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

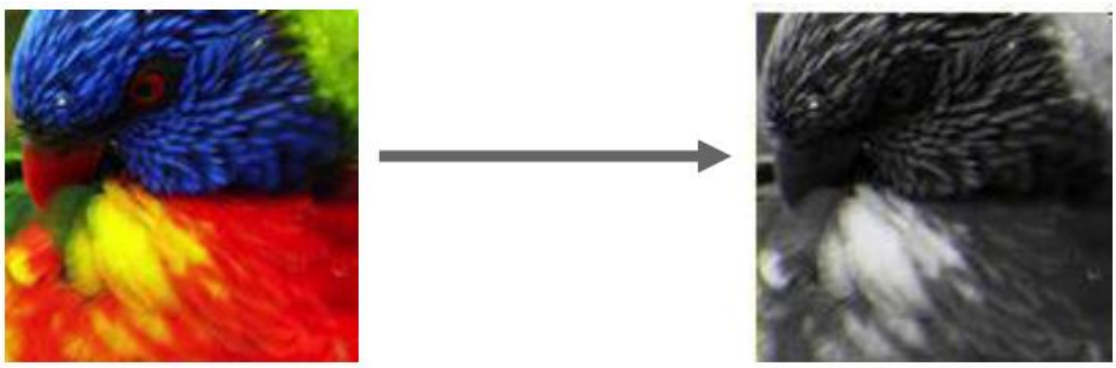
    In all three options, each test case must be marked whether the output is correct or not.

    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.

Problem 1.
[70 pt] 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:
You can choose any of these options:

a. Changing a color image data to black and white image
b. Changing the color to black and white and then rotate in hardware.
c. Changing the contrast of the image.



The project consists of three parts:

1. Transmit to FPGA
    - Send data by traversing each row of pixels, for each pixel 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 or any other example
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.

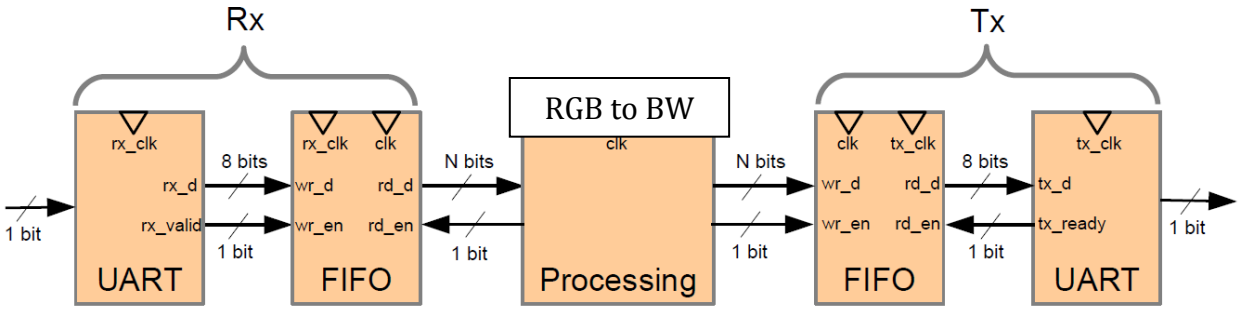
These are helpful functions that you can use:
load(input_filename) --> returns the values in the input file into a matrix
ex: input_mat = load('test.txt');

size(matrix) --> returns the size of the matrix into a matrix
ex: [row col color] = size(input_mat);

imshow(array_name) --> displays the image of the input matrix (which must be in the following format [pixel_row, pixel_col, color])
ex: imshow(input_mat);

imread(input_filename) --> returns a matrix that corresponds to the input file. The matrix will be in the following format [pixel_row, pixel_col, color]
ex: input_mat = imread('parrot128.png');

typecast(original_value, new_type) --> returns the original value converted into the new type
ex: eight_bit = typecast(16, 'uint8');



2. For processing in FPGA you have different options: shown below:

Option a: 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, and blue values with this normalized sum: 0.21*Red + 0.71*Green + 0.07*Blue.

new_pixel(i,j).red = 0.21*pixel(i,j).red +0.71* pixel(i,j).green + 0.07*pixel(i,j).blue;

new_pixel(i,j).green = 0.21*pixel(i,j).red + 0.71*pixel(i,j).green + 0.07*pixel(i,j).blue;
new_pixel(i,j).blue = 0.21*pixel(i,j).red + 0.71*pixel(i,j).green + 0.07*pixel(i,j).blue;

- You need to implement the coefficient multiplication by the closest value that gives the best results with lowest number of bits (maximum 8-bits).

Option b. RGB to B&W and Rotate. For picture rotate, you need to transpose the image matrix

Option c. To change the contrast the image. First need to store all image data, then find min and max for each color pixel then normalize it.

For example: new_pixel(i,j).red = (pixel(i,j).red-Min)/Max*255

- 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.

Problem 2.

[30 pts] Write a matlab code, which decodes a received LDPC block from channel using iterative MinSum decoding algorithm. Use the LDPC code and parity check matrix example that we discussed in class. The code is defined by a 6x12 parity check matrix that is shown below:

$$H = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

- LDPC code length=12, information length=6, column weight=2, row weight is 4.
- Assume Sfactor= 1.0.
- Write your matlab code such that you can use it for the next problem.

a) [14 Pt] Test the decoder using the received block example λ= [-9.1 4.9 -3.2 3.6 -1.4 3.1 0.3 1.6 -6.1 -2.5 -7.8 -6.8], which was discussed in class . The outputs are in Minsum_result.m file.

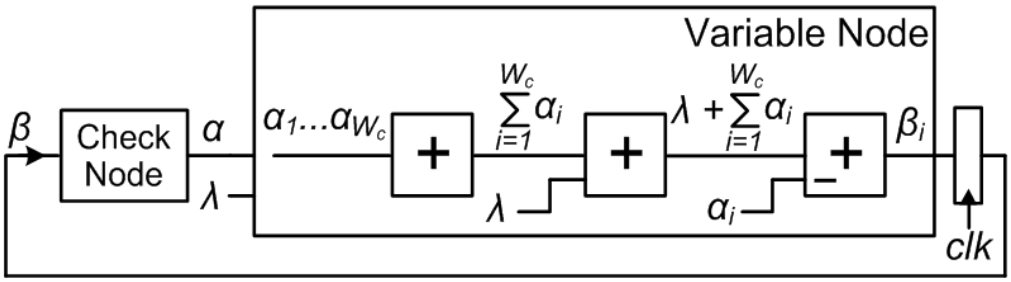
b) [16 pts] Print and submit the output of variable node processor (z, β), check node processor (α), estimation block ($\hat{v}$) and syndrome for three iterations using the received block from channel λ= [-0.5564 2.5132 -2.8639 1.7114 -0.7370 -0.0790 -1.3578 1.2827 1.9527 1.2852 -2.1138 1.5042].

c) [10 Pts] Full parallel LDPC Decoder Design: Draw complete block diagrams for LDPC decoder, check node processor, variable node processor for the H matrix given in this problem. For your block diagrams you need to provide all the details, number of inputs/outputs and labels which closely match with your matlab variable names.

1. [100 pts] Design, synthesize and implement a full-parallel LDPC decoder for the code in problem 2.

a) [10 pts] Given your λ inputs in problem 2 part b, what is the minimum word length of λ in fixedpoint? How many integer bits and fractional bits are required to cover the values within a good approximation?

b) [10 pts] Using the format of λ that you found in part a, specify the data path wordlength of each block in the decoder. As a starting point, use the block diagram below and show the wordlength of each path. For this design, assume the β values are in 3.3 format and obtain the size of other signals.



c) [10 pts] change your matlab code which can model the fixedpoint implementation of the decoder in hardware. Use the same λ inputs from Part a. Add the fixedpoint conversions for each part so your ourput signals are close to your hardware implementation results. Print and submit the output of variable node processor (z, β), check node processor (α), estimation block ($\tilde{V}$) and syndrome for 5 iterations.

d) [5 pts] Calculate the throughput of the full parallel decoder as a function of clock period. Assume there are 15 decoding iterations and the design has a single pipeline (as shown above).

e) [40 pts] Write verilog for the full-parallel decoder and test the decoder using lambda inputs that were given in Part a. Generate the output of the decoder ($\tilde{V}$) for 5 iterations using your testbench and compare the results with your matlab output.

f) [10pts]Synthesize and place and route your design.

g) [15 pts] Report maximum clk that the hardware can run and the power consumption of whole design at that clk by performing timing analysis and power analysis in ISE tool.

The project needs a full report of your design and status of the design i.e working or not working.