University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2017	HW2: Profiling	Wednesday, September 6

Due: Friday, September 15, 5:00PM

In this assignment, we will profile an application on the ARM core of the ZedBoard.

Importing the Application

To import the application into SDx, we have to follow these steps:

1. Download the archive from here and extract it.

2. Launch SDx, and open the workspace that you used before. You can also create a new workspace.

3. Import the code into SDx. You can find how to import code in Eclipse manual. Make sure that you import the files as *file system* in the import dialog.

4. Download the data file from here. Extract the file to the root of an SD-card. Place the SD-card in the card reader of the ZedBoard.

5. You can now build and run the code on the target as before.

Collaboration

In this assignment, you work with partners that we assigned. You can find the assignment on Canvas in the *Partners* map under the *Files* section. In the event that the partner assignment does not work out (e.g., your assigned partner has already dropped the course), contact the instructor or TA as soon as possible. Partners may share code and results and discuss analysis, but each writeup should be prepared independently. Outside the assigned groups, only sharing of tool knowledge is allowed. See the course policies on the course web page http://www.seas.upenn.edu/~ese532 for full details of our policies for this course.

Homework Submission

Your writeup should follow http://www.seas.upenn.edu/~ese532/writeup_guidelines.pdf Your writeup should include the following:

1. **Measure**

   (a) Report the latency of the application. For this, you will have to instrument the code. You can find out how to do this in the SDSoC Environment User Guide. Assume that the ARM runs at 667 MHz. Do not include the time spent on loading input and storing the output. (1 line)

   (b) Create an execution profile of the encoder using TCF profiler. Add the profile to your report. You are allowed to make a screenshot of the TCF Profiler view. Profiling is described in the same section of the user guide as instrumentation. Do not include disk I/O again.

   (c) Estimate the latency of `Filter_horizontal`. (1 line)

2. **Analyze**

   (a) Which function has the highest latency? Ignore disk I/O again. (1 line)

   (b) Assuming that the innermost loop of the first `for` statement of `Filter_horizontal` is unrolled completely, draw a DFG of the body of the loop over `X`.

   (c) Determine the critical path length of the same loop in terms of compute operations. (4 lines)

   (d) Estimate the latency of `Filter_horizontal`. Start with your critical path estimate. Assume that each operation takes one clock cycle at 667 MHz. (4 lines)

   (e) If you would apply a 2× speedup to one of the stages, which one would you choose to obtain the best overall performance? (1 line)

   (f) Use Amdahl's Law to determine the highest overall application speedup assuming you accelerate the one stage that you identified above. (1 line)

   (g) Assuming a platform that has unlimited resources and you are free to exploit associativity for mathematical operations, draw the DFG with the lowest critical path delay for the same loop body as before.

   (h) Assuming a platform that has 4 multipliers, 2 adders, and a shifter, report the resource capacity lower bound. (4 lines)

3. **Refine** As you hopefully noticed, our model did not estimate `Filter_horizontal` very accurately. Let's see whether we can improve it.

   (a) Report all the instructions in the loop body of the innermost loop. (Do not assume that it is unrolled this time.) You can see the instructions by opening the *Disassembly* view.

(b) Estimate the latency based on the number of instructions in the loop body. (1 line)

(c) What is the purpose of all the extra instructions in the loop? Note that we do not expect an explanation for each individual instruction. (3 lines)

(d) Assuming the constant offset loads complete in one cycle, estimate the average latency in cycles of the remaining load(s) together? (3 lines)

(e) How many of these non-constant-offset loads are to values not loaded in the recent past? (e.g., not loaded during this invocation of `Filter_horizontal`)? (3 lines)

(f) Assuming memory locations loaded in the recent past also take a single cycle, what is the average number of cycles for the remianing non-constant-offset memory references? (3 lines)

This gives us a model for the runtime of this filter computation:

$$T_{filter} = N_{instr} \times T_{cycle} + N_{fast-loads} \times T_{cycle} + N_{slow-loads} \times T_{slow-load} \qquad (1)$$

$N_{slow-load}$ is what you estimated in 3e and combining the cycle time with the cycle estimates from 3f, you can estimate $T_{slow-load}$. The real model is still more complicated than this, but this is a first-order model that can start helping us reason about the performance of the computation including memory access.