Teaching Basics of Instruction Pipelining with HDLDLX

Miloš Bečvář
Department of Computer Science and Engineering,
Faculty of Electrical Engineering,
Czech Technical University in Prague
Karlovo nám. 13, Prague 2, Czech Republic
becvarm@fel.cvut.cz

Abstract: *HDLDLX is a graphically described VHDL model of 5-stage integer pipeline of well known DLX processor. It can be used as a platform explaining logic-level implementation of pipelined processor as a complement to SW functional simulators. Students can interact with model by implementing hazard resolution logic or modifying the pipeline structure. Even though that the model is internally represented in VHDL, the previous knowledge of this language is not required. HDLDLX can be used in conjunction with HDL Designer and Modelsim tools from Mentor Graphics corporation. Article also discusses pros and cons of using commercial EDA tools in undergraduate computer architecture course.*

1. INTRODUCTION

Good understanding of instruction pipelining is essential for current computer architecture students. RISC processors DLX [1] or MIPS64 [2] are used as examples explaining the main principles. It is a common practice to reinforce understanding of this topic by practical assignments done by students. One way to help understand of pipelining is making students to optimize programs for these processors. Cycle accurate simulators [3] with visualization capabilities such as DLXView [4], WinDLX [5] or WinMIPS64 [6] or MIPSIt [7] are used for program verification. Although these simulators give a good view of pipelined instruction execution, actual implementation of pipeline and associated hazard detection logic is usually hidden. Moreover the pipeline implementation in these simulators is mainly fixed although some parameters could be changed (e.g. functional unit latencies). Main question is how the students could experiment with actual pipeline structure on the logic design level. Some teachers presented dedicated tools which requires the students to specify the pipeline structure using specialized Hardware Description Language. These tools usually generates a complete VHDL or Verilog netlist of the processor specified. Example of such tool is ASIP Meister [8].

It is obvious that development and maintenance of similar tool is relatively complex task. We present another approach which leverages commercial EDA tools as teaching aids in undergraduate computer architecture course.

We were looking for a tool which allows design of simple pipelined processor and its simulation without knowledge of Verilog or VHDL. This requirement is a result of the fact that VHDL is introduced only to hardware oriented students after the undergraduate computer architecture course.

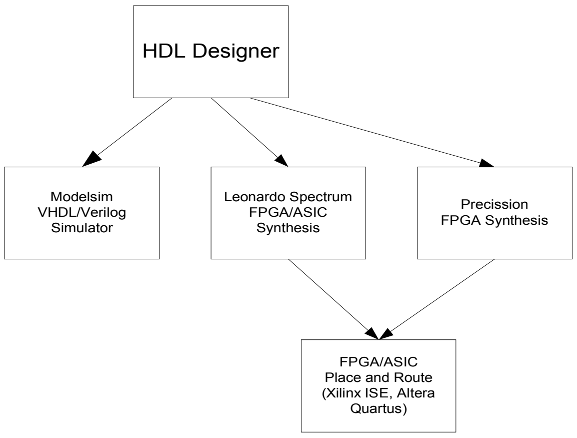
Finally we decided to use a graphical VHDL entry tool – HDL Designer from Mentor Graphics corporation for our experiments. In this article HDLDLX – a graphical VHDL model of well known integer DLX pipeline is presented. This model can be used together with HDL Designer and Modelsim for simple experiments with instruction pipeline.

The rest of the paper is organized as follows – section 2 presents an overview of HDL Designer and its use in computer architecture course. Section 3 outlines the developed HDLDLX model. Section 4 describes the use of this model in undergraduate computer architecture course. Section 5 presents conclusions and future work.

2. HDL DESIGNER OVERVIEW

HDL Designer is a professional EDA tool intended to be a “designers cockpit”. It offers a graphical VHDL entry and integrates several downstream design tools in a single GUI – namely simulator Modelsim, synthesis tool Leonardo Spectrum, Precession and others (see fig. 1) Out intention was to build a flexible graphical model of integer DLX pipeline and simulate it using a common VHDL simulator Modelsim. The reason why we decided to use this tool was in fact that we used it in specialized design courses and it was possible to extend the number of licenses without increase of maintenance fee (offered in Mentor Graphics High Education Program).

Figure 1: HDL Designer flow


HDL Designer
Modelsim VHDL/Verilog Simulator
Leonardo Spectrum FPGA/ASIC Synthesis
Precission FPGA Synthesis
FPGA/ASIC Place and Route (Xilinx ISE, Altera Quartus)

One of important question was how the processor model will be represented assuming that students do not know the VHDL or Verilog. HDL Designer offers several different types of graphical VHDL entry :

- **Block Diagram**
  It was obvious that top-level representation of the processor should be in block diagram. This diagram should be the same or similar to block diagrams used during lecture to simplify orientation of students and save time.
- **Truth Table**
  This view is useful for defining combinatorial components of the pipeline. It is also well known abstraction and easily understandable by students.
- **State Diagram**
  This abstraction is very useful for Finite State Machine specification. However, our integer pipeline does not use any state machines and this abstraction currently is not used.
- **Flow Chart**
  Flow Chart is a graphical equivalent of VHDL process. It has been used for sequential elements in design such as memories and registers

From all these components, the tool generates VHDL files which are submitted to Modelsim for simulation. Thanks to tight integration of Modelsim and HDL Designer, it is possible to cross-probe between Modelsim waveforms and graphical representation in HDL Designer. Namely, it is possible to observe a values of signals directly in the block diagram.

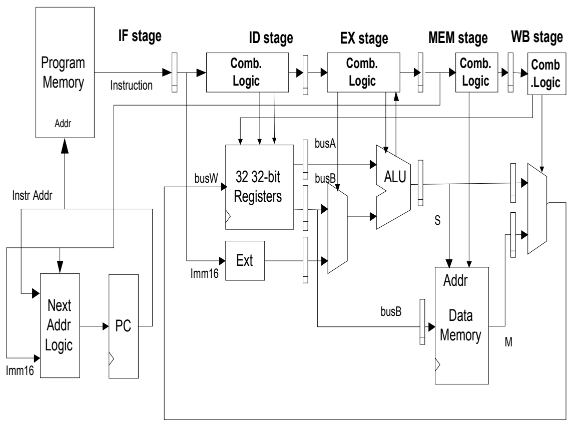
Although the creation of the model was relatively easy task for us, the big question was whether students are able to learn how to use it. Even when we use only limited functionality of the tool the overhead to learn how to use it can overweight actual benefits. For this reason high effort was spent in preparation of documentation and step by step user guide.

We return to this issue later in section 4. Another disadvantage of using commercial EDA tool is that it runs only with connection to licensing server. It limits the ability of students to run this tool from their home computer if they do not have an access to the Internet.

3. HDLDLX MODEL

Although the latest edition of Hennessy-Patterson book [2] switched to MIPS64 processor, we decided to implement 32-bit DLX processor because we use WinDLX and DLXV simulators in rest of the semester and DLX ISA during lectures. However, resulting model can be relatively easy modified to 64-bit MIPS due to similarity of pipeline structure.

Figure 2: HDLDLX pipeline



3.1 *HDLDLX Pipeline Overview*

DLX is well known 32-bit RISC processor used in computer architecture courses and many different simulators of this processor exist. These tools differs in the variant of DLX implementation because it evolves throughout the book.

Our main intention was to create the same variant of DLX as the one used during our lectures.
In the first pipelining lecture, the evolution of integer DLX processor datapath and controller from single-cycle non pipelined processor through multicycle processor to 5-stage integer pipeline.
The implemented model corresponds to this simple 5-stage integer DLX pipeline (Instruction Fetch – Instruction Decode – Execute – Memory – Write Back).

Following section outlines the implementation of DLX pipeline components.

3.2 HDLDLX Pipeline Components

HDLDLX consists of pipelined datapath and controller. Datapath is created by PC, program memory, register-file, ALU, data memory, multiplexers and pipeline registers. Controller consists of combinatorial logic in every stage and pipeline registers.

- **Program Counter (PC)** is fed by the value from Next Address Logic. Depending on control signal, the PC is either incremented by 4 or jump to branch target address.

Important control signal of PC is *IF_stall* which can prevent the PC from changing its value. This signal is very useful in implementation of pipeline stalls.

- **Program Memory** is implemented as a ROM. Its content can be preloaded from the file (containing instructions encoded in hexadecimal form). The size of Program Memory is limited to 16K x 4B (higher bits of PC are ignored).

- **Register file** implements 32 32-bit registers. It can be described as 3-ported synchronous RAM with R0 hardwired to zero. To be conform with book, the register file contains internal bypassing – data from write port could be *forwarded* directly to one of read ports in the same clock cycle (in exception of write to register R0 which is never forwarded). Register file ignores any attempt to write to register R0 which can be used to simplify the pipeline control.

- **ALU** is implemented using Truth-Table abstraction as a black box. It supports only limited set of binary and unary operations (see section 3.3). However number of operations can be expanded by increasing the size of ALU control bus and expanding the truth-table. Besides the result of operation, it also produces zero indication which can be used in branch evaluation.

- **Data Memory** is implemented as ideal synchronous RAM. Memory model pre-loads initial data from the text file and dumps its content into file after the simulation is finished. The size of Data Memory is currently limited to 64KB and higher address bits are ignored.

- **Controller.** As could be seen from fig. 2, the controller is implemented as a sequence of combinatorial logic sliced by pipeline registers. In real implementation, the instruction is decoded in ID stage into internal representation which flows through the pipeline stages. However, it would be very difficult for students to observe the pipeline behavior in this case. For this reason we let the complete 32-bit instruction code to flow through the whole pipeline. In every stage, the necessary control signals are decoded from this 32-bit instruction code. Although, this is slightly redundant implementation, it allows to understand the pipeline much easily**.** Control combinatorial logic in every pipeline stage is described as truth-table and can be easily modified.

- **Pipeline Registers** are implemented as rising-edge triggered D flip-flops. Two types of flip-flops is used in the pipeline - normal D-flip-flops in the datapath and specialized in the controller. Specialized D-flip-flops in the controller have two control signals – *stall* and *clear*. There are dedicated *stall* and *clear* signals for every pipeline stage (e.g. *ID_stall, ID_clear, EX_stall, EX_clear* etc.) If the *stall* signal is asserted, the pipeline register retains its value, if *clear* signal is asserted, the pipeline register is synchronously cleared. Clearing pipeline register in controller efficiently means that NOP is inserted to this pipeline stage.

As could be seen from fig.2 , the pipeline evaluates branch instruction in the MEM stage and no forwarding and no hazard detection is implemented. These features has to be added by students.
Main methodology to implement these changes is defining of *stall* and *clear* signals behavior either by drawing schematics or writing VHDL equation. We would discuss this in detail in the section 4.

3.3 HDLDLX Instruction Subset

Only limited instruction subset is implemented in HDLDLX. However, all types of instruction are supported (Register-Register, Register-Immediate, Load, Store and Branch).
Number of supported instructions can be expanded by changing combinatorial logic in controller and ALU.

4. EXPERIENCE WITH HDLDLX

We currently use the HDLDLX in our undergraduate computer architecture course. This course is obligatory for all computer science and engineering students and current capacity is around 300 students. The course has a single 90-minute lecture and single 90-minute laboratory seminar per week. HDLDLX is used in laboratory seminars during 4 sessions as could be seen in table 1. Experiments with HDLDLX were done every second week and interleaved with simulations on WinDLX. It means that students can compare two models of the same architecture. WinDLX also helps in understanding what must be implemented in HDLDLX. The ultimate goal is that both simulators process the same integer program equivalently.

Following subsections outline the actual use of HDLDLX during the course.

Table 1 HDLDLX in undergraduate CA Course.

Week	***Lab Overview***
1	Introduction to HDL Designer and HDLDLX model Simulation of HDLDLX with pipeline data hazards
2	Implementation of RAW hazard resolution logic (pipeline interlock)
3	Adding of control hazard resolution logic into DLX pipeline with stalls. Implementation of forwarding
4	Finalizing of forwarding

4.1 Introductory Session

As could be seen from table 1, a first session is spent in introduction to the tool and model. An ultimate goal of the first session is a brief explaining of HDL Designer and Modelsim and more detailed description of HDLDLX model.

Although only limited functionality of HDL Designer is used, some time must be spent in setting up the tool and explanation of necessary steps in using this tool in various situations. Tool setup was relatively easy task accomplished only by downloading and expanding of HDL Designer library from a web into a user directory. All HDLDLX model components were stored in shared library and students had read-only access to this library. It means that only top-level part of HDLDLX was stored in student's libraries and only this part can be modified. This restriction saves a lot of time possibly spent in tracking of peculiar bugs unintentionally introduced by students modifying of model components.
After setting-up the library in the user directory, the tool use was relatively simple and consists mainly from sequence of mouse clicking. In the beginning, students must open a project, open a library within this project and finally open structural view of HDLDLX. Luckily, the majority of these steps is performed only during a first HDL Designer run and later the tool opens the library automatically. Although it was not completely necessary, we briefly explained the concept of projects, libraries and blocks and their different views to students. The rest was not difficult to understand and students considered HDL Designer just as any other schematic editor. Generation of VHDL model, compilation and invoking of Modelsim was automated by a single clicking on Modelsim icon in HDL Designer. Overall the tool setup and necessary explanation took around 30 minutes.

Problem of explaining HDLDLX pipeline is more demanding. However, it was simplified by the fact that the same (simplified) DLX pipeline is explained in the same week during lecture. In the first lab, the effort is spent mainly in explaining how each type of instruction flow through pipeline and purpose of various parts of DLX datapath. This is illustrated by running of simulation of sample program.
Next, the concept of inserting stalls into pipeline using *stall* and *clear* signals is explained. Initial model of HDLDLX does not contain any hazard detection and resolution logic and students may actually observe the effect of RAW hazards in the pipeline. A good teaching aid is the ability of cross-probing signals between HDL Designer schematic diagram and Modelsim.

4.2 Experiments performed with HDLDLX

A list of experiments possible with HDLDLX is presented in table 1. Simple experiments are suitable for undergraduate course where students have limited access to the tool. More complex experiments can be performed in graduate course as a half semester assignments assuming that graduate students will have more knowledge of the HDL Designer and better access to the tool.

All simple experiments lead to specification of some form of combinatorial logic into pipeline. Typically, equations for *stall* and *clear* signals must be specified for implementation of pipeline interlocks and branch instruction. Forwarding is implemented by adding of multiplexers into datapath and specifying control of these multiplexers.

Table 2 Experiments with HDLDLX

Simple experiments possible with HDLDLX
• Implementation of data hazard detection logic and stalling of the pipeline • Implementation of pipeline flushing after branch instruction • Implementation of data forwarding • Moving branch evaluation into ID stage and implementation of delayed branches
More complex experiments with HDLDLX
• Implementation of Program and Data Caches and pipeline stall due to "Cache miss" • Implementation of multicycle operations (e.g. multiplication) and associated WAW, structural hazard resolution logic • Implementation of exceptions

During the first run of the course, we proposed use of schematic diagram or VHDL subset for specifying this combinatorial logic. It was expected that students would prefer schematic diagrams over learning of subset of a new language. However, majority of students decided to directly write VHDL parallel signal assignments. We thought that students preferred a text description because it is faster and it reminds them software programming.

Overhead of learning subset of VHDL syntax was not high. Students received a one-page simplified description of the VHDL parallel statements and some of them were even able to write these statements before the end of the first session with HDLDLX. Parallel assignments have smallest learning overhead in VHDL . Their another advantage is in fact that they introduce a dataflow way of thinking.

The major difficulty encountered by students was distinguishing between *std_logic* and *boolean* statements which use the same overloaded operators (e.g. AND, OR, NOT). It suggests that using Verilog can be even more straightforward in this application.

4.3 Experience from First Run on HDLDLX

After first introductory session, students work in groups of two autonomously and tried to complete assigned tasks. A role of teaching assistant during these sessions was in helping students to overcome difficulties with VHDL and trying to push them on a way to find solution. Although the fact that students can work on the simulator only in the school complicated their task, it also limited the possibility of cheating by copying solution of other groups.

Experience shows that around 50 % of groups completed the assignments during the expected time and obtained a full number of points. The rest of students required more time but majority of them were finally able to complete it also. It was an important role of teaching assistant to check that students understand “their” solution to limit the possibility of cheating.

5. CONCLUSIONS AND FUTURE WORK

Current experience shows that HDLDLX is a good teaching aid in explaining basics of instruction pipelining. The use of commercial EDA tool allows relatively fast model development comparing to building custom simulator. Although the model is based on VHDL, the students were able to use it without previous knowledge of this language. Students of undergraduate computer architecture course were able to learn a limited subset of this language relatively easy and preferred using it over schematic diagrams.

Students who completed the assigned tasks get better understanding of complexity of hazard detection logic implementation. Moreover, they were also introduced to contemporary tools and language used in design of digital circuits. Students mostly stated that their task was relatively difficult but very interesting. A fact that a commercial tool is used was also positively appreciated and some students were interested to use this tool in the future courses.

A good experience we had with HDLDLX confirmed our decision to introduce Hardware Description Languages in early courses of logic design. It means that in the future, students will come to undergraduate computer architecture course with basic knowledge of VHDL which will offer new possibilities.

HDLDLX model will be soon available on the internet - http://service.felk.cvut.cz/hdl_dlx.html

REFERENCES

[1] Patterson, D., Hennessy, J.,Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers 1996, 2nd edition

[2] Patterson, D., Hennessy, J.,Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers 2002, 3rd edition

[3] Yurcik, W., Wolffe, G., Holliday, M. , A Survey of Simulators Used in Computer Organization/Architecture Courses**,** In: *Proc. of the 2001 Summer Computer Simulation Conference* (SCS 2001), Orlando, USA

[4] Zhang, Y., Adams, G.B.: An Interactive Visual Simulator for DLX pipeline, *Newsletter of IEEE Computer Society Technical Cometee on ComputerArchitecture,* September 1997

[5] Gruenbacher, H., Khosravipour, M.,WinDLX and MIPSim Pipeline Simulators for Teaching Computer Architecture, In: Proc. of *IEEE Symposium and Workshop on Engineering of Computer Based Systems* (ECBS'96) Friedrichshafen, 1996, GERMANY

[6] http://www.computing.dcu.ie/~mike/winmips64.html

[7] Brorson, M.., MipsIT – a Simulator and Development Environment using Animation for Computer Architecture Education, In: *Proc. of Workshop of Computer Architecture Education*, Anchorage, USA , 2002

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