CMPE 691/491 – Homework/Project #2

Fall 2014

Work individually, but I recommend working with someone in the class nearby so you can help each other when you get stuck, with consideration to the **course collaboration policy (please read it in the course website)**. Please send me email if something isn't clear and I will update the assignment. Changes are logged at the bottom of this page.

Before getting started, you should go through the verilog notes located under Course Readings on the course home page.

A paper copy of everything and electronic copies of all your code and testing files (all in one zipped file) are due at the beginning of class on the due date.

Notes:

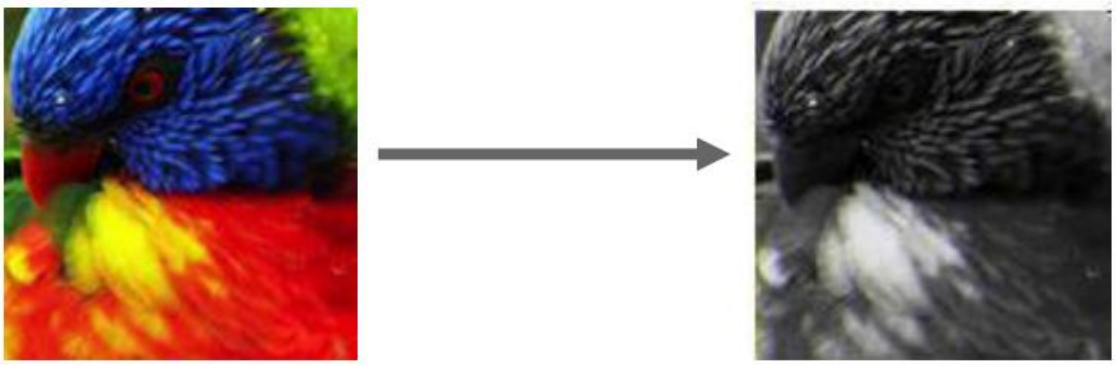
- [15% of points] Clearly state whether your design is fully functional, and state the failing sections if any exist.
- Make sure your design and code are easily readable and understandable (clear and well commented).
    - Up to 5% extra credit will be given for especially thorough, well-documented, or insightful solutions.
- *** Where three '*'s appear in the homework, perform the required test(s) and turn in a printout of either:
    1. a table printed by your verilog testbench module listing all inputs and corresponding outputs,
    2. an Isim waveform plot which clearly shows (labeled and highlighted) corresponding inputs and outputs, or
    3. a section of testing code which **clearly** compares the designed circuit and a simple reference circuit, and two short cut & paste sections of text from your simulation (one for pass, and one for fail where you purposely make a slight change to your reference code to make it fail) that look something like this:
       Error: input=0101, out_module=11110000, out_ref=11110001

  In all three options, each test case must be marked whether the output is correct or not.

  Keep "hardware" modules separate from testing code. Instantiate a copy of your processing module(s) in your testing module (the highest level module) and drive the inputs and check the outputs from there.

This project is for sending a128x128 pixel image data from PC to FPGA through UART, perform some processing on FPGA and then send data back to PC through UART.

Example of Processing:
Changing a 128x128 pixel color image data to black and white image.



The project consists of three parts:

1. Transmit to FPGA
    - Send data by traversing each row of pixels sending 8-bits for red channel, 8-bits for green channel, and 8-bits for blue channel.

2. Processing by FPGA: Convert image from colored to grayscale
3. Send image data from FPGA to PC though UART using same order.

Detail Description

1. Use MATLAB to communicate between PC and FPGA. The following matlab functions are provided:
    imageRGB2BW():
        DESCRIPTION: Converts image from RGB to grayscale in MATLAB.
        INPUTS:
            - imageFile: The location of the source image file to convert.

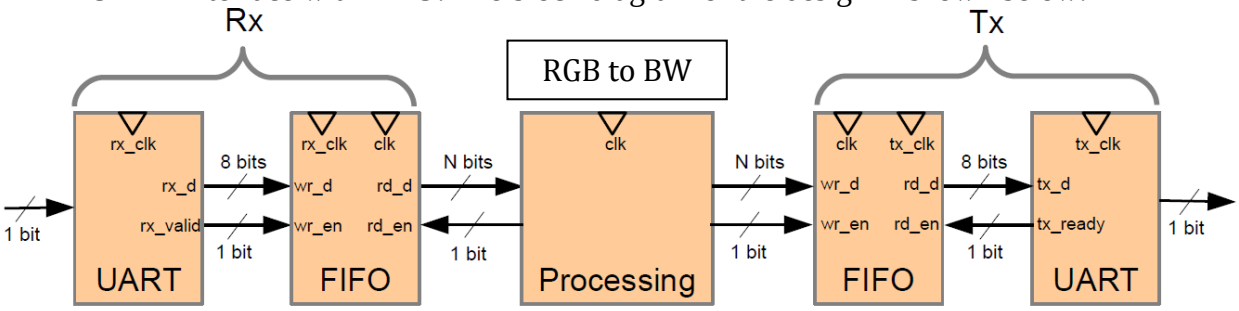
        OUTPUTS:
            - origImageData: Matrix representation of source image.
            - newImageData: Matrix representation of destination image.
    RGB2BW_FPGA():
        DESCRIPTION: Sends src image over UART. Received dest image over UART. Displays src and dest images.
        INPUTS:
            - imageFile: The location of the source image file to convert.

- PORT: The serial port (EX: ‘COM20’ , ‘/dev/tty._name_’).
- BAUD: The baud rate of the UART.

OUTPUTS:

- origImageData: Matrix representation of source image data.
- newImageData: Matrix representation of destination image data.

2. For FPGA implementation you need to design the RGB to B&W conversion and UART interface with FIFO. The block diagram of the design in shown below:



- RGB to B&W conversion, every pixel consists of three element (Red, Green and Blue) to convert to Black and White you need to replacing red, green, & blue values with this normalized sum: 0.21*Red + 0.71*Green + 0.07*Blue.

  new_pixel(i,j).red = pixel(i,j).red + pixel(i,j).green + pixel(i,j).blue;
  new_pixel(i,j).green = pixel(i,j).red + pixel(i,j).green + pixel(i,j).blue;
  new_pixel(i,j).blue = pixel(i,j).red + pixel(i,j).green + pixel(i,j).blue;

You need to implement the coefficient multiplication by the closest value tha

- For UART (universal asynchronous receiver-transmitter):
  - Download a Verilog implementation of an unbuferred UART from the internet: http://www.asic-world.com/examples/verilog/uart.html
  - Specs: 8-bit data, no parity bit, no hardware or software flow control

  Note that the verilog doesn’t have the FIFO implementation and you need to add it. But for initial test you don’t need to have the FIFO implemented.

What to return

Fully tested design with testbench and hardware demonstration.

- Verified design in simulation
  - First get the RGB-BW part in FPGA working by a simple testbench. Then test UART with a testbench separately. Finally, test UART and RGB-BW complete design with a complete testbench.

Note that each design module must be accompanied by a testbench and be verified. A testbench for the UART is provided but you need to modify it in order to make it compatible with your design module.

- Verified design in Hardware and demonstration

Use matlab and serial interface or any other serial protocol software to send data from PC to FPGA ML5 board and receive data and show the image using matlab.

Part 2 will be posted