RESEARCH ARTICLE OPEN ACCESS

# Leon Processor Architecture Implementation with LISA

P.V. Bhandarkar1, Dr.S.S.Limaye2
1Assistant Professor, RCOEM, RTM Nagpur University
Nagpur, M.S, India pvb_bhandarkarpv@rdiffmail.com
2Principal, JIT, RTM Nagpur University
Nagpur, M.S, India shyam_limaye@hotmail.com

**Abstract-**
This paper presents the machine description language LISA for the generation of bit and cycle accurate models of LEON processors. Depending on a behavioral operation description, the architectural specification and pipeline operations of modern LEON processors can be successfully implemented. The behavioral model LISA of includes other architecture related information like the instruction set. Also the information provided by LISA models enables automatic generation of simulators. It is mainly focused on ADL based verification of Leon processor.
**Keywords-**LISA, Leon processor, Verification

## I. INTRODUCTION

Recent advances & progress in the semiconductor technology has enabled the electronics industry with the dreams of realizing more ambitious design goal. With the tremendous progress in the fabrication technology & the computer-Aided-Design (CAD) tools, there is an remarkable growth in the market of embedded processors & the application specific Instruction-set Processors (ASIPs).There is great demand of these processors in different domains. More interesting part is that nowadays the major innovations in these processors are taking place in the Architecture level. In the architectural level the decisions regarding the pipeline depth, caching strategy, priority arbitration , efficient power, area management etc are to name a few.

These architectural decisions are the design intent of the system-designer, in fact design intent are like pillars of any successful implementation of the desired system. Verification of these large systems is still a major verifying the whole system simultaneously. Hence, industry resorts to the verification of the smaller RTL blocks only. Through the verification of the local properties, validation engineer tries to guess whether the global intent has been implemented or not in the RTL. The design intent expression is usually made from a higher abstraction level. We have chosen LISA language to model the Leon processor using it since provide the higher abstraction

## II. SIMILAR WORK

To explore architectural tradeoffs during the design phase of embedded processors [2,3] has led to an increased interest in ADLs for processor design. [3,8] .The ADL are offering promising avenues for fast design-space exploration with enough room for optimization for target-specific architectures.
The advantages offered by ADL-based design are as follows1.

Faster design-space exploration.
2. Seamless integration of the components through the automatic generation of the software tool-bottleneck to Electronic Design Automation (EDA) industry. Although the demand for the processors is huge, due to short time-to-market span, there is a huge competition among the vendors.
All these circumstances make the situation more complex. In the technical front, the capacity limitations of the various verification methodologies largely depend on chain as well as RTL description of the processor.
3. The higher abstraction doing away with implementation.

The processor design ADLs can be categorized into one of three categories. These categories include languages that focus on describing the processor at the architectural level (RTL or structural level), languages that focuses on the instruction level (behavioral level), and the languages that incorporate a joint behavioral and structural design approach.[2,5] MIMOLA (Machine Independent Microprogramming Language) is an example of an ADL that describes the processor at the RTL level.[2, 8] The MIMOLA language is very much similar to that of Verilog or HDL. The software tool suite utilizes the structural definition of the processor that

results in poor quality compilers and assemblers.[8] Languages such as MIMOLA do not support the exploration of different processor architectures also.[2] [9] nML and Instruction Set Description Language ( ISDL ) are examples of ADLs that describe the design of an embedded processor at the behavioral level.[2] [8] nML will produce an assembler based on the defined model, however the generated simulator does not support cycle accurate pipeline or VLIW architectures.[6] The nML processor model can be used with the separate CHESS compiler to generate processor specific code from a higher level language source file.[3,2 ,10] ISDL is similar to nML and requires a separate compiler tool to generate processor specific assembly files.[7] Since these languages model the processor from a instruction set view, the ability to group common functionality together into dedicated hardware units is severely limited; which in return limits the amount of resource sharing that can occur.[2]

Many newer ADLs are available that describe the embedded processor in a combined behavioral and structural manner. Examples of such ADL tools include EXPRESSION, Xtensa by Tensilica, CoWare Inc.'s LISA.[2] Xtensa is built on a predefined RISC core and is limited in the architectures that it can model.[10] EXPRESSION and LISA are more flexible ADLs and allow arbitrary processor architectures and their memories to be designed and simulated. Both languages provide automatic generation of a complete tool suite and RTL code generation from the model description.[12] [10] [8] .

EXPRESSION is a public-domain product. No results have been published on the efficiency of the generated tool suite or RTL code based on an EXPRESSION model. CoWare's LISA tool suite is the leading commercially supported ADL. CoWare is the only such toolset that has both UNIX and Windows versions.[2] [8].

**III. LISA LANGUAGE**

The language LISA [13] is aiming at the formalized description of programmable architectures with their peripherals and interfaces connections. 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 Single Input Multiple Data (SIMD), Multiple Input Multiple Data(MIMD) and VLIW-type architectures. Other than this, processors with complex pipelines can be easily modelled. The LISA machine description provides information consisting of the following model components:

1. The *resource model* describes the available hardware resources, for example registers or functional units and the resource requirements of operations to be performed.
2. The *instruction set model* identifies valid combinations of hardware operations and admissible operands. It is given by the assembly syntax, instruction word coding, and the specification of valid operands and addressing modes for each instruction.
3. The *behavioral model* abstracts the activities of hardware structures to operations changing the architecture of the processor for simulation purposes. The abstraction details of this model can range widely between the hardware implementation level and the level of high-level language (HLL) statements.
4. The *timing model* specifies the activation sequence of hard ware operations and units.
5. 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 units, multipliers units, 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 System C. Certainly a working set of software tools can be generated from all levels of abstraction.

LISA can be used to model any processor that is defined by an instruction set, such as an Simple Reduced Computer (SRC) or a Digital Signal

processor (DSP). The LISA language allows for the easy representation of pipelined (cycle-accurate) and VLIW processors. The LISA tool suite includes Processor Designer, Processor Debugger, Processor Generator, and a C Compiler.

The full description of the hardware and instruction set of the processor can be developed with Processor Designer. This combining of hardware and software design for an embedded processor greatly reduces the complexity and time of modelling. LISA can be used to describe the instructional hierarchy, which lends itself to easy addition of instructions to an already defined design. The behaviour of each instruction within LISA is coded in ANSI-C. Processor Designer tool of LISA generates an assembler, linker, disassembler, an instruction set simulator, and a debugger, based on the LISA design description. Along with these tools, Processor Designer tool can generate an instruction set users manual. The instruction set users manual lists the complete instruction set in the architecture along with syntax and a short description of how each instruction is implemented. The LISA language model can be tested and debugged using the Processor Debugger tool, before the HDL code is generated. Within the debugger tool the user has the ability to view all the hardware resources, including registers and memories, as the assembly code is executed. The debugger tool also keeps track of statistical information about the processor, such as the percentage of time a pipeline stage spent executing on data or how many times a block of assembly is run i.e keeps iteration counters on every line of the assembly code. Once the LISA model is verified with Processor Designer, HDL code can be generated with the Processor Generator tool. The Processor Generator tool generates HDL code in either Verilog or HDL. Options are also given for the optimization of the generated HDL code; which includes the degree of resource sharing between functional Units.

## IV. 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 language 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.

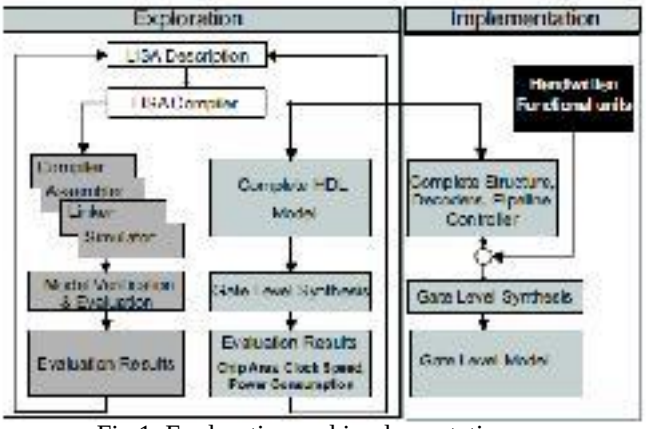


Fig 1. Exploration and implementation

The LISA processor design platform (LPDP) [12,14] 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 consist of the following programs

1. The LISA language debugger for debugging the instruction-set as well as the behavior with a dedicated graphical debugger frontend.
2. The Assembler which translates text-based instructions into object code for the respective Programmable architecture.
3. The linker which is configured by a dedicated linker command file.
4. 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.

## V. 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[13] 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 as given in following section illustrated in figure 2:

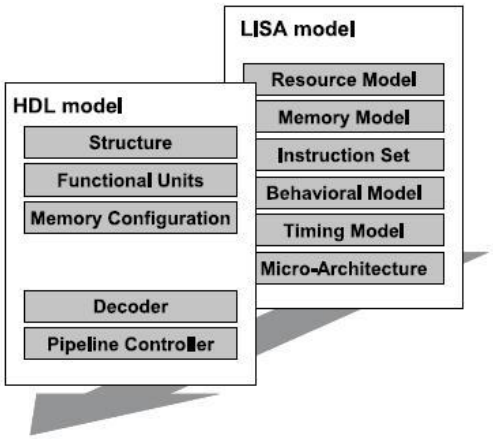


Fig 2: LISA model and correspondent HDL model components

1. The memory configuration, which summarizes the register and memory sets including the bus configuration, is directly derived from the LISA memory model.
2. 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.
3. 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.
4. The decoders are resulting from the coding information included in the instruction set model and the timing model.
5. The pipeline controller is also generated from instruction set model and the timing model.

## VI A.LEON PROCESSOR ARCHITECTURE

The LEON architecture is a Scalable Processor Architecture V8 compatible architecture, which was initially developed by the European space Agency [20]. The complete RT-level VHDL source code is freely available from Gaisler Research [21].

To begin with, the integer pipeline and memory configuration of the LEON were modeled in LISA. SPARC is a CPU instruction set architecture (ISA), derived from a reduced instruction set computer (RISC) lineage. SPARC is an instruction set architecture (ISA) with 32-bit integer and 32-, 64-, and 128-bit IEEE Standard 754 floating-point as its principal data types. It defines general-purpose integer, floating-point, and special state/status registers and 72 basic instruction operations, all encoded in 32-bit wide instruction formats.

### B. LEON PROCESSOR pipeline

Leon Processor [20,21] is a pipelined architecture. Pipelining is one of the key concepts in architecture of processor, defined as an implementation technique in which multiple instructions can be executed in overlapping way. This is possible if only operation tasks performed in each cycle of a multi-cycle architecture are clearly defined and independent from one another [24].

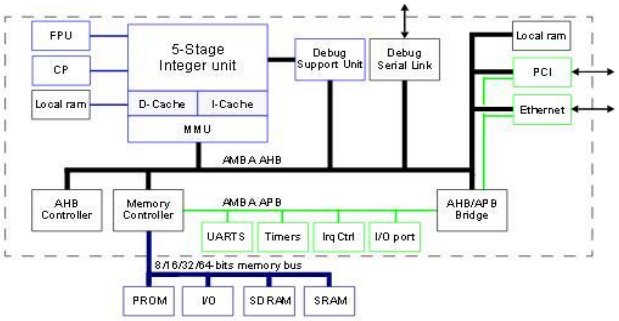


Fig.2 Block Diagram of Leon Processor

The Leon Processor pipeline is a five stage pipeline.

Each instruction's execution follows all or some of the pipeline stages corresponding to these cycles: Instruction Fetch (IF), Instruction Decode/Register Read (ID), Execution/Effective Address (EX), Memory Access/Branch Completion (MEM), and Write-back (WB) cycle. Communication among the various stages is done using pipeline registers, as shown in Fig.

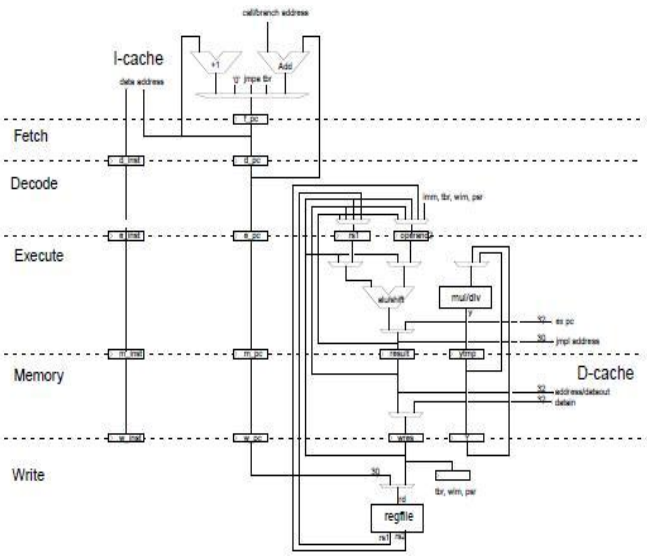


Fig. 3. Pipelined LEON architecture.

As shown, LEON load/store architecture features a 5-stage instruction pipeline including: instruction fetch (IF), instruction decodes (ID), execution (EX), and memory access (MEM) and write back (WB).

Instruction Fetch is a primary pipeline cycle in which a new instruction is fetched from the instruction memory and then sent to the Instruction Register (IR), then the Program Counter (PC) is incremented by 4. Next, during the Instruction Decode/Register Read (ID) cycle, the fetched instruction is decoded and register file is accessed in order to initialize internal registers.

### C.LISA Implementation

Considering the above description of the Leon processor, we are planning to implement it through definition and ADL implementation ( LISA) of several groups of instructions. We primarily focus on integer instructions . Leon architecture also supports floating-point operations.

In order to make the implementation easy, we decided to divide integer instructions into six groups, based on functions they perform:

1. load/Store instructions
2. arithmetic/logical/shift instructions,
3. read/write control instructions,
4. Floating point operate
5. Coprocessor operates instructions.

All of them have some different characteristics. Some registers and memory are also mandatory for the Leon processor to function correctly. I n i mp l e m e n t a t i o n of the IU may contain from 40 to 520 general-purpose 32-bit ***r*** registers. This corresponds to a grouping of the registers into 8 *global r* registers, plus a circular stack of from 2 to 32 sets of 16 registers each, known as register windows. Since the number of register windows present (N WINDOWS) is implementation-dependent, the total number of registers is implementation-dependent. We are planning to implement this general purpose register with pipeline register to implement the above instructions.

### VII. Proposed Leon Processor verification

The verification of Proposed Leon Processor functionality can be done by comparing the result obtained in simulation and the one attained using existing and already verified processor architecture.

### VIII. CONCLUSION AND FUTURE WORK

LISA is a language which aims at the formal description of programmable architectures, their peripherals, and interfaces. The language supports different description styles and models at various abstraction levels. Its development was necessary since existing approaches are not able to produce cycle-accurate models of pipelined DSP architectures and to cover their instruction-set. Furthermore, LISA enables the principle of fast compiled simulation of embedded processors. This paper provides an overview of the LISA language and discusses modeling issues.

In future work we will focus on modeling further real-life processor architectures and the generation of fast simulators.

## Reference

[1.] Uwe Meyer-B¨ase, Alonzo Vera, Suhasini Rao, Karl Lenk, and Marios Pattichis FPGA Wavelet Processor Design using Language for Instruction-set Architectures (LISA). Independent Component Analyses, Wavelets, Unsupervised Nano-Biomimetic Sensors, and Neural Networks V, Proc. of SPIE Vol. 6576, 65760U, (2007).

[2.] A. Hoffmann, F. Fiedler, A. Nohl, and Surender Parupalli, *A Methodology and Tooling Enabling Application Specific Processor Design*, IEEEConference on VLSI Design, 2005

[3.] A. Hoffmann, H. Meyr, and R. Leupers,

*Architecture Exploration for Embedded Processors with LISA*, Kluwer Academic Publishers, Boston, 1 ed., 2002.

[4.] P. Ienne and R. Leupers, *Customizable Embedded Processors*, Morgan Kaufmann, Boston, 1 ed., 2006

[5.] A. Hoffmann, A. Nohl, G. Braun, and H. Meyr, *A Survey on Modeling Issues Using the Machine Description Language LISA*, , 2001

[6.] G. Hadjiyiannis, S. Hanono, and S. Devadas, *ISDL: An Instruction Set Description Languagefor Retargetability*, in Proc. of the Design Automation Conference (DAC), Jun 1997

[7.] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau, *EXPRESSION: A Language for Architecture Exploration Compiler/Simulator Retargetability*, In Proc. of the Conference Design,Automation & Test in Europe, Mar. 1999

[8.] Peter Marwedel, *The MIMOLA Design System: Tools for the Design of Digital Processors*, In Proceedings of the 21st Design Automation Conference, pages 587-593, 1983

[9.] S. Yang, Y. Qian, H. Tie-Jun, S. Rui, and H. Chao-Huan, *A New HW/SW Codesign Methodology to Generate a System Level Platform Based on LISA,2005*

[10.] R. Gonzales, *XTensa: A Configurable and Extensible Processor*, IEEE Micro,Mar. 2000

[11.] Vincent P. Heuring and Harry F. Jordan, *Computer System Design and Architecture Second Edition*, Pearson Education Inc., 2004

[12.] 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

[13.] 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

[14.] Zivojnovi, S.Pees & H.Meyr;LISA–machine description language and generic machine model for HW/SW codesign in Proceedings of the IEEE Workshop on VLSI Signal Processing,San Francisco,Oct 1996

[15.] http//www.ertrwth_aachende.com

[16.] Integrated Verification Approach during ADL-driven Processor Design, Anupam Chattopadhyay, Arnab Sinha, Diandian Zhang, Rainer Leupers, Gerd Ascheid, Heinreich Meyr, Microelectronics Journal, Volume 40, Issue 7, July 2009.

[17.] S. Pees, V. Zivojnovic, A. Ropers, and H. Meyr, Fast Sim-ulation of the TI TMS 320C54x DSP," in Proc. Int. Conf. on Signal Processing Application and Technology (IC-SPAT), (San Diego), pp. 995{999, Sep. 1997.

[18.] Stefan Pees,Andreas Hoffmann,Vojin Zivojnovic, Heinrich Meyer Meyr LISA -Machine Description Language forCycle_Accurate Models of Programmable DSP Architectures

[19.] J. Rowson, \Hardware/Software co-simulation," in Proc. Of the ACM/IEEE Design Automation Conference (DAC), 1994.

[20.] esa:LEON http://www.estec.esa.nl/wsmwww/leon/.

[21.] Gaisler Research. http://www.gaisler.com/