SimPL: An Object-Oriented Methodology for Modeling the Precise Behavior of Processor Architectures

João Cláudio S. Otero, Flávio R. Wagner
{jcotero, flavio}@inf.ufrgs.br

Federal University of Rio Grande do Sul - UFRGS
Computer Science Institute
Porto Alegre – RS – Brazil

Abstract

This paper presents SimPL, an object-oriented methodology for modeling processor behavior with precise timing, which may be used as a basis for both teaching and design environments.

1 Introduction

Environments for modeling current state-of-the art processor architectures serve one of two main purposes: teaching and design. Teaching environments (such as WinDLX [WIN] and Escape [ESC]) are usually oriented to a base processor that can be modified in a limited way. Design environments, in turn, usually offer powerful languages (such as Lisa and Expression) for modeling various processor architectures classes and must automatically generate retargetable tool suites. SimPL is a processor design *methodology* (and not a language with some fixed set of constructs), based on a specialized class library developed on top of a general-purpose modeling and simulation framework – SIMOO [SIM 97]. In SimPL, the processor modeling process automatically results in a simulator with built-in interactive resources for tracking and controlling experiments. For the moment, the methodology does not generate other tools, such as compilers or assemblers, and generation of the simulator code is not aimed at high performance.

2 The SimPL Modeling Methodology

The SimPL modeling methodology explicitly divides a processor into MAIN_CONTROL and DATAPATH blocks. Each of these blocks is an aggregation of *control elements (INSTRUCTION_SET and EXECUTION)* and *functional elements (MEMORY, REGISTER...),* respectively**.** Futhermore, *information elements* are used for building a bridge between the control and datapath blocks.

The datapath block does not model physical connections between the functional elements. Connections are implicitly defined by means of the behavior specified by the control elements. This behavioral approach makes the architecture definition very flexible. Functional elements may be more easily replaced or introduced, because their interconnections do not need to be modeled.

Each functional, information or control element is implemented as a *class,* which presents *micro-operations* as an interface to the other elements. These micro-operations are messages capable of firing specific methods that implement operations within the elements.

Control elements define the processor behavior as sequences of these micro-operations, but they do not directly call methods of the functional elements. They communicate only with the aggregating DATAPATH element. This element thus redirects the messages to its aggregated functional elements, by calling the corresponding methods.

With this approach, the class diagram that models the processor does not contain direct connection relationships between a control element and all the functional elements it may control. If these direct relationships were necessary, the class diagram would be much more complex.

Micro-operations typically define interactions between functional elements. As an example, suppose we need to transfer the contents of register REG to input *Inp* of the functional element ALU. A control element would execute the following micro-operation, wich is implemented as a method that is local to this element:

ALU_SET(*Inp*,REG_CONTENTS( ))

As a result of this method, a message is sent to the aggregating element DATAPATH, requesting it to execute method SET of functional element ALU with two parameters (the last one is another micro-operation REG_CONTENTS with a similar format). These micro-operations implicitly create a path from register REG to ALU.

3 Defining Processor Control

Control elements, from types MAIN_CONTROL, INSTRUCTION_SET and EXECUTION, interact with each other to define the overall processor control behavior.

INSTRUCTION_SET is an element that defines each processor instruction as a set of *instruction steps.* In turn, each instruction step is a set of micro-operations to be executed in parallel. As already explained, each micro-operation specifies operations in functionl elements and implicitly defines datapaths between the functional elements. An instruction step is executed at once each time the instruction is *fired*. However, the INSTRUCTION_SET element does not enforce any timing relationship between the steps.

An INSTRUCTION_SET element may define complete instructions as well as *instruction fragments,* to be hierarchically used within other instructions or instruction fragments. This feature may be used to model instruction fragments that are common to various instructions, such as a fetch operation or a memory access using a particular addressing mode.

EXECUTION is the control element that orders the execution of all instructions steps, according to a selected *execution mode.* Depending on the mode, several *firings* may be necessary to complete the execution of all steps defined for an instruction. Each firing may be executed, for instance, at a consecutive clock cycle, but alternative timing schemes may be implemented. EXECUTION may use one of the following execution modes: *single-cycle* mode – all instruction steps of an instruction are executed in a single firing; *multi-stepped* mode – each firing executes a certain number of instructions steps; and *multi-cycle* mode – each firing executes a single instruction step.

The execution of the instructions or instruction fragments is ordered by the MAIN_CONTROL element, wich is the topmost level of the control block hierarchy and defines the overall processor control, as well as a concrete timing behavior.

MAIN_CONTROL uses micro-operations of the EXECUTION element to implement the overall control, according to a given timing behavior.

4 Case Study: Modeling the DLX Processor

DLX is a general-purpose, hypothetic RISC processor, defined by Patterson and Hennessy [PET 96]. The processor has a 5-satage pipeline, where stages are identified as Instruction Fetch, Instruction Decode, Execution, Memory Access, and Write Back to Register or Memory. The first two stages are identical for all instructions.

Modeling the DLX datapath with the SimPL methodology is straightforward. The DATAPATH element aggregates all functional elements. The basic SimPL library already contains classes that implement all datapath elements of the DLX processor. These classes must be only instantiated and parameterized. After all datapath elements have been instantiated, they make a large set of micro-operations available to the control elements.

```
op2 = NOP; op3 = NOP;      // pipeline is initially empty
while(true) {
   EX(InstrFetch);
   EX(InstrDecode);
   EX(op1 = DECODE (),1); // executes step 1 of instruction decoded in op1
   EX(op2,2);     // executes step 2 of instruction op2
   EX(op3,3);     // executes step 3 of instruction op3
    op3 = op2;    // instructions advance through the pipeline...
    op2 = op1;
    wait(1);}       // simulation advances to the next clock cycle
```

Figure 1 - MAIN_CONTROL for the DLX processor.

The MAIN_CONTROL element contains an EXECUTION element, which will be responsible for invoking micro-operations that correspond to each instruction, as defined in the INSTRUCTION_SET element, which is instatiated within EXECUTION.

The INSTRUCTION_SET element defines the four DLX instructions types in a hierarchical way. For each instruction, this element defines three instruction steps. They correspond to the pipeline stages EX, MEM, and WB, which are distinct for each instruction. Each of these stages is defined as a set of micro-operations. The pipeline stages IF and ID, which are identical for all instructions, are modeled by two instruction fragments InstrFetch and InstrDecode and invoked directly from the MAIN_CONTROL element. The EXECUTON element invokes instructions (in fact: invokes micro-operations that are made available by INSTRUCTION_SET) in a multi-cycle mode, where each processor cycle executes a single instruction step.

The overall control behavior and timing are defined within MAIN_CONTROL, as shown in Figure 1. Instruction fragments InstrFetch and InstrDecode are invoked first, to execute micro-operations corresponding to stages IF and ID.

The *information element* DECODE monitors all updates to INSTRUCTION_REGISTER, a datapath element defined within the hierarchy of

DATAPATH, and returns an instruction *inst* to be executed by micro-operation EX(op1 = DECODE(),1). This information element is necessary because the control block, according to the SimPL approach, does not have a direct association to each functional element within the datapath block. The information element may obtain the information and pass it to the control block.

The movement of instructions through the pipeline is modeled by auxiliary variables *op1* thru *op3*. All 5 pipeline stages are executed in parallel, and instructions move to the next stage also in parallel. The SIMOO function *wait(1)* advances the simulation clock and leads the processor to the next cycle.

5 Conclusion

This paper presented the SimPL methodology for modeling the precise behavior of processor architectures. SimPL is built on top of SIMOO, a general-purpose framework for object-oriented modeling and simulation of discrete systems. From a given processor model, the methodology makes it extremely easy to obtain a new one, with a different timing behavior, or with a different set of instructions. Full exploitation of object orientation greatly enhances the capabilities for deriving new processor models from old ones. A model of the DLX processor is being implemented as a validation strategy.

6 References

[ESC] VERPLAETSE,P. ESCAPE v1.1 Manual. Available by WWW at http://www.elis.rug.ac.be/escape

[EXP 99] HALAMBI,A. et al. EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability. In:Proceedings of DATE 99 – Design, Automation and Test in Europe. Munich, Germany, March, 1999.

[LIS 99] PEES,S. & HOFFMANN,A. & ZIVOJNOVIC,V. & MEYR,H.: "LISA – Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures". In: Proceedings of the 36th Desgin Automation Conference. New Orleans, USA, June 1999.

[PET 96] PATTERSON, D. A . & HANNESSY, J.L. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc, 1996.

[SIM 97] SIMOO – An Environment for the Object-Oriented Discrete Simulation. In: Proceedings of ESS 97 – 9th. European Simulation Symposium. Passau, Germany, October 1997.

[WIN] GRENBACHER,H. WinDLX Tutorial – A First Example. Available by E-mail at maziar@vlsivie.tuwien.ac.at