# ADL-Based Methodology for Extensible ASIP Cores: a Case Study

E.K. d'Ávila, V.A. Carlos[*] , L.C.V. dos Santos[†]
Computer Science Department
Federal University of Santa Catarina
Florianópolis, SC, Brasil
{eduardok,vcarlos,santos}@inf.ufsc.br

## ABSTRACT

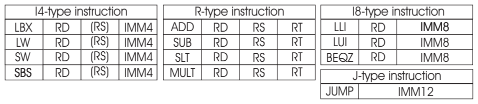
This paper applies an ADL-based methodology to the design of an ASIP and the generation of its software toolkit (instruction-set simulator, assembler, cross-compiler). For the sake of case study, a simple digital audio processing application was chosen. The methodology starts from an untimed functional description of the instruction set architecture, which is then refined into a cycle-accurate functional SystemC model. This model is then gradually refined towards an HDL RTL model, the starting point for ASIP synthesis on a FPGA platform.

## 1. INTRODUCTION

The growing complexity of embedded systems asks for higher levels of abstraction, design reuse and scalable verification. The design of a system starting at the RT level as provided by most HDLs is unable to cope with the demand of platforms containing one or more CPUs, several busses, IP blocks, memories and I/O devices. Although most of the EDA community agrees upon SystemC [8] as the future standard language for system-level design [5], the gap between a transaction-level (TL) model written in SystemC and a RTL model written in VHDL is enormous. Although SystemC allows the refinement of models from TL to RTL, some mechanisms have been introduced to simplify and speed up the creation and maintenance of functional models for CPUs with the introduction of architecture description languages (ADLs) [1] [2] [4] [6] [7]. The role of an ADL is especially important when a general-purpose CPU is not adequate to meet real-time constraints or power restrictions of a given application and an ASIP has to be used instead. ADLs are crucial to ASIP usability, since there is no pre-existent software development toolkit (e.g. cross compiler, instruction-set simulator, assembler, linker, etc.) able to comply with the tailored instruction set. This makes ADLs a common starting point for model refinement, CPU synthesis and automatic toolkit generation.

This work presents a case study whose ultimate goal is the automatic generation of ASIPs (comprising their toolkits). To pave the way towards full automation, two key elements are addressed. First, a design methodology is defined and used during the design of an ASIP for a simple digital signal processing application. Second, the main refinement steps are traversed: some of them are actually automated; others, although provisionally performed manually, contribute to identify the key issues for further automation.

The application consists of a set of audio effects (delay, distortion, flanger, phaser, tremolo), as explained in [11].



| I4-type instruction | | | |
|---|---|---|---|
| LBX | RD | (RS) | IMM4 |
| LW | RD | (RS) | IMM4 |
| SW | RD | (RS) | IMM4 |
| SBS | RD | (RS) | IMM4 |

| R-type instruction | | | |
|---|---|---|---|
| ADD | RD | RS | RT |
| SUB | RD | RS | RT |
| SLT | RD | RS | RT |
| MULT | RD | RS | RT |

| I8-type instruction | | |
|---|---|---|
| LLI | RD | IMM8 |
| LUI | RD | IMM8 |
| BEQZ | RD | IMM8 |

| J-type instruction | |
|---|---|
| JUMP | IMM12 |

**Figure 1: Instruction-Set Architecture**

## 2. RELATED WORK

Several ADLs have been reported in the literature. The early ADLs were developed for compiler retargetability (e.g. ISDL [4]). Later, the evolution of system-level design has given rise to ADLs designed for the automatic generation of both efficient compilers and cycle-accurate CPU models. Some ADLs, like EXPRESSION [2] and ArchC [1], reach this goal by providing separate views of the instruction-set: a semantic view (for compiler generation) and a behavioral view (for simulator generation). Other ADLs, like nML [6] and PEAS-III [7] aim at the same goal by combining both views into a more restrictive grammar. Essentially, the work in this domain focuses on improvements to overcome such grammar restrictions and redundant views [3], the extension of generic processor cores [9] and the characterization of embedded applications to allow such instruction-set extension [10].

In this work, we adopt the ADL ArchC [1], an open-source architecture description language which has the advantage of generating functional models in SystemC, thereby allowing direct model integration with other component models within a platform described in SystemC. Our main goal is to envisage how the ASIP hardware can be synthesized from an ADL description, along with its software toolkit.

## 3. THE CORE ARCHITECTURE

Since our goal is the automatic generation of a software toolkit, it seems better to avoid the well-known problems of creating efficient compilers for classical DSP architectures. That's why our core architecture is based on a load-store RISC-like machine, as summarized in Figure 1.

To minimize the amount of memory, we have chosen an

[*]supported by CNPq IC grant.
[†]Advisor.

8-bit representation for audio sample storage. However, to alleviate overflow and rounding effects, extended arithmetic precision (16 bits) is used within the core. By providing 16 general-purpose registers, good register allocation is expected.

## 4. DESIGN FLOW METHODOLOGY

Figure 2 summarizes our design methodology.

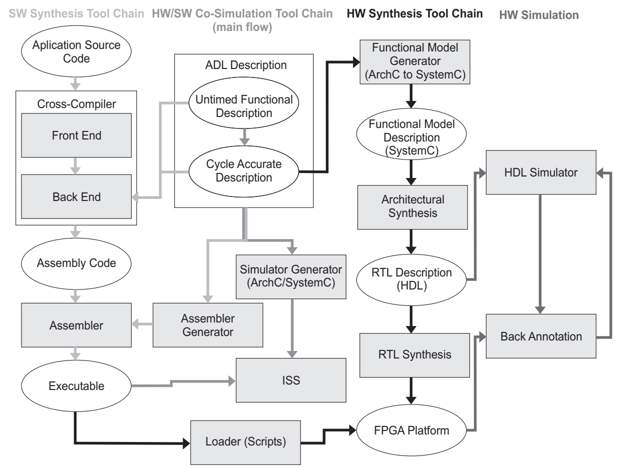


**Figure 2: The design flow methodology**

The ADL description is firstly provided as an untimed functional (UF) model, then refined to cycle-accurate (CA) model. A plain instruction-set simulator (ISS) or a cycle-accurate simulator is automatically generated from the proper ADL description and fed with the executable code generated by the SW synthesis flow. This automation steps are already implemented within the ArchC toolbox.

The SW synthesis flow embodies the tools responsible to allow easy and efficient high-level ASIP programming. Since the core architecture is extendible both the compiler and the assembler have to be automatically modified to comply with the new instructions. That's why the compiler's back-end must extract information automatically from the ADL description either for instruction selection (semantics from UF or CA models) or for code scheduling (latencies from the CA model). For the same reason, an assembler generator is needed. In our implementation, the GNU GCC compiler is adopted as a front-end and its back-end is modified to comply with core extensibility. An assembler was implemented for the architecture core.

We envisage a contemporary HW synthesis flow with using an ADL as starting point. From the CA ADL description, a CA model written in SystemC is generated. Architectural synthesis from SystemC is performed resulting in a RTL description from which classical RTL synthesis is performed targeting a FPGA platform. Then a Loader is used to combine netlist and code storage information for FPGA prototyping. In our implementation, VHDL is used for the RTL description and Mentor Graphics tools for FPGA mapping. The Architectural Synthesis step is performed manually in this case study. Since first-generation architectural-synthesis tools assume an HDL behavioral description as starting point, most commercial tools can not be used directly to automate this step from a SystemC description. Besides, the need for a second-generation architectural synthesis has been advocated by the EDA community [5], where HDLs are used for RTL synthesis and not for behavioral architectural synthesis. When performing "manual" architectural synthesis, we are making a first attempt at evaluating the challenges and needs of such a second-generation of tools. The HW synthesis flow traversal is in progress.

## 5. EXPERIMENTAL SET UP

Initially, audio effect algorithms were written in Java. Several audio (wave) input files were used to check if a given effect was audible by an average listener. Then, each effect algorithm was coded in assembly language and the binary code was fed, along with the same audio input files, to the ISS. The resulting audio output files were compared to the one generated at the algorithmic level. Finally, this procedure was repeated, replacing the plain ISS by the CAS. As all generated output files matched, both the core architecture and the core assembler were properly validated.

## 6. CONCLUSION AND PERSPECTIVES

This paper shows the current status of research work in progress. The design methodology follows the EDA trends. The proposed core architecture was properly validated and minimum software infrastructure was provided. Besides, an assembler generator and a cross-compiler will be available in the near future. In the long run, the main contribution of this case study is to give clues on how some manually performed system-level design steps should be automated, thereby paving the way for further research activities.

## 7. ACKNOWLEDGMENTS

We would like to acknowledge the ArchC support of the Computer Systems Laboratory at the Institute of Computing of the University of Campinas.

## 8. REFERENCES

[1] ArchC. http://www.archc.org.
[2] A. H. et al. Expression: A language for architecture exploration through compiler/simulator retargetability. In *DATE Conference*, Mar 1999.
[3] G. B. et al. A novel approach for flexible and consistent adl-driven asip design. In *DAC*, Jun 2004.
[4] G. J. et al. An instruction-set description language for retargetability. In *DAC*, Jun 1997.
[5] G. M. et al. System level design: Six success stories in search of an industry. In *DAC Panel, session 22.1*, Jun 2004.
[6] M. H. et al. Generation of software tools from processor descriptions for hardware/software codesign. In *DAC*, Jun 1997.
[7] M. I. et al. Peas-iii: An asip design environment. In *ICCD*, Sep 2000.
[8] SystemC. http://www.systemc.org.
[9] Tensilica. http://www.tensilica.com.
[10] P. Yu and T. Mitra. Characterizing embedded applications for instruction-set extensible processors. In *DAC*, Jun 2004.
[11] U. Zlzer. *DAFX: digital audio effects.* West Sussex: Wiley, 2003.