# Accelerating Image Processing Algorithms with Microblaze Softcore and Digilent S3 FPGA Demonstration Board

Computer Electronics

1st Semester, 2010

## 1 Introduction

This project guide introduces examples of image processing algorithms to be accelerated in the MicroBlaze (MB) softcore processor [3] for Xilinx Field Programmable Gate Array (FPGA) devices [4]. The introduced concepts of the tutorial in [1] will be used to create a system for image processing that also incorporates interfaces for an image capture camera and a VGA display, allowing for the visualization of the implemented real-time algorithms.

The system design will be accomplished employing the Xilinx ISE and Embedded Development Kit (EDK) tools [5], version 10.1.03. The implementation will be supported on the Digilent S3 starterkit board [2] which embed a Xilinx Spartan 3 FPGA (part XC3S1000-4), and two custom interfaces that connect to the S3 starterkit board through its general purpose I/O pins that allow for image capture and display.

Before going through this project guide the students are suggested to:

- have completed the tutorial in [1];
- have a comprehensive reading on the MB processor architecture and supported instructions;
- study some image processing basic routines;
- know basic concepts of C and assembly language;
- know basic concepts of VHDL hardware description language.

After the completion of this project the students are intended to know how to:

- efficiently characterize an algorithm in software and hardware components for an FPGA system;
- efficiently accelerate image processing algorithms with an MB based system.

This tutorial is organized as follows. In section 2 the background on the hardware modules, namely the image capture and display modules, is described. In section 3 the flow to obtain a complete system for image capture, processing, and display is described, based on the introductory design described in [1]. In section 4 the image processing algorithms are addressed, and different approaches for their implementation are discussed, namely different partitioning between software and hardware resources.

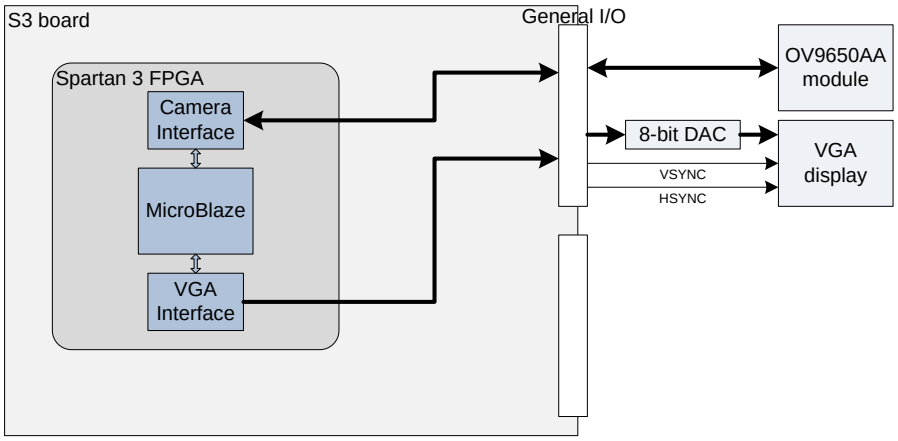



Figure 1: MB based image processing system configuration

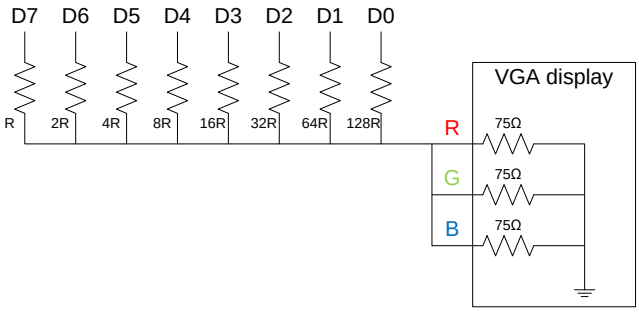



Figure 2: 8-bit DAC layout.

Table 1: Temporal properties of the VGA signals.

| Symbol | Parameter | Vertical sync [$\mu$s] | Horizontal sync[ns] |
|---|---|---|---|
| $T_S$ | Period | 16,700 | 32,000 |
| $T_{disp}$ | Display time | 15,360 | 25,600 |
| $T_{pw}$ | Sync pulse width | 64 | 3,840 |
| $T_{fp}$ | Sync front porch | 320 | 640 |
| $T_{bp}$ | Sync back porch | 928 | 1,920 |

# 2 Preliminaries

In this project guide we aim the real-time image processing. In order to evaluate this requirement, we provide two hardwired peripherals that allows for real time video acquisition and display. These peripherals connect to external video acquisition/display devices. The video acquisition device is a CMOS SXGA digital camera with 1.3 MegaPixels with a maximum spatial resolution of 1280 x 1024 with 24 bits for color. However in this project we limit the image resolution to 64 x 128 and the colors to 256 gray-levels. The video output display is a VGA computer monitor. The MB peripheral that connect to the external devices were developed in VHDL specifically for this application.

While the camera provides the output in digital format, for the VGA display a simple Digital to Analog Converter (DAC) is required to obtain the results in analog format. In Figure 1 is depicted the system configuration for the image processing application. The layout of the DAC circuit is presented in Figure 2. The VGA signals properties are presented in Figure 3 and Table 1. The schematic of the synchronization section of peripherals developed in VHDL that interface the MB softcore with the external video acquisition and display devices is presented

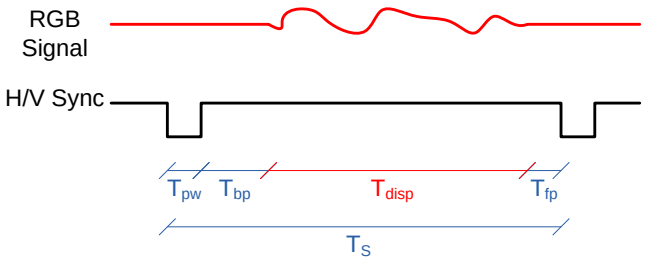
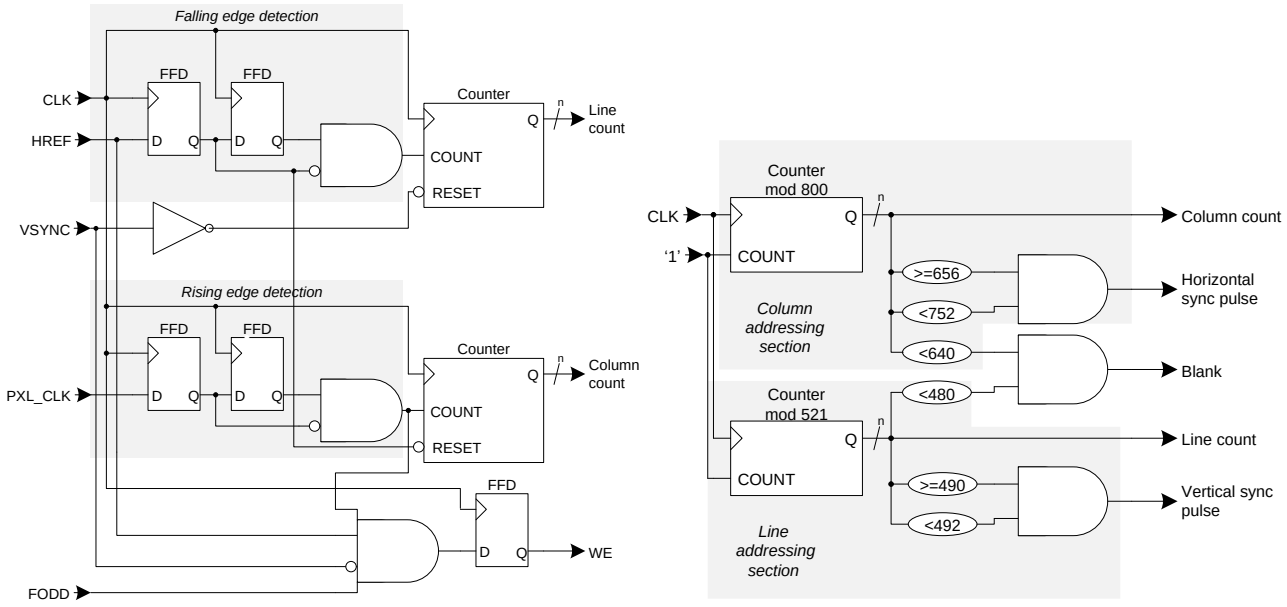



Figure 3: VGA signals properties.




(a) Input (b) Output

Figure 4: Schematic of the external video I/O devices' interfaces.

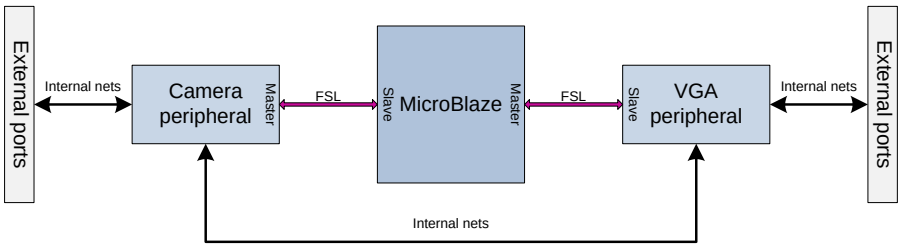



Figure 5: MB based EDK project overview for image processing.

in Figure 4.

# 3 Image processing with Microblaze

In this section we guide the design of a system, with the structure defined in Figure 1, in order to evaluate image processing algorithms. We will employ the knowledge acquired in the tutorial [1] regarding the EDK project creation and attachment of peripherals. The project to be developed (or incremented to the project in the tutorial [1] ) in the EDK environment is represented in Figure 5.

Table 2: Internal net connection information.

| Camera peripheral | | | | | VGA peripheral | | | | | External ports | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ports | Dir. | Net | Range | Class | Ports | Dir. | Net | Range | Class | Ports | Dir. | Net | Range | Class |
| clock | I | sclk | | CLK | clock | I | sclk | | CLK | clock | I | sclk2dcm | | CLK |
| c15 | I | sclk | | CLK | v07 | I | sn | | | io01 | I | si | 7:0 | |
| c14 | I | sn | | | v06 | I | sc | 7:0 | | io02 | I | sj | | |
| c13 | I | sm | | | v05 | O | sb | 12:0 | | io03 | I | sk | | |
| c12 | I | sl | | CLK | v04 | O | sa | | | io04 | I | sl | | CLK |
| c11 | I | sk | | | v03 | O | sq | 7:0 | | io05 | I | sm | | |
| c10 | I | sj | | | v02 | O | sp | | | io06 | I | sn | | RST |
| c09 | I | si | 7:0 | | v01 | O | so | | | io07 | O | so | | |
| c08 | O | sh | | | | | | | | io08 | O | sp | | |
| c07 | O | sg | | | | | | | | io09 | O | sq | 7:0 | |
| c06 | O | sf | | | | | | | | io10 | O | sf | | |
| c05 | O | se | | | | | | | | io11 | O | sd | | |
| c04 | O | sd | | | | | | | | io12 | O | se | | |
| c03 | O | sc | 7:0 | | | | | | | io13 | O | sg | | |
| c02 | I | sb | 12:0 | | | | | | | io14 | O | sh | | |
| c01 | I | sa | | | | | | | | | | | | |

**Implementing the image processing system**

a) Create a project similar to the one you created in the tutorial [1], or use the same if you already have the files.

b) Rename the reset and clock external ports and nets to the same names used in the project in the tutorial [1]. Rename the clock in 'clock_generator' to 'sclk' as well.

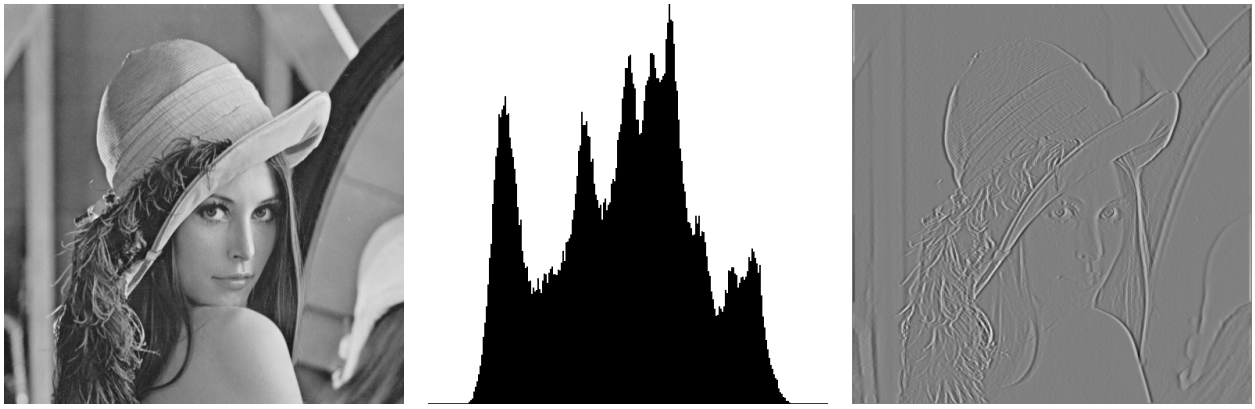
c) The peripherals for the camera ('camera_interface_v1_00_a') and VGA ('VGA_interface_v1_00_a') interfaces were provided to you with this project guide. Take a look on the VHDL files (specially the top level VHDL file) of these projects and try to identify the similarities to the project implemented in the tutorial [1], namely the FSL bus signals. Copy the folders that contain these peripherals to the 'project_path/pcores' folder.

d) Click 'Project->Rescan User Repositories'. Now you can find the camera and VGA peripherals at the end of the 'IP Catalog'. Add a camera and a VGA peripheral to your project. Also, add two FSL buses. Rename one of the FSL buses to 'camera2mb' and the other one to 'mb2vga'.

e) Configure the MB ('microblaze_0') instance to accept another FSL connection. Also, configure the MB 'Instructions' tab to use a barrel shifter, 32-bit integer multiplier, and integer divider. We do not need floating point support nor patterns comparator. We will optimize the design for area, which means that a 3-stage pipeline will be used.

f) Connect the buses according to Figure 5. Connect the clock ('sclk') and reset ('sn') nets to the FSL buses in the 'Ports' tab. Add and rename external ports and connect the internal nets between peripherals and external ports as suggested in Table 2.

g) Check the file 'mb.ucf' and copy the pin assignment and constraints related to your project's external ports to the '*.ucf' of your project. This completes the hardware configuration for your system. Hence you can generate the software libraries for your system by selecting 'Software->Generate Libraires and BSPs'.



(a) Original Image. (b) Histogram. (c) Embossing.

Figure 6: Image processing output examples.

h) Create 2 extra software projects with the names 'camera_sw' and 'camera_sw_asm'. Copy the file 'negative.c' to the 'src' folder of the 'camera_sw' project, and the file 'equalization.s' to 'src' folder of the 'camera_sw_asm'. These files contain a sample application to compute the negative of an image and equalize an image, respectively. The latter is written in assembly language using the MB instructions, hence it is an optimized implementation (check [3] for details on these instructions). Analyze these samples carefully and try to perform your own application.

i) Compile the software and obtain the bitstream to program the FPGA. Remember to select the software project you want to use in a current configuration bitstream by selecting 'Mark to initialize BRAMs'. Also, in the Compiler Options of your project set the optimization level to 'O2'.

j) Download the bitstream to the FPGA and check the results. On the FPGA there are 4 buttons. The button 2 ('BTN2') makes the camera to configure its registers, hence you must push this button otherwise you will not be able to see the captured images.

# 4 Software and hardware image processing

In this section two examples of image processing applications are thoroughly described in order to be implemented by the students using the system set in section 3. For this task, the students must be aware of basic concepts of C language, assembly (see the supported MB instructions in [3]) and VHDL. The two applications that are intended to be implemented are the histogram and image embossing. The output of these applications are depicted in Figure 6 for a 256-gray-levels image with 512x512 pixels size. In the image processing system an image size of 64x128 must be used instead, thus the required modification in the algorithms must be performed.

## 4.1 Histogram

The histogram is a graphic representation of the distribution of the gray levels of the image. It is obtained by browsing all image's pixels and counting the number of pixels in each of the possible gray-levels. The representation of the histogram is a bar like graphic, where each bar corresponds to a gray-level and the size of the bar corresponds to the number of pixels with

that gray-level. Summarizing, in order to obtain the histogram, the following procedure should be followed:

- Define and initialize to zero a memory range (variable) for the histogram, which should be an integer array with the size of the number of possible gray levels.
- Define a memory range (variable) to store the image, which should be a byte array (assuming 256 gray levels) with the size of the number of pixels in the image.
- Read the image;
- Browse each pixel of the image and increment the histogram entry that corresponds to that pixel.
- Draw the histogram picture to the image array.
- Write the histogram image to the output device.

Special attention must be payed to the histogram drawing. Note that the image width may not be exactly the number of gray levels, thus the the entries of the histogram must be replicated or collapsed, depending of the image width to be larger or smaller than the number of gray levels, respectively. Also, the image height may be smaller or larger than the maximum number of pixels with the same gray level, which may result the histogram to overflow the image boundaries or very difficult to visualize (very small bars). This means, that a scale factor should be established to dimension the histogram data to the image. Regarding this issue, two options can be taken:

- Set a fixed scale factor, which should be a compromise for the histogram bars to be easy to visualize within all the image height;
- Set a variable scale factor, such that the larger bar correspond to the entire image height and the other bars to be scaled appropriately.

## 4.2 Image embossing

When embossing an image we are interested in turning light/dark boundaries in highlights and shadows, while setting low contrast areas to gray background (middle of the gray levels range). The identification of light/dark boundaries is accomplished by computing the image first derivative. The first derivative magnitude is more pronounced in the boundaries we are interested to identify. After computing the derivative, the obtained values can be added to the background gray level in order to light or dark the boundary regions, while maintaining the low contrast areas almost unchanged. Finally, the computation result must be scaled such data the boundary pixels are scaled to fit the range $[0 : N_G - 1]$, where $N_G$ is the number of gray levels. Summarizing, in order to emboss an image, the following procedure should be followed:

- Define a memory range (variable) to store the image $I$, which should be a byte array (assuming 256 gray levels) with the size of the number of pixels in the image.
- Define a memory range (variable) to store the derivative $D$, which should be an integer array with the size of the number of pixels in the image.

- Define a memory range (variable) to store the squared convolution kernel of width $k_W = 3$, which should be an integer array with the $k_W$x$k_W$ size and initialize it with:

$$K = \begin{bmatrix} -2 & -2 & 0 \\ -2 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1}$$

- Read the image;
- Compute the derivative on one side of the boundary by computing the convolution $D = I * K$.
- Add the derivative in the other side of the boundary accumulating the convolution $D = D - I * \tilde{K}$, where the kernel element $(\tilde{K})_{i,j}$ corresponds to $-(K)_{k_w-1-j,k_w-1-i}$:

$$\tilde{K} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -6 & 2 \\ 0 & 2 & 2 \end{bmatrix} \tag{2}$$

- The values in $D$ are in the range $[D_{min} : D_{max}]$. A scale factor $s$ must be applied ($D = sD$) to assure that the values are in the range $[-N_G/2 : N_G/2 - 1]$.
- At this time the values of $D$ are around zero. However, we want the background level to be the gray level $N_G/2$. Hence, the resulting image is obtained by adding the background level $N_G/2$ to each value in $D$.

Note that the image embossing performance can be greatly improved by taking into account that the two convolution operations involved can be merged into only one. Also, the convolution kernels have several entries equal to powers of 2, thus multiplications can be efficiently replaced by shift operations.

The derivative computation can be also performed on a different direction which would result for the dark image areas to become light and vice-versa. To change the direction, the convolution kernel $K$ can be changed accordingly:

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 6 & -2 \\ 0 & -2 & -2 \end{bmatrix} \tag{3}$$

## 4.3 Implementing the algorithms

a) Most of the effort to compute the algorithms involves software development. Hence, the students are suggested to implement the proposed algorithms in sections 4.1 and 4.2 in standard C using as experimental setup a personal computer and their favorite development and debugging tools. For this task, a start point implementation is provided with this project guide in the file 'image_processing_sample.zip'. The current implementation in this file computes the negative of an image. The students should update this project with the proposed algorithms. This procedure is preferable than implementing the project directly on EDK since it involves hardware placement which is time consuming and does not enhance the debug. Note that the MB based system as constrained memory, thus the algorithms should make an efficient use of the memory (do not use unnecessary arrays, and establish the smallest possible dynamic range for the arrays' entries).

b) Once the algorithms are correctly implemented, the routines can be adapted to the image size that will be handled in the MB base system. With this, the software implementation becomes closer to the implementation to be set in the MB system. Thus, after completing the image processing algorithms the transition to our image processing system will be no more than C code porting to the EDK environment, with small modifications regarding the image reading/writing and the removal of system calls.

c) After updating the software project in the EDK environment, compile the program and load the updated bitstream to the FPGA device and check both histogram and image embossing applications.

## 4.4 Optimizing the algorithms

Despite the possible optimizations in the algorithms, other approach to enhance the system performance rely in the description of the algorithm as close to the hardware as possible. The first approach towards this goal, is the description using the MB assembly instructions. The second approach is the implementation directly in hardware using an hardware description language such as VHDL. In this project the students are suggested to implement the image embossing with assembly instructions and accelerate the histogram computation with dedicated hardware.

### 4.4.1 Image embossing with assembly

a) Change the application in the 'equalization.s' file and implement the same image embossing algorithm you programmed in C with assembly instructions.

### 4.4.2 Histogram with VHDL

a) The histogram relies in computing a temporary array containing the number of pixels with the same gray-level (the histogram) and draw a picture that represents it (bar graphic that depicts the histogram content). In this project guide it is proposed to accelerate the histogram computation by obtaining the histogram directly in hardware and drawing the bar graphic in software. In order to do this, the students have to intercept the pixel stream in the camera interface with a buffer and correspondent control logic to support the computation of the histogram with dedicated hardware. With this project guide is provided a start point to help in this task in the file 'camera_interface_histogram.vhd'. Substitute the content of the 'camera_interface.vhd' in the folder 'project_path/ pcores/ camera_interface_v1_00_a/ hdl/ vhdl' of your EDK project, with the content in the file 'camera_interface_histogram.vhd'. Check this content carefully in order to understand how does it work.

b) In the EDK project, click 'Hardware->Clean Hardware' in order to inform the tool that the hardware sources have changed. Perform the required modifications in the camera interface VHDL source and on your software in order to compute the histogram with the aforementioned partition between hardware and software. Note that the 'camera_interface_histogram.vhd' computes the histogram for 256 gray-levels while the graphical representation of the histogram must be done in 128 columns. Thus, the appropriate modifications must be per-

formed in the VHDL source and software code in order for the hardware/software partitioning to focus the computation as most as possible in the dedicated hardware.

c) Resynthesize the design and load it to the FPGA to check the results. If any syntax errors are found by the tool, they will be reported in the EDK log.

### 4.5 Further optimizations

The students are free to perform any other optimization extra this project guide. Any effort regarding the developing of a deeper knowledge on the subject covered in this project is facultative but valuable.

## 5 Writing a report

The students should write a detailed report of up to 15 pages containing the summary of the image processing system implementation and its characteristics. Also, the options followed during the development as well as the justification, mainly the options referring to the algorithms implementation, should be included. The algorithmic and implementation optimizations performed should also be reported. The software and hardware sources developed should be appended to the report and clearly identified. The students are also free to state any suggestion or comment concerning the developed work.

## References


[1] Computer Electronics Course, DEEC, IST. Microblaze Softcore and Digilent S3 FPGA Demonstration Board: Tutorial, 1st Semester, 2010.

[2] Xilinx Inc. Digilent S3 Starterkit board. http://digilentinc.com/ Data/ Products/ S3BOARD/ S3BOARD_RM.pdf.

[3] Xilinx Inc. Microblaze Reference Manual, version 10.1. http://www.xilinx.com/ support/ documentation/ sw_manuals/ mb_ref_guide.pdf.

[4] Xilinx Inc. Xilinx FPGA Documentation. http://www.xilinx.com/ support/ documentation/ index.htm.

[5] Xilinx Inc. Xilinx ISE and EDK tools. http://www.xilinx.com/ support/ download/ index.htm.