SEP: SIMULATION FRAMEWORK TO EVALUATE DIGITAL HARDWARE ARCHITECTURES.

Frédéric Mallet, Fernand Boéri
Laboratoire Informatique, Signaux et Systèmes (I3S)
UPRES_A 6070 du CNRS
Les Algorithmes - bât. Euclide B – BP 121
06903 Sophia Antipolis Cedex France.
Frederic.Mallet@unice.fr, boeri@unice.fr

Jean-François Duboc
DSP Architect – Embedded Processor Group
Philips Semiconductors
505 route des Lucioles - Sophia Antipolis
06560 Valbonne France.
Jean-Francois.Duboc@vlsi.com

KEYWORDS

Performance evaluation, Architecture design, Rapid-prototyping, Simulation, Object-oriented modelling.

ABSTRACT

Know-how is the most useful mean for designing new processors before a complete hardware description. The integration rate is increasing very quickly and the time-to-market has to be dramatically reduced because of the rapid evolution of technology. Therefore, reuse and rapid-prototyping are definitely a major issue to integrate existing architectures and to design new ones.
SEP is an object-oriented framework which attends these problems. This paper intends to show major problems and solutions in simulation due to our simplification choices and in particular due to a not typed specification. It also presents the service feature which is a major enhancement to SEP and allow some validation of static properties about the rapid-prototyped model and its associated instruction-set. In this paper some examples related to the modelling of an industrial bi-core architecture from VLSI Technology – a subsidiary of Philips Semiconductors – are used to illustrate our method.

INTRODUCTION.

Nowadays, paper and know-how are the only useful tools for designing new processors before a complete hardware description. The normal designing way is to highlight restrictions of existing architectures. Then, designers specify new functional and time requirements to perform new manageable applications. Most of the time, these new requirements consist in the integration of several existing architectures. When designers think they solved every problem of integration, they make a hardware description using a classical hardware description language (VHDL, VERILOGHDL). Finally some tools allow simulation and benchmarking to validate the hardware model.
Until now the integration of existing architectures and resolution of problems were manageable with paper simulations because of the relative simplicity of computer architectures. However, the integration rate is increasing very quickly and the time-to-market has to be dramatically reduced because of the rapid evolution of technology. Moreover every processors have superscalar capabilities. Therefore, reuse and rapid-prototyping are definitely a major issue to design the next architectures.
Then we defined in collaboration with VLSI Technology – a subsidiary of Philips Semiconductors - the specification of a tool able to fix those problems. The basic principle of this tool called SEP has been presented in a previous paper (Mallet & al. 1998). Based on an object-oriented method, SEP allows a rapid prototyping of architectures (including both applications and hardware) and performs evaluations by simulation. This paper intends to show major problems and solutions in simulation due to our simplification choices and in particular due to a not typed specification. It also presents the service feature which is a major enhancement to SEP and allow some validation of static properties about the rapid-prototyped model and its associated instruction-set. These mechanisms are due to our object-oriented method and give to the designer some confidence in the built model as a first step toward behavioural and structural proof.
We firstly give an overview of SEP as an object-oriented framework and introduces what we called service of hardware components. Then we point out the extension of the service concept to the description and simplification of logic in structural components. Then, we show the extension of the service feature to describe the instruction-set of architectures and properties that can be deduced. In this paper we will give some examples related to the modelling of an industrial bi-core architecture from VLSI Technology.

THE SEP COMPONENTS.

Our method shows that object-oriented and component-based techniques may be useful for digital architectures modelling and simulation. This model demonstrates good reusability and flexibility properties. Autonomy of components may guaranty the reuse and protection of intellectual-property components (see 4.4). Other approaches consist in adding the object-oriented paradigm into already existing hardware description languages: OO-VHDL (Benzakki and Djaffri 1997; Schumacher and Nebel 1995), Objective-VHDL (Putzke & al. 1998). We think that classical hardware description languages were designed with synthesis goals and most of them are strong-typed languages.

Then adding object-orientation should result in a difficult to handle (Schumacher and Nebel 1995) and a very constraining language. As a consequence we think that a good approach consists in adding some hardware description capabilities to already existing object-oriented languages like we proposed in (Mallet & al. 1998).

Our graphical framework is based on its ability to manage components and links described by our generic object-oriented model. By generic, we mean that the proposed model is not on dependency of some architecture specific features. So, we should be able to model every architecture types (RISC, DSP, CISC, VLIW) as well as control systems, multiprocessors or distributed architectures. By now, SEP was used to successfully model and evaluate a DSP, a RISC, a bi-core ASIC and a software sprinkler controller.
The generic model consists in the definition of the minimal communication interface between different autonomous components (SEP-Components). This interface imposes some rules for event propagation and data transfers independently with the operating system or programming language. This generic model allows the internal description of basic components as well as the gathering of existing components connected by communication lines (control signals, data busses). Then, each component is designed independently from the others. Thus, it becomes definitively reusable.

A 'SEP-Component' is a set of ports and an internal behaviour. The behaviour can be described in two different ways. The first solution is the structural representation by gathering some components. Components can also be described with a behavioural description. A sensitivity can be added to ports of behavioural components. Sensitivity describes the sequence of events needed to make the component reacting. The reaction consists in the execution of the attached service (method). The composition of several services constitutes the behaviour of the component.

Example – Load/Store Register :

```
package component.basic;
public class LS Register extends
            modele.simulation.EdgeComponent {
  public LS Register() {
    addInactivePort ("in", LEFT);
    addInactivePort ("out", LEFT);
    addRisingEdgePort("load",TOP).setService("load");
    addRisingEdgePort("store",TOP).setService("store");
  }
  public void load() {
      setPriority(1);
      emit("out", read ("value"));
  }
  public void store() {
      setPriority(0);
      emit("value", read ("in"));
  }
}
```

The preceding example shows the load/store register behaviour described with a Java description.

The remaining problem is to be sure that asserted component constraints remain in other architectures. As we can see in this example, there is no type associated to SEP ports and there is no constraint on links between ports. As a consequence, each port and each component may receive any data type. Each data inherits from the abstract class Value and has the ability to mute to match another attempt type. This property increases reusability of components. In order to avoid dangerous connections, an attempt type is inferred for each connection from the Java code. If the $\top$ type is inferred then no constraint are deduced. If the $\bot$ is inferred, then the connection is rejected.

For instance the load/store register has some constraints. The 'in' port needs a producer to serve some values, so the 'in' port cannot be connected to a data bus where no potential producer may emit some data. In the same way, system will deduce that load and store ports have to be connected to ports able to provide some level values since they are edge sensitive ports.

This is the first step of the structural verification. Then, since we use inheritance and sub-typing to describe values more than one operator may apply for a specific operation. The arithmetical and logical unit (ALU) is just defined as a component owning two input data ports (in1, in2), one input control port (cop) and two output ports (out, flag), which represent the result and the flag result. Then, whatever effective driven data types are (string, integer, float, instruction), the components has to compute the required operation (addition, multiplication). By example, an addition can be seen as integer addition, float addition, string concatenation or others. Data have the ability to mute in accordance to the defined sub-type relation. And the right operator must be inferred dynamically in simulation. This is done quite easily parsing the sub-type tree selecting the lowest operator which matches parameter requirements. If no operator can be inferred then the NONE Value is computed and a message is sent to designer. This can be due to the lack of an adapted description for the concerned operator and the description must be completed. Or this can also indicate that some data paths are wrong and should be corrected.

STRUCTURAL COMPONENT SERVICES AND INHERITANCE.

Since a component is a set of ports and a set of services, inheritance of hardware components means inheritance of communication ports and inheritance of services which can be overridden. The service concept is extended to structural components by the way of the service port. In a structural component, a service port is a sequential and parallel composition of services provided by contained components. This composition is

abstracted by a textual identifier which can then be considered as a higher level service. By the way, it also makes sense to inherit behaviour from a structural component. This extension is more powerful than the simple structural inheritance as soon as in this way behavioural components may inherit behaviour from a structural one.

Moreover, a service port allows the abstraction of every control ports in a structural component. Management of numerous ports and signals is difficult and number of not interesting signals increases quickly in hierarchical descriptions. So, this abstraction greatly reduces complexity of models since it reduces the number of ports and signals. Most of ports in hardware components just allows the implementation of low-level communication protocols. There are lots of enabling or chip-select signals, configuration signals as well as compatibility signals with previous architectures. Most of those do not intervene in performance evaluation (except in area and consumption calculus).

For instance, let us imagine we want to model a multiplication unit (MU) for the Oak processor – a DSP Group trademark. The MU contains two input load/store registers X and Y, and a multiplier 16x16. Allowed operations with this MU are storing values in each of the registers. There are two input busses GDP and XYDP. The multiplier performs the multiplication with the X and Y current values and drives the result on the 'p' port. At any time, one must be permitted to read X or Y values. Moreover some chip-select signals are required to reduce consuming or to implement bus communication protocols. A structural SEP model for this MU could be as shown by Figure 1 or a much more complex one.

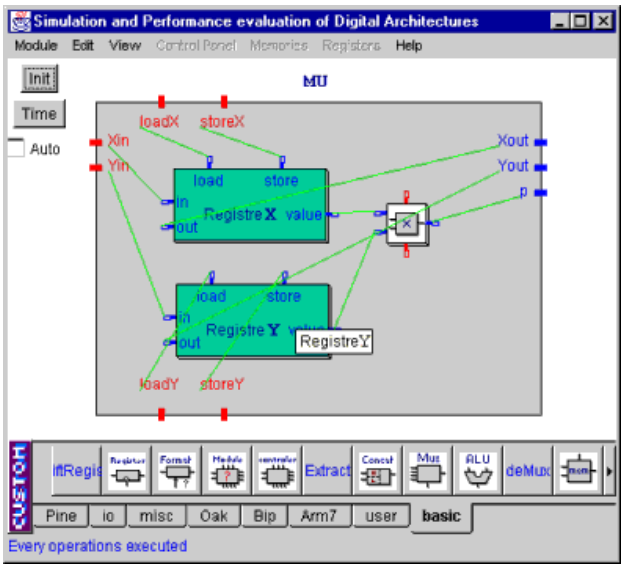

Figure 1 – MU SEP model (1).

This model uses four control signals (loadX, storeX, loadY, storeY). Since this component is included in three description levels and since the decoder is included in two description levels, there are three intermediate components, which have to drive those four signals. Then, it must appear 16 (4*4) signals and 20 (4*5) ports relative to these control signals.

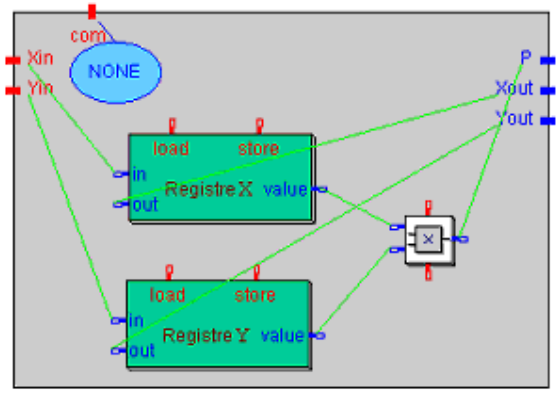

Figure 2 – MU SEP model (2).

Therefore, service ports were created. They allow multiplexing those control signals as soon as we do not want to evaluate performances about them. Indeed, using service ports do not let external components to see differences between multiplexed signals. Figure 2 shows the resulting MU model with SEP using a service-port. Then, MU is an entity providing four basic services.

Using that model, paths are dynamically built depending on required services. To use the 'loadX' service, decoder has to send a "loadx" string value to the 'com' port. And a rising-edge event is automatically sent to the 'load' port of the 'Registre X' component (see Figure 3). In the same way, decoder can drive the 'storeY' service as shown by Figure 4.

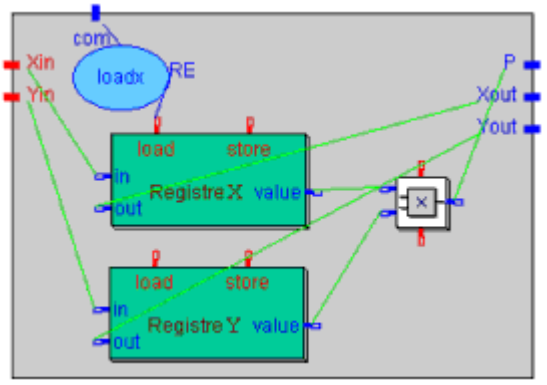

Figure 3 – Driving the 'loadX' service.

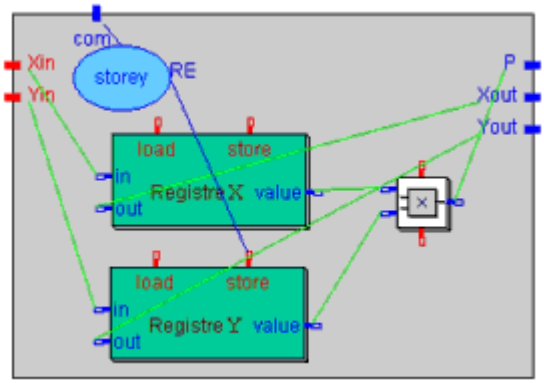

Figure 4 – Driving the 'storeY' service.

The SEP graphical interface allows a graphical description of services.

SEP SERVICES TOWARD A GENERIC INSTRUCTION DECODER.

Using SEP, data paths can be constructed very quickly. They only consist in gathering some of the on-the-shelves components and connecting them with busses and signals. But this is not enough. Indeed, designers also have to construct control paths and decode block. The control part can greatly be simplified using service ports.

This chapter intends to show that the description of the decoding part can also be simplified. There are many ways to perform this task and each solution has its own complexity. The decoding part of RISC (Reduced Instruction-Set Component) processors is very simple to implement. But the decoding part of CISC (Complex Instruction-Set Component) and DSP (Digital Signal Processor) are much more complexes. There are lots of other methods, and they are more or less complexes: VLIW (Very Large Instruction Word) or micro-programmed processors. But they all have the same objective, they are supposed to transform an assembly or machine-specific instruction into some concurrent control orders.

The service concept can be extended to the core of a processor, then services are still sequential and parallel composition of a set of low-level services. At this level those services can be considered as pseudo assembly instructions. Then, a MAC instruction is the parallel composition of both addition and multiplication. Using this technique we model a decoder for a DSP (OAK+) and for a RISC processor (ARM 7 from ARM Limited, inc.) in about a week. In that way, instruction-set can be built incrementally just adding graphically some new services.

The first benefit of this mechanism is obviously to increase reusability. Using SEP we modelled both the PINE and the OAK processor. The OAK processor is the successor of the Pine processor from DSP group. In that case data paths were almost the same, and the instruction-set of the OAK includes Pine's one.

In addition, this mechanism is not only a wizard but also a verification tool in order to produce a valid instruction-set. Indeed, when a new pseudo-instruction is created material dependencies are checked in order to verify if physically the both services can be called concurrently or sequentially. By example, with the MU component, all of its four services can be called concurrently at the first sight. But into the OAK core both the 'Xout' port and the 'Yout' port are connected to the same global bus GDP. As a consequence, the parallel composition of loadX and loadY services is strictly forbidden. This checking may detect lots of problem about material dependencies and in particular, dependencies due to the pipeline structure. The construction of physically bad instructions is forbidden. This method will never prove the functional rightness of the composition. But this is a first step for the designer to detect missing data paths or invalid prototyping.

Finally, information about pipeline structure can easily be taken into account since we can add some delays which represent pipeline stages between a sequential execution of several services. Then we can combine information from instruction-set and information from structural constraints to perform a cycle-accurate simulation.

CONCLUSION.

As a conclusion, SEP is a method to increase reusability and abstraction of hardware architecture models. A user-friendly graphical interface allows designers to easily use this method and gives them a simple tool to quickly model hardware architectures. It was often said to be a good pedagogical tool to teach hardware architectures to beginners. However, it has the power to model completely industrial architectures (DSP core with standard signal processing applications, bi-core architecture: RISC processor controlling a DSP processor to perform a multimedia application). A high-level description of the DSP core to obtain cycle-accurate simulation took less than one week and the high-level description of the RISC processor took three days. These cores were interacting through a communication interface with several communication protocols.

SEP is not just another simulation tool, it is mainly a computer-aided design tool for computer architectures. Since designers need to evaluate performances of the designed architecture as quick as possible, SEP provides simulation features from a flexible and reusable model. However simulation tools cannot completely validate a model, because tests might never been exhaustive. It is very difficult to apply theorem-proving techniques to complete industrial problems. Then, model checking seems to be the most promising

way to perform some property verifications about a given model. Nevertheless, simulation and validation of properties are complementary approaches and will both be useful in tools for the next architectures generation.
It seems to be quite easy to generate some VHDL code from SEP components but since the abstraction level is not the same, this operation can only be semi automatic. The generation of synthesisable code from a behavioural description is completely out of the scope of SEP by now. Obviously, some parts of an architecture described with SEP are very closed from a VHDL description. The generation of synthesisable VHDL description of the decoding part of a processor seems to be possible and detailed researches will be performed.

As we showed in (Mallet and Boéri 1999), the abstraction power of SEP allows the integration of an execution machine into SEP components, then we are able to integrate some ESTEREL (Berry 1998) descriptions into our model. This is a good beginning in the way of validation. Actually increasingly model-checking tools use synchronous languages as specification languages of reactive control systems and they use temporal logical as specification of properties to be validated. Many of those tools use the synchronous reactive language ESTEREL and provide some property validation features (safety, fairness, reachability and equity). Then, we are able to have confidence in some of our components. We have to investigate a way to propagate deduced properties to the entire model and to validate other components and composition of those components in order to complete the information about misuse of physical resources by an information about good scheduling of service execution to perform a specified behaviour.

REFERENCES.

Putzke-Roming W.; Radetski M. and Nebel W. 1998.“Modeling communication with Objective VHDL”, *Proceedings of International Verilog HDL Conference and VHDL International Users Forum Santa Clara*, CA, USA 16-19 March 1998.

Berry G. 1998. “The foundations of Esterel” http://www.inria.fr/meije/esterel .

Mallet F.; Boéri F. and Duboc J-F. 1998. “ Hardware Modelling and Simulation using an Object-oriented Method “. *Proceedings of the European Simulation Multiconference*, June 98. 166-168.

Benzakki J. and Djaffri B. IFIP 1997. “Object-Oriented Extensions to CHDL : The LaMI Proposal.“. *Chapman & Hall*. 334-347.

Schumacher G. and Nebel W. “Inheritance Concept for Signals in Object-Oriented Extensions to VHDL“. *Proceedings of the EURO-DAC’95 with EURO-CHDL’95*.

Mallet F. and Boéri F. 1999. "Esterel and Java in an Object-oriented framework for Heterogeneous Software and Hardware system Modelling and Simulation. The SEP approach." *Proceedings of the 25th Euromicro conference*. Vol I, 214-222.