SIMULATION PLATFORM IN TLM OF SYSTEM ON CHIP USING RETARGETABLE ISS

M.KHETATBA & R. BOUDOUR

Dept d'informatique, Université d'Annaba, Algérie
khetatbam@yahoo.fr,racboudour@yahoo.fr

ABSTRACT

System-on-Chip (SoC) designs are increasingly becoming more complex. One of the major constraints is the time to market New design methods are necessary, and the tendency is with the integration of the software and hardware parts on the same chip. Efficient on-chip communication architectures are critical for achieving desired performance in these systems Thus, the development of codesign's modern methods and the appearance of hardware description languages (HDL) based on C/C++ such as SystemC or specC allowing to employ the same language to describe the software and the hardware, and returning of this fact easier and more effective Co-simulation. These methods would be able to generate an optimal solution starting from a functional specification by reducing the time and the cost of the design. Thus, one of the main objectives of this paper is the development of a SystemC platform for multiprocessors architectural exploration at the compromise level (TLM) by using SystemC/TLM. It must lead to partition system into hw/sw and also to validate it by simulation or to move easily modules from hardware to software (or vice versa) during the architectural exploration. Except for the software task priorities that could be modified, we only need to recompile and simulate.

KEYWORD: ISS, RTOS, SystemC, TLM, initiator, target, socket, generic payload, interconnect component, Loosely-timed, approximately-timed communication manager

1 INTRODUCTION

The principle of systems on chip consists in gathering on the same integrated circuit all the data-processing components. As the hardware does not exist with the starting of the project, simulation tools are necessary to develop the software on a simulated hardware, to validate the hardware design, and to study system performances and behavior. It appears today that SystemC is a very powerful language that allows to describe systems at different levels of abstraction, i.e. from transaction-level down to the gate-level [1]. SystemC allows the use of a common language for software and hardware specifications, and the simulation of the whole system. However, during the simulation, the scheduler, responsible for determining which thread will run next, manages identically both software and hardware threads. It means that systems with hard real-time constraints requiring an RTOS (Real-Time Operating System) based on a preemptive priority-based kernel cannot be modelled naturally. Such RTOS [2] provide a very useful abstraction interface between applications with hard real-time requirements and the target system architecture. As a consequence, availability of RTOS models is becoming strategic inside hw/sw (hardware/software) codesign environments [3].

Thus, one of the main objectives of this paper is the development of a SystemC platform for multiprocessors architectural exploration; as well as the realization of the communication by using TLM 2.0 between the various components of the system.. It must lead to partition system into hw/sw and also to validate it by simulation or to move easily modules from hardware to software (or vice versa) during the architectural exploration. Except for the software task priorities that could be modified, we only need to recompile and simulate.

In the remaining sections of this paper, in section 2 of the article, we particularly discuss the various works in progress in the field of codesign partitioning, cosimulation and the interest of TLM. In section 3, we present the principles of TLM in SystemC, an overview of TLM 2.0, partitioning with the importance of the estimators. and SystemC gaps. In section 4, we present our architecture of simulation by exposing the objectives of the platform, then the hardware and software components of the platform. The Paper is completed with conclusion and future works.

2 RELATED WORKS

System level modeling has become an important issue, as a means to improve the hybrid design process. SLDLs (System Level Description Language) or capturing such models have been developed (e.g. SystemC [4], SpecC [5]). Different frameworks for system level design and software synthesis have been developed.

Indeed, for the simulation of software modules, the SystemC simulator does not offer all the necessary functionalities, such as preemption or scheduling by priority, generally present in any RTOS: a joint refinement of the software and hardware parts is thus a tedious task in SystemC 2.0. A possible area for consideration in extending the SystemC core language is to provide better software support. Unfortunately, the specification for this future release is not yet available. Consequently with SystemC 2.0, there are tools able to synthesize hardware modules, but this is done at the expense of the software part.

A possible refinement is to simulate more accurately the software interaction with the hardware, using ISS. The Instruction Set Simulator (ISS) accepts an assembler code obtained by the cross-compilation of the software modules. Several researches were already undertaken to integrate an ISS with SystemC [6,7]. The resulting simulation is reliable and realistic because it depends on the actual architecture. This decreases largely the number of delta cycles necessary and accelerates significantly the simulation. On the other hand, these solutions focus more on the simulation aspects than on the partitioning methodology: the use of an ISS seems to take place after the partitioning phase. Furthermore, no proposal suggests the use of an RTOS making it possible to schedule several software modules on the ISS, so programs being executed remain of the foreground/background type, which is today a substantial limitation in SoC (System on Chip) designs for example.

Nicolescu and al[8] propose a component-based SoC methodology using a wrapper generation flow to automatically link various heterogeneous cores of different abstraction levels in the same simulation. Fummi and al.[6] present two cosimulation methodologies, using SystemC on an instruction set simulator (ISS) as a processor model. The first methodology, GDB-Kernel (GNU debugger kernel), works at the SystemC kernel level and exploits the GNU suite's potentialities. The second, Driver-Kernel, uses features offered by the operating system running on the ISS. Both the Nicolescu and al. and Fummi and al. methodologies require that modules simulating the software be modified specifically; it is thus not possible to easily transfer modules between hardware and software to test various configurations. The MPARM multiprocessor simulation platform has been recently proposed in [9] for cycle-accurate power performance estimation. The MPARM platform has been used in [10] for analyzing several on-chip communication or a communication channel. Cycles Accurate System Simulator (CASS) [11], proposed by Denis Hommais and Frederic Pétrot, targets the fast simulation of integrated systems on chip. Jerome Chevalier and al [7] proposed the method "SPACE" which makes it possible to have the optimal partitioning of a system.

Traditionally, systems were captured at a cycle and pin-accurate level in RTL and then simulated for performance estimation before synthesis. However SoC designs today are large and very complex, so not only does it take a lot of time to capture them in RTL, but the resulting simulation speed is too slow for meaningful performance exploration. To overcome these limitations, system designers have raised the abstraction level of system models. High level models (usually written in C/C++) give an early estimate of the system characteristics before committing to RTL development

The rise in complexity, size and heterogeneity of modern embedded system designs has pushed modeling to new abstraction levels above RTL. Transaction level modeling using SystemC is emerging as a new paradigm for system modelling. The introduction of Transaction Level Modeling (TLM) allows a system designer to model a complete application, composed of hardware and software parts, at several levels of abstraction. The simulation speed of TLM is orders of magnitude faster than traditional RTL simulation. TLM has gained a lot of attention recently ever since it was introduced [1] as part of high level SystemC [2] modeling initiative. Several use models and design flows [12, 13] have been presented centering around TLM. The separation between the communication and the computation aspects of a design can be effectively achieved using transaction level models (TLMs). The concept of TLM first appeared in the domain of system modeling languages, like SystemC [1] and SpecC [5]. In [1], a TLM is defined as a model where communication between modules is modeled in a way that is accurate in terms of what is being communicated, but not in a way that is structurally accurate (i.e., the actual wires and pins are not modeled). They have also insisted that the communication between modules be modeled using function calls.

An overview of TLM is presented by Cai [14] and an overview of TLM flows by Donlin [12]. Different approaches to SystemC modeling are presented by Colgan [15] and Kogel [16]. The differences between RTL and TLM have been studied by Calazans [17]. Commercial tools such as the Incisive Verification Platform [18], ConvergenSC System Designer [19] and Cocentric System Studio [20] have also started adding support for system modeling at the higher TLM abstraction, in addition to lower level RTL modeling.

3 BASIC CONCEPTS

3.1 Principles of TLM in SystemC

SystemC is a system modeling language that can model and operate hardware at system-level. SystemC can easily express a complex SoC core at a high level while having all the merits of hardware description languages. It was developed using C++ classes. Hence, SystemC can be effectively used for simulation environment in checking not only hardware operation but also software one. Also, it supports TLM (Transaction-Level Modeling). SystemC class libraries provide essential classes for modeling system structure. They supports hardware timing,concurrency, and reaction, which are not included in standard C++. SystemC allows developers to describe hardware and software, and their interface under C++ environment.

The goal of transaction-level modeling is to speed up simulation time and the time it takes to develop the models.The transaction-level model is built as set of interfaces that define how models communicate. In its most primitive form the TLM basic interfaces provide with the fundamental communication and synchronization constructs that can be used to create TLM models. The basic idea of Transaction Level Modeling (TLM) is to establish communication through function calls that represent transactions rather than signals as at the Register Transfer Level (RTL). Transaction Level Modeling (TLM) has been introduced in the recent past as a modeling style to describe communication channels at a higher abstraction level with respect to Register Transfer Level. Although Transaction Level (TL) models offer high simulation speed, in some cases, they do not capture enough details about on-chip behavior. At TLM level, the system bus behavior can be viewed as an abstract channel independent of the target bus architecture or protocol implementation.

3.2 TLM 2.0

The focus of OSCI TLM-2.0 in particular is the modeling of on-chip memory-mapped busses. TLM-2.0 has a layered structure, with the lower layers being more flexible and general, and the upper layers being specific to bus modelling. In TLM-2.0, an initiator is a module that initiates new transactions, and a target is a module that responds to transactions initiated by other modules. A transaction is a data structure (a C++ object) passed between initiators and targets using function calls. The same module can act both as an initiator and as a target, and this would typically be the case for a model of an arbiter, a router, or a bus. In order to pass transactions between initiators and targets, TLM-2.0 uses sockets. An initiator sends transactions out through an initiator socket, and a target receives incoming transactions through a target socket. A module that merely forwards transactions without modifying their content is known as an interconnect component. An interconnect component would have both a target socket and an initiator socket. The default transaction type for the socket classes, implied in the absence of any template arguments, is tlm_generic_payload. The generic payload is an important part of the TLM-2.0 standard because it is another of the keys to achieving interoperability between transaction level models. The generic payload serves two closely-related purposes. It can be used as a general-purpose transaction type for abstract memory-mapped bus modeling when you are not concerned with the exact details of any particular bus protocol, offering immediate interoperability between models off-the-shelf. Alternatively, the generic payload can be used as the basis for modeling a wide range of specific protocols at a more detailed level, the beauty of this approach being that it is relatively easy to bridge between different protocols when both are built on top of the same generic payload type. The generic payload supports two commands, read and write. The goal of the OSCI TLM2.0 standard is to ease and enable interoperability between high-level SystemC components. TLM 2.0 defines modeling styles, several interfaces, a generic payload, and more than 150 rules to define the expected interface behavior during simulation. While some of those rules clarify the semantic of a transaction, a large part specify restrictions and expected behavior.TLM2.0 defines two modeling styles. The Loosely Timed (LT) modeling style is targeted for system and platform models, where timing and data are only loosely connected. For the correct functionality, it is not important to get specific data at exactly the specified time. Resource conflicts and contentions are not modeled in the LT modeling style. For the LT modeling style, TLM2.0 defines two timing points, the beginning time and the end time of a transaction. The second modeling style is called Approximated Time (AT). With this modeling style, it is easy to model resource contention and arbitration. It is used for systems where the dependency of timing and data is very strong, like in systems with very stringent and specific hard real-time requirements. The AT modeling style defines four timing points for a transaction: the begin request, end request, begin response and end response. Therefore, systems with delay due to concurrent resource access conflicts and arbitration can be modeled very accurately.

Beside the two modeling styles, TLM2.0 also defines two interfaces. One is a blocking transporter called b_transport (TRANS&, sc_time&). The specification defines that the caller module does not continue any processing while issuing a transport call. It waits until the transport calls return before continuing execution. In addition, TLM2.0 defines a non-blocking interface. Two non-blocking transport calls exist, a forward path nb_transport_fw (TRANS&, PHASE&, sc_time&) and the backward path nb_transport_fw (TRANS&, PHASE&, sc_time&). Both transports return immediately so the caller module can continue its processing while it waits for the recipient to respond. TLM 2.0 defines a direct memory interface (DMI), as well as a debug transport interface.

3.3 Partitioning

The hw/sw partitioning allows the transformation of the parts specifications system into hw and sw components. Partitioning is a well-known NP-complete problem depending on a big number of parameters. To solve this problem, most automatic methods reduce the number of of parameters and use a heuristic based on a cost function pondered by the retained criterias and checked constraints. However, the feasibility of these methods depends exclusively on the accuracy of parameters for the values to be optimized, in the main part execution time, hardware size, power consumption, etc. for every part of a design [21].

Global optimization is the task of finding the absolutely best set of parameters to optimize a goal function. In general, there are solutions that are locally optimal but not globally optimal. Consequently, global optimization problems are typically quite difficult to solve exactly. The criteria chosen for this estimation vary from a tool to another. With each criterion is associated a metric value just as a factor which is used to balance them and which depends on technology used and the applicability.It is thus difficult to define an estimate function which is realistic even for a specific applicability.

Metrics allow evaluation of design quality and design space exploration. Partitioning into hw and sw of a system is performed by evaluating a cost function assembled from competing metrics (see figure 1). These metrics are obtained by static and dynamic code analysis on the highest level of abstraction - the algorithmic description of the system. The common metrics can be applied on all objects of the specification. For example a size, execution time, cost, and so on. Closeness metric measures the probability that two closest objects of the specification should be implemented on the same system component. For example, if two system objects use the same data, execute sequentially, and have the same hardware requirements, then implementing them on the same system component would likely lead to a good design [21,22].

A CDFG (Control Data Flow Graph) is an abstraction of the real system. In general an algorithm in form of sequential code can be decomposed into its control flow graph (CFG), built up of interconnected objects. Each object (for example basic block) contains a sequence of data operations ended by a control flow statement as last instruction. This sequence of data operations forms itself a data flow graph (DFG) [22].

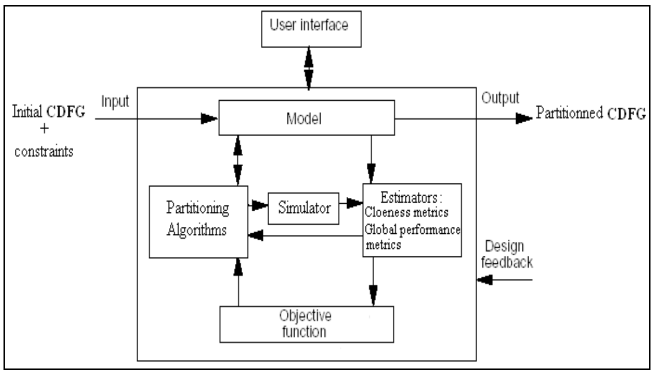


Figure 1: Typical partitioning system configuration

Estimators are necessary in partitioning step. Indeed, the values given by the estimators inform the designers about the quality of the found solutions, in order to predict the results of the design without going until the total realization. It is a fast estimate of the performances or characteristics of architecture. The problem is then to find the metric ones to evaluate the solutions, and these metrics is generally limited to the technological criteria.

This is a summary of software estimation used in the software simulator:

- Execution time (RunTime)
- Create basic blocks (SystemC) and compile them to a specific instructions set.
- Estimation of blocs execution time.
- Calculate specification's execution time
- Data memory size:
- Program memory size
- Task maximum hold time (context switch, preemption)
- Maximum number of preemptions for each task

A good system performance measure is a difficult task, thus it depends of the modelling of target processor and the instructions flow execution. For simple processors, it is possible to calculate execution time of each instruction on target processor. The global processor runtime is calculated by multiplying execution frequency of each instruction by the execution time of that instruction.

Software simulator, provides a cost estimate of the partition result while using like metric evaluation: Execution time, memory size, Memory access time and Context switching

3.4 SystemC Gaps

Let us start from a system description in a set of SystemC modules. After partitioning, the hardware modules will be simulated by the SystemC hardware simulator, while the software modules will be carried out by the SystemC software simulator (see figure 2). But, in spite of the fact that we can qualify the language SystemC on the level system, several gaps concerning the modeling of the software part of an application remain. All the devices are in place to allow refinement of the hardware, but not for the software. The problem comes owing to the fact that the scheduler is the same one for the software and the hardware during simulations. Concurrency is not modelled, neither préemptive scheduling nor the support of the priorities. It is thus impossible to manage the systems with hard real time constraints. However, the real time systems are used much in the systems on chip. To be able to model them as parts of a complete system is an obvious strategic asset for the codesign.

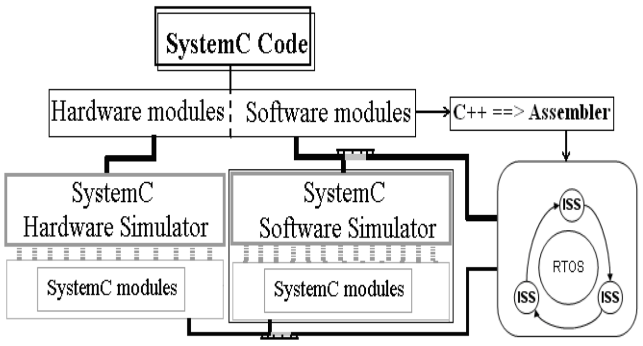


Figure 2: SystemC simulator

We propose an integrated ISS with a RTOS core in order to replace the weak software simulator of SystemC. ISS in a basic architecture of simulation in SystemC which could be used as assistance tool in the systems on chips design starting from raised levels of abstraction.

4 SIMULATION PLATFORM

The approach we present here helps architecture exploration for SoC development. Our platform allows simulation and performance assessments to facilitate architectural exploration, particularly for hardware-software partitioning. The building of SystemC modules in platform follows a coding guideline that facilitates transparent module transfers between hardware and software. This approach wants to be innovating in the direction where two simulations are carried out on level TLM, the first quick to check only the system functionality by using a particular case of the modeling style LT (Loosely-timed) and second is precise by using the style of modeling AT (approximately-timed). The platform facilitates the work of the designer when this last wishes to test various software/hardware configurations, because on this level it is possible to move the modules of the software part towards the material part and conversely, with a minor effort and a minimum time on behalf of the designer. To guide the originator in his partitioning, it is possible for him to carry out simulations of its application.

In summary, no method currently exists that makes it possible to easily simulate at high level various hardware/software configurations, in order to obtain results leading to the optimum partition of a system. Three main conditions are required to reach this goal:

The possibility of moving modules between the software part and the hardware part without changing modules' code. A simulation of the whole system giving realistic results to validate or invalidate a partition choice.

A multiprocessor approach giving a realistic embedded system simulation.

Our architecture is made up of the following hardware components(see figure 3) : processor,hardware modules , memory,interrupt manager , timer,stop peripheral

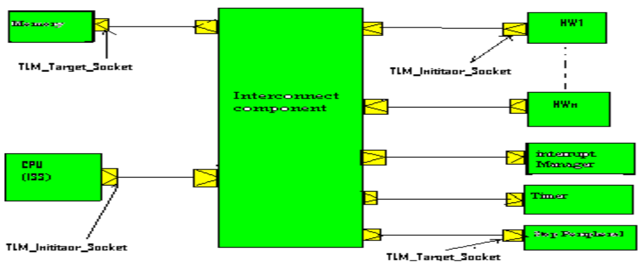


Figure 3: Hardware components

The software components (figure 4) are: software modules, RTOS, communication manager

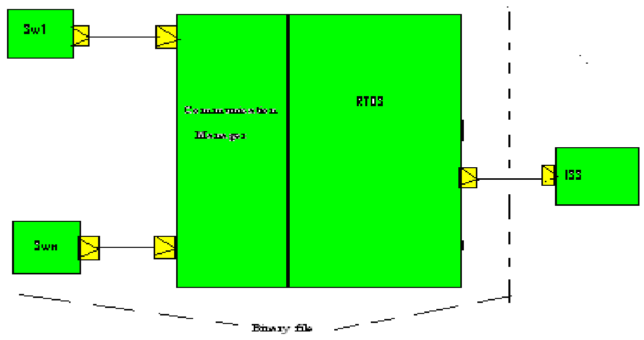


Figure 4: Software components

4.1 A Retargetable ISS

An instruction set simulator runs a program by simulating the effects of each instruction on a target machine, one instruction at a time. Instruction set simulators are attractive for their flexibility: they can in principle, model any computer, gather any statistic, and run any program that target architecture would run. Additionally, it can provide debug access to internal state information of the processor that are usually hidden in a real hardware. Using an ISS eases software development and debugging by providing a deterministic execution; especially useful for analyzing race conditions.

Retargetability is now an important concern, particularly in the area of the embedded systems and SoC design. A retargetable ISA (Instruction Set Architecture) simulator requires a generic model, supported by a language, to describe the architecture and its instruction set. The simulator uses the architecture description to decode instructions of the input program and execute them. The challenge is to have a model that is efficient in terms of both quality of the description and performance of the simulator. To have a high quality description, the model must easily capture the architectural information in a natural, compact and manageable form for a wide range of architectures. On the other hand, to generate a high performance simulator and to reduce the operations that the simulator must do dynamically at run time, the model should provide as much static information as possible about the architecture and its instruction set.

Designing an efficient model that captures a wide range of architectures is a hard problem because such architectures have different instruction-set format complexities. There is a tradeoff between speed and retargetability in ISA simulators. Some of the retargetable simulators use a very general processor model and support a wide range of architectures but are slow, while others use some architectural or domain specific performance improvements but support only a limited range of processors. Also in some description languages, deriving a fast simulator requires lengthy descriptions of all possible formats of instructions.

Our Software simulator as described in Figure 5 has the main features:

- multiple processors support
- running code, written in assembler
- extensive configuration
- simulating x86 and 68000 processors
- step by step execution
- visual representation of the registers state
- detailed simulation results
- OS window showing all events with precise timing (hour, min, sec, msec)

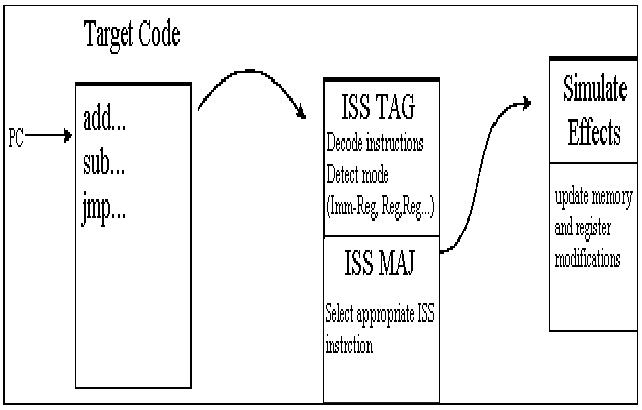


Figure 5: Principles of software simulator

Figure 5 illustrates the main principles. The target code consists of instructions for the target architecture. Execution is modelled by simply incrementing the intermediate code pointer. Branch instructions must calculate a new intermediate pointer first.

SystemC makes it possible to suitably simulate hardware, but not software. We thus chose to use of ISSs in our architecture to jointly simulate the software part with the hardware part. For our architecture, our ISS is encapsulated in a SystemC module (sc_module) and thus SystemC deals with making carry out the hardware processes and the ISS. As the ISS is also itself a simulator, this SystemC code (software) is not carried out by the same simulator as the SystemC code of the hardware part. Thus we have two simulators of SystemC: SystemC hardware simulator of SystemC and the software simulator which we designed

4.2 Memory

It contains the binary code, which gathers the code of the operating system and the software modules and tasks. The code memory is loaded in starting from a file at the time of simulation initialization. This memory is also used to store software dynamic information.

4.3 Interrupt Manager

It informs the processor by activating the interruption. The processor must then communicate with the manager by the interconnect component in order to know the identity of the module which caused the event.

4.4 Timer

Any operating system which allows parallel tasks needs a timer to indicate the moment to him when it must pass from one process to the other. It provides as a basis interruptions being used of time to the RTOS which is carried out on the ISS. The tics of scheduling are essential to have a préemptive core such that of MicroC/OS-II.

4.5 Stop Peripheral

This special peripheral makes it possible to finish simulation. A simple access memory directed to the address of this peripheral generates the execution of the function sc_stop () of SystemC.

4.6 RTOS

A Real-Time Operating System (RTOS) allows realtime applications to be designed and expanded easily. However, the RTOS introduces overhead, which may prevent some real-time systems, such as high-speed packet switches, from working efficiently. As a result,deadlines may be missed. The overhead can be reduced by migrating kernel services such as scheduling, time tick (a periodic interrupt to keep track of time during which the scheduler makes a decision) processing, and interrupt handling to hardware. This will significantly improve the response time and the interrupt latency, provide accurate timing, and increase the CPU utilization.

If more than one behavior is mapped to a processor, the mapped behaviours have to be scheduled. A processor allows only sequential execution at a time.

One scheduling approach is dynamic scheduling as executed by an RTOS on the target system. However, it is not desirable to execute a complete RTOS at an early design stage due to the simulation overhead. Therefore, an abstract RTOS model is used for exploration of different scheduling policies.

µC/OS-II is a core real time that permits to do an execution of several tasks on a microprocessor. We needed a RTOS in order to permit a multiprocessor execution. Our choice was about µC/OS-II because its source code is available, what facilitated the understanding of the working principle of the RTOS in general. In spite of its advantage, µC/OS-II have been deprived of some aspects as the multiprocessors, therefore we rewrote our OS entirely (see figure 6) in order to manage the wanted appearance (See Table 1), without including functionalities of µC/OS-II as the management of the TCP/IP and some aspects of communications of which we didn't have need in our partitioning architecture.

The RTOS is as much as possible independent of the hardware. The main function of our adapted RTOS is the scheduling of software tasks. It offers many scheduling policies,

Table 1: RTOS comparative

Properties	Adapted RTOS	µC/OS-II
Multiprocessors	Yes	No
Priority sharing (tasks)	Yes	No
Heterogeneous	Yes	No

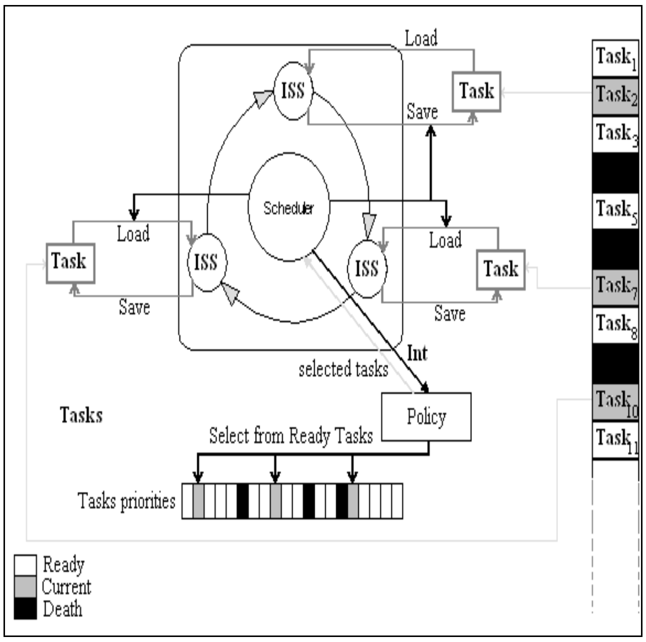


Figure 6: represents the core of the operating system.

A module can be in one of the three states: finished, elected, ready.

4.7 Modules

Modules represent the object code of the SystemC modules belonging to a CDFG (Control Dated flow Graph), intended to be implemented in software and/or hardware. Each module has a single identification number given by the user of our simulation platform. In our case, the grains must contain a code fragment, written in assembler language . The modules implemented in software are tasks carried out by ISSs. For the modules which will be implemented in hardware, they are established to communicate through the interconnect component. Our platform guarantees the communication of the entities, as well as the change of the nature of the modules. The following figure 4.13 watch an example of the modules of code obtained starting from a CDFG, the left part contains those to implement in software, and the right part those to implement in hardware.

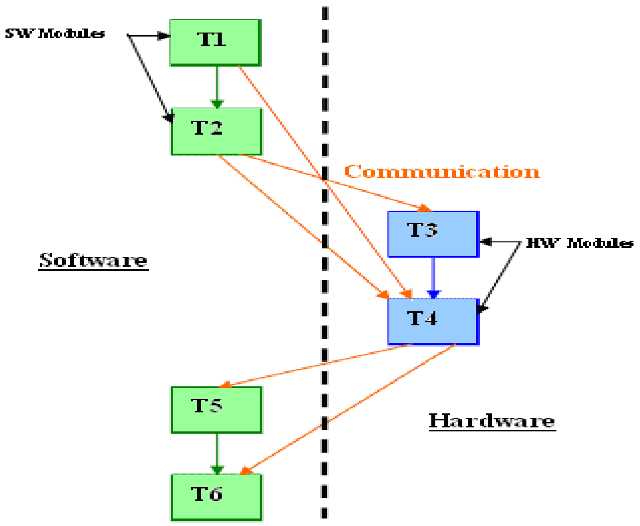
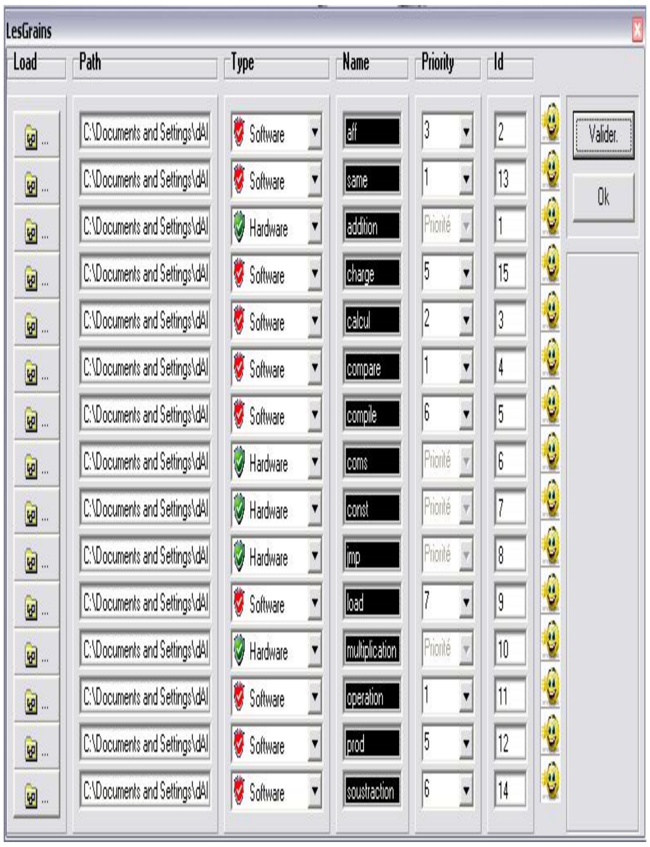


Figure 7: Example of modules faisant partie d'un CDFG.

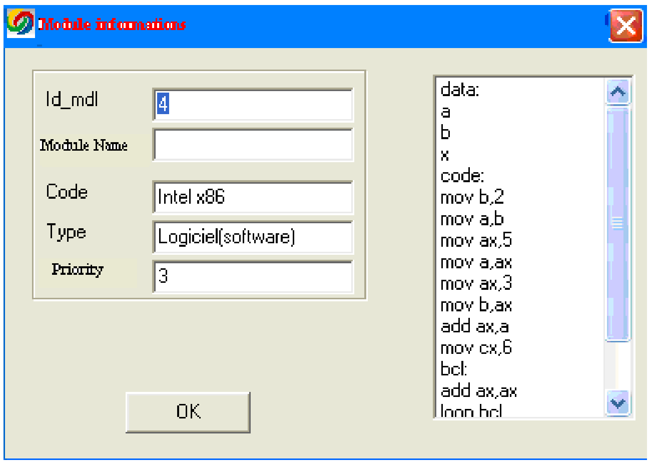
The unit managing the modules allows the choice and the management of the modules, by assisting the user to choose the code modules written and stored on the disc. This unit ensures also the management of the hardware and software modules, by simplifying the passage of a module from the hardware part to the software part and conversely by a simple click (figure 8).



Load	Path	Type	Name	Priority	Id
...	C:\Documents and Settings\dAl	Software	aff	3	2
...	C:\Documents and Settings\dAl	Software	same	1	13
...	C:\Documents and Settings\dAl	Hardware	addition	Priorité	1
...	C:\Documents and Settings\dAl	Software	charge	5	15
...	C:\Documents and Settings\dAl	Software	calcul	2	3
...	C:\Documents and Settings\dAl	Software	compare	1	4
...	C:\Documents and Settings\dAl	Software	compile	6	5
...	C:\Documents and Settings\dAl	Hardware	coms	Priorité	6
...	C:\Documents and Settings\dAl	Hardware	const	Priorité	7
...	C:\Documents and Settings\dAl	Hardware	imp	Priorité	8
...	C:\Documents and Settings\dAl	Software	load	7	9
...	C:\Documents and Settings\dAl	Hardware	multiplication	Priorité	10
...	C:\Documents and Settings\dAl	Software	operation	1	11
...	C:\Documents and Settings\dAl	Software	prod	5	12
...	C:\Documents and Settings\dAl	Software	soustraction	6	14

Figure 8: Modules reset

Figure 9 shows an example of module and some information attached to this module


Module informations

Id_mdl: 4

Module Name:

Code: Intel x86

Type: Logiciel(software)

Priority: 3

```
data:
a
b
x
code:
mov b,2
mov a,b
mov ax,5
mov a,ax
mov ax,3
mov b,ax
add ax,a
mov cx,6
bcl:
add ax,ax
loop bcl
```

Figure 9: Example of module informations

4.8 Interconnect Component

It makes it possible to connect all the hardware modules and the peripherals of the platform and ensures the communications. The simulation of these modules is much faster, because there is not all the arsenal to make carry out software (OS, ISS, etc). All the hardware modules can communicate and send their transactions (hw/hw communication).

The interconnect component has a role to carry out the routing of the transactions; therefore it is necessary that it is equipped with a transactions queue to store the communication calls.

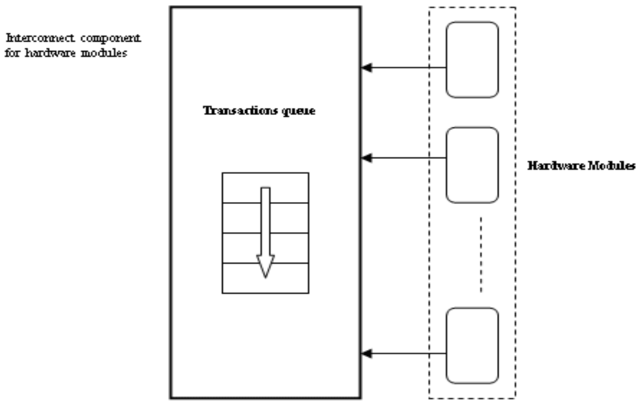


Figure 10: interconnect component

4.9 Communication Manager

It establishes connections between software modules and platform hardware. It answers task requests to communicate with other software or hardware modules. The software communication manager handles communications between any connected modules (hardware or software) and provides the same communication model as the hardware channel. A module connected to this manager will be registered as a software module.

5 CONCLUSION

Thus, according to the TLM 2.0 concepts, modules can be initiators, targets, or targets and initiators at the same time. Thus, in our architecture, All the hardware modules and processor (ISS) are regarded as initiators, on the other hand, the memory, timer, stop peripheral and interrupt manager are regarded as targets.

We also used the two blocking interfaces of transport and not blocking. Thus, we carried out a first validation of the system with a particular case of the style of modeling "loosely_timed", it is the untimed model i.e. the second parameter sc_time of b_transport is null and that to quickly check the functionality of the system and a second validation of the architecture of simulation with the modelling style approximately_timed to have precise results.

We have define, through this article, the importance of the concepts of standard TLM 2.0 to carry out a simulation architecture on TLM level .with the respect of TLM 2.0 rules, we have obtained interopable components. We also see how to integrate a retargetable ISS on the platform which allows simulation an high level of an application M/L, with an aim of validating a partitioning M/L and to have an optimal solution. We also made communicate our software simulator with the hardware simulator of SystemC. To optimize the process of partitioning, works remain to be made. The addition of other estimators for example remains a big step to be achieved.

BIBLIOGRAPHIE

[1] T. Groetker, S. Liao, G. Martin, S. Swan, System Design with SystemC. Kluwer Academic Publishers, 2002

[2] M. Besana, M. Borgatti, Application Mapping to a Hardware Platform through Automated Code Generation Targeting a RTOS: A Design Case Study. pages 41–44.Proc. of DATE Conference and Exhibition Design Forum, March 2003

[3] J. Staunstrup, W. Wolf, Hardware/Software Co-Design, Principles and Practice. Kluwer Academic Publishers, 1997

[4] OSCI. SystemC. www.SystemC.org.

[5] D. D. Gajski, J. Zhu, R. D¨omer, A. Gerstlauer, S. Zhao, SpecC: Specification Language and Design Methodology. Kluwer Academic Publishers, 2000

[6] F. Fummi, S. Martini, G. Perbellini, M. Poncino, "Native ISS SystemC integration for the Co-simulation of multi-processor SoC," DATE 2004, Vol. 1, pp. 564-569.

[7] J. Chevalier, O. Benny, M. Rondonneau, G. Bois, M. Aboulhamid, F.-R. Boyer, "SPACE: A Hardware/Software SystemC modeling platform including an RTOS," Forum on specification and Design Languages, 2003.

[8] G. Nicolescu et al., "Validation in a Component-Based Design Flow for Multicore SoCs". Proc. 15th Int'l Symp.System Synthesis (ISSS 02), ACM Press, pp. 162-167, 2002

[9] M. Loghi, M. Poncino, and L. Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In Proceedins of the 14th ACM Great Lakes symposium on VLSI, pages 410–406. ACM Press, 2004

[10] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a mpsoc environment. In Proceedings of the conference on Design, automation and test in Europe, page 20752. IEEE Computer Society, 2004.

[11] F. Petrot, D. Hommais, and A. Greiner. "Cycle precise core based hardware/software system simulation with predictable event propagation". In Proc. of the 23rd Euromicro Conf., pages 182 -187, Hungary, Sep. 1997.

[12] A. Donlin. Transaction level modeling: Flows and use models. In A. Press, editor, CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hard-ware/software Codesign and System Synthesis, pages 75–80, New York, NY, 2004.

[13] F. Ghenassia. Transaction-Level Modeling with SystemC:TLM Concepts and Applications for Embedded Systems.Springer, November 2005

[14] L. Cai and D. Gajski, Transaction level modeling: an overview, Proc. Intl. Conf. Hardware/Software Codesign and System Synthesis, October 2003, pp. 19–24.

[15] J. Colgan and P. Hardee, Advancing Transaction Level Modeling – Linking the OSCI and OCP-IP Worlds at Transaction Level,Open-Systems Publishing (November 6, 2006); www.opensystems-

[16] T. Kogel, A.Haverinen, andJ. Aldis, OCPTLMfor Architectural Modeling. Methodology: White Paper, OCP-IP (November 6, 2006); www.ocpip.org.

[17] N. Calazans, E. Moreno, F. Hessel, V. Rosa, F. Moraes, and E. Carara, From VHDLregister transfer level to SystemC transaction level modeling: a comprehensive case study, Proc. Symp. Integrated Circuits and Systems, September 2003, pp. 355–360.

[18] Cadence NCSystemC. http://www.cadence.com.

[19] CoCentric System Studio. http://www.synopsys.com.

[20] CoWare. http://www.coware.com.

[21] F. Vahid, D.D. Gajski, Closeness metrics for system-level functional partitioning. O-8186-7156-4/95 IEEE, 1995.

[22] B. Knerr, M. Holzer, M. Rupp, HW/SW Partitioning Using High Level Metrics. Copyright International Institute of Informatics and Systems, published in the proceedings of the International Conference on Computing, Communications and Control Technologies (CCCT), pp. 33-38, Austin, 2004