Teaching pipelining and concurrency using hardware description languages

Roy Melton
Tsai Huang
Sudhakar Yalamanchili
Philip Bighman

Follow this and additional works at: http://scholarworks.rit.edu/other

Recommended Citation
Melton, Roy; Huang, Tsai; Yalamanchili, Sudhakar; and Bighman, Philip, "Teaching pipelining and concurrency using hardware description languages" (1999). Accessed from http://scholarworks.rit.edu/other/370

This Conference Proceeding is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Presentations and other scholarship by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Teaching Pipelining and Concurrency using Hardware Description Languages

Tsai Chi Huang, Sudhakar Yalamanchili, Roy W. Melton, Philip R. Bingham, and Cecil O. Alford
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia

Abstract

Relating to a previous simplified VHDL processor model [1], a more advanced synthesized VHDL pipeline microprocessor model was developed and has been used in the second term computer architecture course offered in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. This paper will first describe the pipeline processor model and its VHDL implementation. Then, it presents various implementation extensions that have been assigned and completed within a satisfactory period by participating students.

Introduction

The advent of faster microprocessors and cheaper personal computing systems has led to change in teaching methodology. This is especially true in the field of technological learning. An example can be seen in the course of computer architecture [1] where the digital modeling language VHDL is used to help students grasp hard to realize computer architecture concepts by implementing the actual gate level computer system logic in VHDL. As an exercise, a student could be asked to implement an adder that produces the correct overflow flag from the input operands, signed or un-signed. Furthermore, an enhanced adder could mimic the STE (set-on-less-than) operation of the MIPS instruction set as described in [2].

From our teaching experience, another advantage gained using VHDL in computer engineering courses is a feel for real world computing. A student who is familiar with block-structured imperative programming languages such as C++, FORTRAN, or BASIC, cannot always relate this knowledge to the learning of VHDL. This is because VHDL marshals the concepts of sequential programming into concurrency via an object-oriented methodology. As the by-product from such learning, a student is equipped with a better understanding of the hardware and software aspect of computer engineering.

In the past the capabilities of available VHDL platforms and pricing were issues of concern [1]. The ideal VHDL platform for this purpose should be a modeling/synthesis tool that is freely available to students to install and use on his/her personal system and that has the ability to compile and simulate a moderate core microprocessor model plus extra capacity for logical extension.

Currently, we have developed a simplified VHDL pipeline processor model from [2] for a junior level course in computer architecture. This model can be compiled and simulated using the student version of Altera MAX II software. Since MAX II is a limited synthesis tool, only the capability of one FPGA device, an Altera FLEX 10K, can be used to map the VHDL logic. The final mapped logic counts for about 30% of the total space. As a result, the MAX II student version serves our purpose well because there is still room for the students to add hardware extensions to the model.

The VHDL model and feasible classroom extensions are described in the following sections.

A Microprocessor Pipeline Model in VHDL

The base microprocessor model is developed to closely follow the pipelined SPIM model in Chapter 6 of [2]. This is a 5-staged pipeline architecture: fetch, decode, execute, memory, and write-back stages. The core datapath is implemented in five VHDL modules: fetch.vhd, decode.vhd, control.vhd, execute.vhd, memory.vhd. These functional modules are implemented in propagational logic where the delay elements such as DFFs and latches are minimized. Additional VHDL modules, pipe_reg1.vhd, pipe_reg2.vhd, pipe_reg3.vhd, and pipe_reg4.vhd, are constructed in delay element models for the F/D, E/M, and M/W pipeline registers. Finally, the main module, spim_pipe.vhd, integrates all VHDL modules mentioned above. We chose to explicitly instantiate pipeline registers rather than permit the Altera synthesis compiler to infer pipeline register storage. We found that explicit instantiation and use of pipeline registers facilitated learning of the architecture concepts by not “hiding” elements of the datapath.

32-bit MIPS instructions are hand assembled into machine language and stored as VHDL constants in the instruction memory which is implemented as byte...
addressed memory. There are only 8 instructions stored for the base model.

The control.vhd module decodes the instruction and generates the appropriate control signals for other modules. For example, a signal such as RegWrite is asserted when load word or ALU write back to register instructions are encountered.

Decode.vhd implements the register file and generates immediate operands which include updating registers from the write-back stage. Four 8-bit registers are implemented in the base model.

Execute.vhd contains an ALU and operates on register and immediate operands to computer data value results or memory addresses for the next pipeline stage.

Both the decode.vhd and memory.vhd modules contain memory elements where the register values and memory values are preserved throughout the processor operation. Such storage elements are modeled as concurrent processes for data write and data read operations. Each of the storage locations is initialized to a unique value to save instruction cycles at the beginning of execution. This can shorten the execution time when performing functional tests.

Finally, the write-back stage only contains the memory delay element implemented by pipe_reg4.vhd. The multiplexor that feedback the memory data or ALU data is moved into the module decode.vhd.

The block diagram of this base model is shown in Figure 1.

Figure 1. Base VHDL Pipeline Processor Model

Extension to VHDL Base Pipeline Model

The base pipeline model does not implement data hazard-detection or data forwarding, nor does it allow concurrent read/write to the same register location on the same clock cycle, i.e., writes in the first half of the cycle with read in the second half. In the base model NOPs must inserted between instructions with dependencies to ensure execution correctness. For the base model, a read/write hazard between two instructions requires 3 NOPs between dependent instructions. It can be reduced to 2 cycles when the concurrent read/write operations are permitted in a clock cycle.

Class assignments focused on removing these intermediate NOPs by implementing hazard detection and forwarding. Output from each stage such as the delayed ALU output or memory data read can be forwarded to the execution stage eliminating most classes of the data hazard stalls.

Additional assignments addressed inclusion of conditional branch instructions and support for control hazards through detection and insertion of pipeline stall cycles. Due to the nature of the pipeline structure, an extra stall cycle may be required to execute a conditional branch. This is caused by an instruction bubble between the fetch and execute stages.

From an educational viewpoint (as well as in industry’s), the testing of the model is as important as the implementation itself. Signal traces capture pipelined operation and provide a medium for students to force themselves to address the detailed sequencing of operations. The ability to understand the temporal behavior captured in a trace is prerequisite for generating test cases. We have found that requiring the students to generate test cases was a tremendous aid towards their understanding and development of intuition about such complex architectures.

Conclusion

A VHDL model of a core pipelined microprocessor datapath has been developed for junior level course in computer architecture in the Computer Engineering curriculum at the Georgia Institute of Technology. The model is extended via class assignments to enable students to generate an understanding of, and intuition for, advanced pipelining concepts. This model is an improvement over the non-pipelined model described in a previous paper [1].

Altera Corporation has been generous in supplying the VHDL environment. The experience has identified many benefits of modeling and studying a microprocessor design using hardware description language such as VHDL over conventional lectures.

References