A Design Methodology for the Exploitation of High Level Communication Synthesis

Francesco Bruschi, Politecnico di Milano, Italy
Massimo Bombana, CEFRIEL, Italy

Abstract

In this paper we analyse some methodological concerns that have to be faced in a design flow which contains automatic synthesis phases from high-level, system descriptions. In particular, the issues related to the synthesis of the communication between the system elements are considered. The context in which the analysis is performed is the design flow proposed in the ODETTE project: in this ambient, SystemC is exploited in order to provide efficient system-level models; after that, the SystemC+ SystemC subset and extensions can be used to get a refined description that, despite the use of object oriented features such as polymorphism and inheritance, can be automatically synthesised by means of the ODETTE tools. Still, the problem of interfacing the hardware synthesised with the other elements of the design (memories, peripherals) remains an important issue. In order to face this problem, we propose a pattern that can be used to design bus interfaces that allow both an high level of abstraction in the communication on the "user" side, and automatic synthesis by the ODETTE tools. In order to do this, OSSS global objects are exploited to implement the communication between the application and the interface. After presenting the general methodology, a specific library interface is presented, that could connect the device under design to a PCI bus. In order to prove the viability of the approach, an example of synthesis of an example, from the system level down to the RT level is performed.

1. Introduction

The introduction of new languages capable of modelling systems at abstraction levels not possible before is imposing deep transformations of the design flows adopted in the industry. Among the most important innovations introduced by languages such as SystemC there is not only the possibility to exploit all the high level modelling features present in C++, such as object orientation, but also the ability to describe the communication between the various structural units of the design (the modules) in a very abstract way, relieving the designer from the need to specify protocol details that are irrelevant and even misleading at the beginning of a project. As the interest of the design world towards these possibilities is growing, a urge is felt to define methodologies that make a fruitful use of them, really enabling a faster and easier way of developing complex systems. In a long term perspective, an effective design flow will comprehend automatic synthesis steps that will heavily assist the designer in refining the system model from one level of abstraction down to another closer to implementation. At the moment, there are some attempts to achieve this, by the definition and implementation of synthesis tools that accept as input very high level system descriptions and produce representation that can be synthesised with more "traditional", behavioural level synthesis tools.

One of the projects with the most innovative synthesis features is the tool developed in the context of the ODETTE project. Within this project, a language has been defined that adds to SystemC object oriented constructs synthesisable by a tool currently in prototype phase. SystemC+ descriptions can make use of constructs with "late-binding" procedure invocation semantics, thus introducing the possibility of applying the polymorphism, which is a concept whose usefulness is widely accepted in the software world, in the hardware modelling and synthesis task. Another important feature of this tool is the possibility to synthesise communication between structural elements (modules) of a model, described with a formalism that is syntactically and semantically very similar to the invocation of methods of a class. This possibility, offered in SystemC by the Interface Method Call (IMC) mechanism, is proposed in

1530-1591/04 $20.00 (c) 2004 IEEE

the ODETTE SystemC+ language subset with an ad-hoc formalism, and dramatically rises the abstraction level at which the communication can be not only modelled, but also synthesised. SystemC+ language definition will also be presented to OSCI for standardisation.

In this paper we present some modelling ideas that can contribute to the full exploitation of the synthesis capabilities of this new formalism. In particular, we show how these expressive features can be used to design a set of IP modules that can interface high level description of components of the system to the complex bus elements that often compose the communication infrastructure. By using such interfaces, the designer would be able to write the system model at the highest level of abstraction possible for the intermodule communication, that is by means of functions and procedure. This is what is actually recommended for the exploitation of most system level languages. After that, it would be possible to refine this communication in the quickest way by just picking the right interface IP among those provided by a library, and applying some straightforward syntactical adjustments. The methodology proposed constraints the structure of the interface elements with the following requirements:

1. the interface must offer a set of functionalities that encapsulate the transfer modes of the bus protocol;
2. these functionalities must be offered to the application in form of *guarded methods*; *guarded methods* are the mechanism provided by SystemC-Plus
3. to implement the high-level communication mechanism by which a process can invoke a method in the context of another process; the interface must implement the service offered to the application at pin-level accuracy towards the bus signals.

After describing the design methodology, we show the implementation of a representative element of such library. The chosen component is a PCI bus interface, that receives requests by an application in the form of function and procedure invocation and translates them into pin-level PCI operation requests. First the structure of the interface is defined: the *global object* components offered in SystemC+ to describe the communication between processes of different modules are exploited to connect an application module with an inbound interface module. The interface module consists of one of such global objects, needed to communicate with the application, and of several processes that implement the pin-level PCI protocol. Synchronization with the application is addressed with the use of *guarding conditions* upon the value of which the methods return; this semantics is exploited to obtain a blocking behavior of the interface methods towards the application. An application performing a series of bus transactions is modelled to act as a high-level ”stimuli generator”. After defining the structure of the interface as an example of the methodology proposed, several issues are analysed, and thus synthesis experiments and verification by simulation are performed and the results are analysed.

An interesting future work will be the evaluation of the temporal cost of the method calls: these are implemented with synchronous logic, and the completion of a transaction require an amount of time that depends on different factors (among which the number of concurrent processes accessing the same resource). Compliance with temporal requirements should then be evaluated after synthesis.

2. SystemC+ synthesisable subset extension and Global Objects

In the context of ODETTE project, one the major goals achieved was the definition of an extension of the SystemC subset synthesisable by most of the tools actually on the market. This extension was called SystemC+, and extends the synthesisable subset by adding:

1. the possibility of using classes and templates inside modules;
2. an hardware oriented version of the object oriented polymorphism;
3. a family of classes, called *global objects*, that can be used to implement inter-module communication described at high level;

A description using such features can be synthesised down to a mixed RT-behavioral level using the ODETTE synthesis tool. The result of the synthesis can then be handed to an RTL to gate synthesiser.

Among the presented ones, fhe feature considered for the methodology proposed in this paper are the global objects. These are classes in which some *guarded methods* are declared (syntactically by means of some C++ macros). Different global objects of the same

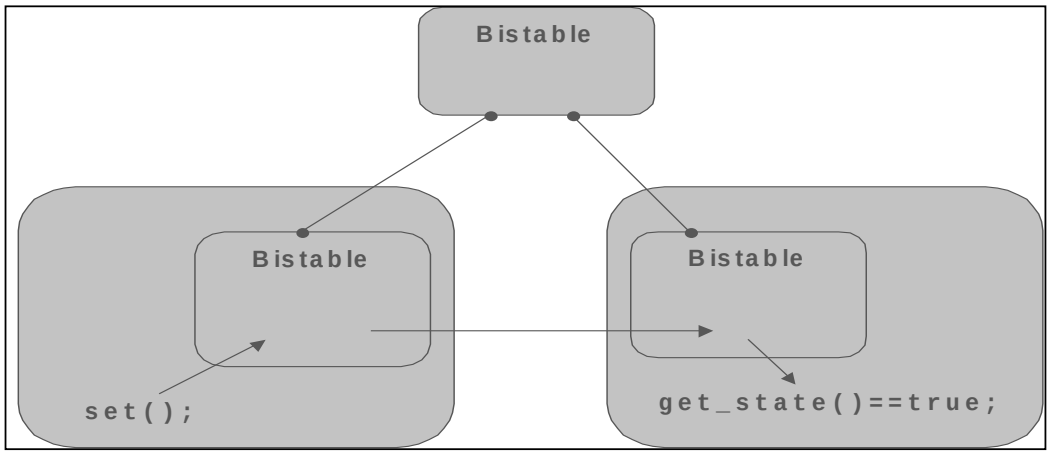

Figure 1 A shared bistable implemented as a global object

class can be instantiated in different modules; once instantiated, they can also be connected.

When two or more global objects are connected, a change in the state space of an object is reflected in the state space of the others. (see Figure 1). In the example of Figure 1, two modules contain a "bistable" class, declared as a global object. Another bistable is present at top level. All three bistables are connected together. When the `set()` method is invoked on the global object contained in the first module, the state change is induced also in the state space of the object belonging to the second module (or, equivalently, the change happens in the state space shared among all the objects). When the second module asks the object its state, the modification induced by the first is observable.

Thus, the semantic of this kind of communication is that all the connected global objects share a common state space.

Moreover, if different modules invoke at the same time the execution of a guarded method of a shared global object, the calls are queued and scheduled according to a user defined algorithm. This way, also concurrent access to resources is implemented.

The last feature is the possibility to guard for a certain condition upon the execution of a method: when declaring a guarded method, a Boolean condition can be specified; if the condition is evaluated true at the time of the method invocation then the call is processed; otherwise, the caller is suspended until the condition becomes true.

3. The bus interface design pattern

One of the possible design flows that could exploit the potentialities of the ODETTE tool chain is the one depicted in Figure 2. In this flow, the design starts from the specifications input. After that, an executable model of the system has to be realized.

The executable model is typically composed by three kinds of elements:

1. the units under design: these are the elements that have to be physically implemented on the target technology;
2. the IPs with which the units to be designed will interact; these can be memories, processors, peripherals, and all the other elements that will be part of the systems and that are already implemented;
3. a set of stimuli generators, that will simulate the working conditions of the system in the model

One of the greatest advantages coming from the use of highly expressive system description languages is the possibility to express the communication between the elements of the system model in a powerful and easy way. This let the designer focus on the definition of the functionality of the system rather than on the communication details, greatly speeding up the modelling task.

In a context such as the ODETTE one, high level descriptions of the elements under design are automatically converted down to abstraction levels (register transfer, for instance) that can be handled by traditional automatic synthesis tools. This approach imply all the "traditional" advantages and drawbacks of the automatic synthesis, but much earlier in the design flow.

One problem in applying such an automatic synthesis flow is the following:

1. in the executable model, functional descriptions of the IPs interacting with the elements under design can be given; in order to speed up the modelling phase and to exploit the high simulation speeds achievable with such descriptions, the communication must be described at a high level of abstraction;
2. after having simulated and validated the system model, the synthesis tools can be directly applied to the elements to be implemented;

The problem is that, while the functionality of the elements is synthesised, the communication between those and the IPs has to be taken care of.
A typical scenario would be the one in which the functional models of the IPs offer a transaction level interface, based on function calls, while in the implementation the interaction is performed through the use of some kind of bus. At this point, the problem of implementing the bus protocol handling would arise: the implementation of the bus protocol handling behavior could be, in principle, done on the high level model; the interactions with the transaction level IPs interfaces should be refined (or, rather, substituted) by a much lower level description of the bus protocol handler. After doing this, all the design should be revalidated with the new communication features; this procedure could be both time consuming and error prone, and could vanish part of the efficiency boost given by the use of automatic synthesis tools.

Performing the refinement on the post synthesis model is usually not feasible, since the synthesis algorithms often produce code that is not easily modifiable.
The methodological solution we propose to address this problem is the definition of a pattern, using which a bus interface can be designed in order to:

1. offer the designer the possibility to model the communication between the units under design and the peripherals at a high level of abstraction;
2. be synthesisable;

The main elements of the interface are:

1. one global object that implements the communication with the units under design;
2. the interface towards the models of the IPs; this can be a transaction level SystemC 2.x port, or it can be a set of signals, according to the kind of IP model;

So, for each communication abstraction level, an interface could be provided in order to connect the units under design to the IPs models dealt with.
The basic idea is that, when a proper library of such interfaces would be provided, in order to refine the communication from a high-level model down to its implementation, it would suffice to replace the high level interface with the appropriate one (see Figure 3).

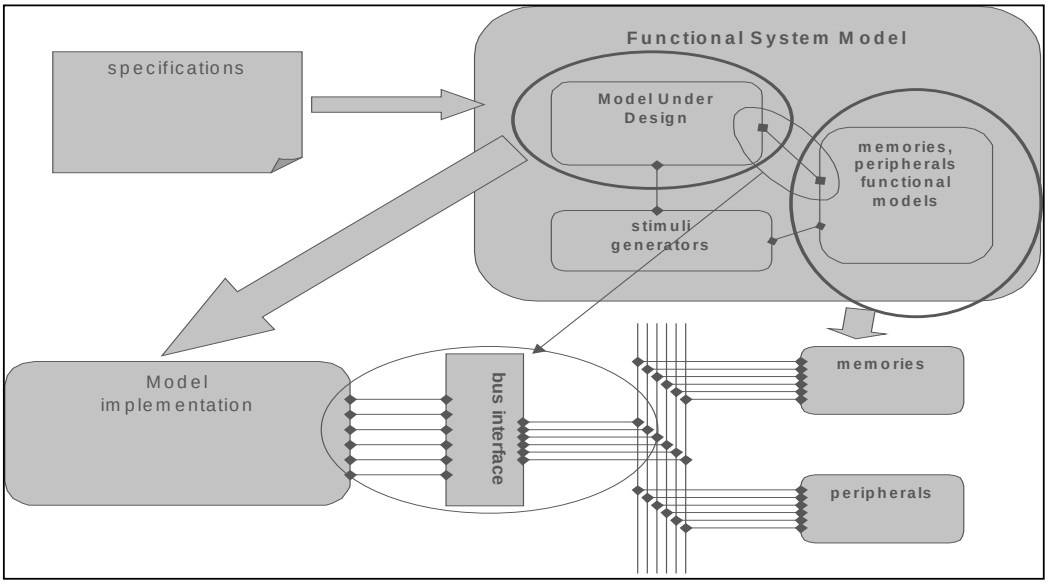

Figure 2 A possible design flow

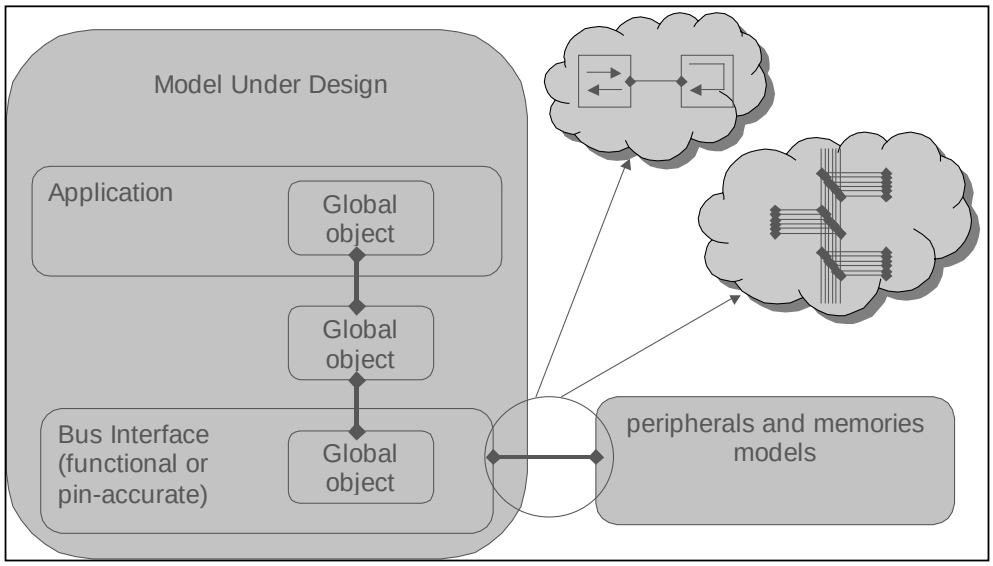

Figure 3 Communication refinement

In order to show how the interface between the application and the interface is design, we show and comment some of the *guarded methods* that define it:

```
GUARDED_METHOD(void,putCommand(CommandType& command),!isPendingCommand)
```

This method is invoked by the application (the module that uses the bus) in order to perform a bus operation. In the blocking version of the interface, the method is guarded upon the condition that there is no other command pending for execution; otherwise, the caller module is suspended until its request can be handled.

```
GUARDED_METHOD(CommandType,getCommand(),isPendingCommand)
```

This method is invoked by the processes that implement the bus protocol handling; it returns the command being asked by the application, if there is one pending; otherwise the calling process is blocked until a command arrives.

```
GUARDED_METHOD(DataType,appDataGet(),isApplicationReadData)
```

This method is invoked by the application in order to get the result of a read transaction. In the blocking version, it suspends the caller until the data is available.

```
GUARDED_METHOD(void,reset(),true)
```

This method is invoked in order to reset the interface. It cancels all the pending commands and perform other initialising operations.

As an example of a library element implementing such an interface, we implemented an handler of a simplified version of the PCI bus.

A simple stimuli generator has been chosen, together with a target PCI device, in order to build a test executable model. In order to verify the viability of the proposed synthesis flow, these steps have been performed:

1. the executable specification has been compiled and simulated;
2. the synthesiser was run in order to get an RT level description of the communication;
3. the resulting model was again simulated to check behavior consistency with the original model, at least with respect to the test set adopted;

The third step showed no problems arisen during the synthesis phase. Part of the simulation waveforms obtained from step 3 are shown in Figure 4.

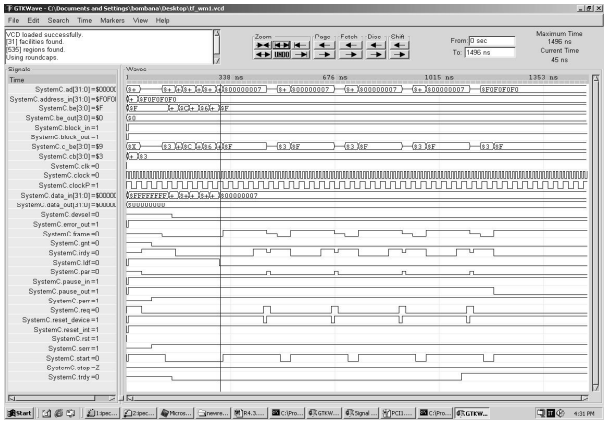

Figure 4 Simulation waveforms

4. Conclusions

In this paper we presented a methodological approach to the high level communication modelling that takes advantage of the SystemC+ SystemC extension provided for the synthesis by the tools developed in the ODETTE project. The problem of modelling the system at high level and being able to automatically synthesise it, together with the communication parts, has been addressed. A pattern has been proposed in order to model the communication between the application and the interface with using synthesisable *global objects.*

Viability of the proposed modelling and synthesis technique has been tested on the example of a PCI bus handler. Pre and post synthesis validation has been performed, showing the consistency of the flow adopted.

References

[RAL01] Grimpe, E., Ashenden, P. J., et al. (2001). *Input language subset specification (formal)*. Technical report, KuratoriumOFFIS e.V. Public result of the IST FP5 project ODETTE.

[GRAL02] Grimpe, E., Biniasch, R., Fandrey, T., Oppen-heimer,F., and Timmermann, B. (2002). *Systemc object-orientedextensions and synthesis features.* In Forum on Design Languages(FDL'02), Marseille.

[CHCO99] H. Chang, L. Cooke, M. Hunt, A. McNelly, G. Martin, andL. Todd. *Surviving the SOC RGevolution: A Guide to PlatformBased Design.* Kluwer Academic Publishers, 1999.

[GRLI02] T. Grotker, S. Liao, G. Martin, and S. Swan. *System Design with SytemC. Kluwer* Academic Publishers, 2002.

[SCC] Synopsys. *CoCentricSystemC Compiler Behavioral Modelling Guide*