University of California, Davis

Department of Electrical and Computer Engineering

Lab 7: Image Processor Using ROM Storage

This lab specifies the design and implementation of circuits that operate on the DE10-Lite FPGA board. Details not specified in this lab should be chosen by you and stated in your lab report.

I. Prelab

Complete the following and submit your work at the beginning of your lab session.

1. Review and make sure you fully understand the Handouts: "Memories" and "M9K memories", as well as all earlier relevant handouts.

II. System Operation

This project uses a ROM built from M9K memories to store a jpeg image. The system displays the image on a VGA monitor and optionally modifies the image as described:

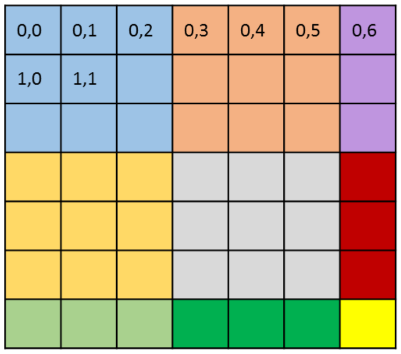
DE10-Lite switches	Parameter	Comments
`SW[9:8]`	zoom image	00: normal 1x zoom; one image pixel per screen pixel 01: zoom 2x; display size 256 × 256 10: zoom 3x; display size 384 × 384 11: zoom 4x; display size 512 × 480
`SW[7]`	Turn off red component (set to zero)	0: red normal 1: red off
`SW[6]`	Turn off green component (set to zero)	0: green normal 1: green off
`SW[5]`	Turn off blue component (set to zero)	0: blue normal 1: blue off
`SW[4]`	Display image in grayscale	0: display in normal colors 1: display in grayscale

The provided matlab script should be used as a starting point to read jpeg (or other format) images and to convert pixel data into verilog statements.

The non-zoomed image is 128 × 128 pixels and can be generated by re-sizing and cropping an image in a tool like gimp, or found on google images by selecting 128 by 128 pixels under *Tools*.

The complete image ROM of 128 × 128 = 16,384 words × 12 bits must be built from 12 independent ROMs. As a reminder, the DE10-LITE's VGA interface requires 4 bits each for red, green, and blue. If your last name begins with A–M, build the memory system from three (R, G, B) 16,384-word × 4-bit ROMs, each of which is made up of four 4096-word × 4-bit ROMs. If your last name begins with N–Z, build the memory system from four 4096-word × 12-bit ROMs each of which is made up of three (R, G, B) 4096-word × 4-bit ROMs.

When SW inputs are changed, they must take effect not immediately, but at the beginning of the next video frame. Figure 2 in the tutorial "Displaying to a VGA monitor using a combinational circuit" shows that either the rising or falling edge of *v_sync* can be used for this purpose to synchronize operation.



When images are zoomed 2×, 3×, or 4×, the image ROM must not be read 2, 3, or 4 times in each row but rather, to save power, read only once and then this pixel data is stored in a special register *pixel_buffer* which is then read 1, 2, or 3 times.

In the figure shown, the image is zoomed 3×. In this example, the ROM must be read to find the color of pixel (0,0). This value is then stored in the single-pixel *pixel buffer* and used to output pixels (0,1) and (0,2). The process repeats similarly for each group of three pixels. Each row is handled independently.

To convert the image to grayscale, display the image's *luma* on *red'*, *green'*, and *blue'* outputs equally using the equation:

$$luma = (0.2126 * red) + (0.7152 * green) + (0.0722 * blue)$$

For example, if *red*, *green*, and *blue* are represented as integers from 0–15 and are 5, 10, 13 respectively, *luma* is then 9.1536, which must be rounded to an integer. Round the result by adding 0.5 and then truncating the fractional part: (9.1536 + 0.5) = 9.6536 → *luma* = 9. So the final output is *red'* = *green'* = *blue'* = 9. Approximate the three constants in the *luma* equation using 6-bit values.

Each multiplier and each adder must be placed inside its own pipestage—that is, there is no other logic inside those pipe stages before or after the multiplier or adder. This is not saying different multipliers and adders cannot be done at the same time.

Register (with FF-registers) all inputs as soon as they enter and all outputs immediately before they leave the top level module.

III. Required Documentation

The following documentation is required and must cover the entire hardware design and all phases of operation (that have unique timing):

1) [14 pts] Detailed pipelined block diagram
2) [16 pts] Timing diagram of all key signals
3) [10 pts] Controller state diagram

IV. Testing in Simulation

[40 points] Design and write a testbench for your top-level module that exercises all functions.

Once your test bench is working correctly, demonstrate it to your TA and have it checked off.

V. Implementation and Verification on the DE10-Lite

Download your design onto the DE10-Lite board and verify it works correctly.

[120 points] Demonstrate it compiling, downloading, and operating to your TA. Your TA will record a `certutil` hash of your top-level files and this must match the hash of the files you upload to canvas so no file modifications are possible after your demo.

To properly initialize the M9K blocks, Quartus must be set up to include memory initialization data with the device's programming data (bitstream) by following these steps in Quartus before compiling:

1) Click Assignments -> Device...
2) Click Device and Pin Options...
3) Go to the Configuration tab.
4) From the Configuration Mode drop-down, select Single Uncompressed Image with Memory Initialization. (typically Quartus is not set to use memory initialization by default)

VI. Extra Credit

[20 pts maximum] Add an interesting capability to your design that has likely not been done by anyone else in the class.

Submitted Work [200 pts total]

With the exception of instructor-provided code, all work must be yours alone.

– **Prelab**

[160 pts] **Lab Checkoffs: Simulation and on FPGA board**

[40 pts] **Lab Report**

Submit all Verilog hardware and testbench code that you wrote. Do not include any code that you did not write such as files generated by Quartus or IP components.

a) [40 points] Required Documentation

b) Upload a copy to Canvas by performing the following steps by the end of your lab session—this is essential to receive credit for the entire lab.

   1. Make a folder on your computer
   2. Copy all verilog files you wrote into the folder—only the ones you wrote
   3. "zip" the folder into a single .zip file
   4. Log onto Canvas, click Assignments, find the correct lab number
   5. Upload the .zip file