COMPARING SYSTEMC AND ARCHC THROUGH THE MIPS PROCESSOR MODELING

Marcio Rogério Juliato, Paulo Cesar Centoducatte

Institute of Computing (IC) – State University of Campinas (UNICAMP)

ABSTRACT

This paper discuss the MIPS processor modeling in SystemC and ArchC, at the same time it compares both languages regarding processor architecture description. SystemC solves many current problems of software-hardware co-design and verification, on the other hand it is not suitable for automatic generation of software tools.

In order to address this problem, a new architecture description language (ADL), called ArchC was created. ArchC's main goal is to facilitate processor description, as well as to provide enough information, at the right level of abstraction in order to allow architecture exploration through the automatic generation of software tools. At the end of this job, we could compare the developed models, and then realize how useful and powerful was to describe processors using ArchC instead SystemC.

1. INTRODUCTION

Considering the increasing complexity in embedded system designs, a tool for evaluation of a new designed instruction set architecture which automatically generates a software toolkit composed by assemblers, linkers, compilers and simulators became mandatory. It is this toolkit that allows designers to get an executable specification of a new architecture to experiment with different instructions sets and resources at very first stages of the design process.

SystemC is among a group of design languages and extensions being proposed to raise the abstraction level for hardware design and verification. The language is suitable to model any kind of hardware at several levels of abstraction, but it is not suitable for automatic generation of a software development toolkit. How could one identify, for sure, how many instructions a processor can execute, which are these instructions and respective formats, whether the processor has a pipeline or not and what and how many are the stages of this pipeline from a *generic* SystemC processor description?

In order to address these problems, our research group at the Computer System Laboratory (LSC) created a new ADL called ArchC. ArchC gets an architecture description and automatically a SystemC model of the architecture, so designer compile this model and get an executable application of the processor.

2. MODELING

The MIPS processor was chosen due to its architecture and instruction set simplicity and regularity, allowing the coverage of the most actual RISC processor characteristics. It has a simple but interesting pipeline, with different instruction formats, and besides, data forwarding and pipeline stall. The modeling was based on the datapath[1] shown in Figure 1.

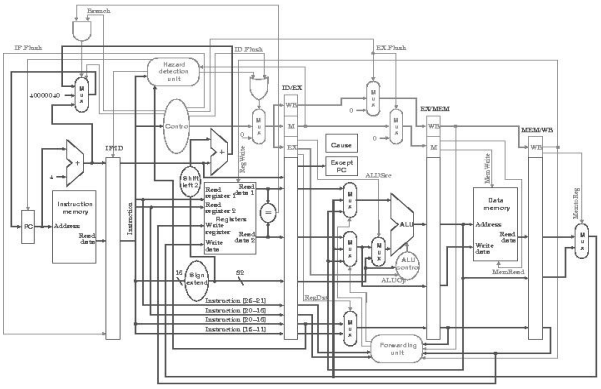

Figure 1: MIPS Datapath

On the MIPS architecture we have three instruction formats, named R, I and J, and five pipeline stages, named IF (Instruction Fetch), ID (Instruction Decode), EX (Execution), MEM (Memory Access) and WB (Write Back). The processor description paradigm was different from SystemC to ArchC models, but having similar abstraction levels.

4.1. The SystemC Model

On the SystemC model was used the paradigm of the instruction moving though the pipeline, like the real processor. This was modeled with SystemC processes corresponding to each pipeline stage, having these processes a switch structure to discover which instruction was received. Like the real processor the description counts on a register bank (RB) and pipeline registers.

The pipeline module implements what has to be done for each instruction at a determined pipeline stage. Part of the pipeline implementation can be observed in Figure 2. It is important to cite that the main goal of this description was simulation, however we have accurately followed the synthesis guidelines for SystemC allowing its synthesis without great changes. The Figure 3, shows the interface of the simulator for this model.

```
void pipeline::execution_stage() {
  switch( reg_ID_EX[ID].read() ) {
    case LW: break;
    case SW: break;
    case ADD:
      reg_bank[reg_ID_EX[RD].read()].write(
      reg_bank[reg_ID_EX[RS].read()].read() +
      reg_bank[reg_ID_EX[RT].read()].read());
      break;
    :
```

Figure 2: Pipeline implementation in SystemC

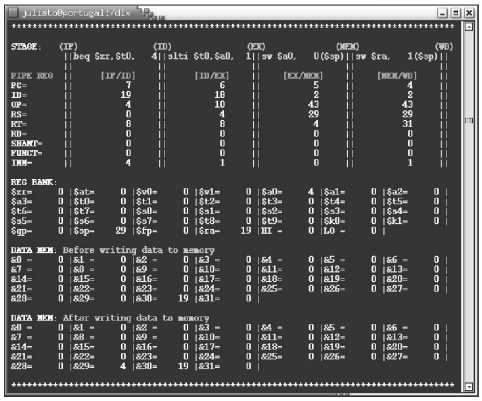

Figure 3: The MIPS model simulator

4.2. The ArchC Model

The first step for modeling in ArchC is to describe the processor's characteristics, regarding the instruction set and architecture resources. The designer has to inform instruction names, assembly syntax and its correspondent formats, as well as some information about the structure of the architecture like register banks and pipeline.

Based on these data is generated a template to be filled with the behavior for each instruction by the designer, thus reducing the modeling time. Part of the Instruction Set Architecture (ISA) description is shown in Figure 4, where we can observe the specification of all the fields of each instruction format, as also the *opcode* for the instruction decoding. Another necessary issue is the pipeline registers specification, what can be done similarly to the instruction format.

```
:
ac_format Type_R ="%op:6 %rs:5 %rt:5 %rd:5
                   %shamt:5 %funct:6";
ac_format Type_I ="%op:6 %rs:5 %rt:5 %imm:16:s";
ac_format Type_J ="%op:6 %addr:26";

ac_instr<Type_R> add, sub,...;
:
add.set_asm("add %rs, %rt, %rd");
add.set_decoder(op=0x00, funct=0x20);
:
```

Figure 4: Describing the instruction set

In ArchC the instruction behavior can be divided in order to represent its execution into a pipeline, i.e., the designer can inform what an instruction does on each pipeline stage separately. This is accomplished through a C++ switch statement, like shown in the Figure 5. In the same figure we can observe the implementation of the instruction *add*. Both models implement all the characteristics of the MIPS processor, among them data forwarding and pipeline stalls. Facilities for pipeline stalls and flushes are also provided by ArchC.

```
void ac_behavior( add ) {
   switch( stage ) {
   case _IF:
     IF_ID.npc = ac_pc + 4; break;
   case _ID:
     ID_EX.data1 = RB.read(rs);
     ID_EX.data2 = RB.read(rt); break;
   case _EX:
     EX_MEM.alures = ID_EX.data1.read() +
                     ID_EX.data2.read();  break;
   :
```

Figure 5: Describing the add instruction behavior

4. RESULTS

There are many ways of modeling processors, but this work focused on cycle accurate models, i.e., we can check out the states of the pipeline as well as the registers at each clock cycle. In both model simulators it is possible to initialize the data memory, register bank, write and execute real programs, and also see what instruction is in execution on each pipeline stage. This can be very useful for students, or designers working in companies or at universities, in order to validate the design through comparison of both models results.

Studying the SystemC description we can realize that, even with a maximum of standardization, is almost impossible to determine a consistent and robust way of describing processors that can simplify the automatic generation of a software development toolkit. All these problems arise due to implementation freedom of the language. However this task is extremely simplified when working on an ArchC description, because we know the instruction formats, the instruction assembly syntax, among other architecture details. These data are collected by a parser and processed resulting in the generation of cycle accurate simulators written in SystemC.

5. CONCLUSIONS AND FUTURE WORK

During the MIPS description we could realize that the SystemC offers many modeling possibilities, however in a non-standardized way. With ArchC we have a well-defined way of describing processors, thus reducing the development time, and also allowing the automatic generation of tools as discussed above. These are very important issues to reduce the time to market of a new product.

The conception of both models presented in this paper was important to contribute to the ArchC's improvement. Next steps include the description of more complex architectures like DSP's and VLIW processors. We have chosen the TMS320C62x processor for this purpose, which combines both characteristics.

6. REFERENCES

[1] John L. Hennessy, David A. Patterson, *Computer Organization & Design: The Hardware/Software Interface,* Morgan Kaufmann Publishers Inc., 2nd Edition, 1997.

[2] Dominic Sweetman, *See MIPS Run*, Morgan Kaufmann Publishers Inc., 1999.

[3] Synopsis Inc., *SystemC User's Guide*, Synopsis Inc., 2nd Edition, 2001.

[4] Synopsis Inc., *Describing Synthesizable RTL in SystemC*, Synopsis Inc., Version 1.1, January 2002.