	5601674012	5800347641	3,54668			

Tabla 10-5. Estimación de número de instrucciones en ARM

	Fallos de cache de instrucciones					
	Sin optimizaciones (-o0)			Con optimizaciones (-o2)		
	Skyeye	SCoPE	Error (%)	Skyeye	SCoPE	Error (%)
Burbuja 1000	15	16	6,666666667	6	5	16,66666667
Burbuja 10000	25	27	8	7	7	0
Factorial	8	7	12,5	5	4	20
Hanoi	20	18	10	12	10	16,66666667
Vocoder 10	46074	46761	1,491079568	25842	28607	10,69963625
H264	677865	589944	12,97028169			

Tabla 10-6. Estimación de fallos de cache de instrucciones en ARM

	Fallos de cache de datos					
	Sin optimizaciones (-o0)			Con optimizaciones (-o2)		
	Skyeye	SCoPE	Error (%)	Skyeye	SCoPE	Error (%)
Burbuja 1000	126	127	0,79365	126	126	0
Burbuja 10000	5199772	5209087	0,17914	5199310	5211595	0,23628
Factorial	375	500	33,3333	375	500	33,3333
Hanoi	38	45	18,4211	41	45	9,7561
Vocoder 10	6018	5908	1,82785	6026	5915	1,84202
H264	5944228	5950255	0,10139			

Tabla 10-7. Estimación de fallos de cache de datos en ARM

Como se puede ver el margen de error se mantiene en estos casos en rangos similares a los anteriores, salvo en el caso del factorial. El error en el programa de cálculo de factoriales se debe a que en cada llamada a función en el computador nativo se guardan en el "stack" más registros que en la plataforma ARM. En particular, el código nativo guarda 6 enteros, lo que un 75% de la línea de cache, mientras que en el ARM se ocupa la línea entera. Por ello, al realizarse hasta 500 llamadas recursivas, se produce la reutilización de líneas en un caso, mientras que en el otro cada llamada requiere una nueva línea de cache.

Sin embargo, podemos decir que este caso es un caso muy particular. Este error se resuelve en cuanto hay variables locales en las funciones y en su defecto se compensa cuando se realicen llamadas a función con distinto número de argumentos o cuando se utilizan funciones que necesitan guardar distinto número de registros. También depende de la profundidad del número de llamadas a función en el "stack", y en general del porcentaje de datos guardados por las llamadas a función con respecto al número de datos totales usados por el programa.

En el caso del factorial, no hay más datos que estos, la profundidad es enorme y dada la simplicidad del ejemplo ninguno de los mecanismos de compensación es aplicado, pero este caso es extremadamente difícil de repetir en cualquier otro ejemplo. Conforme aumenta un poco la complejidad del código los errores se compensan y minimizan al aumentar la cantidad de datos utilizados. Por ello, si bien es importante tener constancia de la causa del error, es enormemente improbable que el uso de la herramienta con cualquier otro programa de lugar a un error tan alto.

Por último se ha realizado una comparativa entre el tiempo requerido para ejecutar el código en el simulador de instrucciones Skyeye y en SCoPE, considerando tanto ambas caches, como únicamente cache de instrucciones y sin caches. En resumen se puede ver como las técnicas propuestas permiten la aceleración de la simulación en unos dos órdenes de magnitud, aumentando al reducir el número de elementos modelados en SCoPE.

	Skyeye	SCoPE		Sin cache de datos		Sin caches	
	Tiempo	Tiempo	Aceleración	Tiempo	Aceleración	Tiempo	Aceleración
Burbuja 1000	0m2.186s	0m0.028s	x78	0m0.028s	x78	0m0.025s	x80
Burbuja 10000	4m6.500s	0m3.486s	x71	0m2.792s	x88	0m1.92s	x130
Factorial	0m1.071s	0m0.014s	x76	0m0.014s	x76	0m0.012s	x90
Hanoi	0m9.426s	0m0.043s	x219	0m0.032s	x294	0m0.020s	x479
Vocoder 10	0m48.793s	0m0.262s	x187	0m0.185s	x263	0m0.105s	x464

Tabla 10-8. Comparación de tiempo de simulación de SCoPE con Skyeye

Una vez garantizado la disponibilidad de la información adecuada de modelado para procesadores ARM, se pasó a utilizar dicha infraestructura en tareas de exploración del espacio de diseño.

10.3 Utilización para Exploración del Espacio de Diseño

Para comprobar la utilizad del trabajo desarrollado en la tesis se procedió a realizar varios experimentos de Exploración del Espacio de Diseño (DSE). Estos experimentos implican la definición de un sistema parametrizable en el cual se realizan múltiples simulaciones aplicando las distintas posibilidades. Una vez obtenidas las estimaciones de rendimiento del sistema conforme a las distintas posibilidades es posible seleccionar la mejor configuración para el sistema.

Cuando las combinaciones a probar son limitadas, es posible la realización del análisis de manera manual. Para ello se generan los archivos XML descritos en el capítulo anterior de forma manual, se ejecutan las distintas simulaciones y se plasman los resultados obtenidos en una tabla que permite detectar la mejor configuración.

Cuando el número de parámetros y valores posibles es mayor, el número de combinaciones a evaluar crece exponencialmente, de tal forma que no es posible la realización manual de los análisis. Para ello se ha probado la integración de la herramienta generada durante la tesis (SCoPE) junto con herramientas de exploración del espacio de diseño, encargadas de seleccionar los experimentos que han de ser simulados con SCoPE y capaces de obtener aproximaciones matemáticas que permiten evaluar todos los puntos del espacio de diseño sin necesidad de ejecutar todas las simulaciones.

A continuación se presentan un par de ejemplos desarrollados durante la tesis, uno por cada caso.

10.3.1 Exploración de ejemplos con parámetros limitados

Para demostrar las posibilidades derivadas del uso directo de la herramienta se realizaron varios experimentos con el Vocoder GSM (figura 10.4) que ya fue presentado junto con la plataforma OpenRISC. Para ello se definieron una serie de plataformas distintas capaces de ejecutar el mismo código SW de codificación y decodificación. Posteriormente se compararon los resultados obtenidos con objeto de comprobar las capacidades de la técnica de estimación.

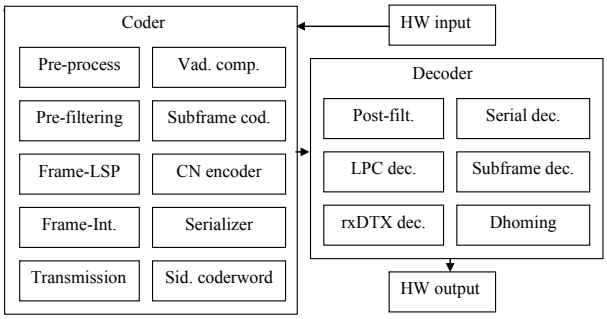


Figura 10-4. Grafo de tareas del sistema GSM

10.3.1.1 Sistema Mono-procesador

En primer lugar se evaluó la ejecución del sistema complejo en un sistema con un único procesador. Para ello, la comunicación entre los módulos codificador y decodificador se realizó utilizando colas de mensajes proporcionadas por el sistema operativo.

Las operaciones de entrada de datos al codificador y de salida del decodificador se modelaron utilizando modelos de componentes HW específicos para entrada/salida conectados al bus del sistema.

10.3.1.2 Modelado de una arquitectura SMP

En segundo lugar se modelo un sistema con dos procesadores simétricos conectados al mismo bus. Los procesadores comparten una memoria común y tienen acceso al mismo HW de entrada/salida que en el ejemplo anterior. En este caso los hilos tanto del codificador como del decodificador pueden ser mapeados por el sistema operativo en el procesador que en cada caso considere conveniente.

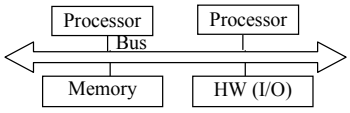


Figure 10-5. Arquitectura SMP

10.3.1.3 Modelado de arquitectura heterogénea

El tercer ejemplo se realizó proponiendo un modelo de arquitectura multiprocesador heterogénea (HMP). En este caso los procesadores no se encontraban en funcionamiento simétrico, aunque fuesen procesadores iguales (AMR9). Esto se debe a que cada procesador tiene acceso a un bus distinto, a una memoria distinta y a unos periféricos diferentes en su bus local. De esta forma no es posible que las tareas sean automáticamente transferidas de un procesador al otro.

La comunicación entre los dos procesadores se realiza mediante una memoria compartida conectada a un tercer bus. Dicho bus es accesible desde los procesadores mediante sendos puentes.

En esta arquitectura, el codificador GSM se mapeó íntegramente en uno de los procesadores y el decodificador en el otro.

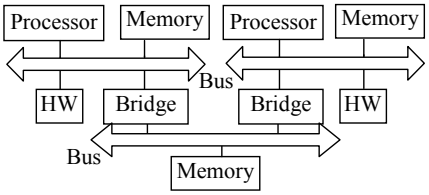


Figure 10-6. Arquitectura HMP

10.3.1.4 Modelado de arquitecturas con redes

Finalmente se modelo una arquitectura basada en dos nodos independientes conectados por una red. De manera similar al caso anterior se modeló un nodo procesador con memoria y periféricos locales encargado de la ejecución del codificador GSM y otro nodo encargado del decodificador.

La transferencia de información entre codificador y decodificador se realiza mediante transferencias por la red. El modelo de red, aunque muy sencillo dado que solo se consideran dos nodos, tiene control de ancho de banda y considera la sobrecarga introducida por los protocolos de comunicación.

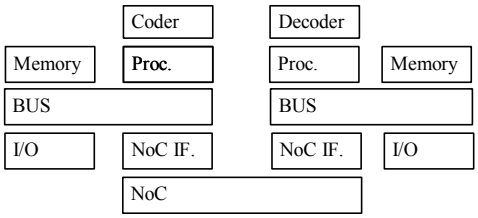


Figure 10-7. Modelo GSM utilizando una red de comunicaciones

10.3.1.5 Resultados de la exploración

Para realizar la comparación de los cuatro sistemas propuestos se obtuvieron datos de tiempo de ejecución, consumo y grado de utilización de los procesadores.

Los análisis realizados demostraron que el codificador, ejecutado en el procesador '1' requería mucha más carga de cómputo que el decodificador (procesador '2'). Por esta razón, la exploración concluyó que la solución SMP era la más idónea para la ejecución del sistema,

ya que la posibilidad de mover hilos de ejecución entre los procesadores balancea el sistema reduciendo el tiempo de ejecución prácticamente a la mitad. Además, la eliminación de los tiempos "idle" del procesador, en los que no realiza cómputos necesarios para la aplicación, pero si consume, produce una reducción del consumo total del sistema.

		MonoP	SMP	HMP	Net
Tiempo estimado	Proc 1	60.2 s	32.2	55.08s	55.15s
	Proc 2	-	28s	5.2s	5.5s
	Total	63.6s	34.7s	59.8s	60s
Energía estimada (mJ)	Proc 1	149	80	142	136
	Proc 2	-	69	77.4	13
	Total	153.4	218.1	219.4	218.2
Utilización del procesador	Proc 1	95%	92%	92%	93%
	Proc. 2	-	82%	9%	8%

Tabla 10-9. Rendimiento de las distintas arquitecturas propuestas

Adicionalmente se realizó una comparación del tiempo de ejecución de la arquitectura monoprocesadora con la ejecución sobre el simulador Skyeye. El resultado demuestra que la ejecución con SCoPE es dos órdenes de magnitud mejor que la ejecución en el ISS.

Tabla 10-10. Tiempos de simulación

Arquitectura	Skyeye	SCoPE
Tiempo de simulación	2m55s	0m1.7s

10.3.2 Utilización de SCoPE en cooperación con herramientas de exploración del espacio de diseño

Para demostrar que SCoPE realmente es una herramienta de diseño útil en exploración del espacio de diseño se integró la herramienta junto con los exploradores M3Explorer y modeFRONTIER.

A tal efecto se utilizó un codificador MPEG4 proporcionado por IMEC y se exploró su optimización sobre una plataforma basada en procesadores ARM9.

10.3.2.1 Caso de Uso MPEG 4

El codificador MPEG-4 es un estándar industrial de codificación de video hibrida basada en bloques. El codificador esta compuesto por una fase de estimación de movimiento y una fase de compensación seguida por fases de transformación y codificación de entropía. La implementación utilizada en el ejemplo está basada en el código de referencia del "MPEG-4 Simple Profile".

10.3.2.1.1 Estructura

La estructura del codificador MPEG-4 se muestra en la siguiente figura (figura 10.8):

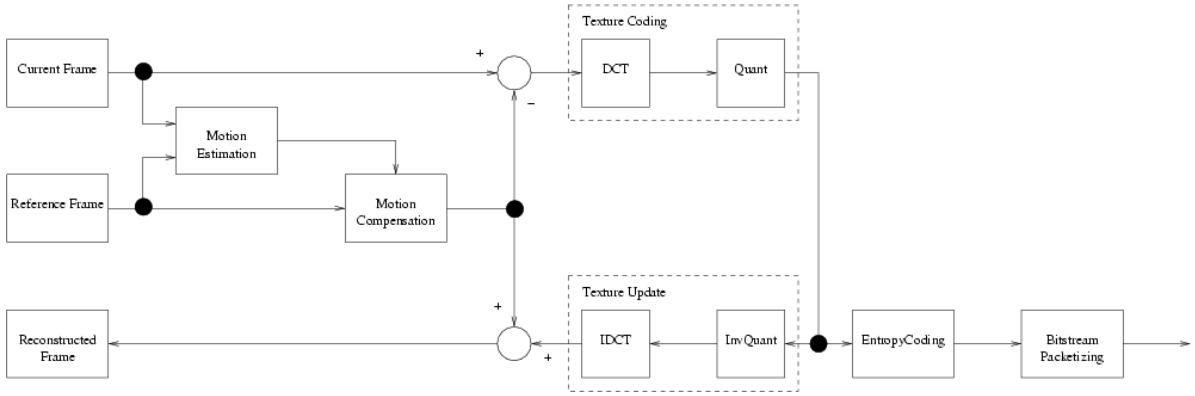


Figura 10.8: Estructura del codificador MPEG-4

Los bloques funcionales que componen el codificador son: MotionEstimation (ME), MotionCompensation (MC), TextureCoding (TC), TextureUpdate (TU), EntropyCoding (EC) y BitstreamPacketizing (BP).

10.3.2.1.2 Versiones

El código de codificador MPEG-4 proporcionado por IMEC está compuesto por seis implementaciones distintas. Cada implementación se compone de un número distinto de tareas concurrentes, de 2 a 8. De esta forma SCoPE puede ser utilizado en el proceso de diseño para explorar cual de las paralelizaciones desarrolladas es más adecuada para el diseño del codificador.

La distribución de los bloques en las distintas tareas para las seis implementaciones puede verse en la siguiente figura:

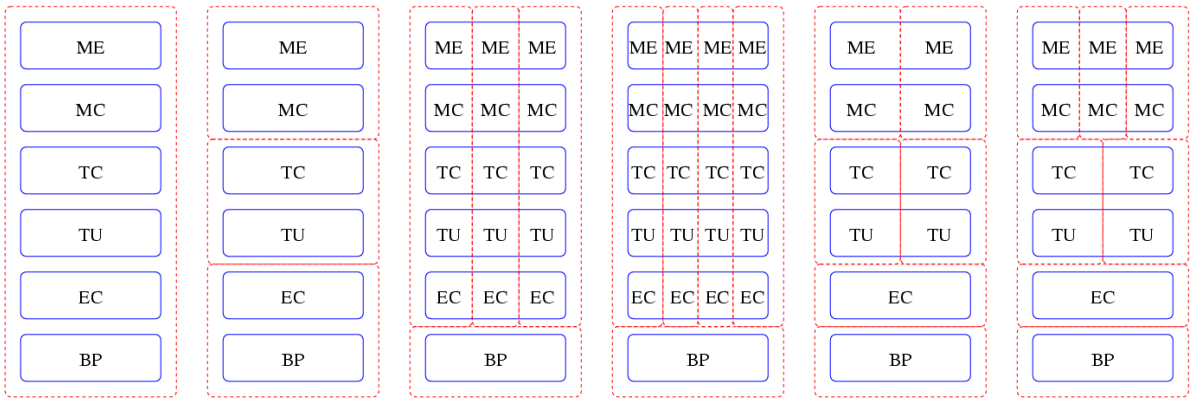


Figura10-9: Versiones del codificador MPEG-4.

En la figura los bloques funcionales están marcados en azul, y los hilos asociados se identifican con las líneas punteadas rojas.

10.3.2.1.3 Plataforma HW

Para la ejecución del codificador se ha desarrollado un modelo de plataforma basado en un número variable de procesadores conectados a un bus y este a su vez a una memoria común para todos los procesadores.

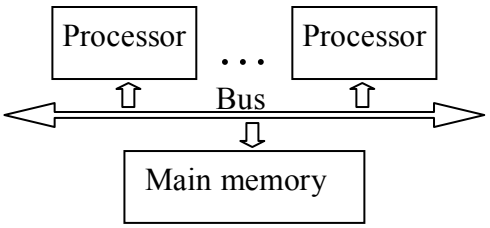


Figura 10-10: Plataforma de ejecución de MPEG-4

10.3.2.1.4 Parámetros de configuración

Para explorar la mejor configuración de la plataforma se ha procedido a explorar el rendimiento del sistema conforme a los siguientes parámetros configurables:

- Parámetros configurables de la plataforma HW:
  - Procesadores: 2-8
  - Frecuencia de los procesadores: 40-200 MHz
  - Tamaño de cache de instrucciones de los procesadores: 4-32K

Considerando todas las combinaciones posibles de los parámetros anteriores, el espacio de diseño obtenido se extiende hasta un total de 122 puntos. Este número se ha considerado excesivo para realizar un análisis manual del problema. Por esta razón se han utilizado herramientas de exploración capaces de gestionar los resultados de las simulaciones para dirigir la búsqueda.

A continuación se muestran los resultados obtenidos al aplicar el ejemplo modelado sobre SCoPE a las herramientas de exploración modeFRONTIER y M3Explorer.

10.3.2.2 Integración con M3Explorer

Al aplicar el ejemplo y enlazar el simulador SCoPE con la herramienta M3Explorer, se obtuvo un análisis completo del ejemplo en cuestión. Como demostración se muestran algunas de las gráficas obtenidas para las distintas configuraciones posibles.

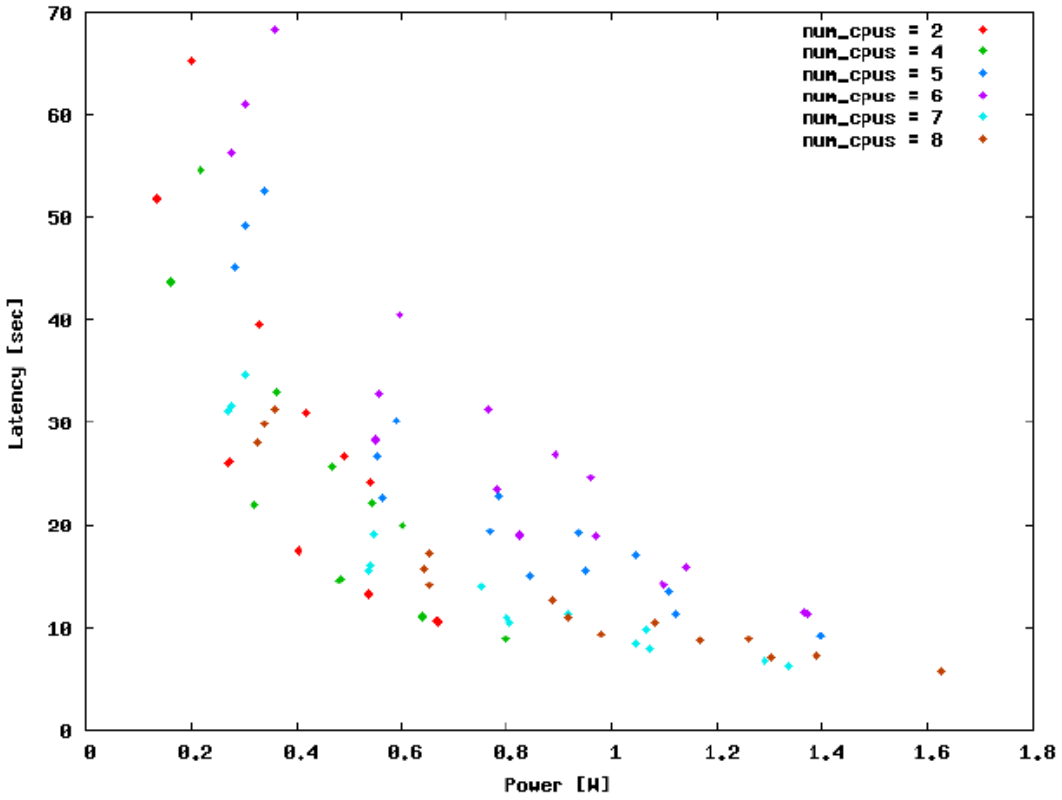


Figura 10-11: Relación latencia/potencia en función del número de procesadores

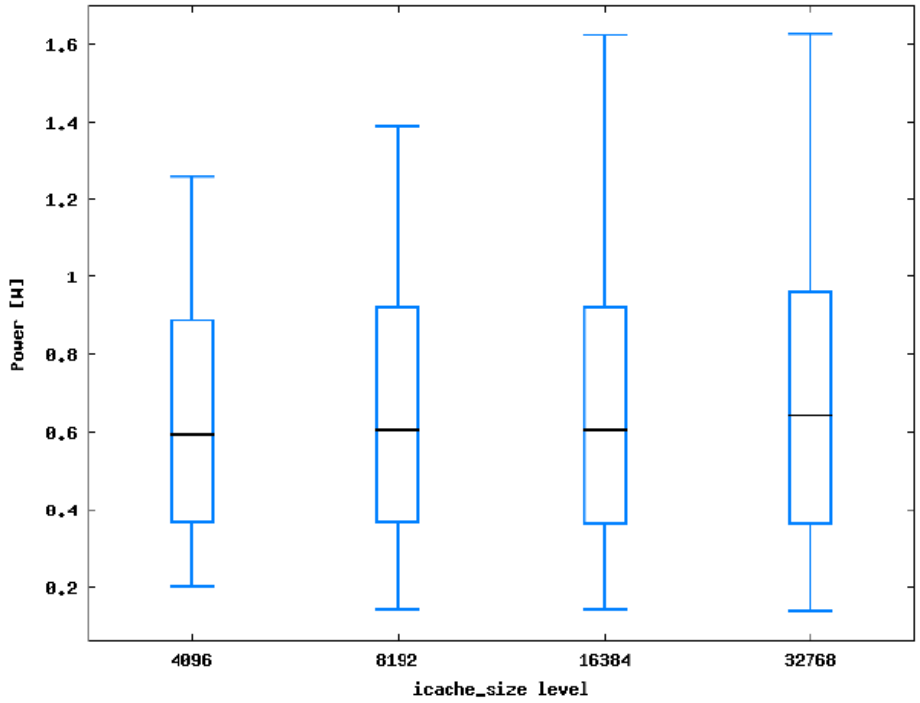


Figura 10-12: Efecto del tamaño de cache en la potencia consumida por el sistema

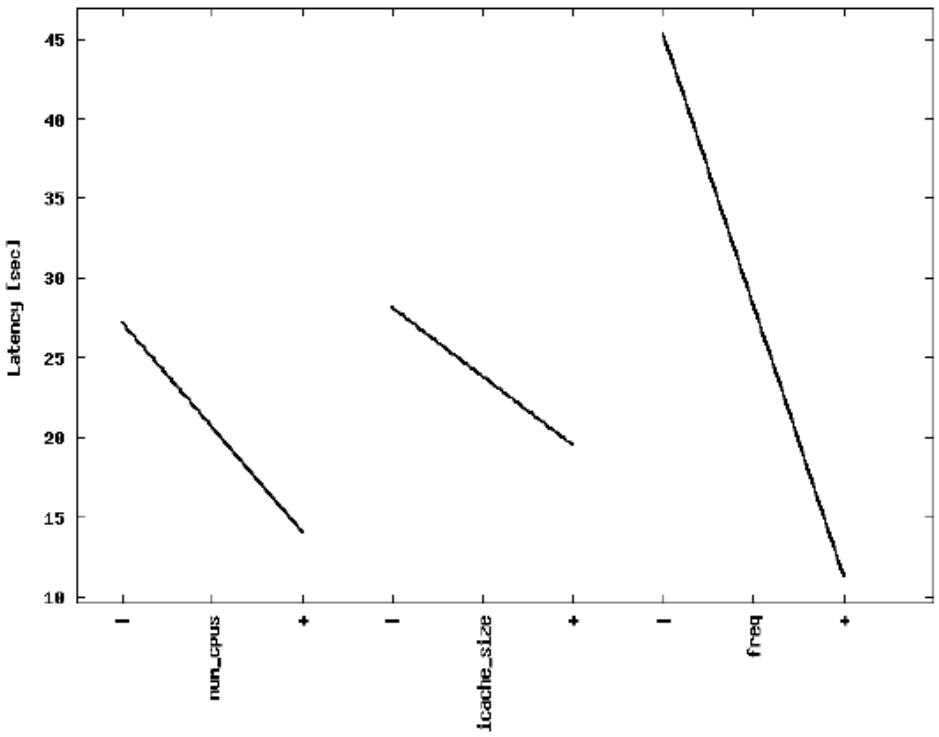


Figura 10-13: Relación entre los distintos parámetros y la latencia.

10.3.2.3 Integración con modeFRONTIER

Además de la integración de SCoPE con la herramienta universitaria M3Explorer para la exploración del espacio de diseño, se procedió a la integración con la herramienta comercial de exploración modeFRONTIER. Como resultado se obtuvo un segundo análisis en profundidad del rendimiento del ejemplo, como se puede ver en las siguientes figuras.

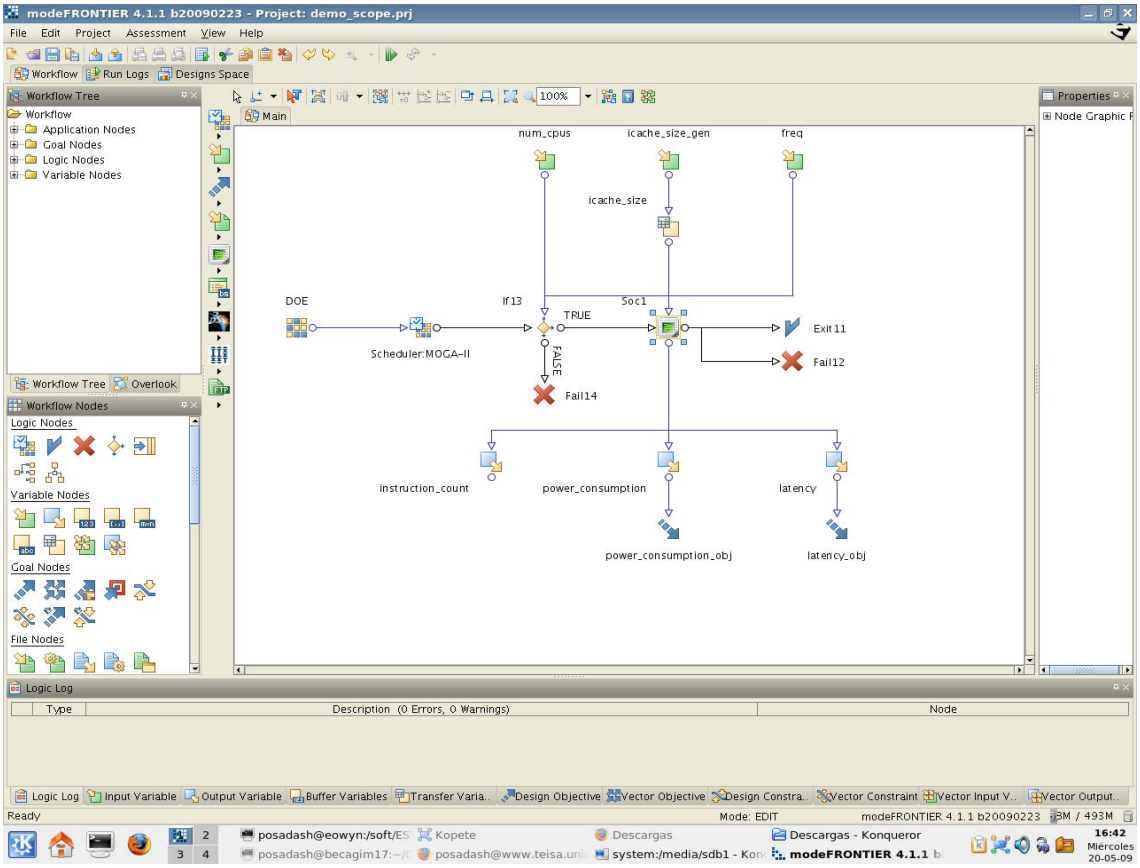


Figura 10-14: Descripción del ejemplo a explorar en modeFRONTIER

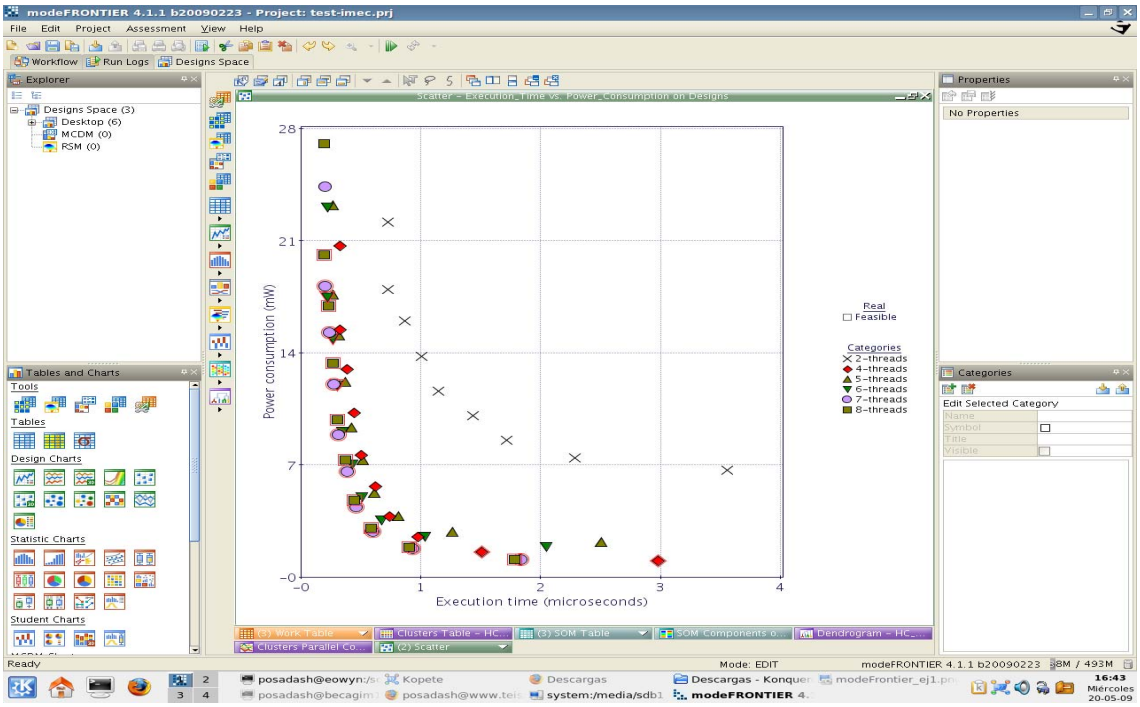


Figura 10-15: Evaluación de los frentes de Pareto en función del número de procesadores

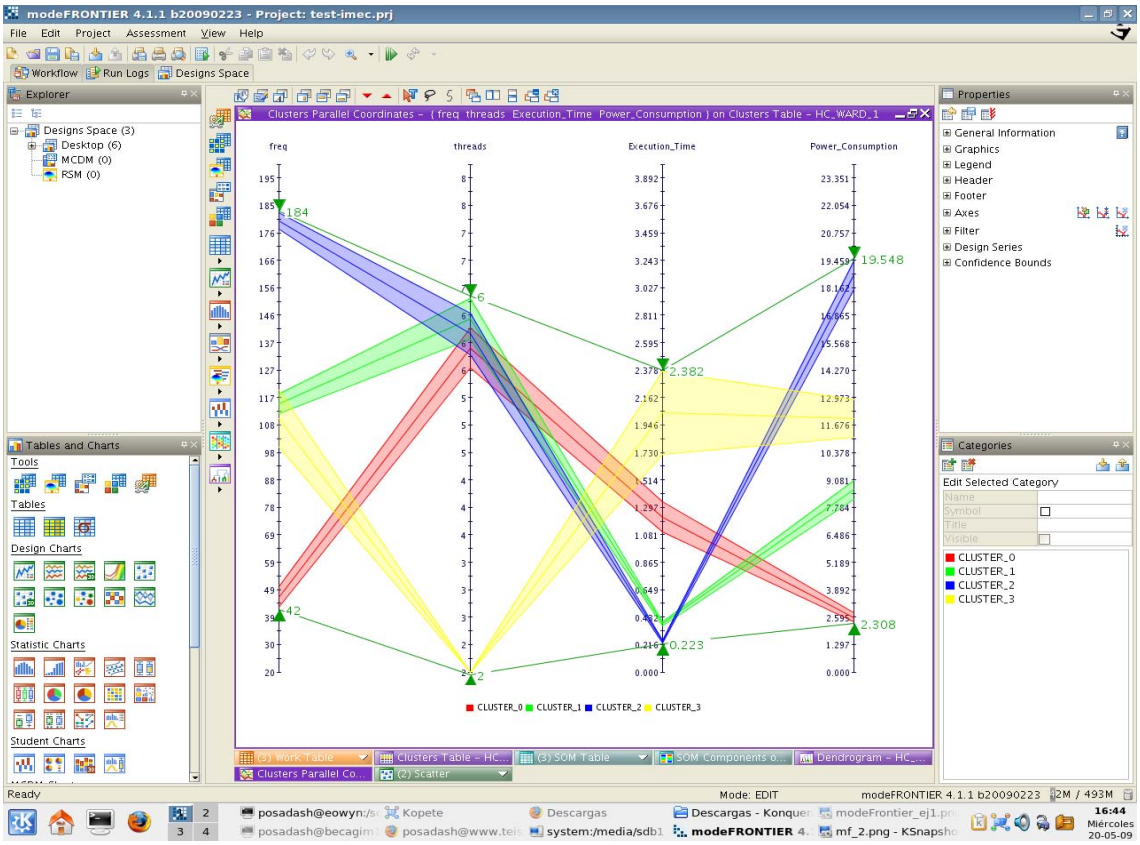


Figura 10-16: Efecto de los distintos parámetros en el rendimiento del sistema

10.4 Estimaciones de componentes HW

Aunque la estimación de tiempos de implementaciones SW a partir de especificaciones de comportamiento mediante la técnica de sobrecarga ha tenido un peso reducido en el desarrollo de la tesis, también se desarrollaron algunos ejemplos para analizar los resultados obtenidos.

Para evaluar el método de estimación de hardware se realizaron dos experimentos con un filtro FIR, un algoritmo de Euler y el módulo de post-procesado del codificador GSM. Los resultados, tanto para mejor caso (B.C.) como para peor caso (W.C.) se muestran en la siguientes tablas. Las resultados de ejecución real se han obtenido de implementaciones sobre implementaciones realizadas por la herramienta de síntesis de comportamiento Cocentric de Synopsys bajo restricciones de recursos y de tiempo.

Benchmarks	Real exec. time (ns)	Estimated exec. time (ns)	Error (%)
FIR (WC)	342	360	5.3
FIR (BC)	945	908	3.9
Euler (WC)	90	88	2.2
Euler (BC)	225	220	2.2

Tabla 10-11: Estimaciones de tiempo para componentes hardware sencillos.

Además se ha realizado una evaluación consistente en comprobar el resultado de implementar los mecanismos de post-procesado del Vocoder GSM presentado en ejemplos anteriores.

Benchmarks	Real exec. time (ns)	Estimated exec. time (ns)	Error (%)
Post. Proc. (WC)	4.875	4.475	8.2
Post. Proc. (BC)	1.975	1.900	3.8

Tabla 10-12: Estimaciones de tiempo para componentes hardware complejos.

11 Participación en proyectos y diseminación

Una de las tareas principales en el trabajo de investigación dar la máxima visibilidad posible a los resultados obtenidos, para que puedan ser aprovechados por el resto de la comunidad. Para ello, los resultados obtenidos durante la presente tesis han sido presentados en varias formas. En primer lugar, el trabajo de la tesis se ha desarrollado en una serie de proyectos europeos. En estos proyectos los resultados se ha mostrado a las empresas y universidades participantes a la vez que se han analizado e integrado sus propuestas y necesidades con objeto de producir una herramienta de modelado lo más útil posible.

Los resultados obtenidos también han sido utilizados como base para otros proyectos en los que ha participado del Grupo de Ingeniería Microelectrónica de la Universidad de Cantabria. Esta utilización, conjuntamente con las webs públicas de acceso a la herramienta SCoPE, los artículos, presentaciones y demostraciones en diversas conferencias han dado al trabajo desarrollado una gran visibilidad. Además, la herramienta ha sido presentada y probada en varias empresas.

11.1 Proyectos donde se ha desarrollado el trabajo

El trabajo presentado en esta tesis se ha desarrollado durante cuatro proyectos europeos. En cada uno de ellos se ha desarrollado una parte distinta, a la par que se evolucionaban las partes desarrolladas con anterioridad. Además, conforme se producía la evolución del trabajo se ha ido modificando el ámbito de aplicación de las herramientas generadas, y sus nombres. Los proyectos son los siguientes:

Medea+ A511 ToolIP

La primera etapa del desarrollo de la tesis se realizo dentro del proyecto ToolIP. Esto tuvo lugar desde el 09-2002 a 12-2003. Durante el mismo se desarrollo el análisis de tiempo de código fuente mediante la técnica de sobrecarga de operadores. Como consecuencia se generó una herramienta denominada PERFidy. Además, se desarrollo la plataforma OpenRISC.

ITEA IP 03002 Merced

La segunda etapa se produjo durante el proyecto Merced. Este trabajo se desarrolló desde el 09-2004 al 12-2005. En esta etapa se desarrolló el modelo de sistema operativo basado en el estándar POSIX. Como resultado de la integración de este trabajo con el realizado en el proyecto anterior se generó la herramienta PERFidiX.

Medea+ 2A708 LoMoSA

La tercera etapa realizada en el marco del proyecto LoMoSA se realizó durante los años 2006 y 2007. En el se integró la gestión de drivers dentro del sistema operativo, el stack TCP-IP, el modelo de ORB, el modelo de plataforma HW, la integración del modelo de red

de Sicosys, estimaciones de consumo SW y la extensión de tiempo real del modelo de sistema operativo. Como resultado de la integración de este trabajo y los anteriores se generó la herramienta SCoPE.

Multicube FP7 216693

A partir de la base de SCoPE se ha realizado, durante 2008 y 2009 una extensión para la utilización del trabajo en procesos de exploración del espacio de diseño (DSE). Para eso se ha adaptado SCoPE con objeto de permitir la generación en tiempo de ejecución del modelo de sistema. A partir de una descripción en XML del sistema, se configuran los componentes del sistema y se interconectan al comienzo de la simulación. Esto permite la ejecución de múltiples configuraciones de diseño simultáneamente y sin necesidad de re-compilación, al no estar el sistema a modelar descrito en el código SystemC de la simulación. Además se han mejorado algunas de las capacidades de modelado de SCoPE, como la técnica de estimación por anotación de código fuente.

Medea+ 2A714 SoftSoc

En este proyecto se pretende desarrollar una metodología de diseño de HdS (Hardware dependent SW) capaz de permitir la reutilización de los códigos entre múltiples plataformas. SCoPE tiene la misión de servir de plataforma virtual desde la que probar los drivers y otros códigos en un modelo completo de sistema. En el desarrollo interno de SCoPE cabe destacar el trabajo hecho en anotación y modelado de caches de instrucciones.

Artemis JU 100029 Scalopes

En este proyecto utiliza SCoPE para el modelado de los sistemas HW/SW. SCoPE ha de servir tanto de plataforma virtual sobre la que probar el código SW generado, como para obtener estimaciones rápidas de rendimiento con las que verificar la partición HW/SW de los sistemas en diseño. Técnicas como el modelado de caches de datos han sido desarrolladas en este proyecto.

Complex FP7 247999

En este proyecto se está explorando la utilización de UML/MARTE para la descripción de sistemas embebidos en base a diversos niveles de abstracción. SCoPE ha de servir de plataforma sobre la que verificar los modelos MARTE generados.

11.2 Página Web y descarga

SCoPE es una herramienta gratuita de código abierto, bajo licencia GPL. Para la difusión de SCoPE a la comunidad científica se ha creado una página Web desde la que se puede descargar la herramienta así como los manuales y otros documentos necesarios para su utilización. La dirección Web es la siguiente www.teisa.unican.es/scope.

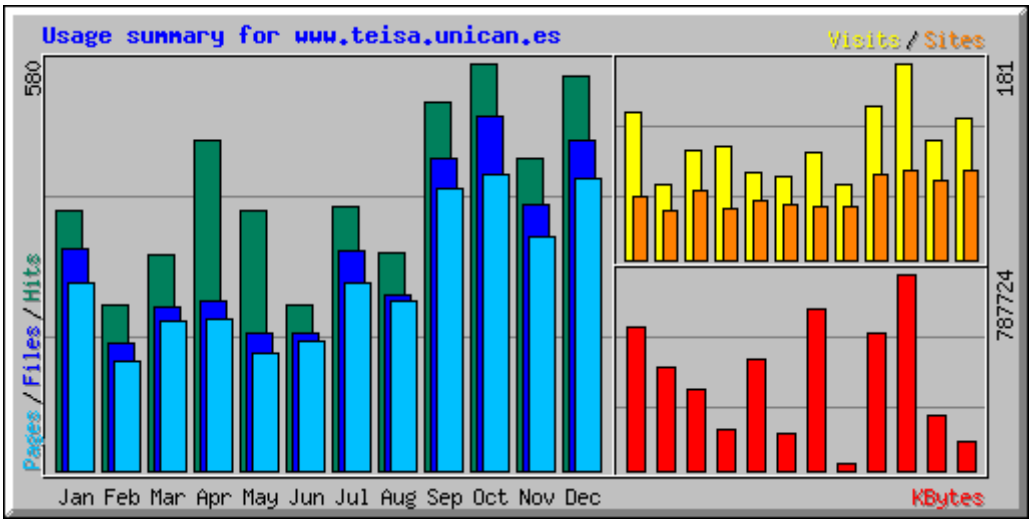


Figura 11-1: Tabla de descargas de SCoPE en 2010.

11.3 Uso de SCoPE en empresas

SCoPE se ha presentado y evaluado en varias empresas:

11.3.1 Marvell Hispania, anteriormente Design of Systems on Silicon (DS2)

Sita en Valencia es un líder mundial en tecnología de transmisión por líneas de alta tensión (Power Line Communication, PLC). Esta empresa ha seguido la evaluación del presente trabajo durante la moyor parte de los cuatro proyectos descritos anteriormente. Como consecuencia, la empresa ha procedido a la integración de la tecnología SCoPE en el proceso de desarrollo de sus nuevos prototipos.

Los enlaces físicos entre los diferentes nodos de una red de línea eléctrica actual tienen características diferentes dependiendo de factores como la distancia entre los nodos, los aparatos eléctricos conectados a la red, la calidad de la infraestructura, etc. Esto se traduce en el hecho de que la velocidad de transmisión, latencia y calidad de servicio puede depender en gran medida de la ruta seleccionada para transmitir información entre dos nodos.

Muchos de los parámetros de diseño del sistema tienen que ser dimensionado teniendo en cuenta la aplicación, los efectos físicos, ruidos, etc. Para ello es necesario realizar muchas simulaciones, con el fin de encontrar el valor óptimo para cada uno de estos parámetros. Además, estas simulaciones requieren cubrir largos tiempo de ejecución real con el fin de hacer aparecer algunos de los efectos que han de ser investigados. De esta manera, no es posible trabajar a bajo nivel (e.g. nivel de transferencia de registros), ya que los tiempos totales de simulación serían prohibitivos. Por tanto, el uso de modelos de alto nivel resulta obligatorio. En este sentido, el trabajo desarrollado en la tesis ha facilitado a los ingenieros de

DS2 la tarea de pasar de un proceso de diseño semi-automático a un flujo completamente automático, capaz de realizar un análisis más exhaustivo del espacio de diseño.

Los resultados de esta evolución, que se resumen a continuación, han sido publicados en "*High-level modeling and exploration of a powerline communication network based on System-on-Chip"*, utilizando la información recopilada por Marcos Martínez, director de proyectos externos de la propia compañía.

11.3.1.1 Necesidades de la compañía

Un paso fundamental en el proceso de creación de un nuevo chipset para comunicaciones de línea eléctrica, después de la definición del estándar UIT G.hn, ratificado a principios de 2010, ha sido la creación de un modelo de plataforma rápido y flexible. DS2 ha desarrollado una plataforma virtual que describe, en un alto nivel de abstracción, los componentes de hardware y software del sistema. Esta plataforma, denominada STORM, hace uso de la tecnología SCOPE para HW / SW de integración. En el modelo resultante, todos los parámetros se pueden configurar con el fin llevar al sistema a sus límites y encontrar la combinación más adecuada. En este sentido, el objetivo de esta plataforma es múltiple, teniendo en cuenta las necesidades de todas las personas que han de usarla.

Cada uno de estos actores busca un resultado diferente del proceso de optimización de la plataforma, sin embargo, cada uno de ellos puede hacer uso de los avances de las metodologías proporcionada por SCoPE. Estas necesidades se enumeran en la figura 11-2.

Actor	Use	Goal
SW designers	Develop application Software over the virtual platform instead of waiting for the FPGA prototype or the ASIC design	Speed up design process
HW designers	Fine tune the hardware sub-system and the interactions between the hardware and the software. Proceed with the refinement process	Optimize HW blocks and the interface between HW and SW
System designers	Explore architectural approaches and freeze design parameters	Obtain the optimum design and architecture

Figura 11-2: Tabla de necesidades de DS2.

11.3.1.2 Modelo de Plataforma

El modelo de alto nivel del sistema "UIT G.hn" implementa una red de múltiples los nodos (denominados nodos PLC) conectados a través de canales de comunicación implementados sobre sobre líneas de alta tensión. Estos nodos PLC incluyen las diferentes capas que se describen en en estándar, centrándose especialmente en la capa MAC, donde se toman las principales decisiones de diseño.

El modelo describe la plataforma de hardware real, basada en un ASIC donde un microprocesador incorporado ejecutará una pila SW PLC que hará uso de los diferentes recursos hardware, accedidos a través de un sistema de buses. Una descripción simplificada (con sólo dos nodos) alto nivel de dicha plataforma se muestra en la figura 11-3.

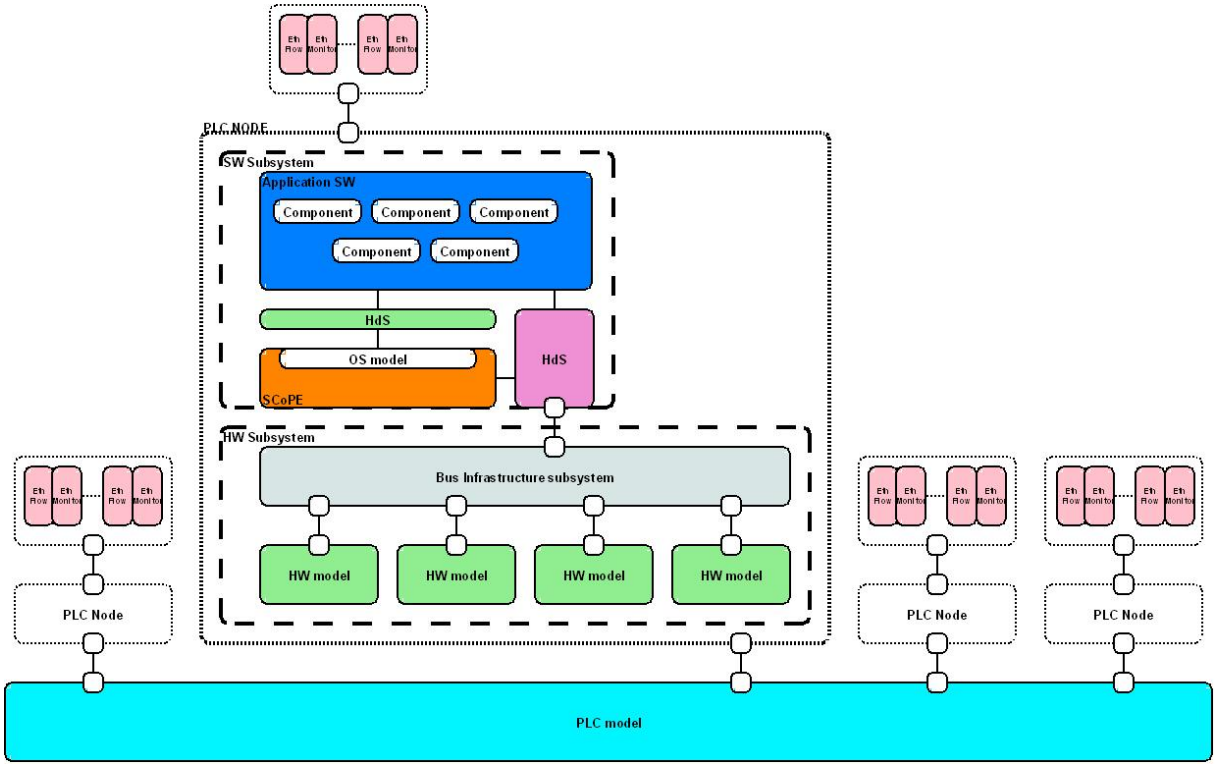


Figura 11-3: Modelo simplificado del sistema PLC

La plataforma, que se conpone de 16 nodos PLC, se ha configurado con el fin de modelar continuamente los flujos de transmisión IP entre pares de nodos del sistema.

La configuración del modelo es la siguiente:

- Características de la simulación:
  - Simulación de 0,5 segundos de tiempo real
- Topología de la red

- 16 nodos están activos durante toda la simulación (todos los nodos están ejecutando y escuchar)
- La comunicación se realiza sólo entre dos de los nodos del sistema: el nodo 0 y el nodo 1
- El ruido de canal puede ser modificado por las herramientas, estando entre 0 y 100% de probabilidad de pérdida

- Tipo de tráfico
  - Tipo de tráfico es multimedia (MPEG) introducido por los generadores de tráfico de la plataforma
  - La comunicación es bidireccional y de caudal constante. No se han configurado hay picos en el flujo
  - El flujo IP se inyecta en la red desde el nodo 0 al nodo 1 es de 40 Mbps
  - El flujo IP se inyecta en la red del nodo 1 al nodo 0 es 1 Mbps
  - Todos los flujos están basados en UDP
- Calidad de Servicio
  - No hay priorización de tráfico entre los nodos
  - No hay priorización de los flujos en un solo nodo

Parameter	Value	Unit	Related concept	Description
MAX_CELL (PKT)	64..4096	Cells	Memory (queue length)	Maximum number of packets that are stored in the modem internal memory
MAX_CELL (FEC)	64..4096	Cells	Memory (queue length)	Maximum number of FEC values that are stored in the modem internal memory
ACK ON/OFF	ON/OFF	Boolean	QoS Policy	Packet Acknowledgement protocol activated or not
PHY_BLOCK_SIZE	120/540	Bytes	Memory	This parameter is called FEC_LENGTH_value in the dse.xml and describes the length of the FEC block in the physical packet sent through powerline channel
LOSS	0..100	%	Channel characteristics	Percentage of lost packets while sent through the physical medium

Figura 11-4: Parámetros a explorar en el diseño

Con esta configuración, se han definido un conjunto de parámetros a analizar durante la exploración, parámetros que se presentan en la figura 11-4. Estos parámetros se han evaliuado utilizando el modelo de plataforma conjuntamente con la herramienta de exploración modeFRONTIER [Modefrontier] (Figura 11-5).

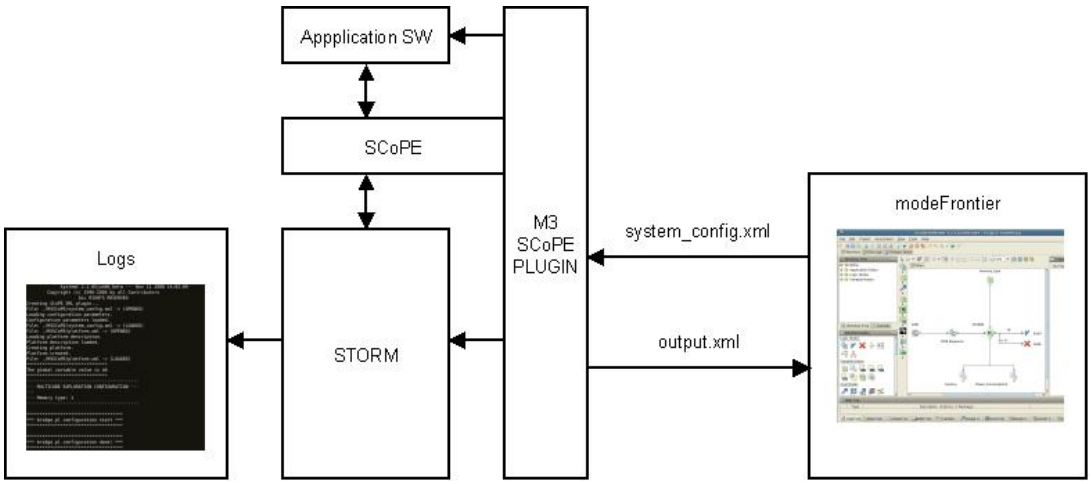


Figura 11-5: Infraestructura de diseño

11.3.1.3 Evaluación

A modo de ejemplo, se ha llevado a cabo una prueba en la que ha tratado de optimizar la plataforma descrita en las secciones anteriores y obteniendo las siguientes cifras (información proporcionada por los diseñadores de DS2).

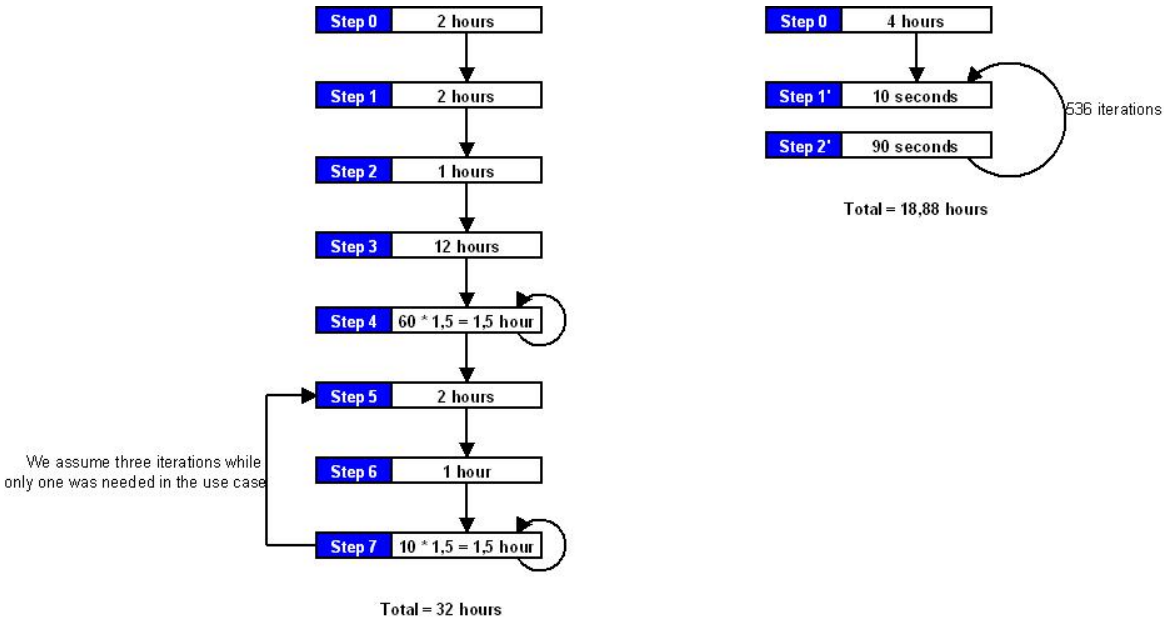


Figura 11-6: Comparación de los procedimientos de exploración

Como podemos ver claramente la forma por encima de la figura, el procedimiento automatizado proporciona:

- Mayor número de puntos: El número de puntos que pueden ser analizados seguimiento la metodología propuesta es mucho mayor que los obtenidos con el procedimiento semi-automático
- Mejor calidad de los resultados: La calidad de los resultados obtenidos a través del uso del procedimiento automatizado es mayor, ya que modeFrontier garantiza la correcta generación de experimentos con el fin de maximizar / minimizar las métricas del sistema siguiendo las directrices del usuario
- Menor complejidad del flujo: La complejidad del flujo es mucho menor en el caso del procedimiento automatizado. El número de pasos para llevar a cabo es mucho más bajo. En este sentido, es interesante mencionar que un número menor de pasos representan una posibilidad menor de los errores.
- Convergencia más rápida: La convergencia hacia el punto óptimo es alcanzado más rápidamente con el procedimiento automatizado que con la semi-automático

11.3.1.4 Conclusiones

Desde el punto de vista objetivo, después de utilizar el procedimiento de exploración del espacio de diseño para la optimización del sistema, la conclusión general es que el flujo propuesto puede ahorrar hasta un 80% de tiempo de diseño, a la ve que permite obtener mejores resultados en términos de rendimiento, ya que el uso de las simulaciones de alto nivel permite ejecutar y analizar 10 veces más combinaciones que con la diseño semi-automática de flujo utilizada anteriormente en la empresa.

Al mismo tiempo, el flujo de diseño es también una herramienta perfecta para garantizar la coherencia y la corrección del diseño.

Por último, desde un punto de vista subjetivo, podemos mencionar que el procedimiento planteado hace que el trabajo del diseñador resulte mucho fácil al proponer una excelente herramienta de front-end y la automatización de todos los procesos necesarios para actualizar la plataforma en cada ciclo de simulación. Además, puede beneficiarse de las poderosas herramientas de análisis de paquetes comerciales como modeFrontier, que han sido demostrados en otros ámbitos técnicos, con el fin de seleccionar la solución óptima.

11.3.2 Thales Communicaciones

A través de las los proyectos Merced y LoMoSA, se ha producido una interesante colaboración con dicha empresa. Como consecuencia se ha procedido al desarrollo conjunto del modelo de ORB y de la extensión de tiempo real del modelo de sistema operativo. La colaboración ha permitido el modelado de las herramientas microCCM y Hyades de Thales en SCoPE.

11.3.3 NXP

Dentro del proyecto LoMoSA se ha trabajado conjuntamente en el modelado de un procesador ARM9 de NXP en SCoPE. Para ello se ha aprovechado los conocimientos del personal de NXP y la herramienta Sleep como fuentes para realizar el modelado. Esta colaboración fue desarrollada por Juan Castillo.

11.3.4 TIMA

Por ultimo se ha realizado una colaboración con el centro de investigación TIMA de Grenoble. Durante esta colaboración se ha realizado la integración del modelo de microCCM de Thales en SCoPE, a partir del trabajo previo de adaptación a SystemC realizado por TIMA.

11.4 Diseminación

El resto de tareas de diseminación se han realizado mediante la presentación de trabajos en congresos, la generación de artículos en libros y revistas y la realización de demostraciones en congresos internacionales.

11.4.1 Demostraciones en congresos

Las diversas etapas de evolución del trabajo presentado han sido mostradas en la zona de exhibición de DATE en varios años:

- 2003 - SystemC Booth:

Se presentó la herramienta PERFidy

- 2004 - University Booth

Se presentó la plataforma OpenRISC 1500

- 2006 - University Booth

Se presentó PERFidiX

- 2008 - University Booth

Se presentó la herramienta SCoPE

- 2009 - Multicube Stand

Se presentó la evolución de SCoPE integrada en el flujo de exploración del espacio de diseño desarrollado en el proyecto Multicube

- 2009 - University Booth

Se presentó el trabajo realizado por la Universidad de Cantabria en Spices. SCoPE fue publicitado como parte de la infraestructura utilizada en el proyecto. La presentación corrió a cargo de Roberto Varona.

11.4.2 Patentes

H. Posadas, L. Diaz, E. Villar
"Método y sistema de modelado de memoria caché de datos"
Oficina Española de Patentes y Marcas. OEPM. 2010-10

11.4.3 Publicaciones derivadas

H. Posadas, L. Diaz, E. Villar
"Fast Data-Cache Modeling for Native Co-Simulation "
Asia and South-Pacific Design Automation Conference, ASP-DAC'11. 2011-01

H. Posadas, S. Real, E. Villar
"M3-SCoPE: Performance Modeling of Multi-Processor Embedded Systems for Fast Design Space Exploration"
C. Silvano, W. Fornaciari & E. Villar (Eds.): "Multi-objective Design Space Exploration of Multiprocessor SoC Architectures: the MULTICUBE Approach", SPRINGER, New York, USA

M. Martínez, D. Ferrúz, H. Posadas, E. Villar
"High-level modeling and exploration of a powerline communication network based on System-on-Chip"
C. Silvano, W. Fornaciari & E. Villar (Eds.): "Multi-objective Design Space Exploration of Multiprocessor SoC Architectures: the MULTICUBE Approach", SPRINGER, New York, USA

D. Calvo, P. Botella, H. Posadas, P. Sánchez, E. Villar
"Automatic Generation of HdS System Models for System Simulation using IP-XACT"
DATE 2011 Workshop HW-Dependent Software Solutions for SoC Design . 2011-03

D. Calvo, P. González, L. Diaz, H. Posadas, P. Sánchez, E. Villar, A. Acquaviva, E. Macii
"A Multi-Processing Systems-on-Chip Native Simulation Framework for Power anD Thermal-Aware Design"
ASP Journal on Low-Power Electronics (JOLPE): Special Issue on Low Power Design and Verification Techniques

H. Posadas, E. Villar
"Native Co-Simulation of TCP/IP-Based Embedded Systems in SystemC"
XXV Conference on Design of Circuits and Integrated Systems (DCIS'10). 2010-11

L. Diaz, H. Posadas, E. Villar
"Obtaining Memory Address Traces from Native Co-Simulation for Data Cache Modeling in SystemC"
XXV Conference on Design of Circuits and Integrated Systems (DCIS'10). 2010-11

P. Botella, P. Sánchez, H. Posadas
"Automatic Generation of SystemC SMP Models for HW/SW Co-Simulation"
XXV Conference on Design of Circuits and Integrated Systems (DCIS'10). 2010-11

S. Real, H. Posadas, E. Villar
"L2 Cache Modeling for Native Co-Simulation in SystemC"
Symposium of Industrial Embedded Systems (SIES'10). 2010-07

H. Posadas, E. Villar, Dominique Ragot, M. Martínez (DS2)
"Early Modeling of Linux-based RTOS Platforms in a SystemC Time-Approximate Co-Simulation Environment"
IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC'10). 2010-05

J. Castillo, H. Posadas, E. Villar, M. Martínez (DS2)
"Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-Simulation "
20th Great Lakes Symposium on VLSI (GLSVLSI'10), Providence, USA. 2010-05

H. Posadas, E. Villar
"Modeling Separate Memory Spaces in Native Co-simulation with SystemC for Design Space Exploration "
2PAMRA Workshop on Parallel Programming and Run-time Management Techniques for Many-core Architectures. 2010-02

H. Posadas, G. de Miguel, E. Villar
"Automatic generation of modifiable platform models in SystemC for Automatic System Architecture Exploration "
XXIV Conference on Design of Circuits and Integrated Systems, DCIS'09, 2009-11

H. Posadas, E. Villar
"Automatic HW/SW interface modeling for scratch-pad & memory mapped HW components in native source-code co-simulation"
A. Rettberg, M. Zanella, M. Amann, M. Keckeiser & F. Rammig (Eds.): "Analysis, Architectures and Modelling of Embedded Systems", Springer, 2009. 2009-09

H. Posadas, J. Castillo, D. Quijano, V. Fernández, E. Villar, Marcos Martínez (DS2)
"SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems"
L. Gomes and J. M. Fernandes (Eds.): "Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation", IGI Global. 2009-07

H. Posadas, D. Quijano, E. Villar, M. Martínez (DS2)
"Protocol Bus Modeling using inheritance with TLM2.0"
Proceedings of the Forum on Design Languages, FDL'07. Barcelona. 2007-09

J. Castillo, H. Posadas, E. Villar, M. Martínez (DS2)
"Energy Consumption Estimation Technique in Embedded Processors with Stable Power Consumption based on Source-Code Operator Energy Figures"
XXII Conference on Design of Circuits and Integrated Systems, DCIS'07 . 2007-11

E. Villar, H. Posadas, Marcos Martínez (DS2)
"SCoPE: Efficient HdS simulation for MpSoC with NoC"
MEDEA+ Design Automation Conference, Grenoble. 2007-05

H. Posadas, J. Adámez, E. Villar, F. Escuder (DS2), F. Blasco (DS2)
"RTOS modeling in SystemC for Real-Time embedded SW simulation: A POSIX model"
Journal on Design Automation for Embedded Systems, V.10, N.4, Springer. 2006-12

H. Posadas, D. Quijano, E. Villar, F. Escuder (DS2), M. Martínez (DS2)
"TLM interrupt modelling for HW/SW co-simulation in SystemC"
XXI Conference on Design of Circuits and Integrated Systems, DCIS'06 . 2006-11

H. Posadas, J. Adámez, P. Sánchez, E. Villar, F. Blasco (DS2)
"POSIX modeling in SystemC"
11th Asia and South Pacific Design Automation Conference, ASP-DAC'06. 2006-01

H. Posadas, E. Villar, F. Blasco
"Real-time Operating System modeling in SystemC for HW/SW co-simulation"
XX Conference on Design of Circuits and Integrated Systems, DCIS'05. 2005-11

H. Posadas, F. Herrera, V. Fernández, P. Sánchez, E. Villar, F. Blasco
"Single Source Design Environment for Embedded Systems Based on SystemC"
Journal on Design Automation for Embedded Systems, V.9, N.4, Springer, 2004-12

M. Bolado, J. Castillo, H. Posadas, P. Sánchez, E. Villar, C. Sánchez, F. Blasco
"Platform based on open-source cores for industrial applications"
Proc. of DATE'04, IEEE CS Press. 2004-02

H. Posadas, F. Herrera, P. Sánchez, E. Villar, F. Blasco
"System-Level Performance Analysis in SystemC"
Proc. of DATE'04, IEEE CS Press. 2004-02

M. Bolado, J. Castillo, H. Posadas, P. Sánchez, E. Villar, C. Sanchez, F. Blasco
"Using Open Source Cores in Real Applications"
XVIII Conference on Design of Circuits and Integrated Systems, DCIS'03. 2003-11

F. Herrera, H. Posadas, P. Sánchez, E. Villar
"Systematic Embedded Software Generation from SystemC"
Proc. of DATE'03, IEEE CS Press. 2003-02

F. Herrera, H. Posadas, P. Sánchez, E. Villar
"Systematic Embedded software generation from SystemC"
"Embedded Software for SoC", Kluwer Academic Publishers. 2003-01

E. Villar, P. Sánchez, H. Posadas
"System-level reusability of microprocessor cores in a SystemC specification environment"
MEDEA+ Design Automation Conference. 2002-09

11.4.4 Publicaciones comunes del proyecto Multicube

C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, R. Zafalon, S.Bocchio, M. Wouters, G. Vanmeerbeeck, P. Avasare, C. Couvreur, L. Onesti, C. Kavka, A. Turco, U. Bondi, G. Mariani, E. Villar, H. Posadas, C. Y. q. Wu, F. Dongrui, Z. Hao and T. Shibin.
"MULTICUBE: Multi-objective design space exploration of multi-core architectures"
IEEE Computer Society Annual Symposium on VLSI – ISVLSI. 2010-07

C. Kavka, L. Onesti, P. Avasare, G. Vanmeerbeeck, M. Wouters and H. Posadas.
"Design Space Exploration for Embedded Parallel System-on-Chip Platforms using modeFRONTIER"
Second Mini Conference on Theoretical Computer Science. 2009-10

P. Avasare, G. Vanmeerbeeck, M. Wouterss, B. Vanthournouts, H. Posadas.
"High-level performance estimation using simulators at multiple abstraction levels"
In Workshop on "Designing for Embedded Parallel Computing Platforms: Architectures, Design Tools, and Applications" held during DATE09. 2009-04

C. Kavka, P. Avasare, G. Vanmeerbeeck, M. Wouters, H. Posadas.
"Design space exploration for embedded parallel system-on-chip platforms using modeFRONTIER"
Workshop on "Designing for Embedded Parallel Computing Platforms: Architectures, Design Tools, and Applications" held during DATE09. 2009-04

12 Conclusiones

En la presente tesis se han propuesto diferentes técnicas que extienden el lenguaje de diseño SystemC para permitir la simulación temporal de código software de tal forma que sea posible la co-simulación de dichos componentes junto el modelo de la plataforma hardware que será utilizada en el sistema.

Para ello, se ha desarrollado una infraestructura de estimación temporal de código SystemC capaz de convertir una simulación atemporal en temporal tanto para descripciones a nivel de sistema como para componentes software. Esta estimación genera valores con errores suficientemente bajos a velocidades mucho mayores que las obtenidas con simuladores de instrucción, que es la técnica dinámica más utilizada en la actualidad.

Además se ha desarrollar una extensión del lenguaje SystemC capaz de modelar de manera adecuada una interfaz POSIX de Tiempo Real bastante extensa. Dicha extensión debe permitir la simulación, refinado y verificación de componentes software desde su especificación al código final. Esto demuestra que el lenguaje SystemC puede ser utilizado en todas las fases del desarrollo de un sistema, tanto para especificación, decisión de partición, refinado hardware, y refinado software.

Adicionalmente, la generación de modelos rápidos de los componentes de la plataforma HW del sistema capaces de realizar estimaciones de rendimiento generales, que permiten el modelado completo del diseño y la evaluación de las comunicaciones HW/SW.

En la tesis se han estudiado diversos tipos de estimaciones temporales, resultando como más eficaz la ténica de anotación de código mediante análisis de código binario. Esta técnica permite obtener unos buenos resultados de estimación, incluyendo el efecto de optimizaciones del compilador con tiempos de simulación cercanos a la ejecución funcional y má de dos órdenes de magnitud superiores a un simulador de instrucciones.

El modelo permite también el análisis del efecto de las caches en el sistema, tanto en cuanto al aumento de la eficiencia del procesador, como de la sobrecarga resultante por las comunicaciones con la memoria. No obstante, el modelado de caches puede reducir hasta en un orden de magnitud la velocidad de simulación.

Por último, los resultados reportados por DS2 (Marvell Hispania), mostrando el efecto de la integración de la tecnología en el flujo de diseño de la compañía, muestran las grandes posibilites de las técnicas de modelado presentadas en esta tesis.

13 Referencias

[Abrar04] S. Abrar, "Cycle–Accurate Model and Source–Independent Characterization Methodology for Embedded Processors", 17th International Conference on VLSI Design, 2004

[Abowd96] Abowd, Gregory, et al. "Architectural Analysis of ORBs." Object Magazine 6, 1 (March 1996): 44-51.

[Anantaraman04] A. Anantaraman, K. Seth, E. Rotenberg, F. Mueller: "Enforcing Safety of Real-Time Schedules on Contemporary Processors Using a Virtual Simple Architecture (VISA)". Rea-Time Systems Symposium, 2004

[ARMulator] The ARMulator, application note, ARM Information Center, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0032f/index.html

[Ascia07] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti. "Efficient design space exploration for application specific systems-on-a-chip". Journal of Systems Architecture, 2007.

[AXLOG] AXLOG, http://www.axlog.fr.

[Bailey09] B. Bailey and G. Martin, "ESL Models and Their Application: Electronic System Level Design and Verification in Practice", Springer, 2009.

[Becker10] M.Becker, T.Xie, W.Mueller, G. Di Guglielmo, G. Pravadelli and F.Fummi, "RTOS-Aware Refinement for TLM2.0-Based HW/SW Designs", in DATE 2010.

[Beltrame06] G. Beltrame, D. Bruschi, D. Sciuto, and C. Silvano. "Decision-theoretic exploration of multiprocessor platforms". In Proceedings of CODES+ISSS, 2006.

[Benini05] Benini et al, "MPARM: Exploring the Multi-Processor SoC Design Space with SystemC", Journal of VLSI Signal Processing n 41, 2005

[Bergamaschi03] R.A. Bergamaschi, Y.W. Jiang, "State–Based Power Analysis for Systems–on–Chip", DAC2003, June 2–6, 2003, Anaheim, California, USA, pp 638–641

[Bison] Bison, http://www.gnu.org/software/bison/

[Bjuréus01] P. Bjuréus, A. Jantsch, "Performance analysis with confidence intervals for embedded software processes", Symposium on System Synthesis, ISSS, 2001.

[Bose02] Bose S.J, "An Introduction to Queueing Systems", Kluwer/Plenum Publishers, 2002.

[Bouchima09] A. Bouchima, P. Gerin & F. Pétrot: "Automatic Instrumentation of Embedded Software for High-level HS/SW Co-simulation. ASP-DAC, 2009.

[Brandolese01] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "Source-level execution time estimation of c programs," CODES 2001.

[Brown96] Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." IEEE Software 13, 5 (September 1996): 39-49.

[Cai03] L. Cai, D. Gajski, "Transaction Level Modeling: An Overview", in proceedings of the Int. Conference on HW/SW Codesign and System Synthesis (CODES-ISSS), 2003.

[Caldari03] M. Caldari, M. Conti, L.Pieralisi, C. Turchetti: "Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0", DATE'03

[CatapultC] Catapult C Synthesis Overview, Mentor Graphics, http://www.mentor.com/esl/catapult/overview

[Chanteperdrix04] G. Chanteperdrix, A. Berlemont, D. Ragot, and P. Kajfasz, "Integration of Real-Time Services in User-Space Linux", 6th RTL Workshop, 2004.

[Chung07] Eui-Young Chung, Hyuk-Jun Lee, Sung Woo Chung: "Scenario-Aware Bus Functional Modeling for Architecture-Level Performance Analysis", IEICE 2007

[Cifuentes] C. Cifuentes. "Reverse Compilation Techniques". PhD thesis, Queensland University of Technilogy, 1994.

[Cobb95] Cobb, Edward E. "TP Monitors and ORBs: A Superior Client/Server Alternative." Object Magazine 4, 9 (February 1995): 57-61.

[Colin02] A. Colin & G. Bernat: "Scope-Tree: A Program Representation for Symbolic Worst-Case Execution Time Analysis", Euromicro conference on Real-Time Systems, 2002

[CoMET] VaST Systems Technology. CoMET R. http://www.vastsystems.com/docs/CoMET_mar2007.pdf.

[CORBA96] The Common Object Request Broker: Architecture and Specification, Version 2.0. Framingham, MA: Object Management Group, 1996.

[Coware] CoWare Processor Designer, http://www.coware.com/products/processordesigner.php

[Dally07] Dally, W et al, "Architectural Support for the Stream Execution Model on General-Purpose Processors", 16th International Conference on Parallel Architecture and Compilation Techniques, 2007

[Dhanwada05] N. Dhanwada, I.C. Lin, V. Narayanan, "A Power Estimation Methodology for SystemC Transaction Level Models", CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, USA

[Dietrich05] S. Dietrich, D. Walker, "The Evolution of Real-Time Linux", Proc. of Real-Time Linux Workshop, 2005.

[Dozio03] L. Dozio, P. Mantegazza. "Real Time Distributed Control Systems Using RTAI", Proc of ISORC, 2003.

[Dowdy04] L. W. Dowdy, V. Almeida, D. A. Menasce: "Performance by Design: Computer Capacity Planning by Example", Prentice Hall, 2004.

[ENEA] ENEA: "OSE Soft Kernel Environment", in http://www.ose.com/products.

[FeWi99] C. Ferdinand and R. Wilhelm. "Efficient and precise cache behavior prediction for real-time systems". Real-Time Systems, 1999.

[Flex] Bison, http://www.gnu.org/software/flex/

[Franke02] H. Franke, M. Kirkwood, and R. Russell. "Fuss, futexes and furwocks: Fast userlevel locking in linux". Proc of OLS, 2002.

[Fummi04] F. Fummi, S. Martini, G. Perbellini, M. Poncino. Native ISS-SystemC Integration for the Co-Simulation of Multi-Processor SoC. DATE, 2004

[Gerlach98]J. Gerlach & W. Rosenstiel: "A Scalable Methodology for Cost Estimation in a Transformational High-Level Design Space Exploration Environment", DATE, 1998

[Gerstlauer03] A. Gerstlauer, H. Yu and D. Gajski: "RTOS modeling for system-level design", in A.A. Jerraya, S. Yoo, D. Verkest and N. When (Eds.): "Embedded Software for SoC", Springer, 2003.

[Gerstlauer03] Gerstlauer, A. Yu, H. & Gajski, D.D.: "RTOS Modeling for System Level Design", Proc. of DATE, IEEE, 2003.

[Gerslauer10] A. Gerslauer, "Host-Compiled Simulation of Multi-Core Platforms", Rapid System Prototyping, 2010

[GhMM99] S. Ghosh, M. Martonosi and S. Malik. "Cache miss equations: a compiler framework for analyzing and tuning memory behavior". ACM Transactions on Programming Languages and Systems, 1999.

[Gleixner05]T. Gleixner. "ktimers subsystem", Linux Kernel Mailing List, 2005. http://lkml.org/lkml/2005/9/19/124.

[Gligor09] M. Gligor, N. Fournel, and F. Petrot, "Using binary translation in event driven simulation for fast and flexible MPSoC simulation", in CODES+ISSS, France, Oct. 2009.

[Ghenassia05] Ghenassia, F.: "Transaction-Level Modeling with SystemC", Springer. 2005.

[Giusto01] P. Giusto, G. Martin & E. Harcourt: "Reliable estimation of Execution Time of Embedded Software", DATE, 2001.

[GreenSocs] GreenSocs: http://www.greensocs.com/

[Grötker02] Grötker, T., Liao, S., Martin, G., Swan: System Design with SystemC, Springer, 2002

[GSM] Digital cellular telecommunications system (Phase 2) (GSM);
Enhanced Full Rate (EFR) speech transcoding, EN 301 245 del ETSI,
http://www.etsi.org/WebSite/Technologies/Codecs.aspx

[H264] H.264 : Advanced video coding for generic audiovisual services, UTI Recommendation, http://www.itu.int/rec/T-REC-H.264-201003-I/en

[Hadjiyiannis97] G. Hadjiyiannis, S. Hanono & S. Devadas. ISDL: An Instruction Set Description Language for Retargetability. Design Automation Conference, 1997.

[Hartoog97] M. Hartoog J.A. Rowson, P.D. Reddy, S. Desai, D.D. Dunlop, E.A. Harcourt & N. Khullar. Generation of Software Tools from Processor Descriptions for Hardware/Software Codesign. Design Automation Conference, 1997.

[Hassan05] M.A. Hassan, K. Sakanushi, Y. Takeuchi and M. Imai: "RTK-Spec TRON: A simulation model of an ITRON based RTOS kernel in SystemC", Proceedings of the Design, Automation and Test Conference, IEEE, 2005.

[He05] Z. He, A. Mok and C. Peng: "Timed RTOS modeling for embedded System Design", Proceedings of the Real Time and Embedded Technology and Applications Symposium, IEEE, 2005.

[Henke02]J. Henke and Y. Li: "Avalanche: An Environment for Design Space Exploration and Optimization of Low-Power Embedded Systems", Ieee Transactions On Very Large Scale Integration (Vlsi) Systems, 2002

[Hergenham00] A. Hergenham, W. Rosenstiel: "Static Timing Analysis of Embedded Software on Advanced Processor Architectures" DATE, 2000.

[Herrera03] F. Herrera, H. Posadas, P. Sanchez & E. Villar: "Systematic Embedded software generation from SystemC", "Embedded Software for SoC", Kluwer Academic Publishers, 2003.

[Honda04] S. Honda, T. Wakabayashi, H. Tomiyama and H. Takada: "RTOS-centric HW/SW cosimulator for embedded system design", Proceedings of CoDes-ISSS'04, ACM, 2004.

[HRT] High Resolution Timers project, http://sourceforge.net/projects/high-res-timers

[Hu06] W. Hu, T. Chen, B. Xie, and Q. Shi, "Embedded Real-Time Linux on Chip: Next Generation Operating System for Embedded System", Proc. of Real-time Linux workshop, 2006.

[Hwang08] Y. Hwang, S. Abdi, D. Gajski. Cycle-approximate Retargetable Performance Estimation at the Transaction Level. DATE, 2008

[Hyades] ITEA Hyades home page; http://www.hyades-itea.org.

[ImpulseC] ImpulseC, Impulse Accelerated Technologies, http://www.impulseaccelerated.com/

[Ipek06] E. Ipek, S. A. McKee, R. Caruana, B. R. de Supinski and M. Schulz. "Efficiently exploring architectural design spaces via predictive modeling". SIGOPS, 2006.

[ITRS07] ITRS - Executive Summary. http://www.itrs.net/Links/2007ITRS/Home2007.htm.

[Joseph06] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In HPCA '06.

[Kaul99]M. Kaul & R. Vemuri: "Temporal Partitioning combined with Design Space Exploration for Latency Minimization of Run-Time Reconfigured Designs", DATE ,1999

[Kanellos03] Kanellos, M.: "Moore's Law to roll on for another decade", CNET News, 2003, http://news.cnet.com/2100-1001-984051.html

[Ke07] H. Ke, D. Zhongliang: "Verification of AMBA Bus Model Using System Verilog", ICEMI'2007.

[Kelly75] F.P. Kelly: "Networks of Queues with Customers of Different Types", Journal of Applied Probability, Vol. 12, No. 3, 1975

[Kempf06] T. Kempf, K. Karuri, S. Wallentowitz, G. Ascheid, R. Leupers, H. Meyr. A SW Performance Estimation Framework for Early System-Level-Design Using Fine-Grained Instrumentation. DATE, 2006

[Kirchsteiger08] C.M. Kirchsteiger, H. Schweitzer, C. Trummer, C. Steger, R. Weiβ, M. Pistauer. A Software Performance Simulation Methodology for Rapid System Architecture Exploration. ICECS, 2008

[Khare99] A. Khare, N. Savoiu, A. Halambi, P. Grun, N. Dutt, A. Nicolau, "V-SAT: A visual specification and analysis tool for system-on-chip exploration". Proc of Euromicro, 1999

[Knudsen96]P. Knudsen and J. Madsen, "PACE: A Dynamic Programming Algorithm for Hardware/Software Partitioning,", International Workshop on Hardware/Software Co-design, 1996.

[Kriaa06] L. Kriaa et al, "Service Based Component Design Approach for Flexible Hardware/Software Interface Modeling", IEEE International Workshop on Rapid System Prototyping, 2006

[Leupers99] R. Leupers, J. Elste, and B. Landwehr. Generation of interpretive and compiled instruction set simulators. Asia South Pacific Design Automation Conference , 1999.

[Leyva06] L. E. Leyva, P. Mejia, and D. de Niz, "Predictable Interrupt Management for Real Time Kernels over conventional PC Hardware", Proc. of the RTAS, 2006.

[Lu10] Y. Lu, S. Sezer, J. McCanny, "TLM2.0 Based Timing Accurate Modeling Method for Complex NoC Systems", ISCAS, 2010

[LuSt99] T. Lundqvist and P. Stenstrom. "An integrated path and timing analysis method based on cycle-level symbolic execution". Real-Time Systems, 1999.

[M3Explorer] Multicube-Explorer, Politécnico de Milán, http://home.dei.polimi.it/zaccaria/multicube_explorer_v1/Home.html

[Malik97] S. Malik, M. Martonosi, Y. Li, "Static Timing Analysis of Embedded Software", DAC, 1997.

[Mead70] "The Technical Impact of Moore's Law". IEEE solid-state circuits society newsletter. 2006. http://www.ieee.org/sscs-news.

[Mehra94] R. Mehra and J. Rabaey, "Behavioral Level Power Estimation and Exploration," International Workshop on Low Power Design, 1994.

[ModeFRONTIER] ModeFRONTIER, ESTECO, http://www.esteco.com/home.html

[Molnar] Ingo Molnar real-time preempt patch, http://people.redhat.com/mingo/realtime-preempt/

[Moore65] Moore, G. E.: "Cramming more components onto integrated circuits". Electronics Magazine. 1965, from ftp://download.intel.com/museum/Moores_Law/ Articles-Press_Releases/Gordon_Moore_1965_Article.pdf

[Moore75] Moore, G.E.: "Progress in digital integrated electronics", IEEE International Electron Devices Meeting, IEDM Technical Digest 1975, pp. 11-13

[Mueller00] F. Mueller, "Timing analysis for instruction caches", International Journal of Time-Critical Computing Systems, Kluwer, 2000.

[Mueller09] D. Mueller-Gritschneder, H. Graeb and U. Schlichtmann: "A Successive Approach to Compute the Bounded Pareto Front of Practical Multi-objective Optimization Problems". SIAM Journal of Optimization, 2009.

[Niehaus97] D. Niehaus, R. Menon, S. Balaji, F. Ansari, J. Keimig, and A. Sheth. "Microsecond resolution timers for Linux", 1997. http://www.ittc.ku.edu/utime/.

[Niemann96] R. Niemann & P. Marwedel: "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming" ,ED&TC, 1996

[Nohl02] A. Nohl, G. Braun, O. Schliebusch, R. Leupers, H. Meyr & Andreas Hoffmann, "A Universal Technique for Fast and Flexible Instruction-Set Architecture Simulation", DAC, 2002

[OCP] Socket Transaction Language, OCP-IP Libraries, http://www.ocpip.org/faqs_simulation_and_test.php

[OPTS] Open POSIX Test Suite 1.5.0, http://posixtest.sourceforge.net/

[OR1000] OpenCores – Project: OpenRISC 1000, www.opencores.org/projects/or1k

[Oyamada07] M. Oyamada, F.R. Wagner, M- Bonaciu, W. Cesario, A. Jerraya. Software Performance Estimation in MPSoC Design. ASP-DAC, 2007

[Peixoto97]H. P. Peixoto and M. E Jacome: "Algorithm and Architecture-level Design Space Exploration Using: Hierarchical Data Flows", 1997

[Perez04] I. Perez, S. Searty, D. P. Howell and B. Hu. "I would hate user space locking if it weren't that sexy...". Proc of OLS, 2004.

[Perrier03] V. Perrier: "CoFluent joins OSCI and SPIRIT", Cofluent Design, http://www.cofluentdesign.com/index.php/news/143/121/ CoFluent-joins-OSCI-and-SPIRIT.html, 2003

[Popovici10] Popovici, K. and Jerraya, A. "Virtual Platforms in System-on-Chip Designs" , DAC.COM KNOWLEDGE CENTER ARTICLE DAC 47, 2010

POSIX] POSIX® 1003.1 (and former 1003.2) standards, ISO/IEC 9945, IEEE Standards Association

[PowerVM] IBM PowerVM, http://www-03.ibm.com/systems/power/software/virtualization/

[Puschner89] P. Puschner, C. Koza, “Calculating the maximum execution time of real-time programs”, The Journal of Real-Time Systems, N. 1, 1989.

[Qemu] Qemu, http://www.qemu.org/

[RaMu05] H. Ramaprasad and F. Mueller. “Bounding worts-case data cache behavior by analytically deriving cache reference patterns”. Real-Time and Embedded Technology and Applications Symposium, 2005.

[Real10] S. Real, H. Posadas, E. Villar, "L2 Cache Modeling for Native Co-Simulation in SystemC", Symposium of Indus

[Reddy95] Reddy, Madhu. "ORBs and ODBMSs: Two Complementary Ways to Distribute Objects." Object Magazine 5, 3 (June 1995): 24-30.

[RedHawk] RedHawk Real-time Linux, http://www.ccur.com/

[RMI97] Remote Method Invocation. http://java.sun.com/products/jdk/1.1/docs/guide/rmi (1997).

[RTMS] Enterprise Real-Time Management System (RTMS), http://www.fsmlabs.com

[Sangiovanni01] A. Sangiovanni-Vincentelli and G. Martin: “Platform-based design and software design methodology for embedded systems”. DAC, 2001

[Sangiovanni04]. Sangiovanni-Vincentelli, A.: “The Tides of EDA”, keynote at the 40th Design Automation Conference, 2004, http://embedded.eecs.berkeley.edu/research/hsc/class/papers/d6sang.lo.pdf

[Sawaragi85] Y. Sawaragi, H. Nakayama and T. Tanino: “Theory of Multi-objective Optimization”, Journal on Mathematics in Science and Engineering, 1985.

[Schirner06] G. Schirner, R. Dömer: “Quantitative Analysis of Transaction Level Models for the AMBA Bus”, DATE’06 [Schirner07] G. Schirner, A. Gerstlauer, and R. Dömer. “Abstract, Multifaceted Modeling of Embedded Processors for System Level Design. Asia and South Pacific Design Automation Conference (ASP-DAC), 2007.

[Schnerr05] J. Schnerr, O. Bringmann & W. Rosenstiel: “Cycle Accurate Binary Translation for Simulation Acceleration in Rapid Prototyping of SoCs”, DATE, 2005.

[Schnerr08] J. Schnerr, O. Bringmann, A. Viehl, W. Rosenstiel. High-Performance Timing Simulation of Embedded Software. DAC, 2008

[Sewal07] R. Sewal & H. Stump, “Gain Abstraction and Accuracy from RTL Power Estimation Highlights”, Electronic Design, 2007, http://electronicdesign.com/article/eda/gain-abstraction-and-accuracy-from-rtl-power-estim.aspx

[Shafi03] H. Shafi et al, "Design and validation of a performance and power simulator for PowerPC systems", IBM Journal Research and Development, Vol 47, No 5/6, September–November 2003

[Sheldon07] D. Sheldon, F. Vahid, and S. Lonardi. Soft-core processor customization using the design of experiments paradigm. In DATE ’07.

[Shiva06] K.P.Shiva Kumar and A.Satya. “RTOS Core Concepts and RT-safe Synchronization Mechanisms”, Proc. of Real-time Linux workshop, 2006

[Sinha01] A. Sinha, A.P. Chandrakasan, “Joule-Track, A Web Based Tool for Software Energy Profiling”, DAC, 2001.

[Skyeye] SkyEye web page, http://www.skyeye.org/index.shtml

[Soulard07] Philippe Soulard, Yijun Xu. :”Accurate System Level Power Estimation through Fast Gate-Level Power Characterization“, Design & Reuse, 2007.

[Steinke95] Steinke, Steve. "Middleware Meets the Network." LAN: The Network Solutions Magazine 10, 13 (December 1995): 56.

[Synopsys] Platform Architect tool, Synopsys, http://www.synopsys.com/Systems/ArchitectureDesign/pages/PlatformArchitect.aspx

[Timesys] Timesys, http://www.timesys.com/

[Tkach94] Tkach, Daniel & Puttick, Richard. Object Technology in Application Development. Redwood City, CA: Benjamin/Cummings Publishing Company, 1994.0

[TLM2] SystemC-TLM standard , version 2.0. OSCI, http://www.systemc.org

[Tomiyama01] H. Tomiyama, Y. Cao and K. Murakami: “Modeling fixed-priority preemptive multi-task systems in SpecC”, Proceedings of the 10th Workshop on System And System Integration of Mixed Technologies (SASIMI’01), IEEE, 2001.

[Turing50] Turing, A.: "Computing Machinery and Intelligence", Mind journal, October, 1950

[UQBT] UQBT, http://www.itee.uq.edu.au/~cristina/uqbt.html

[VeXu02] X. Vera and J. Xue. "Let's study whole program cache behaviour analytically". In Proceedings of International Symposium on High-Performance Computer Architecture (HPCA 8), 2002.

[Wade 94] Wade, Andrew E. "Distributed Client-Server Databases." Object Magazine 4, 1 (April 1994): 47-52.

[Wallnau96] Wallnau, Kurt & Wallace, Evan. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA)," 168-178. Proceedings of the OOPSLA'96. San Jose, CA, October 6-10, 1996. New York

[WhMu97] R. T. White, F. Muller, C. Healy, D. Whalley, and M. Harmon. “Timing analysis for data caches and set-associative caches”. In Proceedings of Third IEEE Real-Time Technology and Applications Symposium (RTAS'97), 1997.

[Wild06] T. Wild, A. Herkersdorf, R. Ohlendorf. Performance Evaluation for System-on-Chip Architectures using Trace-based Transaction Level Simulation. DATE, 2006

[Wilhelm08] R. Wilhelm et al. :”The Worst-Case Execution Time Problem— Overview of Methods and Survey of Tools”, Trans. on Embedded Computing Systems, 2008

[Willink01] Edward D. Willink. “Meta-Compilation for C++”. PhD thesis, Computer Science Research Group, University of Surrey, June 2001.

[Wolf04] F. Wolf, J. Staschulat and R. Ernst: ”Hybrid Cache Analysis in Running Time Verification of Embedded Software”, Design Automation for Embedded Systems, 2004

[Yaghmour] K. Yaghmour, “Adaptative Domain Environment for Operating Systems”, http://opersys.com/ftp/pub/Adeos/adeos. pdf

[Yoo02] S. Yoo, G. Nicolescu, L. Gauthier, A. Jerraya, “Automatic generation of fast timed simulation models for operating systems in SoC design”, Proc. of DATE, IEEE, 2002.

[Zabel09] H. Zabel, W. Müller, and A. Gerstlauer, “Accurate RTOS modeling and analysis with SystemC”, in “Hardware-dependent Software: Principles and Practice”, W. Ecker, W. Mü ller, and R. Dömer, Eds. Springer, 2009.