Processor Centric Specification and Modeling of MPSoCs Using ArchC

Cristiano Araujo, Edna Barros
cca2,ensb@cin.ufpe.br
Informatics Center (CIn)
Federal University of Pernambuco
Cidade Universitaria 50740-540
Recife PE, Brazil

Rodolfo Azevedo, Guido Araujo
rodolfo,guido@ic.unicamp.br
Computer Systems Laboratory
University of Campinas
Cidade Universitaria Zeferino Vaz
Po. Box 6176, Campinas-SP, Brazil

Abstract

In this paper is presented a processor centric approach for the modeling and simulation of multi-processors platforms for SoC (MPSoC). It describes how the ArchC architecture description language has been extended to allow the description of multi-processors platforms. It is shown that with minor efforts the designer can model the processors and the platform in a unified environment. Together with the language extensions, it is also presented the *acsys* tool. It generates executable simulators of the platform at different abstraction levels. The advantages of this approach are more flexibility as designers can easily integrate and configure the processors and the platform in a single environment; and faster design space exploration as the simulation scheme for the platform is generated automatically.

1 Introduction

Embedded systems are specified to optimally run a single embedded application. Shared memory multi-processor systems-on-chips (MPSoCs) have been widely used in today high performance embedded systems, such as network processors and multimedia processors. They combine the advantages of data processing parallelism of multi-processors and the high level integration of systems-on- chip . The MPSoC performance is not only determined by the capacity of the node processors (e.g. CPU speed, cache size, etc.), but it is also limited by the interconnect network that con-nects the processors and memories. Design and optimization of such interconnect network are critical for MPSoC performance, and modeling and analysis mechanisms of MPSoCs at an early design phase are mandatory for decreasing the time to market.

In order to be support the design space exploration at an early design phase, the application should be mapped into a multi-processor platform model, which should support analysis of functional and non-functional requirements of the application.

The simulation of multi-processors platforms is not a trivial task. Instruction Set Simulators (ISS) for the processors must be integrated in order to simulate the programs running in each processor. It is also necessary synchronize these simulators in accordance with the communication protocols implemented by the interconnection structure of the system. Validation of these components is critical as they define most of the functional and non-functional requirements of the system. Traditional approaches use encapsulation of third party ISS as components of the system [PPB02, cow03, CBG$^+$02]. One problem that arises from component based approaches is that the design space for the designers is normally constrained. In most cases a small set of processors is available and in some just a family of processors. This is specially critical when the processors at disposal do not meet the requeriments for the design.

In this paper is presented a processor centric approach for the modeling and simulation of MPSoCs. It is proposed the extension of the *ArchC* architecture description language to make it suitable a high abstraction level mechanism for modeling multi-processor platforms. The platform is modeled similarly as its processor architectures. Together with the language extension, a tool has been developed that takes the platform description and generates executable simulation schemes for the platform at two different abstraction levels: functional and RTL. The benefits from this approach are the gain in productivity and a more efficient design space exploration. The first benefit comes from the automatic generation of the simulation scheme, while the second is the result of integrating processor and platform modeling in the same environment and using the same language.

The rest of this paper is structured as follows. Multi-processors platform modeling and simulation are discussed in section 2. In section 3 are reviewed the related works. The ArchC language is introduced in section 4. The proposed approach is described in section 5. A case study is given in section 6. Finally, some conclusions and future developments are discussed in section 7.

2 Multi-processor Platform Simulation

Multi-processors platform simulation presents its own nuances and particularities that are depicted in the platform example of figure 1. First, it is necessary to integrate an instruction set simulator (ISS) for each processor instance in the platform. Different ISSs can be used depending on the design space exploration done by the designer. In the example of figure 1 two different RISC processors could be used: MIPS or SPARC; while three instances of the 8051 microcontroller, that has a CISC architecture, are being used.

The embedded application running on a multi-processor platform is composed of concurrent tasks that run on the processors and communicate using the interconnection structure protocol of the platform. The ISSs must be synchronized to run in parallel and also to implement the interconnection protocol of the platform. That means read and write instructions (memory mapped IO) and IO instructions, that use special registers, executed in the simulator, must make use of the platform protocol.

It is also necessary to define how the addresses issued to the interconnection structure protocol will be decoded. The decoding logic determines the address ranges for each slave component connected to the bus.

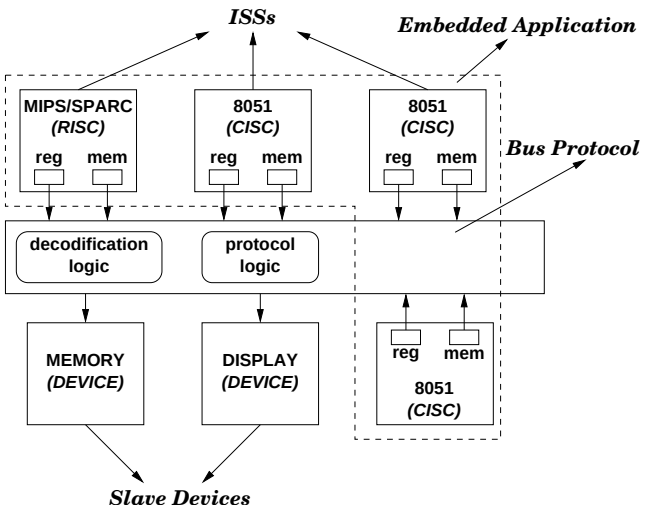

Figure 1: A multi-processor platform model

The implementation of a simulation model for multi-processor platform simulation described above is not a trivial task. The first difficulty is to find the proper ISSs for the processors. The ISSs can be obtained basically from three sources: stand alone simulators, third party components and created by ADL based tools. Stand alone simulators are programs designed to run executable code compiled to a target architecture. The problem in using these ISSs is that two applications need to run in parallel, the platform simulator and the ISS. This results in complicated *ad hoc* schemes that are difficult to handle. A third party component ISS is a processor simulator normally described in some hardware description language and distributed as is by its creator. The most common are described in VHDL/Verilog and recently in SystemC. In this case it is necessary to create one wrapper around the component and add it to the platform model. When the platform simulator is the same as the processor component the integration task is easier. On the contrary, it is also necessary to implement the communication between the platform and the HDL simulator. The third alternative, makes use of an ADL based tool to generate a component from a processor architecture description. The component is then used as a third party simulator. In this case the designer has more flexibility to modify the processor or even to build new ones. Most of the platform simulation environments use the first and second approaches. The consequence is a reduced set of processors available. Others use some ADL languages that allows the modifications in the processors but that are limited to some families. Finally, in cases like the ConvergenSC/Lisatek suite [cow03], a platform simulation and a processor development tools are used together.

3 Related Work

ConvergenSC [cow03] is a tool for platform based design for generic application domains. It comes with a model library that includes ARM and MIPS processor models and also bus models like AMBA. For processors that are not in the library, designers must create them separately using the LISATek product family [HKN+01] and use them as components.

STepNP [PPB02] is a platform exploration tool for the design of Network Processor Units (NPUs). In STepNP designers are restricted to a pre-defined set of processors. They can choose from ARM, PowerPC and DLX simulators. STepNP developers are also planning to include Xtensa [ten03] and LISATek [HKN+01] processor models.

In the *Component Based Design* methodology [CBG+02, Nic02] the processor simulators are wrapped in components. In this approach an *ad hoc* wrapper generation scheme is build and the configuration parameters for the processors instanciation are defined. In a second phase the designers set the configuration parameters values in order to construct the platform simulation scheme.

EXPRESSION [HGG+99] is an architecture description language that is suited for the description of different types of architectures: VLIW, ASIP, DSP and conventional processor architectures like RISC. Functional, cycle accurate and compiled simulation simulators can be generated from an EXPRESSION description. Despite being a powerful ADL, EXPRESSION based simulators are basically used for the processor toolkit generation and not for describing multi-processors platforms.

4 ArchC Language

ArchC [RJA+04] is an architecture description language, initially conceived for processor architecture description, which aims to facilitate and accelerate processor description. Combined with enough expression power to model several classes of architectures (RISC, CISC, DSPs, etc) ArchC allows users to fast explore a new ISA by automatically generating software tools like

assemblers and simulators, which is mandatory in the system-level design scenario.

It has been conceived to have a synthax similar to the SystemC language. The key difference is that is has constructs that make it easier to model processor architectures.

A processor architecture description in ArchC is divided in two parts. The `Instruction Set Architecture (AC_ISA)` description, which includes details about instruction formats, size and names, as well as all information necessary to decoding and to execute the instruction behavior. A detailed discussion on the `AC_ISA` description can be found in [SRA04]. The `Architecture Resources (AC_ARCH)` description specifies the storage devices, pipeline stages and memory hierarchy of the processor.

One example of resource description is given in figure 2. It is characterized by the `AC_ARCH` declaration followed by the processor type name, `sparcv8`. Then is declared the memory hierarchy composed of one 5Mb cache (DM); two register banks (RG and RB) with 8 and 256 registers respectively; and two internal registers (PSR and Y). The word size of the sparcv8, 32 bits, is defined by the `ac_wordsize` declaration. In the `ARCH_CTOR` declaration are defined the file where the ISA is described (sparcv8_isa.ac) and the endianness, in this case big endian, of the processor. The lines marked with (EXTENSION) are discussed in section 5.1.

```
1:  AC_ARCH(sparcv8){
2:    //!Memory hierarchy
3:    ac_cache   DM:5242880;
4:    ac_regbank RG:8;
5:    ac_regbank RB:256;
6:    ac_reg PSR;
7:    ac_reg Y;
8:
9:    //! Processor Architecture word size
10:   ac_wordsize 32;
11:   //!Protocol port declaration (EXTENSION)
12:   ac_protocol<OCP, MASTER> OCP_BUS(32, 32);
13:
14:   ARCH_CTOR(sparcv8){
15:     //! ISA description
16:     ac_isa("sparcv8_isa.ac");
17:     //!Processor endianness
18:     set_endian("big");
19:     //!connection declaration (EXTENSION)
20:     DM.bindsTo(OCP_BUS);
21:
22:     //!Memory map declarations (EXTENSION)
23:     DM.set_range(0x0, 0x500000);
24:     OCP_BUS.set_range(0x500001, 0xa00000);
25:   };
26: };
```

Figure 2: AC_ARCH description and architecture extensions

5 The Proposed Approach

This work proposes a processor centric approach for the modeling and simulation of multi-processors platforms. The approach is based on the extension of the *ArchC* language discussed previously to support descriptions of multi-processor platforms. And, on the automatic generation of the platform simulator.

The phases for describing a platform by using the proposed approach are depicted in figure 3. In the *processor development phase* the designer describes the processor architectures in *ArchC*. The objective is to validate the processor architecture by simulation. The result of this phase is an ISS that can perform compiled or interpreted simulation. The compilation of a processor

description in ArchC by the **acsim** tool results in SystemC code that represents an interpreted simulation ISS. However, using the **accsim** tool, it results in code for a compiled simulation ISS. A more detailed discussion on processor description and simulator performance can be found in [BARA04].

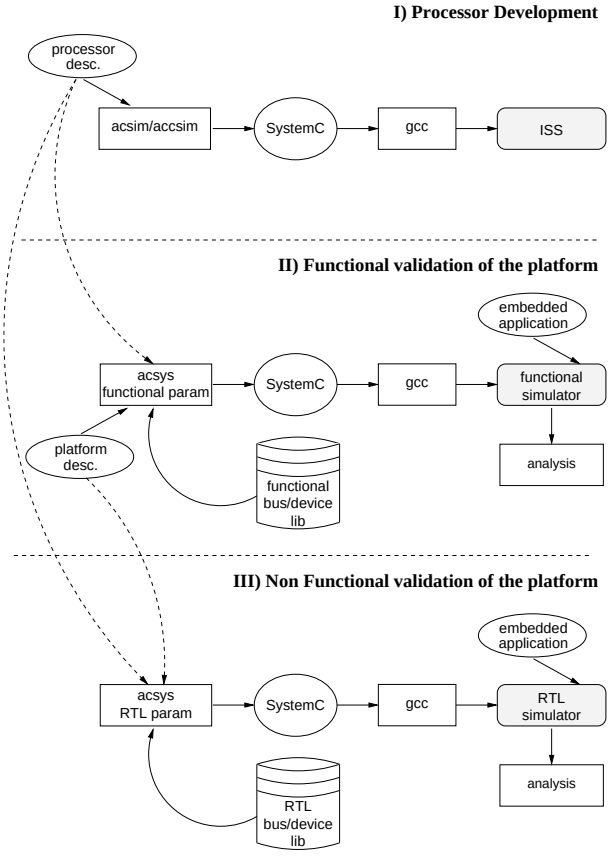

Figure 3: ArchC modeling and simulation flow

The objective of the second phase, *functional validation of the platform*, is to rapidly validate the functionality of the embedded application running on the platform. The designer provides the platform description, also in ArchC, and reuses the previously validated processor descriptions. These are then passed as input to the **acsys** tool with the functional option set. By taking the processors and platform descriptions, and the functional description of the bus and device in the library, the execution of acsys results in SystemC description of the complete platform for simulation. This SystemC code is compiled using a C++ compiler like gcc resulting in an executable simulator of the platform. When executing the simulator the embedded application is executed generating analysis results.

Once the functional requirements of the platform have been validated, the third phase can be started for the validation of non-functional requeriments. In the current version our tool supports performance analysis. An strategy for power analysis is also under development, concerning the platform specification. This phase is quite similar to the previous one. The same processors and platform descriptions are used. The differences are on the bus and device library that is composed of components with RTL interfaces; and on the analysis step that takes in consideration the performance of the system regarding the interconnection structure protocol. The result of this phase is the generation of an executable simulator of the platform where a RTL model of the interconnection structure is used.

The platform simulator (for the platform showed in figure 1) is depicted in figure 4. The entire simulation scheme is implemented in SystemC 2.0. SystemC code for each ISS is auto-

matically generated. The integration of the ISSs is done through the use of protocol ports and protocol connections. Protocol ports are master or slave ports that supports the desired protocol. Currently based on OCP-IP protocol ports are being supported. A protocol connection links one master to one slave port. It is used to connect master ports of the processors to slave ports in the interconnection structure or devices. For instance, in figure 4 the protocol master port of P1 is connected to the slave port of the high performance serial bus. Depending on the selected abstraction level (when executing the *acsys*) funcional or RTL communication can be implemented. The functional implementation of communication does not consider time information as depicted in the lower left part of figure 4. On the other hand, a RTL implementation of the communication is cycle and pin-accurate (see the lower right part of the figure, which shows the basic OCP-IP signals).

The synchronization of the ISSs with the interconnection structure protocol is performed through read/write operations on the storage components. The protocol adapter (PA) converts the read/write requests to one storage element, like the memory DM, into read/write requests on the protocol port. The protocol adapter takes care of protocol, endianness and data width conversions.

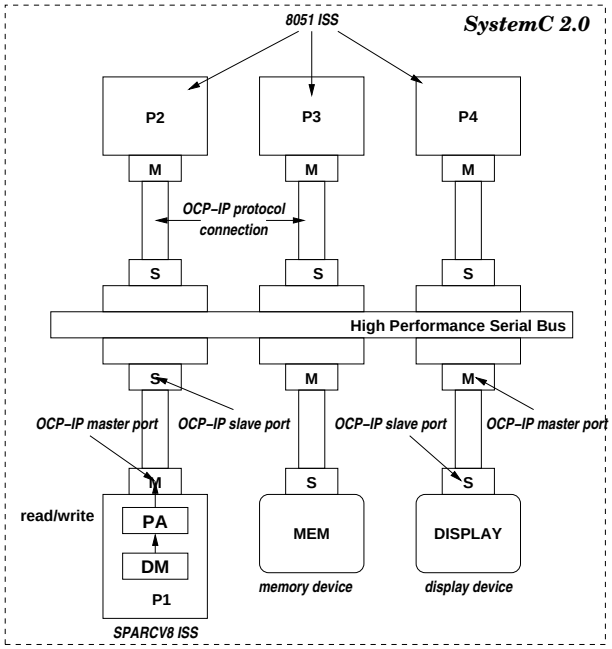

Figure 4: A platform simulator model based on SystemC

5.1 Describing the processor in multi-processor platform

The standard ArchC language supports the description and implementation of a stand alone processor simulator. In order to support the automatic integration and synchronization of the ISS in a multi-processor simulation scheme it is was necessary to extend the ArchC language to support the processor connection with external protocols.

This has been achieved by extending the language with the *ac_protocol* declaration. Using the ac_protocol declaration the designer declares master or slave ports that support the external protocol. A protocol port is defined by the following 5-tuple $< t, d, n, aw, dw, s >$. The parameter t determines the type of the protocol supported by the port. The second, d determine the direction of the port that can be master or slave. Parameter n represents the name of the protocol port and the parameter, aw is the width of the address bus of the port. The parameter dw determines the width of the data bus of the port. The last parameter s is the number of

protocol ports of that type. The protocol port declaration can be seen in line 12 of Figure 2. It declares one master port which is OCP-IP protocol [OCP02]. The port has an address and data bus width of 32 bits.

The protocol port declaration simply defines that the processor has one or more protocol ports. The **bindsTo** declaration is an extension that specifies which storage element of the memory hierarchy will be connected to the processor port (in the case of memory mapped I/O). Line 20 of Figure 2 shows one example of such binding. In this case the cache memory *DM* is connected to the *OCP_BUS* protocol port implementing a memory mapped IO access. The use of the bindsTo statement just connects the memory and protocol port. It is now necessary to define the addresses range used for I/O.

The **set_range** statement is used to define the memory map of the processor. This declaration is used in the case a storage element is binded to a protocol port. It determines the address ranges that are accepted by the storage element and by the protocol port. Lines 23 and 24 of Figure 2 declare that address in the range (0x00000000,0x05000000) are recognized by the memory DM while address in the range (0x0500001, 0x0a00000) are accepted by the protocol port OCP_BUS. The memory map for a component can be composed of several address ranges, i.e, the designer can use the **set_range** declaration multiple times for a single storage element or protocol port.

5.2 Describing the complete platform

In order to support the description of a complete platform, the *ArchC* language has been extended with a set of new specification mechanisms. The extension should support the reuse of processor descriptions, and should be able to model the platform in an similar way as the *ArchC* language (similar syntax) in order to make easier its use by the designer.

Structural Declarations

One extension to the language is the `include` declaration. It has been conceived with two goals. One, support the reuse, as is, of the processor architectures descriptions in *ArchC*. The second goal was to provide support to the use of busses and devices in a library as shown in figure 3. Examples of `include` declarations can be seen in lines 1-5 of figure 5. The first two lines are including the AC_ARCH files for the sparcv8 and 8051 descriptions while the other two lines declare the interconnection structure (OCPSerial) and device (OCPSlaveMem).

The platform itself is modeled using the `AC_SYSTEM` declaration. This declaration has similar syntax to the AC_ARCH declaration for processors. `AC_SYSTEM` is a composed declaration divided in two parts. The first part is used to declare the instances of the platform components and includes the declarations for processors instances, as well as busses and for devices instances.

The processor and device instantiations are similar and follow a similar syntax than the protocol port declaration in the processor description. Processor instantiation is represented by the `ac_processor` declaration followed by the processor type and instances names separated by commas. Examples of the `ac_processor` declarations can be seen in lines 12 and 13 of figure 5. The first statement declares a *sparcv8* processor named P1. The following statement declares three instances of the *i8051* processor. Device instantiation is done in the same way, but using the *ac_device* declaration. Line 19 declares a memory of type OCPSlaveMem.

Using the *ac_system_bus* statement the designer can declare interconnection structure instances. The declaration specify the bus type, instances names and the width of the address and data types supported by the bus. An example of system bus declaration can be seen in line 16 of Figure 5. There is declared an OCPSerial bus type whose instance is named *BUS*. Its address and data width are 32 bits.

```
1://! Processors, display and bus types
2:#include "sparcv8.ac"
3:#include "i8051.ac"
4:#include "OCPSerial.ac"
5:#include "OCPSlaveMem.ac"
6:
7:
8: AC_SYSTEM(platform) {
9:  //Master port
10:  ac_protocol<OCP> OCP_BUS(32, 32);
11:
12:  ac_processor<sparcv8> P1;
13:  ac_processor<i8051> P2, P3, P4;
14:
15:  //Bus declaration
16:  ac_system_bus<OCPSerial> BUS(32,32);
17:
18:  //Device instance declaration
19:  ac_device<OCPSlaveMem> MEM;
20:
21: SYSTEM_CTOR(platform) {
22:
23:  P1.OCP_BUS.bindsTo(BUS);
24:  P2.bindsTo(BUS);
25:  P3.bindsTo(BUS);
26:  P4.bindsTo(BUS);
27:  BUS.bindsTo(MEM);
28:
29:  //Memory map
30:  MEM.set_range(0x600000, 0xA00000);
31:  OCP_BUS.set_range(0xA00001, 0xB00000);
32:
33:  //Sets abi support
34:  sparcv8.set_abi();
35:
36:  //Loading applications
37:  P1.load_obj("control");
38:  P2.load("fibonacci");
39:  P3.load("crc16");
40:  P4.load("bubblesort");
41:  };
42: };
```

Figure 5: AC_SYSTEM and include declarations

The second part of the AC_SYSTEM declaration, delimited by the SYSTEM_CTOR keyword, is used to build the platform. It has a similar syntax than the AC_CTOR part of the processor description, which specifies the connection of memory hierarchy in the processor. The connection of the components is performed with the same *bindsTo* statement used in the AC_ARCH declaration. Despite having a similar syntax, the semantics of the this statement is *master connects to slave*. The master connects to the slave in one of two ways: by expliciting the port to be connected or implicitly (only one port). In the explicit form the designer must state the port of the master and slave devices that are being connected. This is necessary when one or both of the master or slaves have more than one protocol port. One example of an explicit connection is given at line 23 of Figure 5. The port OCP_BUS of the architecture instance P1 is connected explicitly to the BUS. Examples of implicit connections are given at lines 24 to 27 of Figure 5.

Behavioral Declarations

The declarations discussed above are used for specifying the structure of the platform. The behavioral declarations are used to describe how the platform will behave. There are three types of behavioral declarations: `set_range`, `load` and `load_obj`.

The `set_range` declaration is used to define how the decoding logic of the interconnection

structure will decode the addresses issued by the masters elements. Using `set_range` the designer informs in a simple way, the address ranges for each slave in the system. He/she does not have to take care on how the interconnection structure will implement the decodification of the addresses. The `set_range` has a similar syntax and semantics than the memory map declaration used for the memory hierarchy of the platform. Line 30 of figure 5 shows an example of memory map using set_range. Any address in the range (0x600000, 0xA00000) will be decoded to the memory MEM.

The embedded application is composed of executable code for each one of the processors in the system. The mapping of the embedded application is performed using the other two behavioral declarations. The `load_obj` declarations allows the designer to map directly binary code to the chosen processor. Using `load` it is possible to map executable code in the *ArchC* hexadecimal format. Lines 36-40 of figure 5 show the mapping of the embedded application, composed of one executable code in binary format and three in hexadecimal format, to the processors in the platform.

Bus and Device Declarations

The AC_BUS declaration is used to define the interface of the interconnection structures of the platform. In the example of Figure 6 the bus *OCPSerial* is declared. The bus has the following OCP-IP compliant ports, one instance of a master port and two instances of a slave port. As the parameters in the protocol port declaration are literals, the bus address and data ports are parameterized. The implementation of the each declared protocol should be available in a library.

```
1: AC_BUS(OCPSerial) {
2:  ac_protocol<OCP, MASTER> m_port(aw,dw) : 1;
3:  ac_protocol<OCP, SLAVE>  s_port(aw,dw) : 2;
4: };
```

Figure 6: AC_BUS declaration

Similarly, AC_DEVICE declaration states that there is a SystemC module that implements the interface declared. In the example of Figure 7 is declared the OCPSlaveMem device. In this case, however, the parameters of the protocol port are integer values, which means that the device can only be connected to an OCP channel with 32 bits of address and data widths.

```
1: AC_DEVICE(OCPSlaveMem) {
2:   ac_protocol<OCP, SLAVE> s_port(32,32);
3: };
```

Figure 7: AC_DEVICE declaration

5.3 Simulator Generation

The *acsys* tool has been developed that takes the platform description in *ArchC* and generates a SystemC simulation model of the platform. Using a simple command line, the designer can generate code of the platform with the communication at two different abstraction levels: functional and RTL.

The functional implementation is based on the SystemC OCP channel model provided provided by the OCP-IP consortium [OI04]. Using this channel it is possible to one master and one slave component to exchange information that is parameterizable in the data and address types. The main advantage of this type of channel is that they are timeless and provide a much better performance during simulation. The fact that it is timeless is also its weakness. Using this type of channel the designer can validate the functionality of the application running on the

platform but cannot identify synchronization problems or evaluate communication performance. It is also possible to generate communication at RTL level.

The usage of the *acsys* tool is quite simple. It is a command line tool that takes three arguments. The first one is the name of the ArchC file containing the platform description. The second argument determines the abstraction level of the communication protocols ports of the platform. It can assume the values *direct* or *rtl* The last argument determines whether or not the system will generate trace files for the protocols. The usage of acsystem is shown below:

1 > *acsystem input file [-p[abstraction level flag]] [-tp]*

6 Results

The platform of figure 4 has been modeled and compiled using the extensions to the language and the *acsys* tool. The sparcv8 processor and platform descriptions are shown in figures 2 and 5 respectively. The results are summarized in table 1. As it can be seen the modeling effort is minor. The description of the structure of the *sparcv8* and *8051* processors costs 20 and 15 lines of code respectively. The descriptions of the instruction set architecture for the processors have 1582 lines of code for the sparcv8 and 3902 lines for the 8051 processor.

The platform description itself is done by writting aproximately thirty lines of code. This includes structural and behavioral information. In order to use the components in the library, the files containing the AC_BUS and AC_DEVICE declarations have to be added. But again, the effort to write them just once is minimum and results in a total of seven lines of code. It is also important to note that another important benefit is that it is not necessary to make any change in the platform or processor descriptions during the validation of non-functional requirements of the platform. It is only necessary to call the *acsys* with the *-prtl* flag set. The compilation time for any version, functional or rtl, of the platform using *acsys* takes less than a second in a pentium IV machine.

Table 1: modeling effort

SPARCV8 description	20 lines
SPARCV8 instructions	1582 lines
8051 description	15 lines
8051 instructions	3902 lines
platform description	35 lines
OCPSlaveMem description	3 lines
OCPSerial description	4 lines
code changed for RTL	0 lines
ArchC compilation time	$\leq 1sec$

The platform simulator has been used to run four simple applications. The first running on the *sparcv8* processor controls the other three running on the 8051 microcontrollers. When the three microcontrollers finish the execution of their task they communicate with the *sparcv8* that halts the system. The purpose of the application was to validate the communication betweeen the four processors. One characteristic of the application is that it enforces the intensive use of the bus. The objective is make the processors dispute the control of the bus and make the conflicts apparent in order to show the importance of the two abstraction levels used for communication. We have modeled the platform using two different interconnection structures. The first implements a fixed priority scheme for the processors connected to the bus. In case of intensive communication it does not guarantee that all processors take control of the bus. The

second interconnection structure implements a dynamic priority scheme that guarantees that all the processor can take control of the bus.

Table 2 shows the simulation times for the two variations of the platform. The embedded application has work fine on the two variations of the platform using functional simulation. But when using RTL simulation it has shown that version *v1* of the serial protocol is not suitable because the application does not work. This has happened due to bus conflicts that are not properly handled by this version of the protocol. Table 2 also shows the impact of the RTL interconnection structure in the simulation performance. This impact has been approximately of 20 times for the embedded application used, using version *v2* of the serial protocol. It is clear from the results the importance of having the functional and RTL abstraction levels for communication. The first is used to validate the behavior of the application quickly, as it runs 20 times faster than the RTL version. On the other the conflicts on the interconnection structure can only be identified using the RTL version of the protocol.

Table 2: simulation times

master	comm. level	protocol	behavior	time (s)
sparcv8	functional	v1	OK	22 (s)
sparcv8	RTL	v1	failed	-
sparcv8	functional	v2	OK	22 (s)
sparcv8	RTL	v2	OK	567 (s)

In table 3 are given the number of instructions executed per second using functional simulations. The results show that the four processors run about 90K instructions/sec giving a total of aproximately 360k instructions/sec. With this simulation speed designers can validate applications of a reasonable size at an acceptable amount of time.

Table 3: platform functional simulation performance

instance	type	#instructions	performance(Kinstr/s)
P1	sparcv8	1842709	87.00
P2	8051	1897748	90.00
P3	8051	1897748	90.00
P4	8051	1897748	90.00
total		**7535953**	$\sim$ 360.00

7 Conclusions

In this paper has been presented a processor centric approach for the modeling and simulation of multi-processor platforms. The ArchC ADL has been extended to support MPSoCs modeling without disfiguring the language. The results shown that the modeling effort is minimum and presents the advantage is that processor and platform are modeled in a unified environment.

It has also been described the *acsys* tool that generates SystemC simulators at the functional and RTL levels from the platform description. The great advantage of the acsys tool is that the designer does not need to make any change on the platform description to obtain the simulators. It hides all the simulation scheme details from the designer that just have to set command line options for the tool. Simulation results have also shown the impact of RTL simulation performance and on finding errors in the platform.

As further steps we are working on more elaborate analysis tools for the simulator. The objective will be to get in depth information about the communication over the interconnection structure of the platform. This will help the designers to decide what parts of the platform should be modified in case the requirements are not met.

References

[BARA04] Marcus Bartholomeu, Rodolfo Azevedo, Sandro Rigo, and Guido Araujo. Optimizations for compiled simulation using instruction type information. *16th Symposium on Computer Architecture and High Performance Computing (SBAC'04)*, 10 2004.

[CBG+02] W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, and M. Diaz-Nava. Component-based design approach for multicore socs. In *Proceedings of the 39th conference on Design automation*, pages 789–794. ACM Press, 2002.

[cow03] Coware company. available at http://www.coware.com, May 2003.

[HGG+99] Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt, and Alex Nicolau. Expression: a language for architecture exploration through compiler/simulator retargetability. In *Proceedings of the conference on Design, automation and test in Europe*, page 100. ACM Press, 1999.

[HKN+01] A. Hoffmann, T. Kogel, A. Noah, G. Braun, O. Schliebush, O. Wahlen, A. Wieferink, and H. Meyer. A novel methodology for the design of application specific instruction set processors (asip) using a machine description language. In *IEEE Transactions on Computer-Aided-Design*, pages 1338–1354, November 2001.

[Nic02] Eugenia Gabriela Nuta Nicolescu. *Spécification et validation des systèmes hétérogènes embarqués*. PhD thesis, TIMA Laboratory, November 2002.

[OCP02] OCP-IP Association. *Open Core Protocol Specification, Release 2.0, rev 0.95*, January 2002.

[OI04] OCP-IP. available on-line at http://www.ocpip.org, 2004.

[PPB02] Pierre G. Paulin, Chuck Pilkington, and Essaid Bensoudane. StepNP: A System-Level Exploration Platform for Network Processors. *IEEE Design & Test of Computers*, pages 17–26, 2002.

[RJA+04] Sandro Rigo, Marcio Juliato, Rodolfo J. Azevedo, Guido Araujo, and Paulo Centoducatte. Teaching computer architecture using an architecture description language. In *Proceedings of the Workshop on Computer Architecture Education (WCAE'04), Held in Conjunction with International Symposium on Computer Architecture (ISCA)*, June 2004.

[SRA04] Marcus Bartholomeu Sandro Rigo, Guido Araujo and Rodolfo Azevedo. Archc: A systemc-based architecture description language. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing - Foz do Iguacu, Brazil*, October 2004.

[ten03] Xtensa application specific microprocessor solutions - overview handbook. available at http://www.tensilica.com, May 2003.