

# Architecture Implementation Using the Machine Description Language LISA

Oliver Schliebusch, Andreas Hoffmann, Achim Nohl,  
Gunnar Braun and Heinrich Meyr  
Integrated Signal Processing Systems, RWTH Aachen, Germany  
schliebusch,{hoffmann,nohl,braun,meyr}@iss.rwth-aachen.de

## Abstract

*The development of application specific instruction set processors comprises several design phases: architecture exploration, software tools design, system verification and design implementation. The LISA processor design platform (LPDP) based on machine descriptions in the LISA language provides one common environment for these design phases. Required software tools for architecture exploration and application development can be generated from one sole specification. This paper focuses on the implementation phase and the generation of synthesizable HDL code from a LISA model. The derivation of the architectural structure, decoder and even approaches for the implementation of the data path are presented. Moreover, the synthesis results of a generated and a handwritten implementation of a low-power DVB-T post processing unit are compared.*

## 1. Introduction

Today's communication market faces strong competition and multiple new standards. For this reason, several systems, such as mobile devices, networking products or modems require new embedded processors (EP). These EPs can either be general purpose, such as microcontrollers ( $\mu$ C) and digital signal processors (DSP), or application specific, using application specific instruction set processors (ASIP). The decision between flexibility and high optimization to special applications is driven by the time intensive task to develop new architectures. In fact, there is only little margin for design exploration and finding the best-in-class solution.

This results from the fact that the development process of new ASIPs is separated into several development phases, such as design exploration, software tools design, system integration and design implementation. Moreover, the design phases are assigned to different design engineer groups with expertise knowledge in their field. Design automation is mostly limited to the dedicated design phases; even

software tools and description languages vary from phase to phase. Thus, a very important but time-intensive factor is communication and verification between different design groups or at least design phases. The development time can be decreased significantly by employing a retargetable approach using a machine description language. The Language for Instruction Set Architectures (LISA) [10] was developed for the automatic generation of consistent software development tools and synthesizable HDL code.

## 2. LISA language

The language LISA [10] 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 purposes. 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. 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 *resource model* describes the available hardware resources, for example registers or functional units and the resource requirements of operations. Resources reproduce properties of hardware structures which can be accessed exclusively by a given number of operations at a time.
- 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.
- 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 *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.

These various model components are sufficient for generating software tools as well as a HDL representation each with their particular requirements.

Furthermore, LISA models may cover a wide range of abstraction levels. This comprises all levels starting at a pure functional sight, modeling the data path of the architecture, to register transfer level (RTL) accurate models. Besides a proper description of the structure, RTL models include detailed information about the micro-architecture model. Therefore, these models can be used to generate a HDL representation of the architecture, using the languages VHDL, Verilog or SystemC. Certainly a working set of software tools can be generated from all levels of abstraction.

### 3. Related work

Hardware description languages (HDLs) like VHDL or Verilog are widely used to model and simulate processors, mainly with the goal of implementing 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.

Instruction set languages are mainly designed to retarget the software development tools, sometimes a complete tool suite. However, instruction set languages operate on a high level of abstraction to provide as much convenience as possible for fast and efficient design exploration phases. Unfortunately, the required information about the underlying hardware is missing.

The language nML was developed at TU Berlin [2, 1] and adopted in several projects [7, 4]. The nML language supports, similar to the LISA language the grouping of instructions regarding the coding and syntax of the instruction set. Unfortunately, this grouping is also used for the identification of functional units. Thus, resource sharing is not possible. Moreover, implementation results are currently not presented.

The same restriction regarding the generation of functional

units applies to ISDL [6]. 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. As the generation of functional units is the result of an analysis and optimization process of the HDL generator HGEN, the designer has only indirect influence to the generated HDL model. Moreover, no results on the efficiency of the generated HDL code is given.

In contrast to instruction set languages, SystemC [3] is an approach to combine the requirements of system level design and hardware implementation. Semiconductors and embedded software companies announced the Open SystemC Initiative (OSCI) in September 1999. SystemC is based on the C++ programming language and provides via a class library all elements which are necessary to implement hardware. This comprises timing, concurrency and reactive behavior. However, SystemC differs from LISA as the designer is not able to derive the software development tools from the architecture model, since as with VHDL or Verilog information about the instruction set is missing.

### 4. Architecture design

Today's standard architecture development process uses description languages in two fields for the development of new architectures: for architecture exploration, the software development tools are realized using a high level lan-

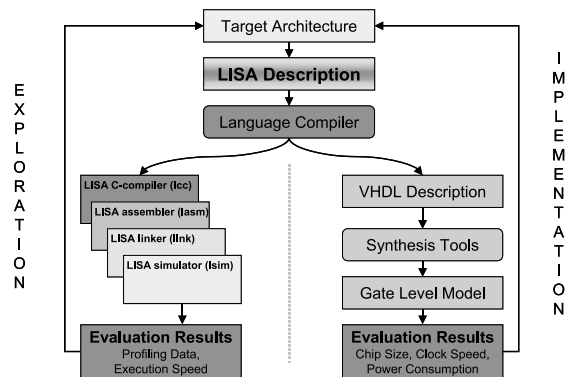


Figure 1. Exploration and implementation

guage as C/C++ to describe the target architecture from the instruction set view, whereas (low level) Hardware Description Languages (HDL) like VHDL and Verilog are used to model the underlying hardware in detail for implementation purposes.

It is obvious that combining the development processes of software tools suite and HDL description is extremely benefiting. As can be seen in figure 1 the LISA language compiler generates both and design changes only influence the

LISA description. By this, consistency problems vanish and the generated software development tools and HDL code are correct by construction.

The LISA processor design platform (LPDP)[9] is an environment that allows the automatic generation of software development tools for architecture exploration and application design, hardware-software co-simulation interfaces, and hardware implementation, from one sole specification of the target architecture in the LISA language. The set of LISA tools comprises the following programs:

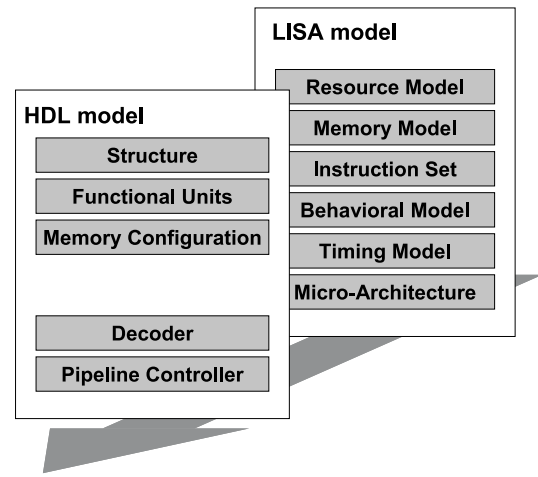
- The LISA language debugger for debugging the instruction-set as well as the behavior with a dedicated graphical debugger frontend.
- The Assembler which translates text-based instructions into object code for the respective programmable architecture.
- The linker which is configured by a dedicated linker command file.
- The Instruction-set architecture (ISA) simulator for cycle accurate simulation including support for deep instruction and data pipelines.

After design exploration and application design the target architecture needs to be implemented, which will be discussed in subsequent part of this paper.

## 5. Architecture implementation

The LPDP platform supports the generation of a HDL representation of the architecture. Since, the generated HDL model does not consist of any predefined components, such as ALUs or basic control logic, the LISA compiler must derive all necessary information from the given LISA description. Thus, the generated HDL model components can be fully compared to the LISA model components given in section 2 and illustrated in figure 2:

- The memory configuration, which summarizes the register and memory sets including the bus configuration is directly derived from the LISA memory model.
- The structure of the architecture, such as pipeline stages and pipeline registers is generated. The required information is gathered from the resource model, behavioral model and the micro-architecture model.
- The functional units are generated from the micro-architecture model. Depending on the HDL language used, the functional units are either generated as empty frames or with full functionality, which is discussed detailed in section 5.3.
- The decoders are resulting from the coding information included in the instruction set model and the timing model.



**Figure 2. LISA model and correspondent HDL model components**

- The pipeline controller is also generated from the instruction set model and the timing model.

This relation between LISA model and generated HDL components is the basis for a comprehensible implementation process. Required changes to the HDL model may be applied either to the LISA model or - if necessary - to the generated HDL model. In fact, the designer has full control over the generated HDL model with all its components. Moreover, the designer has the choice to generate a VHDL, Verilog or SystemC representation of the target architecture. As modern EPs are highly optimized concerning clock speed, chip area or power consumption, the generated HDL code has to fulfill tight constraints to be an acceptable replacement for HDL code handwritten by experienced designers. Especially the data path of an architecture is highly critical and, in most cases, must be optimized manually. Frequently, full-custom design technique must be used to meet power consumption and clock speed constraints. As the LISA behavior model is described using the C/C++ programming language, it cannot be transformed efficiently to VHDL or Verilog. Thus, only frames for the functional units are generated. If SystemC is chosen, the data path will be also generated, as reported in section 5.3. The different parts of the generated HDL model will be discussed now more detailed.

### 5.1. The HDL model structure

The LISA resource model and memory model cover information about the register configuration, memory configuration, pipeline sets and pipeline registers. This information is used to generate the base structure of a HDL model.

The base structure consists of different entities for the register resources, the memory resources and the pipeline, as shown for the case study in figure 5. The register resources are completely generated, including RTL level HDL code to model the register behavior. In contrast, the memory entity is left empty, thus the designer may place a desired memory model into this entity.

The pipeline consists of several entities representing the pipeline stages and pipeline registers. Additionally, the pipeline contains the pipeline controller, derived from the LISA model. Since LISA provides a formalized way to initiate pipeline functions like flush or stall, the HDL generator is able to utilize this information. Furthermore, the pipeline controller is driven by various decoders, placed in the pipeline stage entities.

The pipeline stages contain the entities representing the functional units. More precisely, the functional units implement the data path and will be discussed in detail later. Besides decoder, multiplexers are generated to avoid driver conflicts. These driver conflicts may appear if several functional units require exclusive access to the same resources. The HDL generator derives the information about the exclusiveness from the coding information included in the LISA instruction set model and uses the LISA timing model to generate adequate multiplexers.

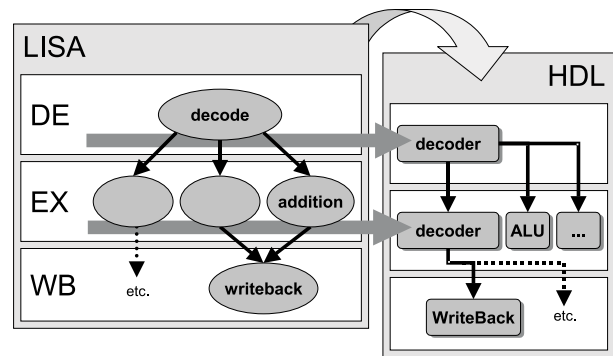
## 5.2. Decoder generation

The decoder generation requires information from various LISA model components. In fact, detailed information about hardware operations and their execution schedule are needed. Thus, the decoder is derived from the instruction set model, the timing model and the micro-architecture model. These models are implemented by several LISA operations, which are ordered in a tree-like structure. Moreover, a single operation may be assigned to a dedicated pipeline stage. Therefore, the behavior of a software instruction, for example the instruction `add`, is distributed over several different LISA operations as shown in figure 3:

- The operation `decode` is assigned to the `DE` stage. This operation loads the operands from a general purpose register into a pipeline register.
- The operation `addition`, which adds the values in the pipeline registers and writes the result back to another pipeline register. This operation is assigned to the `EX` stage
- The operation `writeback`, which is assigned to the `WB` stage, writing the value from the pipeline register to the general purpose registers.

The operation execution depends on the LISA timing model. Whereas, the timing model results from the fact, that LISA operations may activate each other, as indicated by arrows on the left side of figure 3. Thus, a schedule results, which rules the execution of the behavior included in the LISA operations. These activation sequences are translated to control signals in the HDL model, which are set or reset depending on the information given in the coding section of the respective LISA operation.

Decoders are generated in each stage activation signals start. Therefore, two decoders are generated, respectively one decoder in the `DE` stage and the `EX` stage. If the activation sequence would be changed in such manner that the decode operation activates all other LISA operations, only a single decoder in the `DE` pipeline stage would be generated.



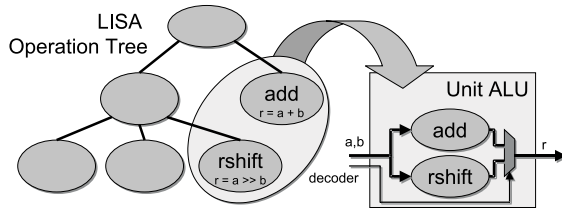
**Figure 3. LISA operation tree and decoder generation**

## 5.3. Data path implementation

The functional units covering the data path result from the micro-architecture model, as certain hardware operations can be grouped in a LISA model to one unit. Figure 4 shows the grouping of the operations `add` and `rshift`. In the case of VHDL or Verilog code generation these functional units are generated as frames with all necessary input and output ports. The data ports, such as operands are derived from the LISA behavior model. Additionally, the units are connected to the respective control signals driven by the decoder. Since all ports are derived from the LISA description the existence of the ports is fully comprehensible to the designer. Changes to the LISA model are immediately reflected in the HDL model. In this example, the designer has to implement the addition and shift manually, and a multiplexer must be implemented to judge which result is written to the output port.

The disadvantage of rewriting the data-path in the VHDL or Verilog 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. The solution for this verification issue is the usage of the SystemC language, which can also be processed by standard synthesis tools. As the behavior description in a LISA model is based on the C/C++ programming language, the functional units can be generated completely if SystemC is used for the behavior description and HDL code generation. In fact, the behavior of the architecture, which is split into different LISA operations is combined in the generated functional units. The control signals are automatically used by the generated multiplexer to ensure a correct execution of the respective functional unit.



**Figure 4. Data path generation from LISA**

One major research topic here is the resource sharing issue. For example the developer may implement the functionality of a shifter in two different exclusive executed LISA operations, which are assigned to the same functional unit. Thus, in hardware only one shifter resource is needed. The HDL generator is currently only able to detect obviously shared resources. The reason for this is that it is difficult to extract the semantic of a C/C++ statement in the LISA behavior model. Thus, the automatic generation of the data path can currently only be used as first step to a more efficient, hand written implementation of the data path.

### 5.4. Implementation results

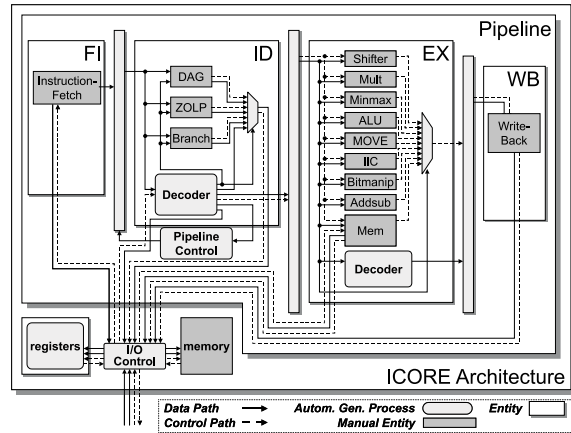
The ICORE is a low-power application specific instruction set processor (ASIP) for DVB-T acquisition and tracking algorithms [5]. It has been developed in cooperation with Infineon Technologies. The primary tasks of this architecture are the FFT-window-position, sampling-clock synchronization for interpolation/decimation 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 spend 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.

The language VHDL has been chosen to implement the architecture. As a consequence the data path within functional units has been written manually, whereas the VHDL code of the remaining architecture has been automatically generated. Figure 5 shows the composition of the model. The gray boxes have been filled manually with HDL code, whereas the white boxes and interconnects have been completely generated.

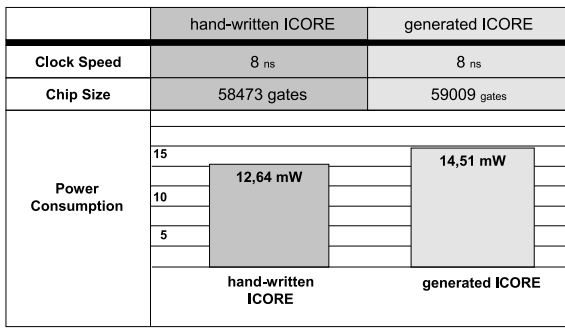
**5.4.1. Design effort.** The LISA model of the ICORE as well the original hand-written VHDL model of the ICORE architecture have been developed by one designer. The initial manual realization of the VHDL model (without the time needed for architecture exploration) took approx. three months. The LISA model was already developed 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 derive 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 VHDL model, could be completed in less than a week. This comparison clearly indicates, that the time-expensive work in realizing the VHDL 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 and does not to be rewritten for implementation purpose.



**Figure 5. The generated VHDL model**

**5.4.2. Gate level synthesis.** To verify the feasibility of automatically generating HDL code from LISA architecture descriptions in terms of power-consumption, clock speed and die size, a gate level synthesis was carried out. The results are summarized in figure 6. To get meaningful data,



**Figure 6. Implementation Results**

the generated model has not been changed (i.e. manually optimized) to enhance the results. Hence, all given results are only for a definition of position of the generated VHDL model in comparison to the hand-written, hand-optimized implementation.

**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 bottlenecks were found in both the hand-written and the generated model. The bottlenecks occur exclusively in the data-path, which highlights the statement that the data-path is the most critical part of the architecture.

**Critical path:** The synthesis has been performed with a clock of 8ns, this equals 125MHz. The critical path violates this timing constrains 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. As can be seen in figure 6 the area of both implementations are nearly the same.

**Power consumption:** In addition to the area and the timing comparison, the power consumption has been also evaluated. Figure 6 shows the comparison of power consumption of different ICORE realizations. The hand-written model consumes 12,64mW, instead the implementation generated from a LISA model consumes 14,51mW.

## 6. Conclusion and Future Work

In this paper we presented the architecture implementation based on the machine description LISA. The components of the LISA model and the HDL model were compared and an overview to the generated elements, as struc-

ture, decoder and data path was given. In a case study it was shown that a real-world ASIP, the ICORE architecture, was completely realized using the LISA based HDL code generation. The results concerning maximum frequency, die size and power consumption were comparable to those of the hand optimized version of the same architecture. Moreover, in earlier work [8] the quality of the generated software development tools was compared to those of the semiconductor vendors. It carried out, that the generated software tool suite can compete well with commercial products.

Our future work will focus on modeling further real world processor architectures. Moreover, the optimized generation of the data path, considering the resource sharing issue, is another major research topic. Additionally, the automatic generation of pipelined functional units in order to combine the advantages of flexible DSPs and application optimized ASIPs is of major interest.

## References

- [1] A. Fauth, M. Freericks, and A. Knoll. Generation of hardware machine models from instruction set descriptions. In *Workshop on VLSI Signal Processing*, pages 242–250, 1993.
- [2] A. Fauth, J. Van Praet, and M. Freericks. Describing instruction set processors using nML. In *Proc. European Design and Test Conf., Paris*, Mar. 1995.
- [3] J. Gerlach and W. Rosenstiel. System Level Design Using the SystemC Modeling Platform. Source: [www.systemc.org](http://www.systemc.org).
- [4] Geurts, W. et al. Design of DSP systems with Chess/Checkers. In *2nd Int. Workshop on Code Generation for Embedded Processors*, Mar. 1996.
- [5] T. Glökler, S. Bitterlich, and H. Meyr. ICORE: A Low-Power Application Specific Instruction Set Processor for DVB-T Acquisition and Tracking. In *13th IEEE Workshop on Signal Processing Systems (ASIC/SOC)*, Washington DC, Sep. 2000.
- [6] G. Hadjiyiannis, P. Russo, and S. Devadas. A methodology for accurate performance evaluation in architecture exploration. In *Proceedings of the 36th Design Automation Conference*, pages 927–932, 1999.
- [7] 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.
- [8] A. Hoffmann, A. Nohl, G. Braun, and H. Meyr. Generating Production Quality Software Development Tools Using A Machine Description Language. In *Proc. of the Conference on Design, Automation & Test in Europe (DATE)*, Mar. 2001.
- [9] A. Hoffmann, A. Nohl, G. Braun, O. Schliebusch, T. Kogel, and H. Meyr. A Novel Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using a Machine Description Language. *IEEE Transactions on Computers-Aided Design*, Nov. 2001.
- [10] 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.