RTL Processor Synthesis for Architecture Exploration and Implementation

Oliver Schliebusch,
A. Chattopadhyay, R. Leupers,
G. Ascheid, H. Meyr
Institute for
Integrated Signal Processing Systems
Aachen University of Technology
Germany
schliebusch@iss.rwth-aachen.de

Mario Steinert
Infineon Technologies
81541 Munich, Germany
steinert@infineon.com

Gunnar Braun, Achim Nohl
CoWare, Inc.
2121 N. First Street
San Jose, CA 95131
gunnar,achim@coware.com

Abstract

Architecture description languages are widely used to perform architecture exploration for application-driven designs, whereas the RT-level is the commonly accepted level for hardware implementation. For this reason, design parameters such as timing, area or power consumption cannot be taken into consideration accurately during design space exploration. Design automation tools currently used to bridge this gap are either limited in the flexibility provided or only generate fragments of the architecture. This paper presents a synthesis tool which preserves the full flexibility of the architecture description language LISA, while being able to generate the complete architecture on RT-level using SystemC. This paper also presents two real world architecture case studies to prove the feasibility of our approach.

1 Introduction

The key factor of designing Application Specific Instruction Set Processors (ASIPs) is an efficient design space exploration phase. Architecture Description Languages (ADLs) are widely used to find the optimum application specific programmable solution. They allow to apply changes to the architecture model quickly as the level of abstraction is higher than RT-level. Although this higher level of abstraction is the basic reason for the success of ADLs, the link to the physical parameters such as chip area, power consumption or clock speed gets lost. Ignoring physical parameters in the design space exploration phase leads to suboptimal solutions or long redesign cycles. The necessity of combining the high level abstraction and physical parameter evaluation in a single exploration is obvious.

1530-1591/04 $20.00 (c) 2004 IEEE

The ADL used for our framework is the Language for Instruction-Set Architectures (LISA) [1] [2]. As shown in figure 1, a LISA model of the target architecture is used to automatically generate software tools such as C-compiler[3], assembler, linker and simulator. These software tools are used to profile and modify both architecture and application. This exploration loop is repeated until a sufficient cost/performance ratio is reached.

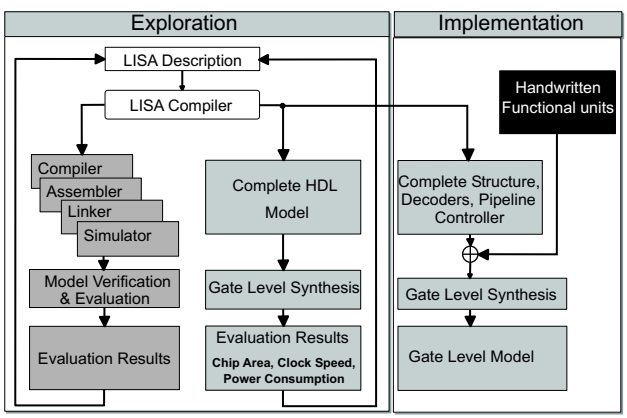

Figure 1. Exploration and Implementation based on LISA

Also shown in figure 1, a complete hardware model is generated in order to get a first estimation about the clock speed, area and power consumption. In this paper we present a framework to take the gate-level synthesis results into account during the exploration phase. The LISA model is used to derive a fully synthesizable model on RT-level. For the first time the designer is able to perform this synthesis flow, without being limited in terms of flexibility. In previous approaches, this flow was realized by using predefined RT-level components or architecture templates.

If the synthesis results of the automatically generated architecture fulfill the given physical constraints, then the hardware model can even be used for the final architecture implementation. As the datapath is often highly optimized and based on in-house IP it may be replaced by the designer manually for the final architecture implementation. This is shown in figure 1 on the right hand side.

LISA provides great flexibility by allowing the designer to describe the state transition of the architecture in ANSI-C code. In order to bridge the gap between the ADL LISA and the RT-level without limiting the flexibility, a ANSI-C related hardware modelling language was chosen. Thus, we are using the SystemC language as RT-level hardware description language. The way we are using the SystemC language does not differ from a description in VHDL or Verilog on RT-level.

The contribution of this paper is to propose an automated synthesis flow from the ADL LISA to the RT-level description for the evaluation of physical parameters. Thus, values such as area, power consumption and timing can be taken into consideration early in the design space exploration. Compared to previous approaches we neither limit the flexibility of the ADL nor utilize predefined RT-level components.

This paper is organized as follows: In the next section, related work is discussed. Previous work on the HDL generation from LISA, which is required for the understanding of this paper, is described in 3. The generation of a complete hardware representation in SystemC is discussed in section 4. Afterwards, in section 5, the results of two case studies are presented. The paper ends with conclusion and outlook.

2 Related Work

Several ADLs support hardware generation from higher levels of abstraction than RT-level. The different languages can be organized into those focusing on the architecture, on the instruction-set or a combination of both.

The languages oriented towards the architecture are close to the RT-level and thus not suitable for a fast and efficient design space exploration phase. For this reason, projects such as MIMOLA [4] are not further discussed here.

Some of the languages strongly oriented towards the instruction-set are ISDL [5] and nML [6]. For example, the synthesis tool HGEN [7] is used to generate synthesizable Verilog code from an ISDL description. The HDL generator GO from Target Compilers Technologies [8], which is an industrial product, is based on the architecture description language nML. The synthesis results are not publicly available. The project Sim-HS [9] is also based on the nML description language and generates synthesizable Verilog models from Sim-nML models. Here, non-pipelined architectures are generated and the base structure of the generated hardware is fixed. Moreover, each functional unit can perform exactly one type of operation each clock cycle. Units such as multiply-add cannot be generated.

Approaches based on an instruction set/architecture combination are mentioned in the following. Information on HDL generation from the EXPRESSION [10] language is presented in [11]. The HDL generation is based on a functional abstraction and thus allows to generate the complete architecture. FlexWare [12] is more related to RT-level than to the level of ADLs and thus not suitable for a fast design space exploration phase. The PEAS-III [13] and the derived ASIP-Meister [14] work with a set of predefined components. These are able to fulfill tight constraints regarding the synthesis results. The limiting factor here is the lacking flexibility.

In addition to the work based on architecture description languages, design systems have to be discussed here. The XTensa [15] environment from Tensilica [16] allows the user to select and configure predefined hardware elements. Hence, the design space exploration can be performed very efficiently and synthesis results are convincing. Also, the PICO (program in, chip out) [17] system developed by the HP-labs is based on a configurable architecture, including nonprogrammable accelerators and cache subsystems. The major issue of these approaches is that the designer is largely limited to predefined processor architectures.

3 The HDL Synthesis Framework

Three different types of hardware descriptions namely, *explicit*, *implicit* and *non-formalized* can be found in a LISA model. Our previous work focused on the usage of the first two categories and has already been presented in [18]. To provide a better understanding a short summary is given here.

3.1 Explicit Hardware Description

This type of hardware description results from language elements with a well defined semantics. Therefore, they define the underlying hardware without any ambiguity. The corresponding RT-level hardware can be generated directly from a given LISA model. The resource section shown in figure 2 is used to directly derive the base structure of the architecture. Entities to cover the memories, registers and the complete pipeline structure are generated. In this example, the pipeline consists of four stages with fetch (FE), decode (DE), execute (EX) and writeback (WB).

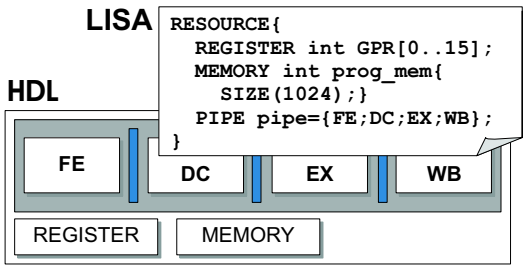

Figure 2. explicit hardware description

3.2 Implicit Hardware Description

The language elements, which provide an implicit hardware description, are not self-explanatory concerning their semantical meaning. The semantics of these language elements is more general and hardware generation requires a deeper analysis of the complete model. Sometimes even additional information not included in the model needs to be taken into account. For example, this information may be the knowledge about the simulation scheduler. This fact can be explained by the LISA timing model which is based on the so-called *activation* of LISA operations. An activated operation is marked for execution and executed by the simulation scheduler at the correct point of time. To derive the RT-level decoder the `ACTIVATION` section of the LISA model and the simulation scheduler have to be taken into account.

```
OPERATION decode in pipe.DC {
  DECLARE  { GROUP instr = {add||sub}; }
  BEHAVIOR { PIPELINE_REGISTER(p,DC/EX).a = R1;
             PIPELINE_REGISTER(p,DC/EX).b = R2;
           }
  ACTIVATION { instr }
}
OPERATION add in pipe.EX {
  DECLARE  { INSTANCE writeback; }
  BEHAVIOR { PIPELINE_REGISTER(p,EX/WB).r =
             PIPELINE_REGISTER(p,DC/EX).a +
             PIPELINE_REGISTER(p,DC/EX).b;
           }
  ACTIVATION { writeback }
}

OPERATION sub in pipe.EX {
  DECLARE  { INSTANCE writeback; }
  BEHAVIOR { PIPELINE_REGISTER(p,EX/WB).r =
             PIPELINE_REGISTER(p,DC/EX).a -
             PIPELINE_REGISTER(p,DC/EX).b;
           }
  ACTIVATION { writeback }
}

OPERATION writeback in pipe.WB {
  BEHAVIOR {R=PIPELINE_REGISTER(p,EX/WB).r;}
}
```

Example 1: Implicit hardware description and non-formalized hardware description

In example 1, the operation `decode` activates `instr`, which in turn leads to an activation of operation `add` or `sub`. From there, operation `writeback` is activated. As every operation in this chain is assigned to a separate pipeline stage, the simulation scheduler executes the activated operations corresponding to the spatial delay given by the pipeline organization. In order to generate hardware from this model that complies with the simulated processor behavior, the control path has to be generated according to the behavior of the LISA simulation scheduler.

4 Generating the Complete RTL Model

In order to generate the complete architecture the non-formalized hardware descriptions of LISA must be taken into account. Combining these two aspects is the novelty of our approach: on the one hand we maintain the full flexibility of LISA and on the other hand we synthesize the complete architecture.

4.1 Non-formalized Hardware Description

The datapath of the architecture is modelled in LISA by the `BEHAVIOR` section. This section contains mainly plain ANSI-C code, enriched by LISA language elements. In example 1, the LISA keyword `PIPELINE_REGISTER` is embedded into the ANSI-C behavior description of the `add` and `sub` operations.

Thus, the `BEHAVIOR` section can be considered as non-formalized hardware description[1]. The LISA language does neither provide language elements nor rules for the description of, for example, an address generation unit in any way. As previously mentioned, the SystemC language is used to generate a RT-level representation of the architecture. The usage of SystemC does not differ from the usage of VHDL or Verilog. For that reason, gate level synthesis results are independent from the choice of language. In the following paragraph, the transformation to SystemC on RT-level is described.

In order to generate RT-level hardware description from the non-formalized elements in LISA, a more complex transformation must be provided compared to the first two hardware description types. This affects the LISA model structure as well as the behavior description given in ANSI-C. To face the various possibilities of hardware description, our approach is to stay in the domain of the ANSI-C programming language and generate a SystemC model of the architecture. The already discussed types of hardware descriptions can also easily be used to generate SystemC rather than VHDL. This move to SystemC as HDL provides

[1]The programming language ANSI-C is of course a formalized language. While using LISA we have to consider ANSI-C as hardware description, which is non-formalized.

the following advantage: only those elements of the LISA behavior description must be identified and replaced which are not standard ANSI-C code. The SystemC code translated from the `BEHAVIOR` section must comply with the SystemC RT-level synthesis guidelines [19].

4.2 SystemC Implementation of the Datapath

The remaining task now is to convert the LISA behavior description itself into SystemC code. As the resources are defined in a global scope, the identified resources can be replaced by the access to ports or signals. Figure 3 shows this conversion. The operands and the destination resources are converted into ports. The postfix `_in` and `_out` indicate the data direction of the port. The data-value `(REG)`, the address `(AW)` and the valid flag `(EW)` are identified by their respective prefixes. Lines are prepended, which describe typical RT-level implementation details, such as writing an address signal or an enable flag. In this example the lines `AW_R_out=5;` and `EW_R_out=1;`.

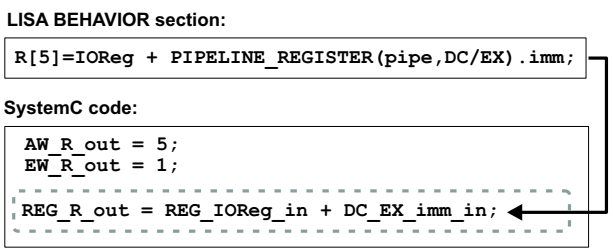

Figure 3. Transformation of Resources

The proposed translation mechanism enables the HDL generation from LISA to support the generation of the complete architecture. As different instructions which operate on the same hardware may be described in different LISA operations, resource sharing is an additional challenge. To solve this issue, the ANSI-C code of the model must be analyzed concerning the underlying hardware and the mutual exclusion of operations. While the latter can be derived from the LISA operation graph, the former is a more complex task, and will be addressed in our future work.

5 Results

Two case studies are presented in this section in order to prove the feasibility of the SystemC RT-level generation from the ADL LISA. The CoWare/LISATek Edge Processor Designer [20] provides the environment for the hardware synthesis tool presented in this paper. This includes, for example, tools generator and simulator frontend.

5.1 The LEON Architecture

The LEON architecture is a Sparc V8 compatible architecture, which was initially developed by the European Space Agency (esa) [21]. The complete RT-level VHDL source code is freely available from Gaisler Research [22]. For comparison, the integer pipeline and memory configuration of the LEON were modelled in LISA. The automatically generated SystemC model and the reference model are synthesized with the Synopsys design compiler [23]. The LISA model as well as the generated SystemC model have been verified with the microSparc Validation suite. The prelayout synthesis results are presented in table 1.

While the achieved clock speed is already in the range of hand-written HDL code, the result regarding the area indicates that there is a significant potential for optimizing the generation process. This topic will be addressed in our future work. However, taking into consideration that all software tools and a hardware model are generated from the same specification, the synthesis results are acceptable for early estimations.

	handwritten LEON	generated LEON	ratio
timing	3.08 ns	4.42 ns	1.44
gates	16.7 kGates	37 kGates	2.22

Table 1. LEON Synthesis results

5.2 The Infineon Technologies ASMD

The Application-Specific Multirate DSP (ASMD) from Infineon Technologies [24] is an ASIP dedicated for interpolation and decimation filters or the CORDIC algorithm.

The ASMD LISA model is derived from an existing version of the ASMD, which has been developed in VHDL on RT-level. Time spent to develop both models are shown in figure 2. The final synthesis results of the ASMD are presented in table 3. The completely generated SystemC model fulfilled all design constraints and thus was taken for the final implementation.

	original ASMD (VHDL)	generated ASMD (LISA + SystemC)
development time	5 months	2 months
model size	1765 lines	738 lines

Table 2. ASMD modelling efficiency

The evaluation of the ASMD with respect to a bluetooth application produced the requirement of one additional instruction to increase the throughput significantly. For that

	original ASMD	generated ASMD	ratio
timing	4.22 ns	5.09 ns	1.21
gates	9549 Gates	11678 Gates	1.22

Table 3. ASMD synthesis results

reason we changed the LISA model, regenerated the RT-level model and verified the new models within one day. We added 53 lines to the LISA model, whereas 208 lines would have to be added in the RT-level model.

Due to the immense advantage in design time and negligible overheads in speed and area, the generated version of the ASMD replaced the existing core and is now being used in a bluetooth device.

6 Conclusion and Future Work

In this paper, we presented a methodology to fully generate a synthesizable RT-level hardware description from LISA. The gap between the ADL LISA and the implementation level can be bridged using the SystemC language for hardware description. The novelty of the presented work is the fact that the complete architecture can be generated from a single specification, without losing the flexibility provided by LISA. Thus, physical parameters, such as clock speed, area and power consumption can be taken into account during the design space exploration phase. Two case studies, in addition to already published example architectures [18], have been presented.

We will add an analysis and optimization step in our future work. The requirements to this optimization process are derived from the various case studies we investigated until now. This will improve the generated control path as well as the data path of the architecture regarding gate-level synthesis results.

References

[1] A. Hoffmann, H. Meyr, and R. Leupers. *Architecture Exploration for Embedded Processors with LISA*. Kluwer Academic Publishers, 2002.

[2] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, A. Wieferink, and H. Meyr. A Novel Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using a Machine Description Language. *IEEE Transactions on Computer-Aided Design*, 20(11):1338–1354, Nov. 2001.

[3] M. Hohenauer, H. Scharwaechter, K. Karuri, O. Wahlen, T. Kogel, R. Leupers, G. Ascheid, H. Meyr, G. Braun, and H. van Someren. A Methodology and Tool Suite for C Compiler Generation from ADL Processor Models. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, Paris, France, Feb 2004.

[4] R. Leupers and P. Marwedel. Retargetable Code Generation based on Structural Processor Descriptions. In *Design Automation for Embedded Systems*, volume 3, no. 1. Kluwer Academic Publishers, Jan. 1998.

[5] 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.

[6] A. Fauth, J. Van Praet, and M. Freericks. Describing Instruction Set Processors Using nML. In *Proc. of the European Design and Test Conference (ED&TC)*, Mar. 1995.

[7] A Fauth, M. Freericks, and A. Knoll. Generation of Hardware machine Models from Instruction Set Descriptions. In *Proc. of the IEEE Workshop on VLSI Signal Processing*, 1993.

[8] Target Compiler Technologies. *http://www.retarget.com*.

[9] V. Rajesh and R. Moona. Processor Modeling for Hardware Software Codesign. In *Int. Conf. on VLSI Design*, Jan. 1999.

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

[11] P. Mishra, A. Kejariwal, and N. Dutt. Rapid exploration of pipelined processors through automatic generation of synthesizable rtl models. In *Rapid System Prototyping (RSP)*, San Diego, USA, 2003.

[12] P. Paulin, C. Liem, T.C. May, and S. Sutarwala. FlexWare: A Flexible Firmware Development Environment for Embedded Systems. In P. Marwedel and G. Goossens, editors, *Code Generation for Embedded Processors*. Kluwer Academic Publishers, 1995.

[13] A. Kitajima, M. Itoh, J. Sato, A. Shiomi, Y. Takeuchi, and M. Imai. Effectiveness of the ASIP Design System PEAS-III in Design of Pipelined Processors. In *Proc. of the Asia South Pacific Design Automation Conference (ASPDAC)*, Jan. 2001.

[14] ASIP Meister. http://www.eda-meister.org.

[15] R. Gonzales. Xtensa: A configurable and extensible processor. *IEEE Micro*, Mar. 2000.

[16] Tensilica. *http://www.tensilica.com*.

[17] V. Kathail and S. Aditya and R. Schreiber and B.R. Rau and D. Cronquist and M. Sivaraman. Automatically Designing Custom Computers. *IEEE Computer*, 35(9):39–47, Sept. 2002.

[18] O. Schliebusch, A. Hoffmann, A. Nohl, G. Braun, and H. Meyr. Architecture Implementation Using the Machine Description Language LISA. In *Proc. of the ASPDAC/VLSI Design - Bangalore, India*, Jan. 2002.

[19] Synopsys. *CoCentric SystemC Compiler RTL User and Modeling Guide*
http://www.synopsys.com, 2003.

[20] CoWare/LISATek. *http://www.coware.com*.

[21] esa: LEON-1. http://www.estec.esa.nl/wsmwww/leon/.

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

[23] Synopsys. *Design Compiler*
http://www.synopsys.com/products/logic/logic.html, 2001.

[24] M. Steinert, O. Schliebusch, and O. Zerres. Design Flow for Processor Development using SystemC. In *SNUG Europe Proceedings - Munich, Germany*, Mar. 2003.