ECSE-487A
COMPUTER ARCHITECTURE LABORATORY
Assignment #1

To be completed individually.

1. INTRODUCTION

The objectives of this assignment are: (1) to introduce you to the VHDL synthesis and simulation tools, (2) to provide some experience in hardware design for a useful application, and (3) to recognize the tradeoffs between behavioural and structural VHDL descriptions.

Image processing involves the manipulation of graphical data, either to alter its appearance or to analyze its contents. Applications of image processing include, among others, high-speed manufacturing, criminal forensics, medical analysis, computer-assisted surgery, aircraft guidance systems, and entertainment. Given the performance demands of many of these applications, a hardware implementation is required to obtain real-time output. Thus, for this assignment, you will design, synthesize, and simulate a basic image processing toolbox using VHDL. You will construct simple modules to operate on pixels, and then use these as building blocks for higher level image processing algorithms.

2. PROBLEM DESCRIPTION

2.1 Image Processing Operation Module

The basic operations you implement will operate on a single pixel at a time. In order to apply an operation to an entire image, you will thus have to iterate over every column of every row. For example, to brighten or darken an image, the same value (PIXEL_OPERAND) may be added to or subtracted from all pixel values (PIXEL_DATA) in turn. Operations are summarized in Table 1, with a possible module (architecture) shown in Figure 1. A brief description of the input and output ports is given in Table 2.

Module	Description
SET	Set pixel to specified value
ADD	Add a value to pixel
SUB	Subtract a value from pixel
AND	AND a value to pixel
OR	OR a value with pixel
XOR	XOR a value with pixel
INVERT	Invert pixel (subtract from MAXVAL)
THRESH	Threshold pixel (output is MAXVAL if pixel is greater than specified value)

Table 1. Module name and description of basic operations.

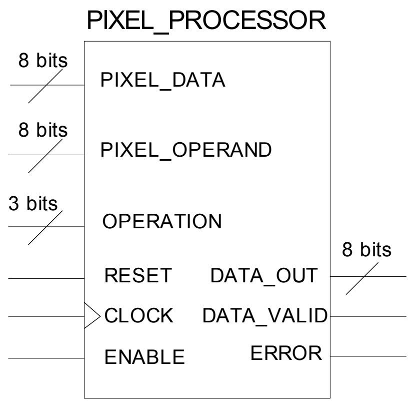

Figure 1. Logic symbol of operation module.

Port name	Width	Direction	Description
PIXEL_DATA	8 bits	Input	Data bits of image pixel value
PIXEL_OPERAND	8 bits	Input	Data bits of operand value (which may be from a second image)
OPERATION	4 bits	Input	Pre-encoded scheme to select the type of operation (ADD, …)
RESET	1 bit	Input	Reset the output when set
CLOCK	1 bit	Input	Clock input
ENABLE	1 bit	Input	Module enable when set
DATA_OUT	8 bits	Output	Data bits of output value
DATA_VALID[1]	1 bit	Output	Flag bit (see footnote)
ERROR	1 bit	Output	Set if error encountered during I/O operation

Table 2. Pinout of the PIXEL_MODULE processing block.

2.2 PGM Image File Format

We will use the portable grayscale map (PGM)[2] file format, which contains the following elements:

- A two-character "magic number" for identifying the file type: 'P5' for binary data or 'P2' for ASCII format. Your design may assume that it will only deal with the 'P2' format.
- One or more whitespace characters (space, TAB, CR, or LF).
- The image WIDTH, expressed as a decimal number in ASCII characters (e.g., 640).
- One or more whitespace characters
- The image HEIGHT, expressed as a decimal number in ASCII characters.
- One or more whitespace characters.
- The maximum gray value (MAXVAL), expressed as a decimal number in ASCII characters. Your design may assume that MAXVAL does not exceed 255.
- A single whitespace character (typically newline).
- A raster of WIDTH x HEIGHT non-negative grayscale values, represented in ASCII characters, between 0 (black) and MAXVAL (white), proceeding through the image in conventional English reading order (row by row, left to right), separated by at least one whitespace character.
- Prior to the line specifying MAXVAL, any characters beginning from '#' until the end-of-line are comments and may be ignored.

Below is a small example of a grayscale map in the PGM format:

```
P2
# feep.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Under Linux, you can view PGM files with `display` or `gimp`. On Windows, you may use `IrfanView` (available as freeware from `www.irfanview.com`).

[1] This bit makes the pixel processing unit interface insensitive to pipeline or internal processing delay. Supposing a single-stage design, the DATA_VALID bit would be : DATA_VALID <= ENABLE; (1 clock delay after the input, the output is valid). In a multi-stage pipeline, latency would make the first clock cycles invalid and thus, DATA_VALID would not become asserted until the ENABLE had time to propagate through the pipeline.

[2] Reference: Jef Poskanzer, UNIX man (5) page, 1991.

3. ASSIGNMENT TASKS

3.1 Design the hardware module(s) that read (see note) and write image files, with careful attention to error handling. You should account for any violation of the PGM file format, raising an exception under such conditions. Some possible (although by no means exhaustive) errors are as follows:

- File does not begin with magic number ‘P2’
- HEIGHT or WIDTH not specified
- Image data exceeds HEIGHT × WIDTH pixels

There are several examples that can be found on-line that demonstrate basic file I/O routines in VHDL and how to call these routines from a testbench. Students may find the material from http://tinyurl.com/okny4zu *useful. Please note that while some of the exception cases are taken into account by that example code, you are responsible for all other cases where an exception can be thrown, e.g., related to the specifics of the PGM image format.*

3.2 Design the basic image processing module as described in Section 2.1.

3.3 Use the basic image processing module from Section 3.2 to design a system that performs useful operations over an entire image, or that performs an operation using the corresponding pixels from two images.

3.4 Use the system from Section 3.3 to implement a simple vertical edge detector that operates on a background-difference image. Such a system has important applications to tracking tasks as it quickly identifies the contours of moving objects.

A horizontal gradient (change in gray level with direction) operator can be used to detect vertical edges. The gradient is formed by taking the difference between column values $J(x,y) \leftarrow I(x+1,y) - I(x,y)$, where I is the input image and J is the gradient image of I. This can be represented by a filter array [-1, 1], which operates over the input image, I. Obviously, you should not attempt to compute gradient values for the rightmost column of J, which should, instead, be filled with zeros. If I consists of grayscale values in the range of `[0,max]` then the filter may produce values in the range `[-max,max]`. These values should be normalized back to `[0,max]` by shifting and scaling. Finally, you must threshold J (with a threshold of 1) to detect any vertical edges. **Note**: Care should be taken in the normalization procedure as an 8-bit subtraction operation will produce an invalid result.

3.5 Test your VHDL modules to ensure correct operation, including handling of error conditions and special cases. In your report, include excerpts of no more than five simulation traces to convince the reader that the design is functioning correctly. Describe the important features of the simulation and annotate your traces to improve readability. Remove any irrelevant signals from the traces. All diagrams must be self-explanatory: the reader should not need to consult your VHDL code in order to understand the meaning of any signals. Provide a representative sample of test bench results as confirmation that your tests passed.

3.6 Synthesize your design without the testbench. In your report, describe the results of the synthesis, including the number of logic cells, look-up tables, and the latency of your circuit. Identify in your design the piece that consumes the most space and the longest path. Discuss what improvements could be made to decrease the size of the design or increase its speed.

4. LABORATORY REPORT

The written report should not exceed 3 pages (excluding title page, diagrams and appendices). The following outline gives an overview of what should be contained in the report. The exact content and organization of the report is left to your discretion.

- front page (course number, lab title, student name, ID, date)
- design description (Sections 3.1 – 3.4)
    - diagram and brief description of each unit
    - brief summary of any important design choices
    - list of supported features
- simulation results:
    - description of testing strategy
    - simulation traces and discussion of results (Section 3.5)
- synthesis results:
    - summary and discussion of results (Section 3.6)
- appendices:
    - VHDL source code
    - simulation traces (concentrate on selecting meaningful traces) and test vectors
    - graphic result of edge-detector applied to sample images (Section 3.4)

Your report and the appendices will be submitted only in softcopy through the course Moodle.

Although your VHDL code will not be graded directly, it is good practice to ensure that it is well documented. Each file should start with a header including the filename, author, last revision date, and a **meaningful** description. All the ports of an entity should be briefly described. Comments should be used to indicate interesting parts of the code or areas that could be improved. Use meaningful names for your variables and signals to ensure readability of your code by others who are not familiar with your design.