Des Autom Embed Syst (2006) 10:181–202
DOI 10.1007/s10617-006-9585-8

A SystemC-only design methodology and the CINE-IP multimedia platform

Guido Araújo · Edna Barros · Elmar Melcher · Rodolfo Azevedo · Karina R. G. da Silva · Bruno Prado · Manoel E. Lima

Received: 5 February 2006 / Revised: 3 June 2006 / Accepted: 8 June 2006

Abstract The increasing complexity of modern System-on-Chip (SoC) platforms has revealed the need for methodologies that enable a rigorous engineering design process, based on a combination of Electronic System Level (ESL) description languages, and IP-core modeling and reuse. On the other hand, ESL modeling has faced designers with the same methodology problems encountered in the design of large computer programs. In this paper, we describe a SystemC-only IP-core design process, called IP PROCESS (IPP). IPP is inspired on two rigorous software engineering processes (RUP and XP), and on well-known hardware design standards (VSIA and RMM). The IPP Verification Methodology (IPV) is based on a careful refinement of the SystemC behavioral description towards RTL. Such approach enabled a continuous co-simulation against the behavioral reference model, while allowed for a SystemC-only environment. As a result, we have experienced a considerable reduction in design time and an improvement in early bug detection. The IPP process has been used by over 70 designers of the BrazilIP Network, a SystemC collaborative partnership, in the design of the Fenix system. An intermediate step in the Fenix design is a real-world multimedia platform called CINE-IP (demo available at http://www.brazilip.org.br/cine-ip), composed of MPEG4, MP3 decoders and an 8051 microcontroller. The application of the IPP methodology in the design of CINE-IP, and its impact in design productivity is thoroughly analyzed.

Keywords A based on SystemC design methodology · System design in SystemC · Engineering design process · Functional verification · IP-core modeling · FPGA prototyping · Multimedia platform

G. Araújo · R. Azevedo
Computer Systems Laboratory, Institute of Computing, University of Campinas, Cidade Universitaria Zeferino Vaz, P.O. Box 6176, CEP. 13084-971, Campinas-SP, Brazil

E. Barros (✉) · B. Prado · M. E. de Lima
Computer Science Institute, Federal University of Pernambuco, Cidade Universitaria, P.O. Box 7851, CEP. 50732-970, Recife-PE, Brazil
e-mail: ensb@cin.ufpe.br

E. Melcher · K. R. G. da Silva
Eletrical Engineering Department, Federal University of Campina Grande, Rua Aprígio Veloso, 882 - Bodocongó, CEP 58.109-900, Campina Grande-PB, Brazil

1. Introduction

The increase in the complexity of modern SoC platforms has created a new set of challenges to system architects. Modeling and simulating such systems is a complex task that can only be achieved by a thorough combination of Electronic System Level (ESL) description languages, and IP-core modeling and re-use [3]. New architectural-level design languages, like SystemC [1, 19] and System Verilog [25] have been introduced to address this problem. On the other hand, the increasing demand for ESL modeling has faced designers with the same methodology problems encountered in the design of large computer programs. Unfortunately, most of the designers lack the proper training on rigorous software-based methodology, since it is not part of the standard IC design curricula.

In this paper, we describe a SystemC-based IP-core design process, called IP PROCESS (IPP) [9]. IPP is inspired on a rigorous software design process and co-verification methodology. It has been used by over 70 designers of the BrazilIP Network,[1] to design a real-world platform called Fenix containing a number of IP-cores: MPEG4, MP3, USB, Bluetooth, Keyboard/LCD, 8051 and Ethernet controller. The goal in designing Fenix was twofold. First to provide the universities involved in the network with a modern platform that could be used for training and research. Second, to enable a test bed where a design methodology could be constructed and evaluated.

The current version of Fenix is an MPEG4/MP3/8051 multimedia platform; called CINE-IP, which was entirely designed using SystemC, from ESL down to RTL synthesis, and the IPP design process. CINE-IP was designed using OSCI SystemC and Synopsys Co-centric, and prototyped with Altera Stratix-II FPGA board and tools, resulting in a robust working multimedia platform (CINE-IP demo is available at http://www.brazilip.org.br/cine-ip).

This paper is divided as follows. Section 2 gives an overview of the IPP SystemC design process, and shows how it relates to other known software engineering design methodologies. Section 3 describes the IPP process in details, and gives an example of its application to an 8051 design. Section 4 provides a description of the IPP Verification Methodology (IPV), and an example on how it was used to verify the MP3 core. Section 5 describes the CINE-IP platform, while Section 6 concludes the paper, listing potential improvements and future work.

2. The IPP SystemC design process

As modern VLSI technology enables the design of complete systems in silicon (System-on-chip), new design methodologies have been increasingly based on pre-designed/verified *Intellectual Property* blocks (IP-cores). IP-core reuse has been an approach to reduce the increasing gap between design productivity and chip complexity of emerging SoC designs [3]. To keep these efforts in check, a design methodology that favors reuse and early error detection is essential. The need of both, reuse and early error detection, demand a design task that is rigorously defined, so that all phases can be clearly identified and appropriate checks enforced.

In order to cope with some challenges of IP-core design, we have developed a well-defined IP-core methodology called IP-Process (IPP). In short, the IPP methodology is a rigorous

[1] BrazilIP Network is a consortium formed by the 8 largest of the Brazilian universities (UNICAMP, USP, UFPE, UFCG, UFMG, UNB, UFRGS and PUCRS).

and thorough engineering process that guides designers through the design process, so that they can acquire a clear and unique understanding of the IP-core functionality and behavior. It has been inspired on the combination of well-known software engineering methodologies, like RUP [8] and XP [30]; with IC design standards like VSIA [29] and RMM [7].

IPP defines the IP-core design task as a set of activities, where each activity determines "what should be done, when and by whom", i.e., the process assigns activities and responsibilities to the right person at a right moment of the design life cycle.

In IPP, the life cycle design of an IP-core starts in the *Conception* phase by eliciting *requirements* and *constraints*. After that the IP-core structure, functionalities and behavior should be defined during the *Architecture* phase. In this phase, the structure, functionalities and behavior are modeled using UML and Real Time UML [26]. Only after that, HDL design takes place. Due to the increasing IP-core complexity it is very important that the design team acquires a clear and unique understanding of the IP-core functionality and behavior, before any design refinement start.

The following phase is the *RTL Design*. It aims at producing an RTL specification of the IP in some hardware description language. This phase starts with a behavioral SystemC/C++ description, which is refined down to structural SystemC RTL and synthesized. Refinement can be done manually or automatically, by using some synthesis tool. Initially we used manual SystemC refinement and the Synopsys Co-centric tool for synthesis. Due to recent problems with Co-centric availability and support, we have been successfully experimenting with behavioral synthesis using Forte Cynthesizer toolset. The results have been quite encouraging.

Concurrently with the RTL implementation, *functional verification* must take place in order to assure that the SystemC RTL description has the same behavior as the original (behavioral SystemC/C++) Reference Model (RM).

The last phase is the *Physical Prototyping* phase, which aims at producing a running FPGA prototype.

By defining all these phases as a set of well defined activities, with *actors* and *roles*, we improve the design productivity, due to: (a) the increase in the probability of earlier error detection since the design starts at a higher level of abstraction; and (b) the continuous co-verification checks enabled by the adoption of a refinement strategy based on a single design language (SystemC).

Moreover, the designer knowledge of the design improves, since IPP makes the design activities more predictable, clarifying the abilities required to execute each one of the phases. The design time is recorded at each phase, so the process can be refined and calibrated for future designs.

IPP is described in SPEM, an UML profile standard, which can be used as a CASE tool input, generating management support for all process activities [6].

2.1. Related work

In line with the design for reuse trend, the Virtual Socket Interface Alliance (VSIA) has been formed by a group of major Electronic Design Automation (EDA) and Semiconductor companies with two goals [29]. First, to establish a unifying vision for the chip industry, and second, to develop the technical standards required to enable the most critical component of that vision: the mix and match of IP-cores from multiple sources. VSIA later expanded that vision to meet the growing needs of the IC industry by including software and hardware IP for SoC design. The VSI Alliance intends to define, develop, ratify, test and promote open specification relating to data formats, with the goal to facilitate the reuse of intellectual property blocks from different sources in the design and development of system on chip.

Another relevant work in this area is the Reuse Methodology Manual (RMM). RMM outlines a set of best practices for creating reusable ASIC designs that can be used in a SoC design methodology. These practices are based on the authors' experience in developing reusable designs, as well as on the experience of design teams from many companies around the world [7]. RMM is a chronicle of the best practices used by the best teams to develop reusable IP-cores and, as the VSIA, addresses the concerns of two distinct audiences: the creators/providers of IP-cores and chip designers who use or integrate these IPs [7, 29].

The Rational Unified Process (RUP) is a software engineering process that provides a disciplined approach to assign tasks and responsibilities within a design organization. At the RUP context, a 'discipline' is a collection of activities that are related to a major 'area of concern'; all activities related with the management of the project are grouped into a discipline called 'Project Management', for example. The RUP goal is to ensure the production of high-quality software that meets the needs of its end users, within a predictable schedule and budget [8]. Furthermore, RUP has a set of 'Best Practices' that are important in the context of reuse and early error detection such as: management of requirements and changes, visual modeling (UML), design based on component architectures and continuous quality verification. Still from the Software Engineering domain, another important approach is commonly used: the eXtreme Programming (XP). XP is actually a deliberate and disciplined approach for software development. This methodology emphasizes teamwork: manager, customers, and developers are all part of a team dedicated to deliver quality software [30]. XP focus on the quality of the final code emphasizing test automation and the use of a test suite for regression and validation testing. The XP approach adopts a set of simple *Rules* and *Practices* related to *Planning*, *Designing*, *Coding* and *Testing* [30].

The first two works cited above, VSIA and RMM, do not intend to develop specifications related to the internal design of IP-cores, architectures of subsystem components nor fabrication processes [7, 29]. VSIA Alliance and RMM are more concerned with the IP-core deliverables, than on how an IP-core is designed.

IPP is based on UML, and its goal is to fill in this gap by defining a design process that describes, step-by-step, how an IP-core should be designed. Based on the RUP and XP processes, IPP proposes a disciplined and documented way to design quality IPs. The use of UML as a SoC specification mechanism has considerably grown recently, both in academia and industry [11, 12, 31].

3. IPP in details

As said in the previous section, IPP has been defined based on two software processes: RUP and XP. The main idea is to re-use and adapt the expertise in designing large software systems, accumulated over the years in the Software Engineering area, to a hardware workflow.

From the RUP processes, two disciplines related to high abstraction level have been adopted into IPP: *Requirements* and *Analysis & Design*. The Requirements discipline aims to eliciting and defining the core requirements, whereas the *Analysis & Design* discipline, includes activities for architecture definition, and behavioral modeling of the requirements. From the XP methodology *Rules* and *Practices* that enforce quality have been adopted.

The reader can observe that the IPP includes various concepts of Software Engineering, which supports to start the design at a very high abstraction level. This is particularly useful to an ESL-based methodology, given that it allows the detection of errors at earlier design phases, while assures the quality of the final implementation.

Table 1 SPEM definitions and Notation

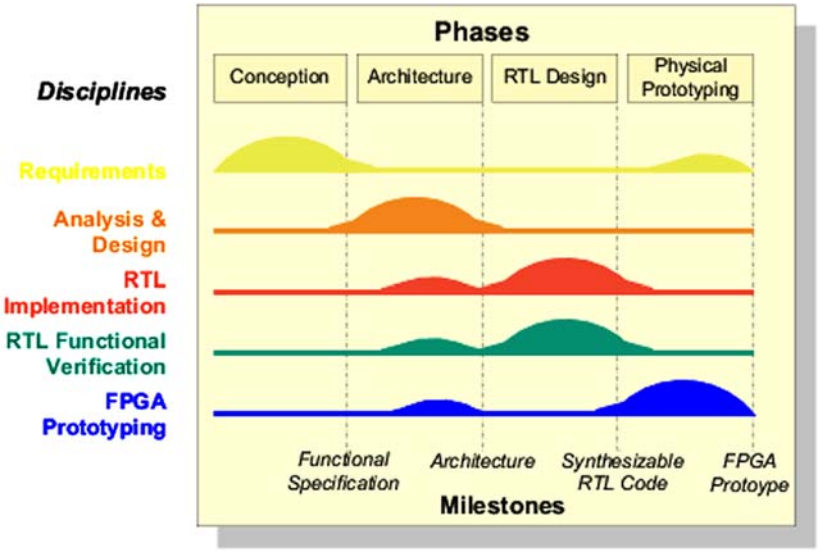
Concept	Description	Notation
Phase	A phase is the time between two major project milestones, during which a well-defined set of objectives is met, artifacts are completed, and decisions are made to move or not move in the next phase.	
Discipline	A discipline is a collection of activities, which are related to a major 'area of concern'.	
Workflow	A workflow is a group of activities, which are performed in close collaboration to accomplish some objective. The activities are typically performed either in parallel or iteratively, with the output from one activity being the input to another activity.	
Activity	An activity is the unit of work that a role may be asked to perform.	
Role	A role is the assignment of abilities and responsibilities to a person from the development team.	
Work-product or artifact	An artifact is anything produced, consumed or modified by a process. Examples include plans, coding, scripts, documents etc.	
Guidance	Guidance is used to provide more detailed information to practitioners about the associated activity. Examples: Guidelines, Techniques, Tool mentors, Checklists, Templates etc	
Document	Document is a type of work-product and represents a text document.	
UML model	This is a type of work-product and represents the UML diagrams.	

The IPP concepts, activities and results have been modeled with the SPEM Metamodel. SPEM, or Software Process Engineering Metamodel, is a UML Profile[2] for defining processes and its components [24]. In Table 1 the main constructs of SPEM are described. These constructs have been used for modeling the IPP process.

Graph 1 is an overview of the application of IPP. It has two dimensions: the horizontal axis represents time and shows the lifecycle aspects of the process, and the vertical axis represents disciplines, which group activities logically according to their nature.

The first dimension represents the dynamic aspect of the process, i.e. how activities are distributed over the time. It is expressed in terms of *Phases* and *Milestones*. The second dimension represents the structural aspects of the process: how it is described in terms of *disciplines*, *workflows*, *activities*, *artifacts* and *roles*. The graph shows how the emphasis on one activity can vary over time. All details about the concepts listed above as well as the IPP process architecture can be found at the IPP website [32].

[2] A UML Profile is a variant of UML that uses the extensions mechanisms of UML in a standardized way, for a particular purpose.



Graph 1 IPP structure overview

3.1. IPP phases overview

From a management perspective, IPP is decomposed over time into four sequential phases, each ended by a major milestone, as show in Graph 1. The first two phases, *Conception* and *Architecture*, are related with the understanding of the problem ("what the IP should be") and with the modeling of its behavior ("how it should do") using UML and Real Time UML as a modeling language. In these two phases, the design team should focus on understanding the requirements and defining the IP behavior before implementing it. The implementation should start at the RTL design phase. This delay to start implementation is intentional, and aims at creating a time slot in the design phase that allows an exhaustive discussion of all functionalities, constraints and possible architectures for the IP-core under design.

In the following, the main objectives and milestone for each design phase of the IPP lifecycle are described.

Conception Phase: the Conception is the first phase when designing an IP-core using IPP. Its goal is to collect the requirements and to define the scope of the project. The primary objectives include: (1) to understand the functional and non-functional requirements; (2) to define the functional specification; and (3) to achieve an agreement about the IP-core scope. The milestone of this phase is a well-defined set of functional specifications. At this point, the evaluation criteria are: (1) an agreement that the right set of requirements have been collected and that there is a share understanding of them; and (2) an agreement that important terms and expressions have been captured and documented.

Architecture Phase: the goal of this phase is to define the architecture of the core in order to provide a stable basis for the next phases. The architecture of the core must take into account the most significant requirements. The primary objectives of the Architecture Phase include: (1) to define the components of the architecture and its interfaces; (2) to demonstrate that the defined architecture will support the requirements of the IP-core;

Table 2 IPP disciplines

Discipline	"Areas of Concern" of an IP-core design
Requirements	Specification
Analysis & design	HW architecture
RTL implementation	Implementation and simulation
Functional verification	Tests (development and automation)
FPGA prototyping	Synthesis and physical prototyping

(3) to plan the components integration; and (4) to plan the RTL Functional Verification. The project milestone at the end of this phase is the IP-core architecture (components and interfaces). The evaluation criteria of this phase must assure that: (1) the defined architecture supports all requirements; and (2) the requirements and architecture are stable enough.

RTL Design Phase: the RTL Design Phase aims at developing an RTL simulation model based on the defined architecture, which can be synthesized. The primary objectives of this phase include: (1) to create the RTL simulation model for the core; (2) to construct the RTL Functional Verification components; and (3) to assure that all detected bugs have been corrected. At the end of the RTL Design Phase the milestone are the RTL simulation model (which should be synthesized) for and its testbenches. The evaluation criteria for this phase consist in answering the following questions: (1) are the IP-core functionalities stable enough? and (2) can all the components of the architecture be synthesized?

Physical Prototyping Phase: the focus of this phase is to create a physical prototype to ensure that the core can be distributed to its end users (system integrators). The project milestone at the end of this phase is an FPGA prototype (soft IP) of the IP-core. The primary evaluation criteria for the Physical Prototyping Phase consist in answering the following questions: (1) are all the IP-core modules synthesized and validated? (2) are the synthesis scripts created? and (3) are the synthesis constraints documented?

3.2. IPP disciplines overview

As mentioned before, a *Discipline* includes all activities you may go through to produce a particular set of artifacts. They are usually described in a detailed level, called 'workflow details'. A workflow detail shows how roles collaborate, and how they use and produce artifacts [8, 24].

In order to help its description, disciplines have been organized into a logical and sequential order of workflow details. Table 2 below shows the relationship between the IPP disciplines and the main areas of concern when designing an IP-core. In the following, each discipline is detailed described as *activities*, *roles* and produced *artifacts*.

Requirements Discipline: the purposes of this discipline are: (1) to establish and to maintain agreement with the costumer on what the IP-core should do; (2) to provide IP-core developers with a better understanding of the requirement; and (3) to define an external interface for the IP-core, focusing on the needs and goals of the end users. The left side of Fig. 1 shows the workflow sequence of the Requirement Discipline workflow, whereas the right side of Fig. 1 shows a detailed description of *Analyze the Problem* (activities, roles and artifacts).

Analysis & Design Discipline: the goals of the Analysis & Design workflows are: (1) to design the IP-core architecture, and (2) to assign the functionalities to the components

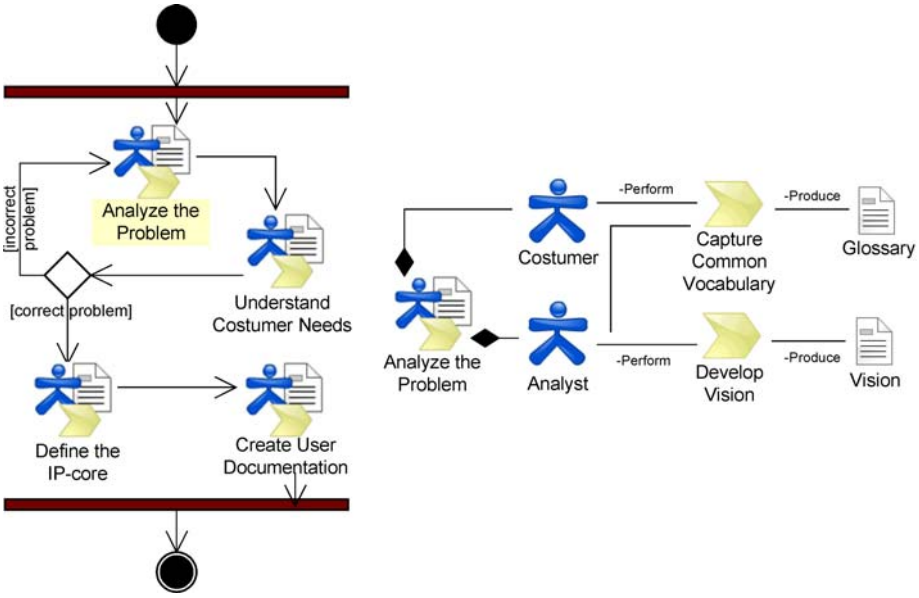


Fig. 1 Requirements discipline workflows

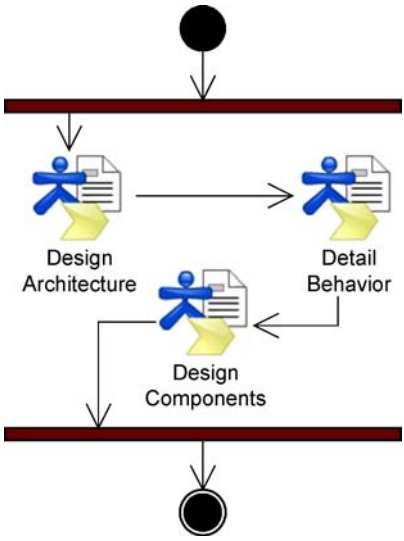


Fig. 2 Analysis and design workflow

of the architecture. Figure 2 shows the workflow sequence of the Analysis & Design workflow.

RTL Implementation Discipline: the goals of the RTL Implementation Discipline are (1) to implement each component defined in the architecture; and (2) to define how they should be integrated. Figure 3(b) shows the sequence of the RTL Implementation workflows. *Integration Planning* is executed during the Architecture phase, whereas the emphasis on *Implement Components* and *Integrate Components* is given during the RTL Design phase.

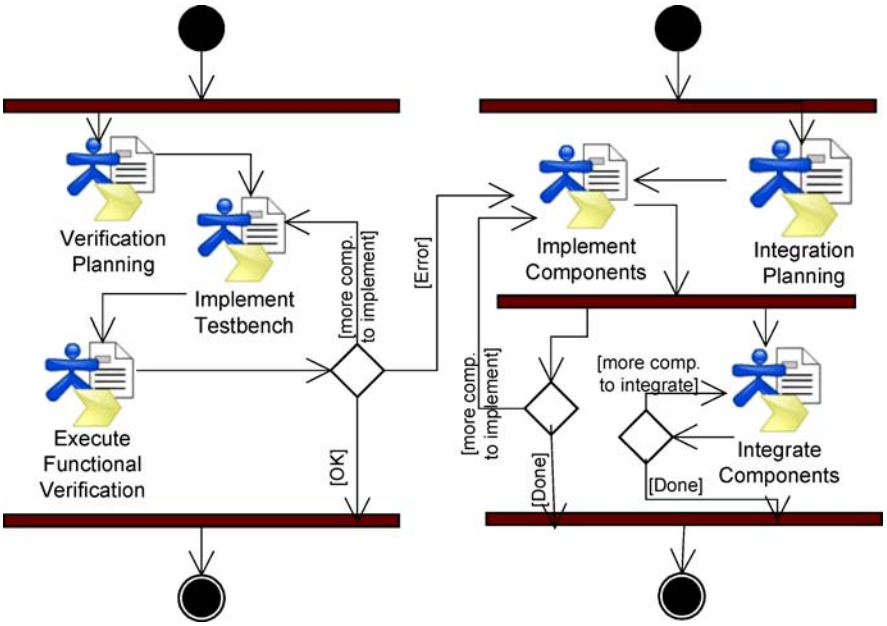


Fig. 3 RTL implementation and functional verification workflows

Functional Verification Discipline: the goals of this discipline are: (1) to validate that the requirements have been implemented appropriately; and (2) to find and to document defects in the RTL simulation model. Figure 3(a), shows the sequence of the Functional Verification workflow. Since verification is the central task in the IPP design process we detail this discipline in Section 4.

FPGA Prototyping Discipline:
this discipline aims: (1) to transform the synthesizable RTL simulation model into a physical prototype in FPGAs, and (2) to validate that the requirements have been implemented appropriately in the physical prototype. Figure 4 represents the sequence of the "FPGA Implementation" workflow. *Integration Planning* is focused during the Architecture phase, whereas the emphasis on *Implement Components in FPGA* and *Integrate Components in FPGA* is given during the Physical Design phase.

3.3. Using IPP to design an 8051 core

The complete IPP methodology includes 5 disciplines, 30 activities, 11 roles, 29 artifacts and 13 templates. The adoption of this methodology in the design of the CINE-IP cores has resulted in a large number of artifacts: source code, documents, scripts, VSIA deliverables, etc.

Unfortunately, due to the lack of space we cannot discuss here the application of all design disciplines of IPP to all CINE-IP cores. Thus, we do this only for the 8051 microcontroller.

The CINE-IP 8051 core has 255 instructions, 256 bytes of internal RAM and 64K bytes of external memory, Clock frequency of 33 MHz, two timers, full-duplex USART, interrupts with 2 priority levels and I/O parallel port [20].

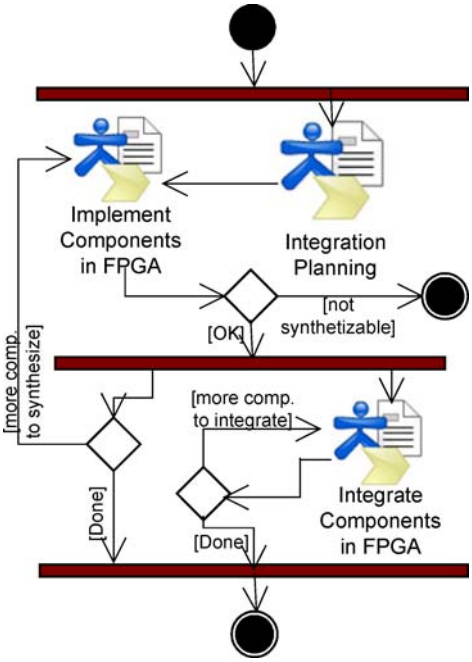


Fig. 4 FPGA implementation discipline workflow

The project has been developed in ten months (including training and design time), and the development team included seven people (senior undergraduate CS students, all of them without design experience, but with some C++ and FPGA design background). The 8051 functionalities and features have been implemented in six modules: CPU, serial interface, interrupt, OCP-IP interface [18], timers and ports. For training purpose, each team member has performed distinct roles during the project. For example, to the same person was assigned the role of analyst during the Conception phase, whereas in the Architecture phase this person has performed the Designer role and so on. As an Analyst this person has specified one module, while as an Implementer he has designed another one. The same happened for the Design role.

An important result was the improvement on design productivity. As mentioned previously the 8051 core was developed in 10 months, 70% of the time for training, conception and architecture. RTL implementation, functional verification and prototyping were done in 30% of the design time. Allocating a considerable amount of design time for conception and analysis has contributed to a reduced number of bugs during implementation. Only 13 bugs during functional verification and co-simulation were found. Prototyping was bug free. This last phase was done in two weeks.

This strategy has provided each team member with: (1) the opportunity to learn the various aspects of the design; (2) a broader understanding of the core being developed; and (3) the possibility to identify specification errors sooner, thus minimizing the communication and integration errors among the core modules. Table 3 lists all artifacts that have been produced at each phase of the 8051's design. The definition of the disciplines Requirements and Analysis & Design is an important feature of the proposed IPP strategy, since they provide a common understanding for all team members of the core that are going to be implemented.

The use of UML-RT for modeling purpose in these disciplines has made easier the specification of a first behavioral specification, which has been used as reference model. During

Table 3 Phases overview of the 8051 Microcontroller design

IPP phase	Artifacts
Conception	Requirements Specification and Functional Specification
Architecture	7 class diagrams (design classes)
RTL Design	Synthesizable RTL simulation model: 28,548 lines of SystemC RTL code Components for RTL functional verification: 12,896 lines of SystemC TL, testbench written in 933 lines of code that produced 817,623 test-vectors (including random), and regression scripts
Prototyping	FPGA Prototype in an Altera StratixII (EPS260F672C5ES), occupying 7% of the ALUTS and 10% of the internal memory. The prototyped 8051 core works at a 33,54 MHz frequency.

the requirements capture, UML-RT has been used for describing use cases. From the use cases model, a class diagram and sequence diagram have been specified in the Analysis & Design discipline. Also, in this discipline a description of the system architecture has been obtained by mapping the analysis classes into design classes. Each design class has been implemented as capsules, which implement sequential behavior. The result of the Analysis & Design discipline is an UML-RT description including capsules, ports and protocols representing the system architecture, which was used for implementing the reference model. In the UML-RT system specification, each capsule (or capsule hierarchy) represented a system module, for which the interface and sequential behavior have been completely specified. This feature has improved the design process, since it allowed the establishing of a relationship between functional requirements, use cases model, class diagram and the system architecture, making it easier errors detection and code maintenance. As an example, the reader can see the class diagram for the OCP interface shown in Fig. 5. The OCP subsystem resulted from the Analysis & Design discipline is composed of two capsules (master and slave), which communicate through a well-specified protocol. The behavior of the slave capsule has been specified as a FSM. From this description a SystemC specification of the OCP subsystem has been easily obtained, which has been used as Reference Model (RM).

4. The IPP verification strategy (IPV)

The most difficult challenge in the design of any system is to make sure that the final implementation is free of implementation flaws. The goal of functional verification is to verify all functionalities of the design and to assure that it behaves according to the specification. Verification can consume over 70% of the overall design effort [2], and thus, tools that can quickly create efficient testbenches are in great demand.

To enable an efficient verification strategy, one must adopt a rigorous design process and create an environment capable of verifying the design output against the output of a Reference Model (RM). Currently it is still common in industry to do functional verification without using a RM, in order to save development time and cost. Based on our design experience, these savings do not pay off, when considering the whole project, since the lack of a RM leads to longer RTL and prototype debugging time, increasing the maintenance costs along the project and product lifetimes and their overall risk. Even when no RM is available at the beginning of the project, it is better to spend the time to develop one than to continue without it. Reference Model is essential to have a verification approach which is self-checking, takes pseudo random data input, and is guided by functional coverage

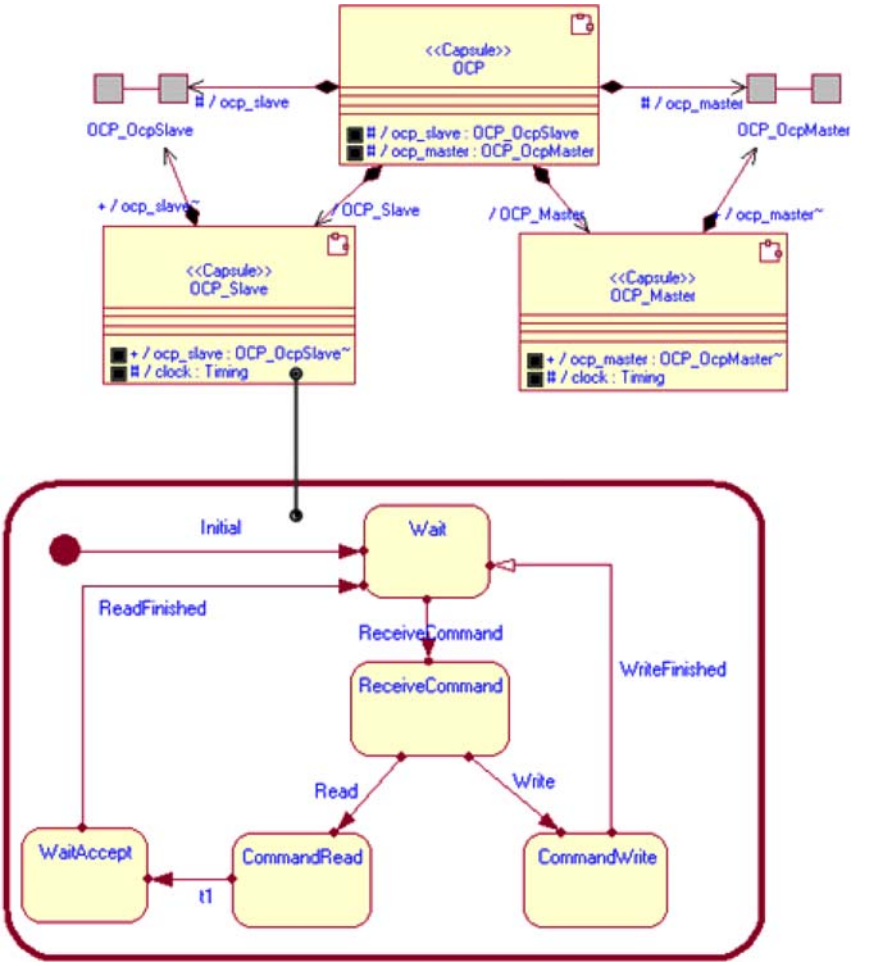


Fig. 5 UML-RT class diagram for the 8051 OCP/IP bus interface

IPP Verification Methodology (IPV) is an implementation of the IPP design process. As shown in Fig. 6, IPV uses SystemC descriptions at two different abstraction levels: (a) behavioral with communication among processes at transaction level, and (b) register transfer (RTL). The transaction level specification has been used for functional verification purpose, while the RTL description focuses on synthesis. In our approach, the SystemC RTL code has been designed by a careful refinement of the SystemC behavioral description towards RTL. Such approach enabled a continuous co-simulation against the behavioral reference model, while allowed for a SystemC-only environment. As a result, we have experienced a considerable reduction in design time and an improvement in early bug detection.

IPV generates an object-oriented environment (Fig. 7), that allows an accurate simulation of the behavioral Reference Model (RM) against the RTL model, which is called Design-Under-Verification (DUV). Simulation covers real, corner-cases, compliance and random test-vectors.

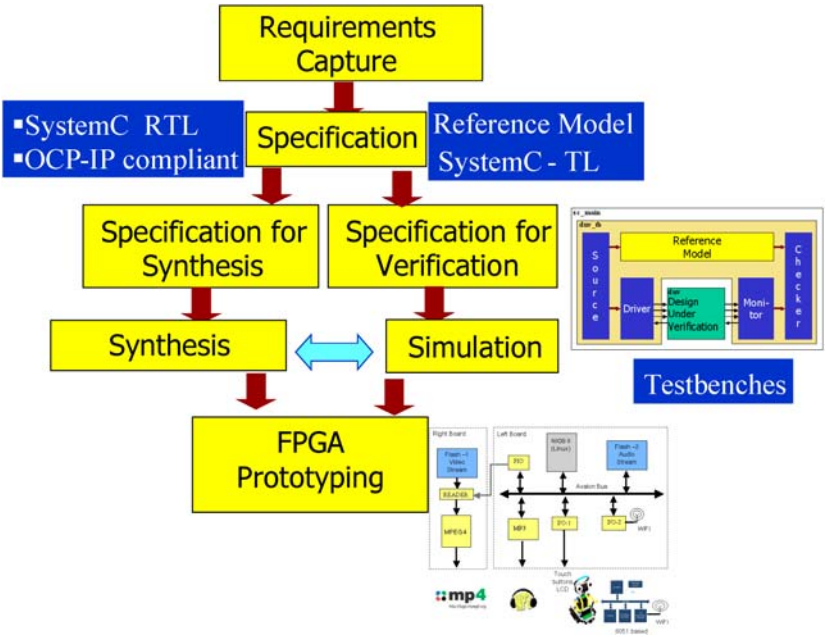
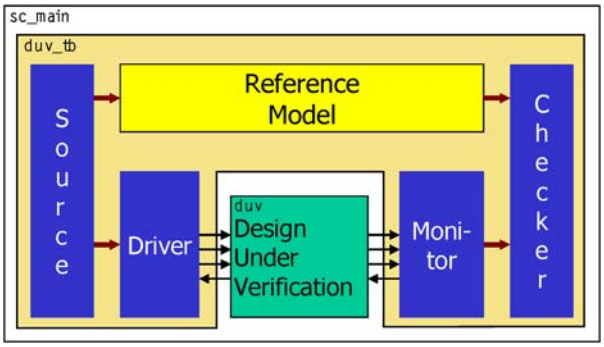


Fig. 6 The IPV functional verification methodology

Fig. 7 IPV testbench setup



After that, each IP-core was synthesized using professional Synopsys Co-centric and Altera Quartus II tools, and prototyped using Altera NIOS-DEVKIT-2s60 Stratix II boards. The synthesis resulting Verilog netlist, is back annotated with timing information from the Altera library, and co-simulated in the testbench, to assure correctness under timing information.

4.1. Related work

In the past some well established hardware description languages, like VHDL and Verilog, have been used to do design verification. Although they have useful hardware programming constructs, these languages lack major functional verification capabilities like constrained randomization, functional coverage and transaction recording. On the other hand, programming languages like C, C++, and Java, allow high-level abstraction constructs, but do not have the mechanisms to account for parallelism and timing, which are required for hardware

description. In order to close this gap, some specialized verification languages have bee created like Verisity, OpenVera, SystemVerilog and SystemC Verification Library (SCV). One of the central features of our methodology was the use of a single design language, and thus we have adopted SystemC.

Due to the increasing complexity of SoC designs, a lot of effort has been concentrated on the research of the verification problem. Historically, several methodologies have been used for functional verification [10, 27], but they lack generality and ease of use. Some methodologies are commercial and cannot be freely used [14, 28].

IPV creates a SystemC based object-oriented environment to perform verification. Other reuse methodologies are based on object-orientation [4]. In [5] the authors describe an approach that uses constraint solving to generate input vectors through a finite state machine. The machine produces all possible inputs to a specific Device Under Verification (DUV). In [22] the authors propose a methodology and a tool to do transaction-based functional coverage. Most of these tools cover some aspect of verification or are specific to some kind of DUV.

IPV creates a template testbench tailored to the DUV under design. It intends to be generic enough to cover all types of synchronous DUV. The IPV approach is implemented in VeriSC, an open-source tool that performs automatic testbench generation. VeriSC uses OSCI SystemC and the SystemC Verification Library (SCV) to create a random-constraint, coverage-driven, self-checking and transaction-based testbench template.

4.2. The VeriSC tool

An important problem in design verification is the need to adapt the testbench to the DUV. In IPV the verification engineer uses VeriSC tool to make such adaptation. VeriSC also allows that the Reference Model (RM) be written in virtually any high-level language, making it simple and easier to maintain. As shown in Fig. 8, input data is fed into the DUV and the RM, and the outputs of both are collected for equivalence checking.

VeriSC offers enhanced productivity to verification engineers by reducing the design time spent in creating testbenches. The resulting testbench templates are compact and easy to understand. By using these templates, the verification engineer can specify signal handshake, functional coverage metrics and input value distributions. For the sake of simplicity, a simple Adder is used in the example of Fig. 8 to illustrate the SystemC testbench modules produced by VeriSC.

VeriSC tool automatically creates a SystemC template testbench according to the particular characteristics of the DUV. It generates all testbench modules: *Source*, *Driver*, *Monitor*, *Reference Model*, *Checker* and all FIFOs that connect these modules. The tool is also responsible for connecting the specific DUV.

In order to implement all these templates, VeriSC tool receives two input files:

- *Template pattern:* which is an ASCII file that contains the SystemC code, common to any template instance of a testbench. This input is part of the tool and not provided by the user.
- *Testbench specification:* a file containing information about which variables will be used as signals in which interface(s) from the DUV and which variables will be used as transactions by the testbench in which interface(s) from the DUV. This specification has a defined format just to be used in VeriSC tool. This specification file is shown in Fig. 9 and is provided by the user.

Based on these two input files, VeriSC tool generates the templates to the testbench. These testbench components are templates with all information about structures, FIFOs, signals and

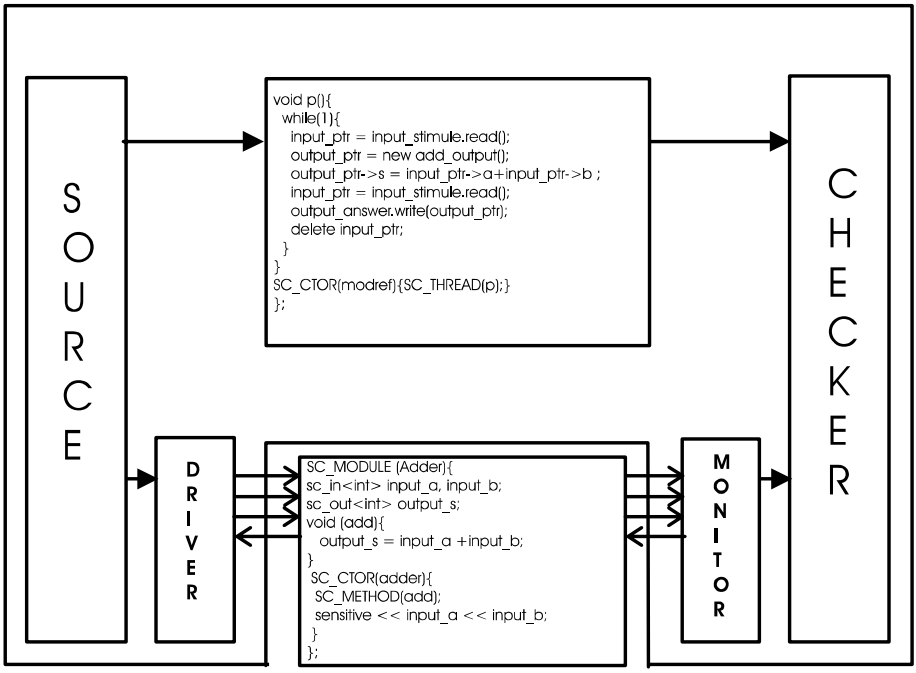


Fig. 8 A simple Adder synthesized testbench

Fig. 9 Specification to input VERISC tool

```
//interface in input
  struct add_input
  {
    int a;
    int b;
  };
 //interface out output
 struct add_output
 {
   int s;
 };
```

specific code to make easier the testbench codification. These components are created with all specific modules information as connection between input/output gates and instances of FIFOs to drive transaction data.

After the generation of these templates, the verification engineer needs to fill in manually implementation specific code into the generated templates to finish the testbench implementation with the information that could not be extracted from the template patterns or from the specification structure.

Fig. 10 The generated Source module

```
class input_constraint_class: public scv_constraint_base {
 scv_bag<int> a_distrib;
 scv_bag<int> b_distrib;
 public:
  scv_smart_ptr<add_input> add_input_sptr;
  SCV_CONSTRAINT_CTOR(input_constraint_class) {
     a_distrib.push(0, 50);
     a_distrib.push(1, 50);
     b_distrib.push(0, 50);
     b_distrib.push(1, 50);
     add_input_sptr->a.set_mode(a_distrib);
     add_input_sptr->b.set_mode(b_distrib);
  }
};
```

Fig. 11 Test coverage setup

```
output_checker_cv.begin();
   BVE_COVER_COND(output_checker_cv, 1, 10);
   BVE_COVER_COND(output_checker_cv, 0, 10);
   BVE_COVER_COND(output_checker_cv, 2, 10);
output_checker_cv.end();
```

Figure 9 shows a specification of our example (an adder), which has just one input interface *in*, with the variables *a* and *b* and one output interface *out* with one output variable b. Without loss of functionality, this file could be used to specify any generic DUV.

4.2.1. Template generation

In this phase, based on both files explained in the previous subsection, the Drivers and Monitors are generated straight from the transaction description file. The tool generates one Driver for each input interface and one Monitor for each output interface.

In Fig. 10, part of the Source module is created as an SCV class that inputs transaction level data into the DUV. VeriSC creates an input class to each input interface that communicates with the DUV through FIFOs.

The Driver is responsible for transforming transaction-level data to handshake signals and passes them to the DUV. The implementation of the specific handshaking protocol for the DUV has to be done by the verification engineer using behavioral SystemC. The Driver records each transaction for visualization.

The Monitor is responsible for receiving the DUV's signals and transforming them into transaction level data. For the structure shown in Fig. 8, with a single output interface, one Monitor was generated. The Monitor sends the data into a FIFO and passes them to the Checker.

The Checker's is to control the functional coverage, indicating which functionalities of the DUV have been tested during a simulation run. By specifying adequate values for the desired coverage the verification engineer sets when verification is finished. The Checker uses a set of bve_cover[3] calls (Fig. 11) to specify what functional characteristics should be verified, i.e. which cover criteria must be reached at the end of the verification task. In the example from

[3] BVE stands for BrazilIP SystemC Extension and is a set of functional verification C++ classes adopted in IPV.

Fig. 11 we show a cover criterion that stops the simulation after the sum reaches 10 times the number 1, 10 times the number 0 and 10 times the number 2. The BVE classes contain optional progress bars that allow the verification engineer to monitor verification progress during the simulation run.

The Checker is also responsible for the self-checking capability, by comparing the results coming from RM and Monitor. This comparison is performed at the transaction-level; The RM receives data from the Source through FIFO(s), and sends data to the Checker also through FIFO(s). All data in the RM are transaction-level.

Any compiled object code that can be linked into C++ can be used as reference model. Input transaction data is used as arguments to subroutine or method calls and the output transactions receive their data from the results. Depending on the operating systems used to run the simulator, pipes can also be used to run the reference model in another process.

4.2.2. Implementation details

We have implemented VeriSC using the SystemC library and SCV. SystemC is based on the C++ programming language and thus it considerably simplifies the creation of a high-level environment. On the top of C++, SystemC adds important concepts such as concurrence, events and hardware data types to enable efficient designs. The SCV library improves SystemC capability by providing APIs for transaction based verification, constrained and weighted randomization, exception handling and other verification features. Furthermore, SCV permits transaction-level programming, a methodology that enables a high-level abstraction, reutilization and simulation speedup.

4.3. Using IPV to verify the MP3 core

The IPV verification discipline of the IPP methodology has been applied to all cores of the CINE-IP system. Table 4 shows the number of lines of SystemC code for the RM and DUV and the total testbench size. It also lists the number of test-vectors applied to each core, and the corresponding application time. The same procedure has been used to verify all cores of the CINE-IP platform, and is currently being used to verify the remaining cores in the Fenix platform.

Again, due to the lack of space, we will only describe our experience of using IPV to verify just one of the CINE-IP cores. Our test case DUV is an MP3 decoder, which was verified using the testbench setup of Fig. 7. For the sake of clarity we have chosen the MP3 WINDOW module to detail (Fig. 12).

The WINDOW module is responsible for generating audio data (PCM) from sub-band samples. It has only two interfaces, one input and one output interface. The testbench source uses floating point to provide random input data to the module. Input data was generated as sets of data pairs with a precision of up to the 9th decimal digit. Random input has been

Table 4 Testbench numbers

IP core	RM Model code lines	DUV RTL code lines	Testbench code lines	Testbench test-vectors	Verification time (sec)
8051	12,896	28,548	933	81,762,300	539.16
MP3	5,572	12,235	892	30,854,230	346.90
MPEG4	12,718	51,382	21,473	50,789,096	9,385,200

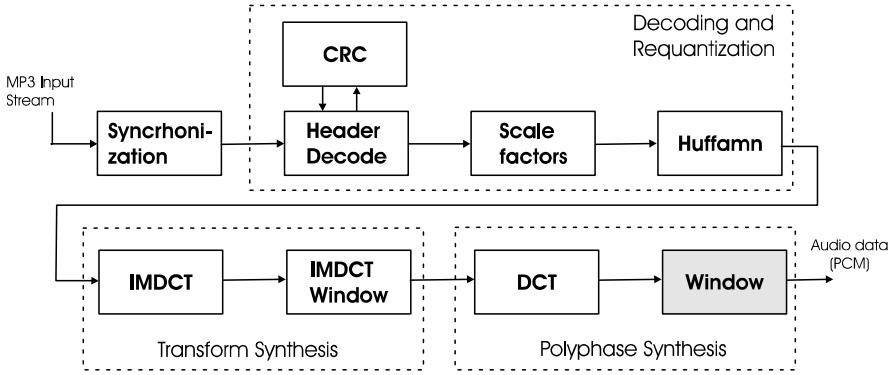


Fig. 12 The MP3 WINDOW module architecture

created using the SCV library's class SCV_CONSTRAINT, with SCV_BAG. A handshake protocol for interfacing Driver-DUV and DUV-Monitor has also been implemented.

The comparison between the RM and DUV models' output, was performed at the Checker Module, by using the Root Mean Square (RMS) formula $\sqrt{\frac{1}{n}\sum_{k=0}^{n-1}(x_k - y_k)}$, where n is the samples number, x is the reference sample and y is the resulting samples from the DUV output. According to the ISO standard, RMS values must not be higher than $\frac{2^{-15}}{\sqrt{12}}$. Moreover, $|x_k - y_k|$ it must not be higher than 2^{-14}.

IPV strategy found three major design flaws that were not encountered during standard simulation. The relevant errors are below:

- In the Finite State Machine (FSM) the reset state was not re-initializing an array in the source code (the *nt*[2][512] vector) to zero during system start-up.
- The first 15 output blocks caused errors when the reset signal is raised.
- The module decoded correctly only stereophonic data, but was not capable of decoding monophonic data correctly.

Notice that subtle design errors, like those above, could hardly be captured if only simulation or a standard verification procedure were used. By using IPV the four designers of the MP3 could considerably speedup the verification of the MP3 design, reducing by 75% the expected design time (from the original 12 months). All errors have been corrected. Verification was repeated and finished without detecting any additional errors.

5. The CINE-IP multimedia platform

One of the central goals of the BrazilIP Network is to design a modern reference platform that can be used as a test-bed for the IPP design methodology. This platform is called Fenix and, as said before, it will contain the following cores: MPEG4, MP3, USB, Bluetooth, Keyboard/LCD, 8051 and an Ethernet controller. The Fenix design roadmap has a number of intermediate steps. One of the relevant steps is the CINE-IP platform.

CINE-IP (Fig. 13) is a system for demonstrating the quality and reliability of the IP-cores developed by the Brazil-IP Network. It can be considered as a memory game for movies, and works as follows. A player chooses, through touch-button panel, clips of audio and video, extracted from trailers of renowned movies. The player's choice is considered correct if the

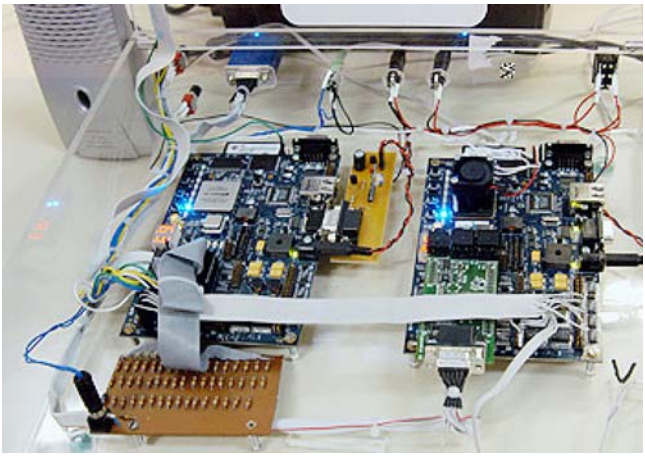


Fig. 13 The final CINE-IP design

selected audio and video clips match, i.e. if they are from the same movie. In such case, the player makes another choice until a mistake occurs, when he/she passes the turn to the next player. CINE-IP is composed of an MP3 decoder and an MPEG4 decoder for audio and video decoding respectively. In the CINE-IP video demo (http://www.brazilip.org.br/cine-ip), a robot works as a movie critic, "dancing" only if the player makes the right choice. The robot is controlled by an 8051 micro-controller, also developed by the Brazil-IP Network using the IPP methodology. A picture of the CINE-IP final design is shown in Fig. 13.

As shown in details in Fig. 13, the CINE-IP architecture uses two Altera NIOS II-DEVKIT-2S60 boards, each one containing an EPS260F672C5ES device. Due to the size of the involved cores (mainly internal memory demand) we had to divide the design into two boards. The board on the right controls the system and stores the MP3 decoder, while the board on the left contains the MPEG4 core.

The board on the right has a NIOS-II processor running the main system control program over Linux operating system. This program uses the I/O-1 device to read two touch-buttons that are used to select the appropriate video and audio clips, and to display the selections through an LCD device. The program sends commands, through a parallel port (PIO), to the READER module on the other board, so it can read the selected MPEG4 stream from the board compact flash card (FLASH-1).

At the same time it fetches the MP3 audio stream, through the Avalon bus, into the MP3 bus interface. After start playing the selected audio and video, the program checks if the choices match. If so, it sends a command to the I/O-2 module that is processed by a WiFi card, and passed to a program running on the 8051 core installed in the robot. If the match is correct, the robot "dances", signaling a correct pick, otherwise, the robot remains stopped. The robot control was also implemented in an Altera NIOS II-DEVKIT-2s60 board.

In Fig. 14, all cores (with the exception of the NIOS-II, the Avalon bus and flash cards) have been designed using the SystemC-based IPP design methodology, and its associated IPV verification methodology. We list below the main features of the NIOS-II, MPEG4, MP3 and 8051 cores involved in this design. All cores have been implemented in Altera EPS260F672C5ES devices.

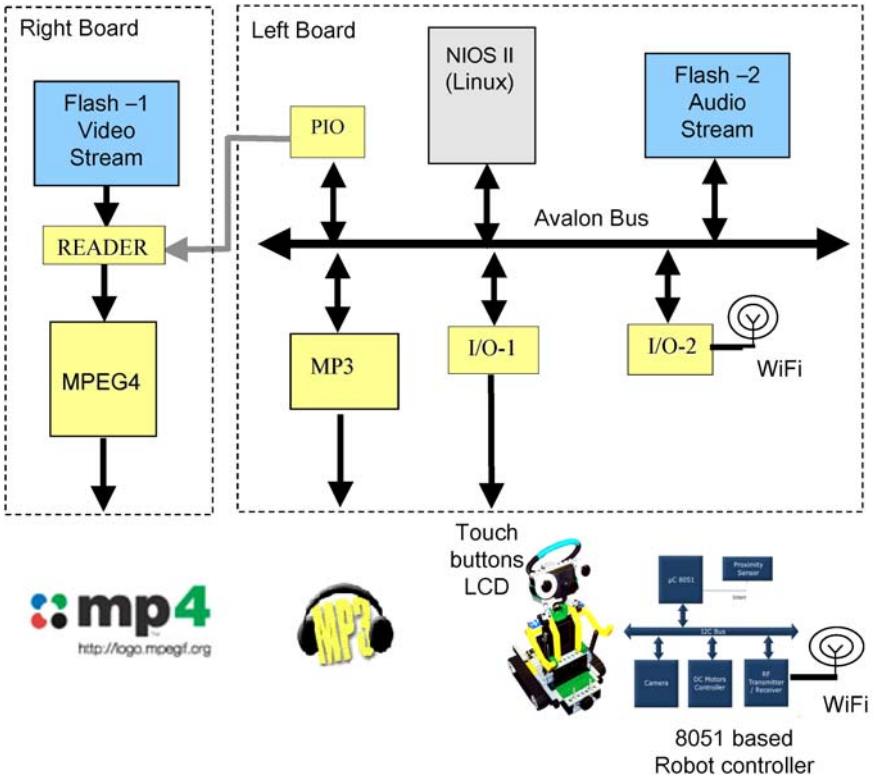


Fig. 14 The CINE-IP architecture

- *NIOS-II processor* is a basic 32-bits RISC processor with 2KB I-Cache and 4KB D-Cache cache memories and a 50 MHz clock. This processor runs the main system application under Linux, and is responsible to manage the touch-buttons, the LCD interface and to read the MP3 audio-stream. It uses 2791 logic cells (5.8% of the available FPGA area).
- *MPEG4 core* is video-decoder, compatible with the "Simple Profile/Level 1" Annex G and N of the ISO-IEC-14496-2 standard [15]. This core requires a 50 MHz clock, providing a 30 Frames/sec throughput. It uses 21 KLEs (Logical Elements), corresponding to 35% of the available FPGA area (including input/output circuits). Most of the 124 KB of internal memory is used by the core to store video frames into the frame-buffer
- *MP3 core* is an audio-decoder, compatible with the MPEG-1 and MPEG-2 layer 3 standard [16]. It uses a 50 MHZ clock, 2,605 logic cells (6.5% of the FPGA area), and 58,368 RAM bits allowing it to decompress 8,000,000 samples/s.
- *8051 micro controller* is a standard micro-controller running at 33 MHz, which has been incorporated into a system that also includes a core to control a DC motor, and another for the RF sub-system. The complete system has been prototyped in FPGA, and is used to control a Lego Mindstorm robot. It requires 2973 ALUTs of the FPGA, corresponding to 7% of its area.

6. Conclusions and future work

A methodology for IP-core development, called IPP, has been proposed. IPP has been structured as an extension of two well-known software engineering processes (RUP and XP) and an IP design standard (VSIA). It is organized in terms of phases and disciplines, where each discipline is related with an 'area of concern' and has been defined in terms of workflows (activities, roles and artifacts) that were constructed from the understanding of what an IP-core design should provide until its physical prototyping.

The IPP methodology has been used to design all IP-cores of the Fenix platform, an on-going project of the Brazil-IP Network. Up to now three cores have reached the final stage and prototyped in FPGA (MP3, MPEG4 and 8051). They have been put together into a multimedia platform, called CINE-IP

We have also shown how IPP has been used in designing an 8051 soft IP-core. It revealed that pre-defined workflows, document templates, pre-validated coding standards and practical tool tutorials are very useful in reducing the learning curve. Furthermore, the organization of related activities into disciplines has provided designers with a thorough understanding of the distinct aspects of the design process. This broad view of the design process, combined with variations in the designer role, has improved their knowledge of the whole IP architecture, improving the team productivity. The adoption of SystemC as the single design language has enabled a smooth design refinement approach, considerably improving team communication and respect to the Reference Model specification.

We have also detailed the IPV Verification Discipline of IPP. We show an approach, based on SystemC and SCV, which allows co-verification between the Reference Model and the RTL design, by means of automatic testbench construction. We describe how IPV was used to capture subtle design flaws in the MP3 design.

Finally we detailed the CINE-IP system design: a real-world multimedia platform that has been entirely designed using SystemC-only tools and methodology.

As future work, the IPP process is being extended to include disciplines for silicon testing and prototyping, targeting the development of a future ASIC flow. Modifying some disciplines to make the IPP process interactive and incremental is also under development.

References

1. Bhasker, J. *A SystemC Primer*, Star Galaxy Publishing, 2002.
2. Bergeron, J. *Writing Testbenches: Functional Verification of HDL Models*. 2nd edn., Kluwer Academic Publishers, 2003.
3. Chang, H. et al. *Surviving the SoC Revolution: A Guide to Platform-based Design*. Kluwer Academic Publishers, Massachusetts, 1999.
4. Drucker, L. Verification Library Speeds Transaction-Based Verification, D&R Industry Articles, EETimes, Feb. 2003.
5. Ferrandi, F., M. Rendini, and D. Sciuto. Functional Verification for SystemC Descriptions using Constraint Solving, *Design, Automation and Test in Europe (DATE'02)*, 704, Paris, March 2002.
6. IRIS Case tool: Iris suite. http://www.osellus.com/, 2005.
7. Keating, M. and P. Bricaud. *Reuse Methodology Manual for System-on-chip Design*. Kluwer Academic Publishers, 2002.
8. Krutchen, P. *The Rational Unified Process*. Addison-Wesley, 1998.
9. Lima, M., F. Santos, J. Bione, T. Lins, and E. Barros. ipPROCESS: A Development Process for Soft IP-core with Prototyping in FPGA , Forum on Design Languages (FDL 2005), Swiss, Sept. 2005
10. Randjic, A., N. Ostapcuk , I.Soldo, P. Markovic, and V. Mujkovic. Complex ASICs Verification With SystemC, 23rd International Conference on Microelectronics, pp. 671–674, 2002.

11. Riccobene, E., P. Scandurra1, A. Rosti, and S. Bocchio. A SoC Design Methodology Involving a UML 2.0 Profile for SystemC, DATE, 2005.
12. Schattkowsky, T. UML 2.0—Overview and Perspectives in SoC Design, DATE, 2005.
13. McGrath, D. EE Times: Design News Unified Modeling Language gaining traction for SoC design, EETimes On Line, April 2005.
14. ModelSim, ModelSim Tutorial. Available at: http://www.model.com/support/documentation/PE/pdf /pe_tutor.pdf, 2005.
15. MPEG4 Document Standard, ISO-IEC, 14496, 2000.
16. MP3 standard, ISO/IEC 11172–3, 1993.
17. Nguyen, K.D., Z. Sun, P.S. Thiagarajan, and W. Wong. Model-driven SoC via Executable UML to SystemC, 25th IEEE Real-Time Systems Symposium (RTSS), 2004.
18. OCP-IP: Open Core Protocol International Partnership, 2006. Available at: http://www.ocpip.org.
19. The Open SystemC Initiative, 2006. See http://www.systemc.org.
20. 80C51 8-bit Microcontroller Family Datasheet, Philips Semiconductor, Jan. 2002.
21. Rashinkar, P., P. Paterson, L. Singh. *System-on-a-chip Verification: Methodology & Techniques*. Kluwer Academic Publishers, Feb. 2001.
22. Regimbal at all, Automating Functional Coverage Analysis Based On An Executable Specification. In *Proc. of the International Workshop on System-on-Chip for Real-Time Applications*, Calgary, June 2003.
23. da Silva, K.R.G., E.U.K. Melcher, V.A. Pimenta, and G. Araujo. An Automatic Testbench Generation Tool for a SystemC Functional Verification Methodology, SBCCI, pp. 66–70, 2004.
24. Software Process Engineering Metamodel—SPEM, version 1.0, Object Management Group. Available at: http://www.omg.org, 2006.
25. SystemVerilog Language Reference Manual Version 3.1. Available at: http://www.systemverilog.org, 2006.
26. U2 Partners, OMG RFPs ad/00-90-01 and ad/00-09-02, Unified Modeling Language 2.0, Version 0.671, 2002.
27. Wagner, I., V. Bertacco, and T. Austin. StressTest: An Automatic Approach to Test Generation Via Activity Monitors, Design Automation Conference, 2005.
28. Verisity, A promising approach to overcome the verification gap of modern SoC designs, 2006. Available at: http://www.verisity.com/resources/whitepaper/soc_nec.html.
29. VSIA: Virtual Socket Interface Alliance, 1997. Available at: http://www.vsia.org.
30. eXtreme Programming, 2005. Available at: http://www.extremeprogramming.org, 2006.
31. Damasevicius, R. and V. Stuikys. Application of UML for hardware design based on design process model, 2004 conference on Asia South Pacific design automation, ASAP 2004, pp. 244–249, 2004
32. IP Process website, 2005: Available at: http://www.brazilip.org.br/ipprocess, 2005