CSE P567 Homework #3 Winter 2010

Due: Tuesday, Jan 26 – Problem 1 at the beginning of class, Problem 2 in Lab

1. Read a subset of the article "**FPGA Architecture: Survey and Challenges",** by Ian Kuon, Russell Tessier and Jonathan Rose. (http://www.eecg.toronto.edu/~jayar/pubs/kuon/foundtrend08.pdf)

Read pages 138 – 139, 151 – 161, 164 (sec. 4.8) –181, 184 – 185, 211 – 216. This article is a good survey of the current state of the art of FPGA architectures. You do not have to read it all, but it is a good place to start if you are interested in FPGAs. We will be discussing the Altera FPGA architecture we are using in lecture next week, and so this is good background material. Write a short (< 1 page) summary that includes answers to the following questions:

a. Let's say you need to design an FPGA architecture. How would you go about choosing how to perform logic in the FPGA? What tradeoffs do you need to consider?

b. Almost all FPGAs have memories; what kind of memories would you provide on your FPGA and why?

c. Page 185 mentions Rent's Rule and Rent's Exponent. This is an empirically observed rule that describes how many I/Os (pins) a circuit needs based on the size of the circuit. Argue informally that Rent's Exponent must be less than 1.0, and that if Rent's Exponent is less than 0.5 then interconnect is no problem when building circuits on a plane (like a chip). You might find Stroobandt's article in the following journal interesting (and optional, of course):

http://www.nd.edu/~stjoseph/newscas/CAS_Dec00.pdf

2. This problem extends the design you implement (or at least start) in Lab 3, which used the average pixel value to produce a threshholded image. You will extend this to design an adaptive image filter that adjusts the image brightness according to the average pixel value in the image. (I should point out that this is something of a hack – see the optional problem later should you want a more challenging design problem.)

You have already designed a circuit that computes the average grayscale pixel value in Lab 3. Instead of using this to threshold the next frame, you will use it to adjust the brightness of the next frame. That is, if the average pixel value is too low, then pixel values will be increased, and vice versa. Use the following linear interpolation equation to do this:

$$\text{newPixelValue} = \begin{cases} 128 * (\text{pixelValue}/\text{average}) & \text{if pixelValue} < \text{average} \\ 128 & \text{if pixelValue} = \text{average} \\ 128 + 127*(\text{pixelValue} - \text{average})/(255 - \text{average}) & \text{if pixelValue} > \text{average} \end{cases}$$

We know that division is expensive, but since we are dividing by an 8-bit value, we can use a 256 entry table to represent the reciprocal to whatever precision we like and use multiplication instead of division.

Note we can apply this brightness compensation to each of the colors independently (which should do a version of color compensation).

Design a circuit module that has the following interface and performs the interpolation described above. We will provide a test fixture that will test your module. Depending on how you implement the equations, you may get slightly different answers than the test fixture does.

```
module interpolate(
  input [7:0] pixelIn,     // Input pixel value
  input [7:0] average,     // Average pixel value (previous frame)
  output [7:0] pixelOut    // Output pixel value
  );
```

We are also giving you a test fixture (hw3_tf.v) that you should use to test your design. We will also show you how to generate a lookup table for the reciprocals.

Note: When you insert this module in the camera pipeline (in Lab 4), you will need to extend the interface to include all three colors (input and output) and replicate the code to perform the interpolation in parallel on all three colors. However, it will be easier to test your module using a single pixel value.

Optional alternative problem

A method known as histogram equalization can be used to perform contrast enhancement on an image. See the lecture notes for a brief discussion, and the Web, e.g. Wikipedia, for more details. If you decide to implement this instead of our simpler interpolation, let us know and we can give you some advice and information about our camera pipeline that can help you. This design is a little tricky, but not too hard. (Don't try to implement parallel scan!) As for simulation test fixtures, I'm afraid you are on your own, although if there are several students that want to pursue this, sharing code is an option.