CS 250B: Modern Computer Systems

Lab 2: Image processing kernel in FPGA

Due: 2022-06-06

Overview

In this lab, you will implement a simple convolution kernel for edge detection, using a low-power embedded FPGA. The FPGA will receive a stream of greyscale pixels from a single image of size (512*256). Your task is to emit another stream of greyscale pixels for a processed image of same size, after performing a sweep of two convolution kernels for edge detection, and then merging them together.

The results of the computation can be seen below. The left image is the original greyscale image, and the right image is the edge-detected image your kernel should emit.

(a) Original input image

(b) Edge-detected image

Figure 1: Results of kernel processing.

Your accelerator will apply two convolution kernels, each of size 1x3 and 3x1. The two convolution filters are shown below, in Figure 2. Each stencil should sweep over the whole image, as described during the convolutional neural network portion of the course. Since the vertical convolution can only be applied to rows between row 2 and row Height-1, and the horizontal convolution for columns 2 to Width-1, let's set the edge pixels of the output image to zeros. (For more information, check out this stackexchange thread, which talks about 3x3 sobel filters: https://dsp.stackexchange.com/questions/35948/why-is-zero-padding-required-for-sobel-edge-eetection)

-1
0
1

-1	0	1

Figure 2: A 3x1 and 1x3 kernel for edge detection.

The results of these two filters will generate two separate 512x256 images. For this lab, you should simply average them together, in order to generate a single, edge-detected image.

Provided Material

First, you should clone the FPGA codebase git repository (ulx3s_bsv) as described in the FPGA toolchain introduction slides. The codebase for this lab is in "projects/image_proc". The single file you need to edit is HwMain.bsv.

In order to compile and run the project, simply run "make runsim". It will create three output files: "output.dat", which is a binary stream of 8-bit greyscale pixels of the processed image, "output.png", which is the processed image file in a png format, and "system.log" which stores the output emitted either by printf in cpp/main.cpp, or by $display in HwMain.bsv. If you have X-forwarding enabled, the makefile will also create a window to show the resulting image.

Initially, the log output will say image processing takes "131,075 cycles". Which is a good number since 256*512 = 131,072. We don't want to increase this number too much with the filter implementation.

Suggested Approach

The current HwMain implementation simply forwards the input image stream to the output. To ease development of the filters, I have already added two row buffers, rowbufferQ1 and rowbufferQ2. Your task is to implement the two filters in a manner similar to the "systolic array design for convolutions", which was introduced in the Lecture 10 slides.

There are two suggested points of importance during implementation:

First, you will have to introduce special handling for some padding pixels. Since, for example, a 3x1 filter (width of 3 and height of 1) can only be applied "width-2" times for each row, simply sweeping the filter over the whole image will result in a smaller number of pixels than the source. Same for a 1x3 filter. To remedy this, you may need to add zero-padding pixels.

Second, be mindful of the pixels offsets generated by the 1x3 and 3x1 filters. For the 1x3 filter (height of 3), the first output pixel will correspond to a pixel in the second row, first column, whereas the first pixel emitted by the 3x1 pixel will correspond to a pixel in the first row, second column.

What to Submit

You do not need to submit code.

Please submit a short documentation describing your implementation (ideally including a schematic), and the performance evaluation.

You can obtain the cycle count from the system.log output, and you can obtain the clock speed form the toolchain guide. From these numbers, you can calculate the latency of each image processing on the real hardware.