Combining Architecture Exploration and a Path to Implementation to Build a Complete SoC Design Flow from System Specification to RTL

M.Anouar Dziri*, Firaz Samet*, Flavio Rech Wagner**, Wander O. Cesário*, Ahmed A. Jerraya*

* SLS Group, TIMA Laboratory
46 Av. Félix Viallet, 38031 Grenoble, France
{Mohamed-Anouar.Dziri,Firaz.Samet,Wander.Cesario,Ahmed.Jerraya}@imag.fr

** UFRGS
Porto Alegre, Brazil
flavio@inf.ufrgs.br

Abstract - This paper presents a full System-on-Chip (SoC) design flow from system specification to RT-level. A new approach to obtain a full path to implementation for SoC design is proposed. This approach combines architecture design space exploration using the VCC design environment and system synthesis using the ROSES design flow, allowing a true and complete system level design flow. The experiment with a VDSL application shows a significant reduction of design time.

I. Introduction

The design of embedded systems-on-chip is a complex process, involving different steps at different abstraction levels. Design steps can be grouped into two major tasks: architecture design space exploration, for hardware/software partitioning and selection of architectural platform and components, and architecture design. The overall design process must consider strict requirements, regarding time to market, system performance, power consumption, and production cost. The reuse of IP components from several vendors is necessary for reducing design time, but their integration into a system also presents various challenges. Most architectural platforms include programmable processors, so that software design in multiprocessor environments becomes a major issue.

A complete design flow for SoCs is then necessary to cover this design complexity. This refinement process requires multiple competences and tools because of the complexity and diversity of the current applications.

Current and previous works try to reduce the gap between different design steps and to master the integration of very diverse components, including hardware and software parts. For a full path to implementation, earlier approaches as COSYMA [16] and Vulcan [17] had a very restricted success. VCC [2] and Artemis [1] are two approaches that deal with the architecture design space exploration step. ROSES [21] and CoWare [6] give a solution for the architecture design step.

Existing approaches deal only with a specific part of the SoC design flow. A full system level flow is quite complex, and to our knowledge very few work cover both system-architecture exploration and system-architecture design.

This paper presents a novel approach that implements a full path including SoC architecture exploration and system design, considering hardware/software architectural solutions along the whole design path. This approach combines the Cadence Virtual Component Co-design (VCC) and the ROSES design environments in a consistent way. It bridges the gap between easy design space exploration supported by VCC and low-level system design supported by ROSES, allowing a true hardware/software co-design.

This paper is structured as follows. The next section presents an ideal system level design flow and reviews related work. Sections III and IV introduce the basic design methodologies, architectural models, and benefits of the VCC and ROSES environments, respectively. The solution for bridging the gap between VCC and ROSES is detailed in Section V. In section VI, the application of this full path to implementation to the design of a VDSL framing is presented and results are analyzed. Section VII summarizes and concludes the paper.

II. A Full System Level Design Flow

A. System level design flow architecture

A full design flow, as shown in Fig.1, starts from a system-level specification and reaches an RTL architecture, from which the synthesis of hardware and software components can be performed with conventional design tools. The first step of the design flow is a contract between the end-customer and system designer dealing with an informal model of the application. System designers then build a formal SoC specification – a behavioral or functional model – that the end-customer can validate. Next, they fix the system architecture, typically reusing IP components such as processors and memories, and decide about the mapping of function to architecture for the application functionality, by assigning behavioral components to architectural ones. This step generally uses an executable specification model that allows designers to go through a performance analysis loop. In order to obtain an executable specification model at this stage, system designers build a simulation model using software profiles for parts of the application and abstract models for the hardware components.

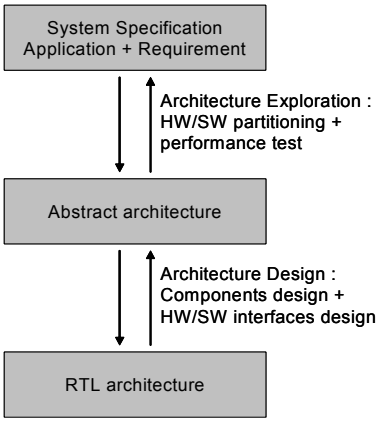

Fig. 1. System-level design flow for SoCs

System architecture exploration fixes the specification of the hardware components (e.g. selection of existing processors or specific hardware), the partitioning of functional tasks into hardware and software components, and the global structure of the on-chip communication network. This step produces an abstract architecture, which does not contain all details of the interfaces between components and is used as a golden architecture model that guides the next step: SoC architecture design.

SoC designers need to interconnect the hardware and software components while respecting the performance constraints described in the golden architecture model. This step results in an RTL architecture (also called a *micro-architecture*), involving the generation of all hardware and software interface details at a pin-and-cycle-accurate level. In parallel, hardware and software designers can implement those components in conformance with the golden architecture model.

B. Related Work

Many academic and industrial works provide tools for SoC design automation that covers many but not all design steps presented before.

There are very few tools on the market that allow a real design space exploration at the macro-architecture level, allowing an easy performance estimation of different architectural choices and hardware/software partitioning.

[20] proposes a framework for object oriented hardware specification, verification, and synthesis an object oriented language 'e' with a powerful run-time environment that enables designer to perform the verification task. COSY [10] proposes a hardware/software communication refinement process that starts with an extended Khan Process Network model for system specification. SystemC [11][19] is a library based on C++ classes for describing hardware/software systems at multiple levels of abstraction and provides mechanisms for system simulation. SpecC [15] proposes a methodology based on an extended C language, but architecture exploration requires manual recoding. Artemis [1] is a modeling and simulation environment aimed to explore the design space of heterogeneous embedded-systems architectures at multiple abstraction levels. VCC [4] performs architectural design exploration, offering fast performance estimations for different macro-architectures and hardware/software partitionings. But this macro-architecture can be hardly synthesized, because available hardware and software models are far from real components.

There are also very few tools that allow a complete and automated generation of pin-and-cycle-accurate hardware and software interfaces between arbitrary IP components, starting from a macro-architecture. Common approaches to IP integration either suppose that IP components follow some bus or core standard and can be thus directly connected, and/or require a manual generation of interface details, and/or consider only hardware interfaces. CoWare [9][18] can be used for hardware/software integration and simulation, but still many architecture details must be implemented manually.

It can be concluded that there is no single design environment that completely covers the ideal design flow depicted in Fig.1. In order to build a complete design flow, this paper introduces an adequate combination of the VCC and ROSES environments.

III. Architecture Exploration Using VCC

The VCC design environment is aimed at architectural design space exploration [3]. This is achieved by exploring hardware/software partitioning alternatives to reach optimal design performance within the given constraints. Once a suitable architecture is found, vendor libraries may be searched to find virtual component models that meet model specifications.

The methodology, as shown in Fig.2, starts with a functional specification and simulation of the system as a network of behaviors, with no implication of an eventual architecture implementation. An architecture platform composed of IP components (CPUs, IPs, memories, buses...etc), is then captured. Besides components corresponding to real IP in the final SoC, an RTOS or scheduler must be attached to each processor. This is a relaxed architecture, for which only the basic topology is defined, with no detailed information on ports and signals.

Behavior components are then mapped to the architectural ones, thereby defining a hardware/software partition. Each communication wire in the behavior diagram may be mapped to a communication pattern, which details how communication is implemented between behavioral components.

For performance simulation purposes, each architecture component must have a performance model, which is described by means of component services, such as a scheduler, for an RTOS, and bus and interrupt requests, for a processor, and read and write transactions, for a memory.

Mapping is a continuous refinement process, whereby new design decisions are taken, for instance to consider bus traffic, behavior delay, and memory accesses, and the

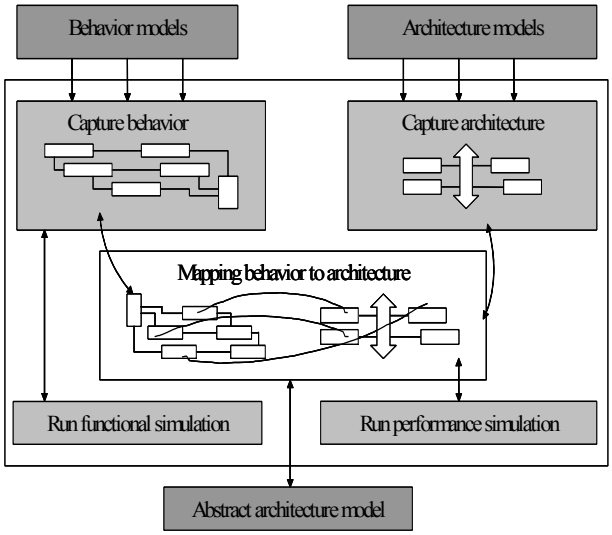

Fig. 2. The VCC design flow

resulting performance is analyzed. Communication patterns may be refined to consider protocols, shared memories for communication, interrupt mechanisms, concrete data types, and bus and memory addressing. DMA and cache components can also be included. Mapping diagrams and communication patterns may be easily modified, so as to explore alternative architectural solutions.

IV. Architecture Design Using ROSES

ROSES [21] is a design methodology with a corresponding set of tools, aimed at bridging the gap between heterogeneous system specification and system-on-chip implementation. The ROSES synthesis process, as shown in Fig.3, starts by capturing an abstract architecture composed of virtual modules (VM), corresponding to processing and memory IPs, connected by a communication network, also encapsulated within a VM. The ROSES virtual architecture model, illustrated in Fig.3.(a), clearly separates behavior and structure from communication. This allows separate and concurrent implementation paths for components and for communication. Virtual modules that correspond to processors may be hierarchically decomposed into sub-modules containing software tasks assigned to this processor. This architecture is described in SystemC.

Wrappers are then automatically generated as point-to-point adapters between the VM and the communication network. They are assembled as combinations of basic components contained in an extensible library. Wrappers may have hardware and software parts [8][12].

The hardware parts are adapters to be physically synthesized, and ROSES specifies their HDL descriptions for the further synthesis process. They may be considered as communication co-processors (CC, in Fig.3.(b)) that operate concurrently with other processing functions. Software wrappers include drivers to the hardware and a dedicated OS including only the functionality needed for

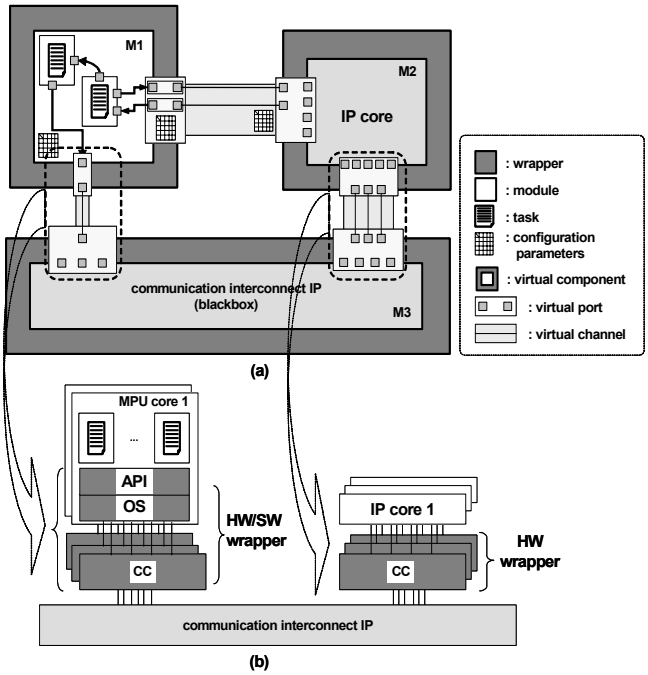

Fig. 3. ROSES design flow

the application.

The synthesis process is guided by design decisions to be previously taken, for instance resulting from a design exploration tool, including protocols, processor types, memory and bus addresses, interrupt levels, and port and net data types.

V. Combining VCC and ROSES

A. Application Domain

VCC is a powerful design space exploration environment offering useful facilities to easily look for different architecture and hardware/software partitionings in order to find optimal design performance.

The architecture resulting from the VCC design flow imperatively contains IP components corresponding to processing functions – processors and hardware IP's – and to the communication network - buses, bridges [4]. The architecture must also contain schedulers or RTOS attached to each processor. It eventually also contains other components that have been included to implement communication patterns, such as shared memories, DMA controllers, cache memories, and interrupt buses.

A data bus may have been included to implement a point-to-point connection between two components. In this case, the corresponding performance model will consider only delays incurred by tokens traversing the path. If the communication path is shared, the bus will also model impact of contention and arbitration. An interrupt bus may have been included to implement an interrupt-based communication, but this must be considered as a modeling abstraction, and in the final implementation it will be physically merged to a chosen data bus.

Unfortunately, VCC does not yet provide a clear path to implementation allowing low-level system synthesis. Design have to be fine-tuned manually until reaching low level system specifications.

Designers must implement hardware and software primitives, selecting CPUs and ASICs, specifying HW/SW communication, allocating addresses to send data between architecture instances and between software and architecture instances, co-verifying hardware and software parts,...etc [5]. This tedious manual effort obliges us to seek other more effective solutions than the VCC link to implementation.

On the other hand, ROSES provides a clear path to implementation, starting from an abstract architecture and reaching an RT-level HDL description for hardware parts and a detailed C implementation for the software parts, and a set of tools automating the design refinement process.

The ROSES initial architecture contains virtual modules (VM) that correspond to all processing and memory IPs of the final SoC. The whole communication structure, including one or more buses, bridges, etc. must be modeled as a single communication network, considered as a black-box VM. IPs must have point-to-point connections to the communication network. The architecture must not contain bus bridges or any other IPs related to protocol conversion or component adaptation, since this functionality will be automatically synthesized within the wrappers.

Unfortunately, ROSES does not yet provide an architecture design space exploration allowing HW/SW partitioning and system performance evaluation.

As we can see, VCC and ROSES have complementary goals and cover different design abstraction levels, shown in Fig.1. A reasonable approach is to combine the VCC model with the ROSES virtual architecture model, making it possible to go into design space exploration process and, once the optimal configuration found, to automatically generate a virtual architecture structure in order to launch the automatic refinement process.

It is necessary that the final architecture in the VCC design flow exactly matches the abstraction level expected at the input of the ROSES synthesis process, which must be guided by particular design decisions, several of them related to communication refinement such as protocols, attached to ports and/or nets, and port and net data types.

Examples of design decisions related to communication refinement that can be optionally taken in VCC are: choice of bus types, widths, and arbitration mechanisms; choice of protocols; choice of scheduling policies for software tasks; inclusion of shared memories to implement communication; definition of bus and memory addressing; and choice of data types.

B. VCC to ROSES Link

A tool prototype has been developed to ensure an automatic link between VCC and ROSES. It extracts information about the system structure in VCC, identifies the system components and assembles them in a ROSES compliant manner.

Many steps are necessary to achieve the VCC-ROSES link: the tool initially analyses the mapping view in order to get the hardware/software partitioning decisions. In other words, it extracts the assignment between behavior and architecture components. During the next step, the tool extracts behavior information. It analyses the behavior blocks and gets the communication network structure, the port parameters and the path to the source codes of each block.

Architecture analysis works in the same way as the behavior analysis, but the difference is that the blocks represent architecture components and do not make reference to source codes. Since the communication network in the architecture model is considered as a modelling abstraction, the ROSES communication network will be determined by the behavior and the mapping decisions:

1- If a net connects two behavior blocks mapped to the same architecture component, the connection is quite straightforward and, as shown in Fig.4.(b)(1), a net is created between the two ports involved in the connection.

2- If the connection is to be made between two behavior blocks mapped to different architecture components, a hierarchical port with internal and external sides have to be added to each architecture module and three connections are established: the first connection, shown in Fig.4.(b)(2), is between the two architecture blocks via the external sides of the hierarchical ports. The two other connections are between each behavior block and the architecture module via the internal side of the hierarchical port.

Finally, the virtual architecture model of ROSES is completed with design parameters from the VCC model (e.g. task priorities).

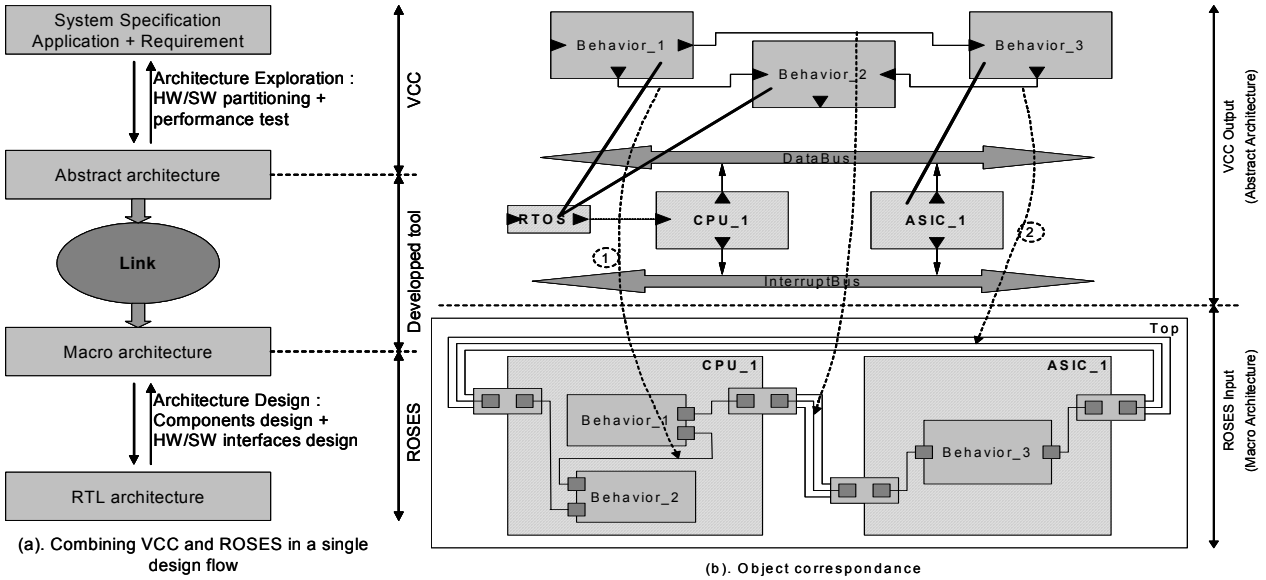

Fig. 4. The translation method

VI. Case Study : framing/deframing unit in a xDSL modem

This section demonstrates the application of the VCC-ROSES full path to implementation using a VDSL application as design example.

A. The VDSL Modem Architecture Specification

The design presented in this section was taken from the implementation of a VDSL modem using discrete components. Fig.5 shows the block diagram for this prototype implementation. The part that was redesigned as a multiprocessor SoC includes two ARM7 processors and part of the data path, the TX_Framer, which is described at the RT-level and used as a black-box hardware IP.

The same partition of processors/tasks as suggested by the design team of the VDSL-modem prototype is adopted [7][14].

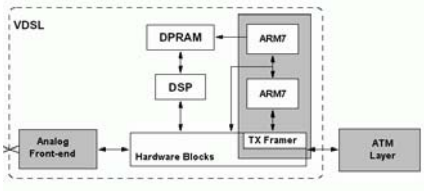

Fig. 5. The redesigned part of the VDSL modem

B. The VCC Model of The VDSL framing

The VCC model, as shown in Fig.6, includes three main diagrams. The first one is the behavior diagram describing the VDSL framing from a functional point of view. The second one is the architecture diagram, which includes the architecture components represented by two processors and an ASIC representing the black-box IP component. And finally the mapping diagram corresponds to the proposed hardware/software partitioning.

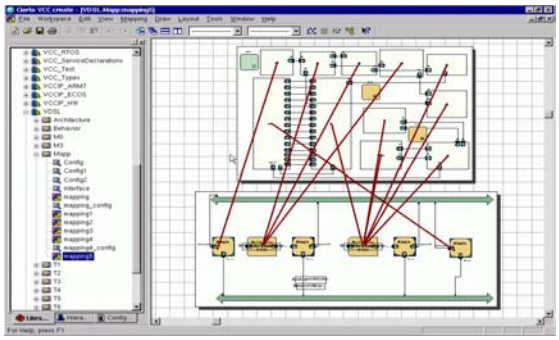

Fig. 6. The VCC model of the VDSL framing

C. The Resulting ROSES Virtual Architecture Model

From the architectural components and mapping decisions in VCC, a ROSES virtual architecture model has been generated, using the translation process explained in Section V. The generated model is based on point-to-point connections to match the current synthesis capabilities of ROSES. Fig.7 shows the overall virtual architecture after the partitioning. This is made of three modules, two software are that will be implemented on two ARM7s and the hardware module.

The communication synthesis has been performed by the application of two different tools. The Application Specific Operating System Generator (ASOG) tool [13] has been applied to the VM1 and VM2 virtual modules,

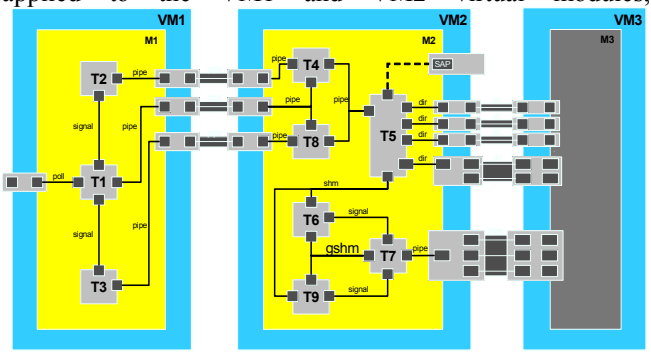

Fig. 7. The virtual architecture model

corresponding to the ARM7 processors, to synthesize the software communication within each module and to generate their dedicated OS. The Application Specific Architecture Generator (ASAG) tool [8], in turn, synthesized the hardware communication channels between the three modules (the ARM7 processors and the IP block). The refined VM2 module is shown on Fig.8. This includes an ARM7 module and a sophisticated interface to manage multipoint communication.

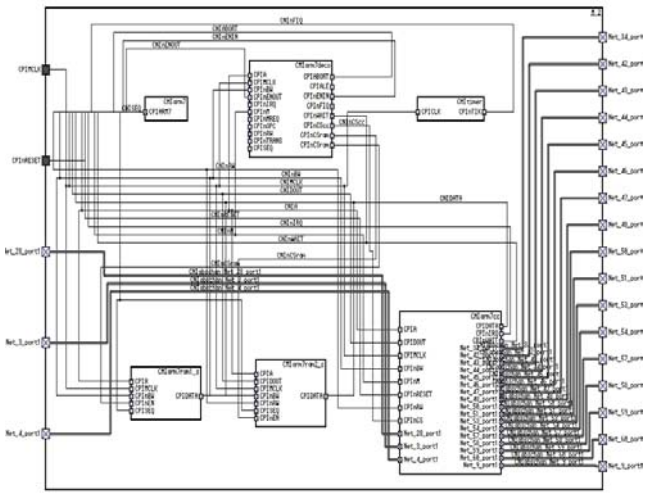

Fig. 8. The VM2 module refined with ASAG

D. Results and Evaluation

The design of a full VDSL modem requires more than one hundred man-years and the manual design of the presented subset was estimated as more than five man-years effort. Previous work [22] has shown that using ROSES brings this time down to six months for implementing a given architectural solution. The design space exploration for the hardware/software partitioning decision takes a very important part of the design time. When using VCC, the time to explore the design space of a specific application can be considerably reduced. It took only 4 weeks to explore a large variety of solutions. Combining VCC and ROSES, we were able to synthesize 10 solutions in one week.

The experimentation results show that our combined approach provides an important design time gain, from 6 months down to one month, simultaneously allowing the exploration of more solutions, by enabling faster generation and evaluation of architectural solutions and providing a complete path from design space exploration to clock-cycle accurate synthesis.

VII. Summary and Conclusions

This paper presented a new approch to obtain a full path to implementation for application-specific multiprocessor system-on-chip architectures. This approach is based on the combination of suites of tools that cover the two major aspects of the design of such system: the Cadence VCC environment supports an easy architecture design space exploration, while ROSES is able to synthesize a complete hardware/software solution, including all details of the communication between heterogeneous components as well as dedicated OS. This approach enabled us to circumvent the difficulty to obtain a complete design flow from system specification to RTL synthesis. The experiment with a VDSL application has validated this approach. Results show that it provides a significant design time reduction and allows a larger architecture design space exploration.

Acknowledgements

This work is sponsored by *Medea+ project INCA A106*. The authors would like to thank Christophe Del-Toso, Benoît Clément, Doha Benjelloun, Fabien Castanier and Silvia Brini from STMicroelectronics who helped in achieving this work.

References

[1] A.D. Pimentel, et al. “Exploring embedded-systems architectures with Artemis,” *IEEE Computer*, Vol. 34, pp. 57-63, 2001.
[2] Cadence, http://www.cadence.com/products/vcc.html
[3] “Cadence Virtual Component Co-design Modeling guide,”
[4] “Cadence Virtual Component Co-design Architecture Evaluation Guide,”
[5] “Cadence Virtual Component Co-design Link to Implementation,” *Cadence Design Systems, Inc*., March 2001.
[6] CoWare N2C, http://www.coware.com
[7] D.J.G. Mestdagh, M.R. Isaksson, P. Odling, “zipper VDSL: a solution for robust duplex communication over telephone lines,” *IEEE Communication magazine,* pp. 90-96, May 2000.
[8] D. Lyonnard, S. Yoo, A. Baghdadi, A.A. Jerraya, “Automatic generation of application specific architectures for heterogeneous multiprocessor system-on-chip,” *Proceedings DAC 2001*, June 2001, Las Vegas, USA.
[9] I. Bolsens, et al. “Hardware/software co-design of digital telecommunication systems,” *Proceedings of IEEE,* Vol. 85, No. 3, pp. 391-418, 1997.
[10] J-Y. Brunel et al. "COSY communication IP’s," *DAC 2000,* Los Angeles, California, USA.
[11] J. Gerlach, W. Rosenstiel, "System Level Design Using the SystemC Modeling Platform," In: R. Merker, W. Schwarz (Eds.), "System Design Automation -Fundamentals, Principles, Methods, Examples", VDI Verlag Düsseldorf, 2000.
[12] L. Gauthier, S. Yoo, A.A. Jerraya, “Automatic generation and targeting of application specific operating systems and embedded systems software,” *5th International Workshop on Software and Compilers for Embedded Systems (SCOPES 2001)*, St. Goar, Germany, March 2001.
[13] L. Gauthier, S. Yoo, A.A. Jerraya, “Application specific operating systems generation and targeting for embedded SoCs,” *the 10th Workshop on Synthesis and System Integration of MIxed Technologies(SASIMI’01),* October2001.
[14] M.D. Nava, G.S. Okvist, “The zipper prototype : a complete and flexible VDSL multi-carrier solution,” *ST Journal special issue xDSL,* September 2001.
[15] M. Fujita, H. Nakamura, “The standard SpecC language,” *ISSS ’01,* October 2001, Montréal, Québec, Canada.
[16] R. Ernst, J. Henkel, Th. Benner, “Hardware software cosynthesis for microcontrollers,” *IEEE Design and Test of computers,* December 1993.
[17] R.K. Gupta, “Co-synthesis of hardware and software for digital embedded systems,” *PhD Thesis, standford*, December 1993.
[18] S. Vercauteren, B. Lin, H. De Man, “Constructing application-specific heterogeneous embedded architectures from custom HW/SW applications,” *33rd DAC 1996,* June 1996, Las Vegas, USA.
[19] SystemC, http://www.systemc.org
[20] T. Kuhn, et al. “A framework for object oriented hardware specification, verification, and synthesis,” *DAC 2001,* June 2001, Las Vegas, USA.
[21] W. Cesàrio, et al. “Component-based design approach for multicore SoCs,” *Proceedings DAC 2002, June 2002*, New Orleans, USA.
[22] W. Cesàrio, et al. “HW/SW interfaces design of a VDSL modem using automatic refinement of a virtual architecture specification into a multiprocessor SoC: a case study,” *Proceedings DATE 2002*, March 2002, Paris, France.