Modern Computer Architectures
Practical Assignments
2016 - 2017

Joost Hoozemans
j.j.hoozemans@tudelft.nl

Jeroen van Straten
j.vanstraten-1@tudelft.nl

Thomas Marconi
t.marconi@tudelft.nl

Stephan Wong
j.s.s.m.wong@tudelft.nl

Sorin Cotofana
s.d.cotofana@tudelft.nl

Chapter 1

Introduction

This document describes the lab assignments for the Modern/Embedded Computer Architectures course (ET4074/IN4340) given at TU Delft starting with the academic year 2016-2017. The purpose of this lab is to let students exercise the design process of a Very Long Instruction Word (VLIW) processor and its integration into a System-on-Chip (SoC) including caches, busses and other peripherals, optimized for a selected set of applications/benchmarks. Instead of following the "cookbook" lab paradigm where the steps of this process are outlined in detail, we decided to make available to the students a parameterized VLIW core and later on a parameterized SoC platform and ask them to identify the best parameter values in order to optimize the application execution in terms of certain metrics, e.g., performance, power/energy consumption, and/or resource utilization. Subsequently, the students are asked to substantiate their design decisions through measured results from either simulation and/or after FPGA synthesis and describe their proposal and findings in a written report.

The lab comprises two parts. In the first part, the student[1] is asked to perform a design space exploration for a VLIW processor to find an optimal processor instance for two given programs. In the second part, the student will use the gained knowledge to create and evaluate an optimal SoC design for a given workload consisting of multiple programs (> 2). Note that the embedding of the VLIW processor in an SoC also means that certain SoC parameters must be determined to derive an optimal solution. In both assignments, the students must quantify and explain the reasons behind their choices.

Your designs will be implemented using the ρ-VEX platform, which contains a VHDL description of a parametrized VLIW processor, by using the parameter values you identified during the design space exploration process. In this way, your designs can be simulated, synthesized, and evaluated on an FPGA board.

1.1 Assignment 1

Goal The goal of this assignment is to perform and document a Design Space Exploration (DSE) process to determine a VLIW processor configuration opti-

[1] In the remainder of this document, the student will be addressed in a more direct form through the use the word "you".

mized for two given application according to specific metrics. The to be considered metrics are: performance and resource utilization.

Tools/platform Each student group is given two applications from the Powerstone benchmark suite and a (software) development platform. The applications from the Powerstone benchmark suite can be run stand-alone without the need for OS support or large software libraries. Within the development platform, the VEX toolchain should be utilized to exercise the DSE process.

Expected Outcome Via the given toolchain, the students must measure the figure of merit of their chosen solution and describe their findings in a report (max 4 pages). The expected content of the report can be found in the assignment description.

A detailed description of Assignment 1 can be found in Chapter 2.

1.2 Assignment 2

Goal The goal of this assignment is to perform and document a DSE process to determine an SoC platform configuration optimized for a given set of applications (called workload) according to specific metrics. The to be considered metrics are: performance, power/energy consumption, and area utilization.

Tools/platform For this assignment, you will be given a new configurable platform template that contains multiple VLIW cores. Using this template, you need to determine its configuration parameters to efficiently execute all given applications, e.g., what type of cores should be used: a single core, multiple homogeneous cores, or several heterogeneous cores. Moreover, other associated aspects of the platform must be considered at the same time, e.g., cache sizes, that impact the aforementioned metrics. You will be given a "dispatch code" as a demonstration how to distribute different tasks to different cores.

Expected Outcome After determining the best SoC configuration, you need to demonstrate that your proposed solution provides the best solution for the application set execution in terms of the chosen metrics. As you need to document the DSE process, you need to include the results of your intermediate solutions leading towards the final one.

A detailed description of Assignment 2 will be disclosed later.

1.3 Grading

The two assignments contribute to the final grade as follows:

$$Grade = 0.4 \times A1 + 0.6 \times A2$$

For each assignment grade the following evaluation criteria are in place:

- The performance of your core/platform. Note that while performance improvement is important, as it captures the capability of your solution

to faster execute the application, we also take into consideration the cost of this improvement. Therefore, higher appreciation will be given to effective solutions which provide a good balance between investment and reward.

- The technical merit of your approach. Aspects as innovation level and implementation quality are considered mostly for Assignment 2.
- The report. Report organization, content, and language are important aspects at this point.

Chapter 2

Assignment 1

2.1 Revision History

Updated 2016-09-09:

- Updated assignment description for the new virtual machine and workspace organization.

2.2 Overview

VEX "Very Long Instruction Word Example") is a system conceived by the authors of the book "Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools" (Joseph A. Fisher, Paolo Faraboschi, Clifford Young 2005). It consists of a flexible instruction-set architecture (ISA), a compiler, and a compiled simulation system. The latter two are included in the VEX toolchain that is included in the Assignment 1 archive and Virtual Machine workspace, and can also be downloaded at: `http://www.hpl.hp.com/downloads/vex`. The system is based on the HP/STMicro Lx family of processors (for example the ST200).

In Section 2.3, we present the assignment definition and state its goals. Section 2.4 describes the tools that you will be using in this assignment. Section 2.5 will show you how to compile and simulate a program using the toolchain. Section 2.6 shows how to view and interpret the simulation results. In Section 2.7, the machine configuration file is described; you can use it to change the design parameters of your processor. Section 2.8 furthermore describes how you can (optionally) tweak the compiler settings as well. Finally, Section 2.9 will describe what you need to turn in to finalize this assignment and the expected content of your report.

2.3 Definition and Goals

In the first assignment, you are asked to design an "optimal" VLIW processor for two given applications taken from the Powerstone benchmark suite. In the design, you must configure a multitude of parameters (see Section 2.7) and

perform measurements to optimize your design in terms of performance and resource utilization. It is important that you understand the interplay between these metrics for your given application. This Design Space Exploration (DSE) process must be documented in a report (max. 4 pages). The expected content is described in Section 2.9.

2.4 Explanation of Tools

2.4.1 Virtual Machine

For the assignments, we have created a Virtual Machine running OpenSUSE that you can download and run using Virtualbox (VMWare should also work). Username and password are both `user`. The root account also has password `user`. The desktop of the VM has some shortcuts on it for maintenance commands:

- **Set VPN netID** When you work from home on assignment 2, you will need to use a VPN connection to get access to the TU Delft license servers for ModelSim and ISE. This application will configure the VPN connection for you. All you need to do is enter your NetID, and then enable the VPN connection when needed as shown in Figure 2.1.
- **Pull toolchain updates** This will use `git` to update your toolchain/-workspace in case we find problems during the lab. You do not need this unless we instruct you to update through blackboard or e-mail.

If you need additional software, you can use YaST (GUI) or `zypper` (command line).

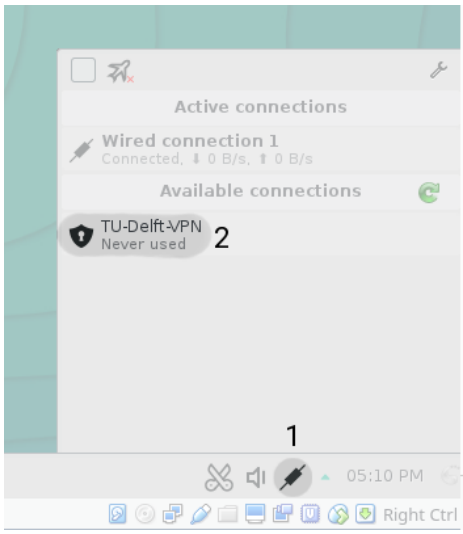

Figure 2.1: Enabling the VPN connection.

2.4.2 VEX Compiler

The compiler can target a range of different machine organizations and it can be configured by supplying a machine configuration file. In this way, the programmer can change the number of clusters, execution units, issue width, and functional unit operation latencies without having to recompile the compiler. By compiling and simulating programs using different parameters in the configuration file and analyzing the results, you can perform the DSE process.

2.4.3 VEX Simulation System

While in normal conditions the compiler produces assembly code according to the VEX Instruction Set Architecture (ISA) description in this case it will perform a cross-compilation for your host machine. This is a standard procedure for not yet completely defined processor architectures which allows you to evaluate the performance of a certain processor instance when executing a given application. As the target VEX processor is not yet defined and implemented, the compiler will generate a program that emulates the execution of the compiled benchmark on it. This means that instead of executing the code on the VEX processor you do that on your own computer. Therefore, the compiler generates an ordinary program for the host machine which performs a simulation of a VEX processor executing the compiled program. The compiler and simulator include many standard library functions, and the terminal output of the simulation is fed back to the host computer. This way, the programmer can still use functions like `printf`.

2.5 Compiling and Simulating Programs with VEX

Each group will be assigned two programs from the Powerstone benchmark suite to analyze. We will use `blit` as an example.

For your convenience, we have created a script that allows you to compile and run VEX simulations easily. Open the file manager and select `Assignment 1` in the places menu (this points to `/home/user/workspace/assignment1/`). Notice the terminal at the bottom of the file manager; if it is not there, press F4. In this terminal, type `source sourceme` and press enter. Now go into the `configurations/example-2-issue` directory. Note that you need to actually enter the directories, not just unfold them in the file tree, for the terminal to change directories as well.

In the terminal, type `run blit -O3` and press enter. The `-O3` is a flag passed to the VEX compiler, which selects optimization level 3. You should see `blit: success` appear in the terminal after some compiler warnings, and the directory `output-blit.c` should have appeared. This directory contains all the output files for the simulation that you just ran. Note that the contents of this directory will be completely overwritten when you run the `run blit` command again. This directory contains the following files.

- `a.out`: this is the x86 executable that simulates the benchmark. The `run` script calls this automatically.
- `blit.c`: this is the C source code of the benchmark, as passed to the VEX compiler. Note that this file is a *symlink*. That means that if you change this file, the original file is also modified, and vice versa.
- `blit.cs.c`: this is the C source code for `a.out`, generated by the VEX compiler to simulate `blit.c` with the given architecture.
- `blit.s`: this is the VEX assembly file corresponding to the simulation. You can (and should) analyze the assembly code with a text editor to get an idea of where the performance bottlenecks are.
- `gmon*.out`: these files contain `gprof` performance logs of the simulated benchmark taking various caches into consideration. Evaluating cache performance is beyond the scope of assignment 1, so we only need `gmon-nocache.out`. Note that these are binary files. To interpret them, run `gprof a.out gmon-nocache.out > gmon-nocache.txt` in the console and open `gmon-nocache.txt` in a text editor.
- `ta.log.001`: this is the simulation log file. Unlike the gmon files, this is already a text file. **The most important number is the number of 'execution cycles'.** The 'total cycles' also takes into account the cache performance, which, again, is beyond the scope of this assignment.
- `vex.cfg`: this is a symlink to the cache configuration file. There is no need to do anything with this file.

2.6 Analyzing Simulation Results

As mentioned in the file description, the most important metric (together with your estimation of the area utilization of a given design) is the 'execution cycles' number in the generated `ta.log.001` files. While optimizing your design though, you may want more information about why a certain design gives certain results.

The VEX toolchain has a number of tools available to this end. These tools can analyze call graphs, control flow graphs, instruction schedules, and visualize them using the VCG (Visualization of Compiler Graph) tool that is also included. Of course, you can also try to analyze the assembly files yourselves. You are advised to read the `vex.pdf` documentation (go to Documentation in the file manager) and use it to your advantage!

As an example, we will demonstrate the use of the `pcntl` program. It can be invoked as
`pcntl blit.s` from the simulation output directory. It should produce the following output (ignoring the `perl` deprecation warning at the top).

```
Procedure: blitf::
 Trace      IPC  Cycles      Oper      Copy       Nop
-------------------------------------------------------
     7     1.53      17        26         0         0
     1     1.52      23        35         0         2
```

```
    17    1.80        5        9        0        0
    16    1.80        5        9        0        0
    14    1.80        5        9        0        0
     8    1.12        8        9        0        0
     9    2.00        9       18        0        0
     4    1.52       23       35        0        2
    22    1.80        5        9        0        0
    21    2.00        5       10        0        0
    19    1.80        5        9        0        0
    11    1.25        8       10        0        0
    10    1.38        8       11        0        0
     3    1.53       15       23        0        2
    23    1.83        6       11        0        0
    20    1.83        6       11        0        0
    18    1.83        6       11        0        0
    12    1.62        8       13        0        0
    13    1.89        9       17        0        0
Operations    = 285
Instructions = 176
Reg. moves    = 0
Nops          = 6
Avg ILP = 1.61932
Procedure: main::
 Trace     IPC  Cycles    Oper    Copy     Nop
-------------------------------------------------
     1    1.09      22      24       0       1
     2    1.00       5       5       0       1
Operations    = 29
Instructions = 27
Reg. moves    = 0
Nops          = 2
Avg ILP = 1.07407
```

As we can see, this utility analyzes information for every function in the program, and every compilation trace per function with statistics. Let us look into some of the metrics stated above:

- IPC: Instructions per Cycle: The number of instructions per second for a processor can be derived by multiplying the instructions per cycle and the clock speed of the processor, indicating the performance of the processor.
- Cycles: Number of scheduled cycles
- Operations: Number of scheduled operations
- Nops: Number of "No Operations" in each trace; The final value is a sum of the total number of operations in each trace
- Average ILP: ILP is a measure of how many of the operations in a computer program can be performed simultaneously

2.7 Changing the Machine Configuration

Your goal for this assignment is to optimize the simulated processor for the given workload. The specification of the processor is given to the VEX toolchain by means of a machine configuration file. Such a file must always be named `configuration.mm` for the `run` command to recognize it, and the `run` command must be run from the directory that contains the file. Two of these machine configuration files are given to you as examples; a basic 2-way VLIW and a basic 4-way 2-cluster VLIW. You may want to make a copy of the configuration directory before you change its configuration.

Note that the VEX compiler seems to silently ignore the configuration file if there are problems with it. The overall issue width and the issue width per cluster must be at least 2, and there must be at least one resource of each type in the processor. Please read the `vex.pdf` file in the Documentation directory for other constraints and information about the machine config file.

You should not change the delay (`DEL`) values; they match the ρ-VEX implementation you will be using in assignment 2. You will see that at some point, increasing the issue width and/or number of functional units will not substantially increase the performance. This is the law of diminishing returns at work. The question you need to ask yourself is: what do I get in return (in terms of increasing performance) by investing in a larger chip (area, power consumption)? You should include one or multiple plots about this in your report.

2.8 Changing the Compiler Flags

In addition to the processor itself, the compiler and its configuration also affect performance. Any flags (the parts of a command line after a `-`, such as `-O3`) passed to the `run` script *after* the benchmark name are passed to the VEX compiler. Again, read `vex.pdf` for information about the available flags. You are also allowed to add `#pragma` statements to the benchmark C files to give more fine-grained information to the compiler if you want, as long as you do not change the actual code. Doing these things is optional.

2.9 What to turn in

In order to successfully complete this assignment you need to turn in the following:

- The machine configuration file;
- The source files for your benchmarks (if you changed them);
- A report.

The report should be maximum 4 pages and should contain at least the following:

- Description of the given benchmarks from a high-level perspective (C-Code);

- Assumptions about the environment that you think that the programs will be used. For example; a desktop computer, a mobile phone, copying machine etc. These assumptions should be used in your design decisions (a desktop machine needs to be fast, an embedded devices needs to be small and low-power);
- Discussion about the processor resources and how they impact the performance of the benchmarks;
- Your final solutions and results;
- Reflection about what you learned in this assignment.

The report must reflect the choices you made during the design space exploration process. Gather results using different configuration parameters and elaborate on them in your report. Analyze the results in your report and create a plot of the most interesting designs. Choose which design you think is the most well-balanced and explain why. The `pcntl` and related tools and the `ta.log` file present you with a lot of information. Part of the assignment is to show that you are able to identify what information is important, and how to base your design decisions on them. So do not produce graphs of all the numbers you can find, but only present the data that you think is most relevant!

Chapter 3

Assignment 2

We will be using an updated version of the ρ-VEX processor starting this year. The tools and assignment description for this have not yet been finalized. They will be made available no later than the third lab session. Note that it should not be necessary to re-download the virtual machine.