Modeling Instruction Semantics in ADL Processor Descriptions for C Compiler Retargeting

JIANJIANG CENG, WEIHUA SHENG, MANUEL HOHENAUER, RAINER LEUPERS, GERD ASCHEID AND HEINRICH MEYR

Institute for Integrated Signal Processing Systems, Aachen
University of Technology (RWTH), Sommerfeldstr. 24 WSH Aachen, Germany

GUNNAR BRAUN
CoWare, Inc. Dennewartstrasse 25-27, Aachen Germany

Abstract. Today’s Application Specific Instruction-set Processor (ASIP) design methodology often employs centralized Architecture Description Language (ADL) processor models, from which software tools, such as C compiler, assembler, linker, and instruction-set simulator, can be automatically generated. Among these tools, the C compiler is becoming more and more important. However, the generation of C compilers requires high-level architecture information rather than low-level details needed by simulator generation. This makes it particularly difficult to include different aspects of the target architecture into one single model, and meanwhile keeping consistency.

This paper presents a modeling style, which is able to capture high- and low-level architectural information at the same time and make it possible to drive both the C compiler and the simulator generation without sacrificing the modeling flexibility. The proposed approach has been successfully applied to model a number of contemporary, real-world processor architectures.

Keywords: architecture description language, processor model, C compiler retargeting, embedded processor design, electronic system level

1. Introduction

Today, an increasing number of Application Specific Instruction-set Processors (ASIPs) are employed as building blocks in embedded System-on-Chip (SoC) designs. Since the architectures and the instruction-sets of ASIPs are highly optimized for specific applications or application domains like image processing or network management, they are very efficient in executing such target applications. On the other hand, compared to a hardwired Application-Specific Integrated Circuit (ASIC), ASIPs’ programmable architectures are much more flexible. Therefore, ASIPs are getting more and more attractive due to their balance between performance and flexibility.

To design an ASIP, one of the most important steps is architecture exploration. In the architecture exploration phase, designers need to find out the optimum architecture for the target application. First, the application is profiled to determine critical portions that require dedicated hardware support by using application specific instructions. This is also often referred to as hardware-software partitioning. After that, the instruction-set is defined based on the result of partitioning and profiling. Then, the micro-architecture of the processor is designed to implement the
instruction-set. The process is partially or fully repeated when different design space parameters are exploited until the design requirement is met.

Such iterative architecture exploration process demands that each time when the architecture is tuned, software tools such as compiler, assembler, linker and simulator should be updated and available as soon as possible so that the tuning result can be examined to find out the potential improvement for the next iteration. It is obvious that manually re-targeting these software tools is a time-consuming, tedious, and error-prone work. For this reason, Architecture Description Languages (ADLs) are developed to aid the design of ASIPs. Different from hardware description languages like VHDL or Verilog which are mainly designed to model and simulate hardware, ADLs characterize processor architecture from different high-level aspects, e.g. instruction behavior, assembly syntax, binary coding, and so on. Base on ADL processor models, required software tools are automatically generated which significantly improves design efficiency.

However, the generation of different software tools requires heterogeneous information on the architecture. It is challenging to make one description to fit all requirements. This is especially true for the C compiler generation and the instruction-set simulator generation. Instruction-set simulators need the knowledge in detail about how the architecture executes an instruction, e.g. internal data manipulations, side effects calculation, cycle-accurate pipeline activities, etc. In contrast, C compilers view these instructions from a much higher, semantics-oriented abstraction level. What they need to know is the purpose of instructions rather than their execution on the micro-architecture level. Due to these orthogonal requirements, it is difficult to include information for both software tools into one single model.

In this paper, we describe an extension of LISA 2.0 [1], a widespread industrial ADL, towards the modeling of instruction semantics for C compiler re-targeting. The design of this extension aims at enabling the description of high-level instruction behavior with minimum design effort. With this extension, embedded processor designers can generate a C compiler conveniently from a LISA processor model. Moreover, our approach is not only useful for the C compiler generation. In [2], we described the technique of simulator generation based on the work described in this paper and the related model consistency issues. Combined with the C compiler generation capability, our approach fulfills the demand for consistent-tool generation from a single ADL model, meanwhile not sacrificing flexibility. As the tool generation exceeds the scope of this paper, here we will focus on the new language constructs.

The rest of this paper is organized as follows: Section 2 shortly discusses the approaches of related works. An overview of the LISA ASIP design methodology is given in Section 3. Section 4 reviews several important principles in the LISA language. Section 5 describes in detail the design criteria and the extension of the language, which is the core of this paper. Section 6 presents the modeling results of several real-world processors. Finally, Section 7 concludes the whole paper.

2. Related Work

Within the last decade, a variety of ADLs has been developed to support ASIP design. However, not all of them support the generation of compilers. One important architecture-specific component in compiler is the code selector. It performs the task of translating the intermediate representation (IR) of the applications into assembly instructions. To generate a code-selector for a processor architecture, the knowledge of instruction semantics, i.e. what instructions do, is needed. Because most of the ADLs known today were originally designed to automate the generation of a specific processor design tool, e.g. simulator, and later extended to other tools, different modeling styles were developed to support the generation of code-selectors. Based on the nature of the information provided, these ADLs can be classified into three categories: structural, behavioral and mixed.

The MIMOLA [3] language belongs to the structural ADL category. Its modeling style is similar to that of the VHDL hardware description language. The instruction set information is extracted from the Register-Transfer Level (RTL) module netlist for use in code selector generation [4]. The software programming model of MIMOLA is an extension of PASCAL.

ISDL [5] is classified as behavioral ADL. It provides the means for hierarchical specification of instruction sets. During the code selector generation, the correlation between the target processor operations and the compiler basic operations comes from the behavior description of each instruction [6]. Because ISDL cannot model the structural details for pipelining, cycle accurate simulator generation is not possible.