A Rapid PCI NoC Prototyping Platform with VCI Interfaces Methodology

Dominique HOUZET
IETR-INSIA
20 av. des Buttes de Coësmes
35043 Rennes France
tel : (33)223238452
taxi : (33)223238439

Salim OUADJAOUT
M3Systems
1, rue des oiseaux
31410 Lavernose France
tel : (33)562231089

Abstract
System-on-chip (SoC) designs provide integrated solutions to challenging design problems in the telecommunications, multimedia, and consumer electronics domains. Much of the progress in these fields hinges on the designers ability to conceive complex electronic engines under strong time-to market pressure. Success will rely on using appropriate design and process technologies, as well as on the ability to interconnect existing components including processors, controllers, and memory array reliably, in plug and play fashion.

This paper presents a prototyping methodology and flexible hardware platform developed to facilitate rapid prototyping of such communication systems.

Keywords: prototyping platform, SoC, NoC, VCI, Interfaces Synthesis

1. Introduction
The transition to SoC design is not straightforward and brings different challenges. While systems complexity is growing, the time-to-market window is decreasing. This can be dealt in two ways: by raising the design level of abstraction or by reusing IP. The first solution requires good high level synthesis tools and description languages. The second is accomplished by socketization. Then companies need an infrastructure for reuse to mix and match IP of different source. Thus, standards are very useful.

This work uses standards for On-Chip-Bus (OCB) communication, developed by VSIA [1], to explore design architectures and synthesis methodologies in a real-time platform.

The first part presents the design flow and methodology that allow a rapid integration of communication by using a standard communication protocol, VCI from VSIA. Software drivers and glue logic are generated to connect processors to peripheral devices, hardware co-processors, or communication interfaces while meeting bandwidth and performance requirement [10].

The second part presents the hardware prototyping platform based on a configurable Network-on-Chip (NoC) [5] implemented with the PCI-X bus. The prototyping platform is used to validate the whole SoC designed around the proposed NoC, with a real-time execution. Numerous papers have described solutions to help the design of SoC based on micronetwork [4, 7].

2. Related Work
Hardware-software co-design approaches differ from one another by their target architectures and by their objectives.

The synthesis from specification allows integration of application-specific and programmable component. High-level descriptions of system are refined into a physical implementation. The focus is put on one of the following points: interfacing, partitioning, refinement, synthesis and simulation.

The platform approach has a goal to map an application into a pre-designed complex SoC architecture, by configuring and extending this architecture.

The last solution, IP assembly must be capable of reuse without redesign work. This ability lets designers treat IP cores as autonomous plug-and-play subsystems in subsequent designs. Designing SoCs based upon reusable IP cores is essential for meeting stringent time-to-market requirements. Once suitable IP cores are identified, the focus shifts
to the integration challenge: how to build a working system from a collection of generic and domain-specific cores that were not designed to work together [9, 10].

One of the key issues in hardware/software co-design is the generation of suitable interprocess communication interfaces, and to determine the amount and type of communication between different components of a digital system. The problem of communication is one of the main obstacles to obtain full benefit of high performance components.

The availability of reconfigurable technologies has enabled the construction of flexible systems allowing run-time reconfiguration of system hardware and software functions.

The complete hardware/software system will be emulated as an architecture–precise prototype on a real-time platform to verify the derived communication architecture as well as the hardware/software partitioning.

Several other approaches have been developed that are based on a similar point of view:

The Cosy [2] approach, based on the infrastructure and concepts developed in the VCC [3] framework defines interfaces at multiple levels of abstraction. The top level, called application, is purely functional. Application-level transactions are then refined into system transaction when a hardware or software implementation of the upper layer functions is selected. This layer deals with selecting communication parameters and solving latency-throughput tradeoffs. Below, the Virtual Component Interface is adopted to interface with specific Physical bus protocols.

Sonics distributes a micro-network tool suite which describe the interfaces at the RTL level. This Sonics FastForward Development Environment[11] constitutes a suite of tools that configure the Sonics μNetwork to meet the specific needs of each application and drive the SOC integration process.

The FastForward Development Environment represents a cohesive methodology for developing SOC designs. The tool suite guides designers through every phase of the SOC integration process.

3. Design Methodology

The design flow supported by the prototyping design methodology is depicted in figure 1. There are the main features of the presented methodology:

**Figure 1:** Methodology design flow
The design flow starts with system-level model described in high-level programming language, that is SystemC.

The functional behaviour of the model system will be simulated with CoCentric tools for specification verification.

During the co-design phase, the system model is partitioned with VCC tools into embedded software for the host processor and the LEON(Sparc) embedded processor, and hardware code for the FPGAs. As the implementation design flow for both targets is based on SystemC, users can easily change the system partition to evaluate several different partitioning tradeoffs.

We use GCC compiler platform for the development of embedded software for the target host and LEON processors. Other standard software development tools are also used for debugging this software system partitioning to get optimised code for host and LEON processors. Soft interface synthesis is also performed for each code. Interface synthesis is further discussed in following section.

In the System–level design flow for hardware system partitioning, the SystemC compiler and FPGA compiler tools translate an input behavioural SystemC description into target technology gate-level netlist. The netlist is then placed and routed by standard Xilinx layout tools, which produce a full FPGA configuration bitstream. In the same time, we generate a hardware interface for each implemented component.

The final product of the design flow is a set of binary files representing programs for host processor, LEON processors and the FPGAs. These can be loaded onto the respective components of the prototyping platform (see section 4) to build a prototype with a real-time communication system.

3.1. Interface synthesis

The main goal in hardware/software interface synthesis is to generate a communication link using minimal glue logic while respecting time constraints. Automating the hardware/software interface synthesis allows designers to focus on higher-level decisions. This approach requires the development of the appropriate abstractions for device interfaces so that device drivers can be generated automatically. The I/O capabilities of processors must be specified in general form. These include directly manageable I/O pins, interrupt mechanisms, and system bus interfaces (PCI, USB, ...). The two sides of the interface given (device and processor), the tool should then be able to determine the best way to interconnect them and generate the corresponding interface software and hardware.

Our approach is to divide the link between hard and soft virtual components (VCs) in several layers (Figure 2) connected through a standard interface. The new system will contain hardware and software components. The communication between the heterogeneous components can be made through classic busses such as PCI, USB, or On Chip Buses (OCB) such as AMBA, CoreConnect, etc. This imposes the existence of a wrapper on each side of the bus. Our physical platform uses PCI for the OCB as well as for the external bus.

![Figure 2: Layers between hard and soft VCs.](image-url)
This wrapper behaves like a bridge by translating the signals between the bus and the component. The wrapper is quite specific to each bus and it must have a compatible interface with any type of component, possibly a VCI interface [1, 9].

With this approach, the software part and the hardware part of the adaptation modules are quite similar. This approach requires to provide two wrappers for each side of the communication bus: a device driver (soft) which is compatible with VCI interface and a bus controller IP (hard) which is also compatible with VCI interface [1].

Thus, the interface synthesis is limited to the generation of the adapt code between the user interface and the VCI interface of the driver and IP. We can see the VCI interface as an intermediate interface.

3.2. Virtual Component Interface
VSIA gives a solution to the standard SoC problem by providing common communication principals, a common design format and unified approach to design quality measurement and assurance. In particular, VSIA provides an OCB Virtual Component Interface (VCI) standard that defines a general interface, such that intellectual Property, in the shape of virtual components of any origin, can be connected to SoC of any chip integrator.

A VCI interface needs a data bus, an address bus and command signals (read, write, req, ack...). The protocol of a VSIA interface corresponds to a DMA protocol sending data blocks, one word at each clock cycle. The word corresponds to the size of the bus.

As an interface, the VCI can be used as point-to-point connection between two units called the initiator and the target, where the initiator issues a request and the target responds. The VCI can also be used as the interface to a “wrapper”, a connection to a bus. This is how the VCI allows the virtual component to be connected to any bus.

The VCI protocol is a handshake point-to-point protocol. Each transfer is done on a rising edge of the clock when valid and acknowledges signals are high. The request-content flows from initiator to target and the response-content flows from target to initiator.

There are three complexity levels for VCI: the peripheral, basic and advanced protocols.

Other generic protocols have been proposed, like Open Core Protocol Interface, developed by Sonics Inc. [11].

The Sonics µ-network approach decouples the design of the communication among IPs. Each IP core communicates with an agent in the Silicon Backplane using a protocol called OCP and agents communicate with each other using network protocols. Both protocols can be customized by the SoC designer who can configure parameters such as data width and buffer depth.

3.3. Objective:
What we are proposing is a CAD tool which aims at generating a VCI interface, from an abstract interface description at a higher lever than RTL level. This description is defined in a high-level language (SystemC). This tool allows a fast integration of IP blocks in a PLUG and PLAY fashion by taking care of the generation of the “Adapt” code.

3.4. Methodology
We propose the use of the µ-network stack paradigm, an adaptation of the protocol stack shown in figure 3, to abstract the electrical, logic and functional properties of the interconnection scheme.

![Figure 3: µnetwork protocol stack](image)

3.4.1. Presentation layer
The presentation layer defines the adaptation solution between user interface and VCI standard interface.
A first solution is to impose a Fifo type programming style. This solution forces the designer to use a certain formalism of programming to model the communication between its virtual components. This modeling is done through configurable objects (FIFOs) defined in the programming environment (VHDL, SystemC). This is, in this case, an extension of a description language.

This approach is used by LINUX to establish the communication between the environment that is real-time and the environment that is not real-time. In this precise case, the programmer has to manage the data flow through the FiFos by serializing the data according to the size of the bus of the FiFos (8, 16, 32, 64 or 128 bits).

The advantage of this solution is that the control signals and operations are defined implicitly, and that the relation between the signals of the same signals group is predefined. But this solution has a major constraint which concerns the freedom of the design and the programming of the interfaces, because the latter forces the designer to follow mandatory formalisms in order to express the communication.

In the second solution, we have used a SystemC [8] code as a starting point. This last one has the merit to cover the RTL and the specification levels, and to build a library which offer support for different semantics.

Figure 4 presents an example of user interface code of a SystemC component.

```c
SC_MODULE (myexample) {
    Sc_in<datatype> seq;
    Sc_in<cmdtype> usercmd;
    Sc_inout<bool> startdata;
    Sc_inout<bool> startcmd;
    Sc_in<bool> clock;
    sc_in<bool> reddefA;
    Sc_out<int> sout[nbcmd];
    int ind;

    Void proc_example();
    SC_CTOR(myexample) {
        SC_THREAD(proc_example);
        Sensitive << startdata;
        Sensitive << startcmd;
        Sensitive_pos << clock;
    });

Figure 4: SystemC code of a component interface (.h)
```

the adapt code requires a knowledge of the roles (types) of signals which constitute this interface: the control signals, the command signals, the addressing and data signals,... and also, a definition of the relations which exist between these signals: signals requested for the execution of a transaction of a particular command.

In the figure 4 example, we can detect transfer requests when sensitive signals receive events (startdata, startcmd or clock). When one of these signals change, a block transfer is performed.

We need to determine the corresponding data which need to be transferred. This is done either with a description file provided by the programmer describing the link between event signals and data signals, or by a compiler which analyzes the code and automatically obtains those links through data dependencies.

If in the code of the module, the use of userdata is dependant of the test of startdata, then there is a link between those two signals. Here it is an example of links description file for this example:

```
Startdata -> userdata
Startcmd -> usercmd
Clock -> nil
```

That means when startdata change for instance, the device driver has to transfer the userdata block, word by word. If several sensitive signals change, the scheduler has to serialize the transfers. All this code can be generated automatically. We have designed IPs and device drivers with VCI interfaces for PCI-X and LEON busses. We are designing the synthesis tool that reads a link description file and generates the corresponding code to adapt a user interface to a device driver or IP with VCI interface.

Our work also consists to adapt the size of the user interface ports to the size of the data bus through serialization, that is block transfer. This treatment is part of the adaptation code we are generating.
The Network control dynamically manages network resources during system operation, striving to provide the required quality of service. Following the micronetwork stack layout shown in figure 3, we describe the control transport layer. We assume that data-link and network layers are provided with the wrapper.

3.4.2. Transport layer:
Atop the network layer, the transport layer decomposes messages into packets at the source. It also resequences and reassembles the messages at the destination. The segmentation is the process of breaking the users data unit into smaller segments that can be handled by the underlying data-link layer service. De-segmentation is the corollary of this process and comprises the re-assembly of the original users data unit from the segments received from the underlying data-link layer service.

The second treatment is the multiplexing and de-multiplexing of the transfers. When we partition a system, there are several components and then several connections between hard and soft components. Those connections have to be scheduled in time to use the common physical bus. This scheduling uses DAG algorithm to optimize delays and meet the timing constraints. The priority management of the different connections depends on the application constraints provided statically or dynamically as quality of service requests (QoS).

4. Hardware prototyping platform
The aim of the previous methodology presented is to map an application on the micronetwork proposed by configuring both the micronetwork interfaces and the application synthesized interfaces in order to meet the application constraints.

Our platform is based on a micronetwork infrastructure which implements the lower level layers of the communication stack protocols. Each layer of the whole micronetwork is configurable in terms of architecture, that is bus numbers, sizes, frequencies and organization.

An overall overview of the micronetwork architecture used in our prototyping platform is presented in Figure 5.

![Figure 5: μnetwork architecture](image)

This micronetwork is based on the PCI-X bus. It is designed to integrate the future PCI-Express standard[6]. The choice of the PCI bus is guided by the maximal re-use of wrappers, boards, test equipments available for the wide-spread PCI bus. With that bus, it is possible to build a low-cost hardware prototyping platform which integrates the busses which will be implemented in the final SoC. The platform can be configured to match the structure of the SoC in terms of busses sizes and organization. In that way, the entire SoC is manageable and testable in real-time from a PC host computer through its PCI bus.

This platform consists of a hierarchy of 64-bit PCI racks (P7T from Magma Inc.) receiving seven 8-FPGA boards each. Figure 6 presents an example of a 2-level hierarchical platform with two racks connected to a host computer used for test purposes or for the modelling of a computer application software part.

![Figure 6: 2-rack platform](image)
The 8-FPGA board designed is presented figure 7. We are using 8 Xilinx XC2V3000 FPGAs connected via the PCI-X bus. This board will be redesigned to use the new PCI-Express standard [6].

An example of a SoC floorplan designed around the PCI NoC with the platform presented here is shown in figure 8.

The external PCI bus is visible on the left of the chip. This example is architected with a 2-level hierarchical PCI-bus with four busses at the second level of hierarchy. The platform used to model this SoC is configured with one rack of four 8-FPGA boards.

We have used this prototyping platform and methodology on a high performance demanding system, that is a multisensors UMTS simulator with adaptive antennas, based on 16 RF modules connected to 32 treatment units. This simulator emulates radio channels fading behaviors. The 32 units are connected to the 16 RF units through the PCI NoC. In this configuration, two 8-FPGA boards were used with two units and one RF module per FPGA. This project is part of the SYMPAA project [12] conducted by TEMEX in collaboration with ST and FRANCE TELECOM. The bandwidth needed by the application was 1Gbytes/s, this is the maximum bandwidth of the NoC.

5. Conclusion

In this paper we have described a platform based methodology to design large and complex systems on a single chip. The communication mechanism (NoC) defines five layered inter-resource communication protocols (physical, data-link, network, transport and presentation layer). These protocols are implemented in the network interfaces for every resource in the NoC. They are generated and configured to be optimized according to the application constraints.

We have also described a physical prototyping platform which implements the NoC and the whole SoC under design, with expandable low-cost racks and boards.

This platform and methodology concepts have been necessitated by the need to amortize the enormous engineering cost involved in designing and testing large chips and by the demand for easy-to-use methods to exploit the parallel processing capacity provided by multiple computational resources.

Currently, we are building various simulators for evaluating various communication options at different levels.

6. References


Salim Ouadjaout is a PhD candidate in electrical and computer engineering at IETR-INSIA Rennes, France. He is also working as research engineer at M3 Systems Inc, Toulouse, France. His research interests include interface synthesis and micronetworks for SoC. Ouadjaout has a MS in computer science from INI Alger, Algeria and a MS from IRIT-ENSEEIHT Toulouse, France. He is a student member of the ACM.

Dominique Houzet is an Associate Professor of electrical engineering at IETR-INSIA Rennes, France. His consulting, training and research interests include logic synthesis, SoC prototyping platforms, micronetworks, and software-hardware co-design. Houzet has a MS and a PhD in computer science from IRIT-ENSEEIHT Toulouse, France. He is an IEEE member.

Salim Ouadjaout is a PhD candidate in electrical and computer engineering at IETR-INSIA Rennes, France. He is also working as research engineer at M3 Systems Inc, Toulouse, France. His research interests include interface synthesis and micronetworks for SoC. Ouadjaout has a MS in computer science from INI Alger, Algeria and a MS from IRIT-ENSEEIHT Toulouse, France. He is a student member of the ACM.

Dominique Houzet is an Associate Professor of electrical engineering at IETR-INSIA Rennes, France. His consulting, training and research interests include logic synthesis, SoC prototyping platforms, micronetworks, and software-hardware co-design. Houzet has a MS and a PhD in computer science from IRIT-ENSEEIHT Toulouse, France. He is an IEEE member.