Lab 4

An Edge Detection Hardware with Sobel Operator

In this lab, you are required to design and implement an edge detection hardware with Sobel operator. Your hardware should apply Sobel edge detection algorithm to a 256×256 8-bit grayscale input image and store output image into a BRAM.

Sobel algorithm calculates derivatives of the image with 3×3 K_x and 3×3 K_y kernels as shown below for input image I and output image O.

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \text{ and } K_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$Sobel_x(I,i,j) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} K_x(k+1, l+1) \times IM(i+k, j+l)$$

$$Sobel_y(I,i,j) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} K_y(k+1, l+1) \times IM(i+k, j+l)$$

The Algorithm for Sobel edge detection is shown in Algorithm 1.

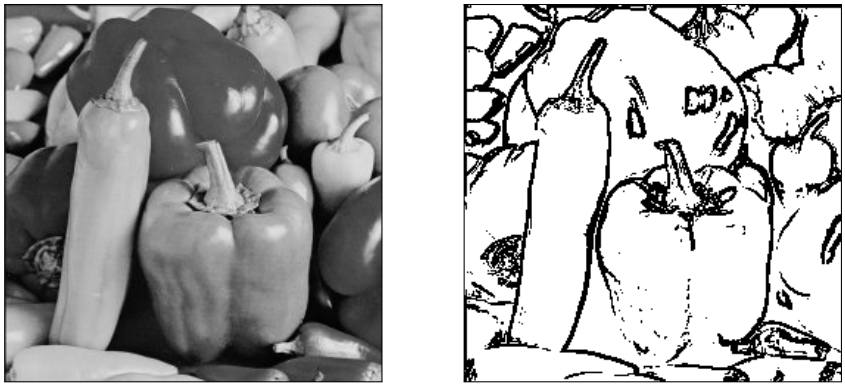
Algorithm 1 Sobel Edge Detection Algorithm for 256×256 Image

Input: 256×256 input image I
Output: 256×256 output image O

for i from 0 to 255 **do** ▷ i is for row of the image
  for j from 0 to 255 **do** ▷ j is for column of the image
    $S = |Sobel_x(I,i,j)| + |Sobel_y(I,i,j)|$
    if $S \geq 128$ **then**
      $O(i,j) = 0$
    else
      $O(i,j) = 255$
    end if
  end for
end for
return O

The Fig. 1 shows example input and output images.

Lab 4



(a) Input Image (b) Output Image

Figure 1: Input and Output Images

Sobel edge detection algorithm is applied to 3×3 windows for each pixel in 256×256 input image. Since the pixels in borders of the input image do not have neighboring pixels, your hardware should duplicate closest pixels for applying Sobel algorithm to these pixels.

You will be given five sample images (as `*.txt` files in binary format) and you are expected to produce edge detected output images. These five images and their expected outputs are shown in Fig. 4-8 on the last two pages. Pixels of these images (their `*.txt` files) will be provided to you.

Since we do not have FPGA boards present, we will not be able to use its VGA port for displaying images. Instead, you will use a Python code for visualizing your output image (and check if your hardware works correctly). Python code `txt_to_img.py` will be provided to you. (It requires `pillow` and `numpy` Python packages. You can use `Google Colab` for running the code, you do not have to have Python installed on your computer. We will provide a very simple `Google Colab` tutorial on SUCourse)

Your input image should be given pixel by pixel (where one pixel is 8-bit unsigned integer) in $256 \times 256 = 65536$ clock cycles and stored in input BRAM with 8-bit data length and depth of 65536. As soon as your input image is stored in input BRAM, your hardware should start its operation and the resulting image should be stored in output BRAM with 8-bit data length and depth of 65536. Finally, your hardware should send the output image to the output pixel by pixel in $256 \times 256 = 65536$ clock cycles. Your hardware should work with the following I/O protocol (please note that *rst* signal is not shown for simplicity):

1. Hardware accepts new inputs if *busy* signal is 0.
2. For providing input, *start* signal should be 1 for one clock cycle and input image should be entered continuously using *pixel_in* signal in the next 65536 clock cycles. Your hardware will not accept any more input until it finishes its operation with current image and outputs the result.

3. Your hardware should store input image in input BRAM and starts its operation as soon as all input image pixels are taken and stored in the BRAM.

4. As *start* signal is set as 1, *busy* signal should be set as 1 (and remains as 1 until the operation is finished.).

5. When your hardware finishes its operation (output image is generated and stored in output BRAM), *finish* signal will be 1 for a single clock cycle. Result (output image) will be at the output continuously after this clock cycle for the next 65536 clock cycles.

6. After the last pixel output is given, *busy* signal should be brought to 0 and the hardware should accept new input. An example timing diagram for input and output operations is shown in Figure 2.

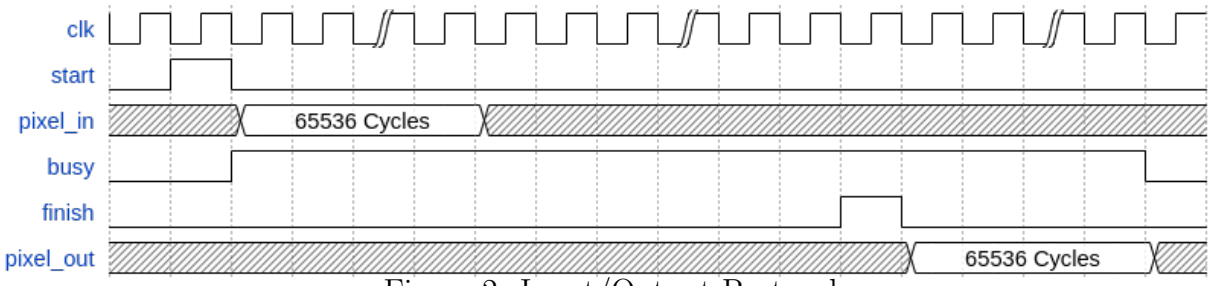


Figure 2: Input/Output Protocol

Write a behavioral Verilog model for your hardware design. In your Verilog model, your Verilog module should have the following interface:

```
module Sobel(clk,rst,start,pixel_in,
                    busy,finish,pixel_out);
    input clk,rst;
    input start;
    input [7:0] pixel_in;
    output reg busy, finish;
    output reg [7:0] pixel_out;

    // ...

endmodule
```

Although an image is 2D array of pixels, it is stored in 1D memory. Therefore, it is important how we store an image in 1D memory for operations. In this assignment, your input images (`*.txt` files) are stored as shown in Fig. 3. Each image is stored row by row. In other words, first row of your image (pixels between positions 0 and 255) are stored in the first 256 locations (0–255), the second row of your image (pixels between positions 0 and 255) are stored in the the second 256 locations (256–511) and so on. Pixel order of your output image in output BRAM should be in the same format with the input image in `*.txt` file.

Lab 4

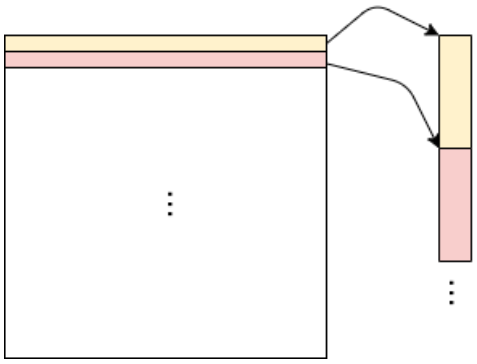
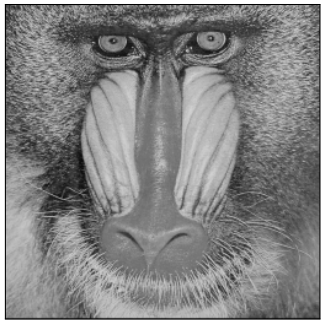


Figure 3: Order of Pixels in Memory

You should give 5 input images (their `*.txt` files) we provided as input to your hardware and obtain output images (their `*.txt` files) in the same format. You should also convert your output `*.txt` files into images using Python code we provided. After designing and verifying the correctness of your design, synthesize (not implement) your design for Artix-7 FPGA (Family:Artix 7, Device:XC7A100T, Package:CSG324, Speed:-3) using Xilinx ISE WebPack 14.7 (**You will not implement your design on an FPGA board. You will just report synthesis results.**). Then, you should write a lab report (in .pdf format) with *readable* English and good structure (dumping many figures and raw synthesis/implementation reports into a pdf file is not a lab report.) and your output images.

Finally, put all of your Verilog projects, your output images (both images and `*.txt` files) and your report into one ".zip" file named Lab3_username(s).zip (e.g. Lab3_berkea_erdinco.zip) and submit this zip file using EE310 SUCourse website. Please note that we will use a plagiarism detection software for your lab assignments.

Lab 4



(a) Input Image

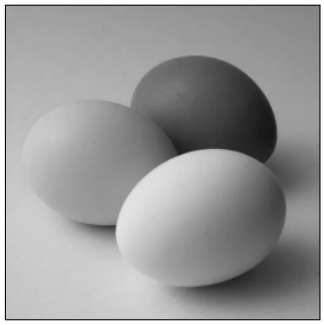
(b) Output Image

Figure 4: Baboon Image (`baboon.txt`)

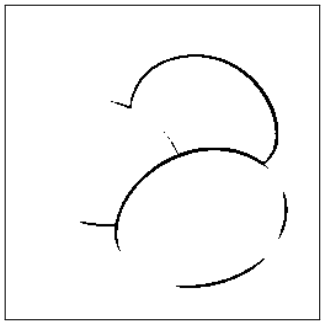
(a) Input Image

(b) Output Image

Figure 5: Cameraman Image (`cameraman.txt`)



(a) Input Image



(b) Output Image

Figure 6: Egg Image (`egg.txt`)

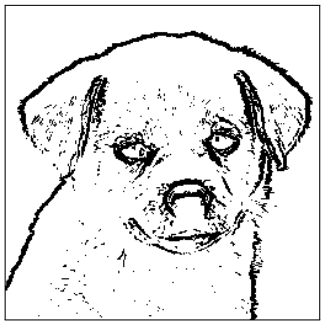
Lab 4

(a) Input Image

(b) Output Image

Figure 7: Lena Image (`lena.txt`)

(a) Input Image



(b) Output Image

Figure 8: Puppy Image (`puppy.txt`)