A Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using the Machine Description Language LISA

Andreas Hoffmann, Oliver Schliebusch, Achim Nohl, Gunnar Braun,
Oliver Wahlen and Heinrich Meyr
Integrated Signal Processing Systems, RWTH Aachen, Germany

Abstract

The development of application specific instruction set processors (ASIP) is currently the exclusive domain of the semiconductor houses and core vendors. This is due to the fact that building such an architecture is a difficult task that requires expertise knowledge in different domains: application software development tools, processor hardware implementation, and system integration and verification. This paper presents a retargetable framework for ASIP design which is based on machine descriptions in the LISA language. From that, software development tools can be automatically generated including HLL C-compiler, assembler, linker, simulator and debugger frontend. Moreover, synthesizable HDL code can be derived which can then be processed by standard synthesis tools. Implementation results for a low-power ASIP for DVB-T acquisition and tracking algorithms designed with the presented methodology will be given.

1 Introduction

In consumer electronics and telecommunications high product volumes are increasingly going along with short lifetimes. Driven by the advances in semiconductor technology combined with the need for new applications like digital TV and wireless broadband communications, the amount of system functionality realized on a single chip is growing enormously. Higher integration and thus increasing miniaturization have led to a shift from using distributed hardware components towards heterogeneous system-on-chip (SOC) designs. Due to the complexity introduced by such SOC designs and time-to-market constraints, the designer's productivity has become the vital factor for successful products. For this reason a growing amount of system functions and signal processing algorithms is implemented in software rather than in hardware by employing embedded processor cores.

In the current technical environment, embedded processors (EP) and the necessary development tools are designed manually, with very little automation. This results in a long, labor-intensive process requiring highly skilled engineers with specialized know-how – a very scarce resource. Most of today‘s processor design is conducted by EP and IC vendors using a variety of development tools from different sources, typically lacking a well-integrated and unified approach. Engineers design the architecture, simulate it in software, design software for the target application, and test the implementation for hardware and software integration. Each step of this process requires its own design tools and is often conducted by a separate team of developers. As a result, design engineers rarely have the tools or the time to explore architecture alternatives to find the best-in-class solution for their target applications. This situation is very expensive, both in time and engineering resources, and has a substantial impact on time-to-market. Without automation and unified development environment, the design process is prone to error and may lead to inconsistencies between the hardware and software representations.

The efforts of designing a new architecture can be reduced significantly by using a retargetable approach based on a machine description. The Language for Instruction Set Architectures (LISA) [1] was developed for the automatic generation of consistent software development tools and synthesizable HDL code. A LISA processor description covers the instruction-set, the behavioral and the timing model of the underlying hardware. Changes in the hardware specification are easily transferred to the LISA model and are automatically applied to the generated tools and hardware implementation. Moreover, speed and functionality of the generated tools allow usage after the product development has been finished. Therefore there is no need to rewrite the tools to upgrade them to production quality standard. In its predicate to represent an unambiguous abstraction of the real hardware, a LISA model description bridges the gap between hardware and software design, since it provides the software developer with all required information and enables the hardware designer to synthesize the architecture from the same specification the software tools are based on.

2 LISA language

The LISA language [1] is aiming at the formalized description of programmable architectures, their peripherals and interfaces. It was developed to close the gap between purely structural oriented languages (VHDL, Verilog) and instruction set languages for architecture exploration and implementation purposes of a wide range of modern programmable architectures (DSPs and microcontrollers). The language syntax provides a high flexibility to describe the instruction set of various processors, such as SIMD, MIMD and VLIW-type architectures. Moreover, processors with complex pipelines can be easily modeled. This includes the ability to describe architectures with complex execution schemes as e.g. out-of-order execution of instructions[1].

[1]A complete reference of the language is provided here: [2]

The process of generating software development tools and synthesizing the architecture requires information on architectural properties and the instruction set definition as depicted in figure 2.1. These requirements can be grouped

	memory model	resource model	behavioral model	instruction set model	timing model	micro-architecture model
HLL-compiler	register allocation	instruction scheduling	instruction selection	-	instruction scheduling	-
assembler	-	-	-	instruction translation	-	-
linker	memory allocation	-	-	-	-	-
simulator	simulation of storage	-	operation simulation	decoder/ disassembler	operation scheduling	-
debugger	display configuration	profiling	-	-	-	-
HDL generator	basic structure	write conflict resolution	-	instruction decoder	operation scheduling	operation grouping

Fig. 2.1: Model requirements of development tools.

into different architectural models - the entirety of these models constitutes the abstract model of the target architecture. The LISA machine description provides information consisting of the following model components.

- The *memory model* lists the registers and memories of the system with their respective bit widths, ranges, and aliasing. The compiler gets information on available registers and memory spaces. The memory configuration is provided to perform object code linking. During simulation, the entirety of storage elements represents the state of the processor which can be displayed in the debugger. The HDL code generator derives the basic architecture structure.
- The *resource model* describes the available hardware resources and the resource requirements of operations. Resources reflect properties of hardware structures which can be accessed exclusively by one operation at a time. The instruction scheduling of the compiler depends on this information. The HDL code generator uses this information for resource conflict resolution.
- The *instruction set model* identifies valid combinations of hardware operations and admissible operands. It is expressed by the assembly syntax, instruction word coding, and the specification of legal operands and addressing modes for each instruction. Compilers and assemblers can identify instructions based on this model. The same information is used at the reverse process of decoding and disassembling.
- The *behavioral model* abstracts the activities of hardware structures to operations changing the state of the processor for simulation purposes. The abstraction level of this model can range widely between the hardware implementation level and the level of high-level language (HLL) statements.
- The *timing model* specifies the activation sequence of hardware operations and units. The instruction latency information lets the compiler find an appropriate schedule and provides timing relations between operations for simulation and implementation.
- The *micro-architecture model* allows grouping of hardware operations to functional units and contains the exact micro-architecture implementation of structural components such as adders, multipliers, etc. This enables the HDL generator to generate the appropriate HDL code from a more abstract specification.

Besides, one of the key aspects in architecture development is the ability to abstract on multiple levels of accuracy. However, it is mandatory that a working set of software development tools can be successfully generated independently of the abstraction level. The LISA language allows the realization of models from ranging data-flow to micro-architecture level in the architecture domain and from the level of boundaries of high-level language (HLL) statements to clock cycles or even phases within cycles in the timing domain. This enables stepwise refinement of the model from functional specification to the micro-architecture implementation.

3 Related Work

Hardware description languages (HDLs) like VHDL or Verilog are widely used to model and simulate processors, but mainly with the goal of developing hardware. Using these models for architecture exploration and production quality software development tool generation has a number of disadvantages especially for cycle-based or instruction-level processor simulation. They cover a huge amount of hardware implementation details which are not needed for performance evaluation, cycle-based simulation and software verification. Moreover, the description of detailed hardware structures has a significant impact on simulation speed [3]. There are many publications on machine description languages providing instruction-set models. The language nML was developed at TU Berlin [4] and adopted in several projects, e.g. [5]. However, the underlying instruction sequencer does not allow to describe the mechanisms of pipelining as required for cycle-based models. Processors with more complex execution schemes and instruction-level parallelism like the Texas Instruments C6x cannot be described, even at the instruction-set level, because of the numerous combinations of instructions. The same restriction applies to ISDL [6] which is very similar to nML. The language ISDL is an enhanced version of the nML formalism and allows the generation of a complete tool-suite consisting of HLL compiler, assembler, linker and simulator. Even the possibility of generating synthesizable HDL code is reported, but no results on the efficiency of the generated tools nor on the generated HDL code are given. The EXPRESSION language [7] allows the cycle-accurate processor description based on a mixed behavioral/structural approach. However, no results are published on simulation speed and the ability to synthesize the architecture.

The PEAS-III system [8] is an ASIP development environment based on a micro-operation description of instructions that allows the generation of a complete tool-suite consisting of HLL compiler, assembler, linker and simulator including HDL code. However, no further information about the formalism is given that parameterizes the tool generators nor are results published on the efficiency of the generated tools. The MetaCore system [9] is a benchmark driven ASIP development system based on a formal representation language. The system accepts a set of benchmark programs and estimates the hardware cost and performance for the configuration under test. Following that, software development tools and synthesizable HDL code are automatically

generated. As the formal specification of the ISA is similar to the ISPS formalism [10], complex pipeline operations as flushes and stalls can hardly be modeled. Moreover, flexibility in designing the instruction-set is limited to a predefined set of instructions. Tensilica Inc. customizes a RISC processor within the Xtensa system [11]. As the system is based on an architecture template comprising quite a number of base instructions, it is far too powerful and thus not suitable for highly application specific processors which do in many cases only employ very few instructions.
Our interest in a complete retargetable tool-suite for architecture exploration, production quality software development, architecture implementation and system integration based on cycle-accurate models for a wide range of embedded processor architectures motivated the introduction of the LISA language which is used in our approach.

4 LISA processor design platform

The design and implementation of an embedded processor, such as a DSP embedded in a cellular phone, requires the following tasks or phases:

- architecture exploration,
- architecture implementation,
- application software design,
- system integration and verification.

The LISA processor design platform (LPDP) is an environment that allows the automatic generation of software development tools for architecture exploration, hardware implementation, software development tools for application design, and hardware-software co-simulation interfaces from one sole specification of the target architecture in the LISA language. The set of LISA tools and their areas of application are as follows:
Hardware Designer Platform – *for exploration and processor generation.* Architecture design requires the designer to work in two fields (see figure 4.1): on the one hand, the development of the software part including C-compiler, assembler, linker and simulator and on the other hand the development of the target architecture itself.

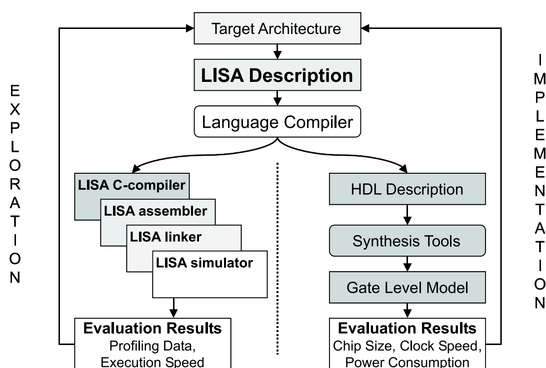

Fig. 4.1: Exploration and implementation.

The software simulator produces profiling data and thus may answer questions concerning the instruction set, the performance of an algorithm and the required size of memory and registers. The required silicon area or power consumption can only be determined in conjunction with a synthesizable HDL model. To accommodate these requirements, the LISA hardware designer platform can generate the following tools:

Language debugger for debugging the instruction-set with a graphical debugger frontend.

Exploration assembler which translates text-based instructions into object code for the respective programmable architecture.

Exploration linker which is controlled by a dedicated linker command file.

Instruction-set architecture (ISA) simulator for cycle accurate simulation including support for deep instruction and data pipelines.

Besides the ability to generate a complete set of software development tools, synthesizable HDL code for the processor‘s control path and instruction decoder can be generated automatically from LISA processor descriptions. This also comprises the pipeline and pipeline controller including complex interlocking mechanisms, forwarding, etc.
Software Designer Platform – *for software application design.* To cope with the requirements of functionality and speed in the software design phase, the tools generated for this purpose are an enhanced version of the tools generated during architecture exploration phase. The generated simulation tools are enhanced in speed by applying the compiled simulation principle [12] – where applicable – and are faster by one to two orders in magnitude than the tools currently provided by architecture vendors.
As the compiled simulation principle requires the content of the program memory not to be changed during the simulation run, this holds true for most DSPs. However, for architectures running the program from external memory or working with operating systems which load/unload applications to/from internal program memory, this simulation technique is not suitable. For this purpose, an interpretive simulator is provided.
System Integrator Platform - *for system integration and verification.* Once the processor software simulator is available, it must be integrated and verified in the context of the whole system which can include a mixture of different EPs, memories, and interconnect components. In order to support the system integration and verification, the LPDP system integrator platform provides a well defined application programming interface (API) to interconnect the instruction-set simulator generated from the LISA specification with other simulators. The API allows to control the simulator by stepping, running, and setting breakpoints in the application code and by providing access to the processor resources.

5 Architecture Implementation

As we are targeting the development of application specific instruction set processors (ASIP), which are highly optimized for one specific application domain, the HDL code generated from a LISA processor description has to fulfill tight constraints to be an acceptable replacement for handwritten HDL code by experienced designers. Especially power consumption, chip area and execution speed are critical points for this class of architectures. For this reason, the

LPDP platform does not claim to be able to efficiently synthesize the complete HDL code of the target architecture. Especially the data path of an architecture is highly critical and must in most cases be optimized manually. Frequently, full-custom design technique must be used to meet power consumption and clock speed constraints. For this reason, the generated HDL code is limited to the following parts of the architecture:

- coarse processor structure such as register set, pipeline, pipeline registers and test-interface;
- instruction decoder setting data and control signals which are carried through the pipeline and activate the respective functional units executed in context of the decoded instruction;
- pipeline controller handling different pipeline interlocks, pipeline register flushes and supporting mechanisms such as data forwarding.

Additionally, hardware operations as they are described in the LISA model can be grouped to functional units. Those functional units are generated as *wrappers*, i.e the ports of the functional units as well as the interconnects to the pipeline registers and other functional units are automatically generated while the content needs to be filled manually with code. Emerging driver conflicts in context with the interconnects are automatically resolved by the insertion of multiplexers.
The disadvantage of rewriting the data path in the HDL description by hand is that the behavior of hardware operations within those functional units has to be described and maintained twice – on the one hand in the LISA model and on the other hand in the HDL model of the target architecture. Consequently, a major problem here is verification and will be addressed in future research.

5.1 LISA language elements for HDL synthesis

LISA descriptions are composed of *resources* and *operations*. The declared resources represent the storage objects of the hardware architecture which capture the state of the system. Operations are the basic objects in LISA. They represent the designer's view of the behavior, the timing, and the instruction set of the programmable architecture.
The following sections will show in detail, how different parts of the LISA model contribute to the generated HDL model of the target architecture.

5.1.1 The resource section

The resource section provides general information about the structure of the architecture (e.g. registers, memories, pipelines). Based on this information, the coarse structure of the architecture can be automatically generated. Example 5.1 shows an excerpt resource declaration of the LISA model of the ICORE architecture [13] which was used in our case study.
The ICORE architecture has two different register sets – one for general purpose use named *R*, consisting of eight separate registers with 32 bits width and one for the address registers named *AR*, consisting of four elements each with eleven bits. The round brackets indicate the maximum number of simultaneous accesses allowed for the respective register bank – six for the general purpose register *R* and one for the address register set. From that, the respective number of access ports to the register banks can be automatically generated. With this information – bit-true widths, ranges and access ports – the register banks can be easily synthesized. Moreover, a data and program memory resource are declared – both 32 bits wide and with just one allowed access per cycle. Since various memory types are known and are generally very technology dependant, however, cannot be further specified in the LISA model, wrappers are generated with the appropriate number of access ports. Before synthesis, the wrappers need to be filled manually with code for the respective technology. The resources labelled as PORT are accessible from outside the model and can be attached to a testbench – in the ICORE the *RESET* and the *MEM_ADDR_BUS*.

```
RESOURCE
{
  REGISTER S32      R([0..7])6; /* GP Registers */
  REGISTER bit[11]  AR([0..3]); /* Address Registers */

  DATA_MEMORY S32 RAM([0..255]);   /* Memory Space */
  PROGRAM_MEMORY U32 ROM([0..255]);/* Instruction ROM */

  PORT bit[1]       RESET;         /* Reset pin */
  PORT bit[32]      MEM_ADDR_BUS;  /* External address bus */

  PIPELINE ppu_pipe = { FI; ID; EX; WB };
  PIPELINE_REGISTER IN ppu_pipe {
      bit[6] Opcode;
      ...
  };
}
```

Ex. 5.1: Resource declaration of a LISA model

Besides the processor resources such as memories, ports and registers, also pipelines and pipeline registers are declared. The ICORE architecture contains a four stage instruction pipeline consisting of the stages *FI* (instruction fetch), *ID* (instruction decode), *EX* (instruction execution) and *WB* (write-back to registers). In between those pipeline stages, pipeline registers are located which forward information about the instruction such as instruction opcode, operand registers, etc. The declared pipeline registers are multiple instanced between each stage and are completely generated from the LISA model. For the pipeline and the stages, entities are created which are in a subsequent phase of the HDL generator run filled with code for functional units, instruction decoder, pipeline controller, etc.

5.1.2 Grouping operations to functional units

As the LISA language describes the target architecture's behavior and timing on the granularity of hardware operations, however, the synthesis requires the grouping of hardware operations to functional units that can then be filled with hand-optimized HDL code for the data path, a well known construct from the VHDL language was adopted for this purpose: the ENTITY. Using the ENTITY to group hardware operations to a functional unit is not only an essential information for the HDL code generator but also for retargeting the HLL C-compiler which requires information about the availability of hardware resources to schedule instructions.

As indicated in section 5.1.1, the HDL code derived from the LISA resource section already comprises a pipeline entity including further entities for each pipeline stage and the respective pipeline registers. The entities defined in the LISA model are now part of the respective pipeline stages as shown in figure 5.1. Here, a *Branch* entity is placed into the entity of the *Decode* stage. Moreover, the EX stage contains an *ALU* and a *Shifter* entity. As it is possible in LISA to assign hardware operations to pipeline stages, this information is sufficient to locate the functional units within the pipeline they are assigned to.

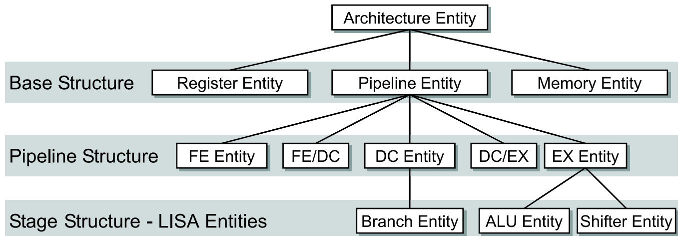

Fig. 5.1: Entity hierarchy in generated HDL model.

As already pointed out, the entities of the functional units are *wrappers* which need to be filled with HDL code by hand. The reason for that is that the functional units mostly represent the critical parts of the architecture both in terms of power consumption as well as in terms of maximum execution speed. Even hand-optimized semi-custom blocks are often not sufficient to fulfill the posed requirements so that frequently full-custom blocks come into operation. Nevertheless, in section 5.2.1 will be shown that by far the largest part of the target architecture can be automatically generated from a LISA model.

5.1.3 Generation of the instruction decoder

The generated HDL decoder is derived from information in the LISA model on the coding of instructions. Depending on the structuring of the LISA architecture description, decoder processes are generated in several pipeline stages. The specified signal paths within the target architecture can be divided into data signals and control signals. The control signals are a straight forward derivation of the operation activation tree which is part of the LISA timing model. The data signals are explicitly modeled by the designer by writing values into pipeline registers and implicitly fixed by the declaration of used resources in the behavior sections of LISA operations.

5.2 Implementation results

The ICORE which was used in our case study is a low-power application specific instruction set processor (ASIP) for DVB-T acquisition and tracking algorithms. It has been developed in cooperation with Infineon Technologies. The primary tasks of this architecture are the FFT-window-position, sampling-clock synchronization for interpolation/decimation and carrier frequency offset estimation. In a previous project this architecture was completely designed by hand using semi-custom design. Thereby, a large amount of effort was spent in optimizing the architecture towards extremely low power consumption while keeping up the clock frequency at 120 MHz. At that time, a LISA model was already realized for architecture exploration purposes and for verifying the model against the handwritten HDL implementation.

Except for the data path within functional units, the HDL code of the architecture has been automatically generated completely. Figure 5.2 shows the composition of the model.

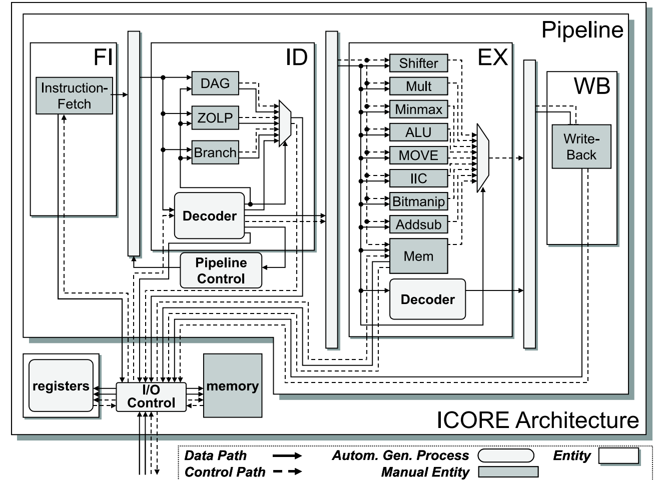

Fig. 5.2: The complete generated HDL model.

The dark boxes have been filled manually with HDL code, whereas the white boxes and interconnects have been completely generated.

5.2.1 Comparison of development time

The LISA model of the ICORE as well as the original handwritten HDL model of the ICORE architecture have been developed by one designer. The initial manual realization of the HDL model (without the time needed for architecture exploration) took approx. three months. As already indicated, a LISA model was built in this first realization of the ICORE for architecture exploration and verification purposes. It took the designer approx. one month to learn the LISA language and to create a cycle accurate LISA model. After completion of the HDL generator, it took another two days to refine the LISA model to RTL accuracy. The hand written functional units (data path), that were added manually to the generated HDL model, could be completed in less than a week. This comparison clearly indicates, that the time expensive work in realizing the HDL model was to create structure, controller and decoder of the architecture. In addition, a major decrease of total architecture design time can be seen, as the LISA model results from the design exploration phase.

5.2.2 Gate level synthesis

To verify the feasibility of automatically generating HDL code from LISA architecture descriptions in terms of power-consumption, clock speed and chip area, a gate level synthesis was carried out. To get meaningful data, the model has not been changed (i.e. manually optimized) to enhance the results.

Timing and size comparison The results of the gate-level synthesis affecting timing and area optimization were compared to the hand-written ICORE model, which comprised the same architectural features. Moreover, the same synthesis scripts were used for both models.

It shall be emphasized that the performance values are nearly the same for both models. Moreover, it is interesting that the same critical paths were found in both, the hand-written and the generated model. The critical paths occur exclusively in the data path, which confirms the presumption that the data path is the most critical part of the architecture and should thus not be generated automatically from an abstract processor model.

Critical path The synthesis has been performed with a clock of 8ns, this equals a frequency of 125MHz. The critical path, starting from the pipeline register to the shifter unit and multiplexer to the next pipeline register, violates this timing constraints by 0.36ns. This matches the hand-written ICORE model, which has been improved from this point of state manually at gate-level.

Area The synthesized area has been a minor criteria, due to the fact that the constrains for the hand-written ICORE model are not area sensitive. The total area of the generated ICORE model is 59009 gates. The combinational area takes 57% of the total area. The hand written ICORE model takes a total area of 58473 gates.

Power consumption comparison Figure 5.3 shows the comparison of power consumption of the handwritten versus the generated ICORE realization.

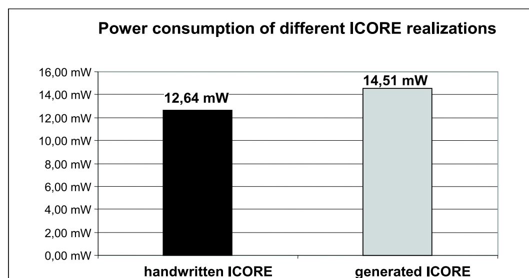

Fig. 5.3: Power consumption.

The hand-written model consumes 12,64mW, whereas the implementation generated from a LISA model consumes 14,51mW. The reason for the slightly worse numbers in power consumption of the generated model versus the hand-written is due to the early version of the LISA HDL generator which in its current state allows access to all registers and memories within the model via the test-interface. Without this unnecessary overhead, the same results as for the hand-optimized model are achievable.

6 Conclusion and Future Work

In this paper we presented the LISA processor design platform (LPDP) – a framework for the design of application specific integrated processors. The LPDP platform supports the architecture designer in different domains: architecture exploration, implementation, application software design and system integration/verification.

In a case study it was shown that an ASIP, the ICORE architecture, was completely realized using this novel design methodology – from specification to implementation. The results concerning maximum frequency and power consumption were comparable to those of the hand optimized version of the same architecture.

Moreover, in earlier work [12] the quality of the generated software development tools was compared to those of the semiconductor vendors. Due to the usage of the compiled simulation principle, the generated simulators run by one to two orders in magnitude faster than the vendor simulators. Moreover, the generated assembler and linker can compete well in speed with the vendor tools.

Our future work will focus on modeling further real world processor architectures and improving the quality of our retargetable C compiler. As the data-path in the architecture implementation phase has to be specified twice – once in C within the LISA-model and once in the hardware description language – verification is a major problem. For this reason we will examine techniques to formally verify the data-path of the different models against each other.

References

[1] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr, "LISA – Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures," in *Proc. of the Design Automation Conference (DAC)*, (New Orleans), June 1999.

[2] LISA Homepage, *http://www.iss.rwth-aachen.de/lisa.* ISS, RWTH Aachen, 2001.

[3] J. Rowson, "Hardware/Software co-simulation," in *Proc. of the Design Automation Conference (DAC)*, 1994.

[4] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nML," in *Proc. European Design and Test Conf., Paris*, Mar. 1995.

[5] Hartoog, M. et *al.*, "Generation of software tools from processor descriptions for hardware/software codesign," in *Proc. of the Design Automation Conference (DAC)*, Jun. 1997.

[6] G. Hadjiyiannis, S. Hanono, and S. Devadas, "ISDL: An instruction set description language for retargetability," in *Proc. of the Design Automation Conference (DAC)*, Jun. 1997.

[7] Halambi, A. et *al.*, "EXPRESSION: A language for architecture exploration through compiler/simulator retargetability," in *Proc. of the Conference on Design, Automation & Test in Europe (DATE)*, Mar. 1999.

[8] Kobayashi, S. et *al.*, "Compiler generation in PEAS-III: an ASIP development system," in *SCOPES 2001 - Workshop on Software and Compilers for Embedded Systems*, Mar. 2001.

[9] C.-M. e. a. Kyung, "Metacore: An application specific DSP development system," in *Proc. of the Design Automation Conference (DAC)*, Jun. 1998.

[10] M. Barbacci, "Instruction set processor specifications (ISPS): The notation and its application," *IEEE Transactions on Computers*, vol. C-30, pp. 24–40, Jan. 1981.

[11] R. Gonzales, "Xtensa: A configurable and extensible processor," *IEEE Micro*, vol. 20, 2000.

[12] A. Hoffmann, A. Nohl, G. Braun, and H. Meyr, "Generating Production Quality Software Development Tools Using A Machine Description Languagexs," in *Proc. of the Conference on Design, Automation & Test in Europe (DATE)*, Mar. 2001.

[13] T. Gloekler, S. Bitterlich, and H. Meyr, "Increasing the Power Efficiency of Application Specific Instruction Set Processors Using Datapath Optimization," in *Proc. of the IEEE Workshop on Signal Processing Systems (SIPS)*, Oct. 2001.