COE4DS4 Lab #4
Hardware Acceleration vs. Software Configuration

Objective

To learn how complex hardware blocks can be plugged into the system-on-a-programmable chip (SOPC) environment (Qsys) and interfaced to the NIOS II processor in order to facilitate the real-time signal processing tailored to the application at hand. Two different embedded image processing approaches will be explored in order to illustrate the importance of both hardware acceleration and software configuration.

Preparation

- Read this document and get familiarized with the source code and the in-lab experiments

Experiment 1

The aim of this experiment is to get you familiarized with a custom hardware block for handling the DE2 peripherals, needed for processing real-time video: the synchronous dynamic random access memory (SDRAM), which has 8 MB capacity, the touchpanel and liquid crystal display (LCD). It should be noted that the program memory for NIOS is stored in the static RAM (SRAM) on the DE2 board, which is 512 KB.

The custom hardware block is an LCD Component that gets data from the touch panel, controls the peripherals, as well as the image buffer from the SDRAM. The physical devices are interfaced to this LCD Component, and NIOS communicates with it only through interrupts and configuration registers. Each of these registers is accessible through a base address (defined in Qsys) and an offset documented in the *readme.component.txt* file in the top-level directory coe4ds4_lab4 (read it carefully to learn which module is used in each of the three experiments in this lab).The top-level figure is given below, explaining also the formatting of data that comes from the SDRAM and the touchpanel.

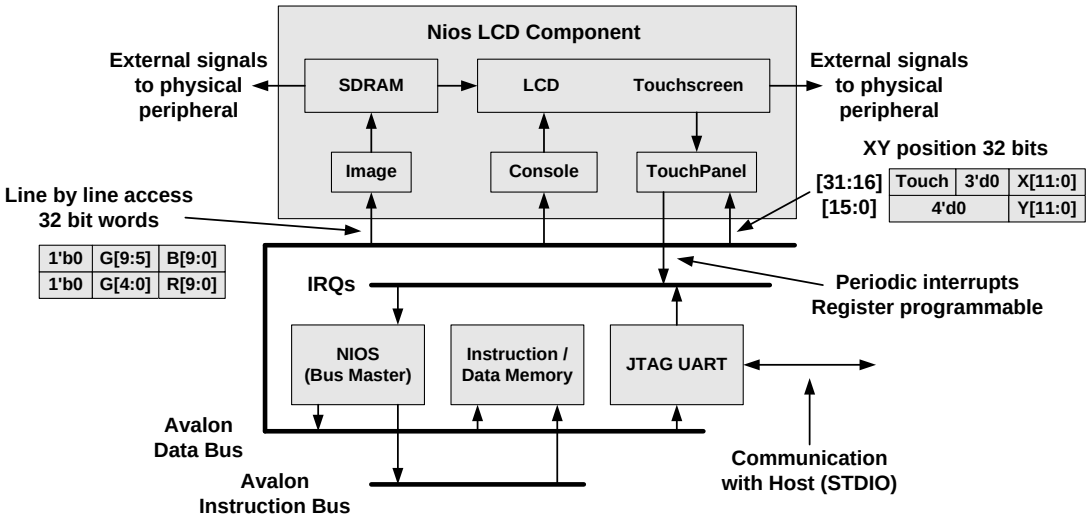

Figure 1: Three peripherals integrated into a single component connected to the NIOS processor.

You have to perform the following tasks in the lab:

- Modify the source code to implement scaling of vertical bars (as opposed to horizontal bars);
- The image area can be divided into 20x15 squares, 32x32 pixels each. Rewrite the source code to display a different color in each square, starting with RGB=000 in the top-left corner and incrementing the RGB code by 1, as you move from the left to the right. For a new row of squares, the color from the rightmost square in the previous row will be incremented.

Experiment 2

In this experiment you will process an image received from the digital camera in software and learn about its performance limitations.

An extension of the custom hardware component from the previous experiment is provided to connect the NIOS processor to the digital camera, the SDRAM frame buffers and the touchpanel/LCD. The top-level figure that shows how the different modules are connected within this component is given below. While the LCD module reads data from the SDRAM buffer, the Camera module converts the raw image data into RGB format and stores it into the SDRAM buffer. Using the provided controller, configurations such as the capture frame rate, exposure, gain of individual color and resolution of the camera can be setup through NIOS. Since the Inter-IC (I2C) bus is used for accessing the configuration registers within the digital camera, an I2C master resides inside the camera controller and it reformats the configuration data from NIOS before being sent to the camera. You can find the information about how configuration data from NIOS is sent to the camera controller using different offset addresses in the *readme.component.txt* file, as well as in the source code given in *Nios_Camera_Interface.v*. Similarly, the documentation for the other modules from the readme file can be verified by browsing the corresponding Verilog source files.

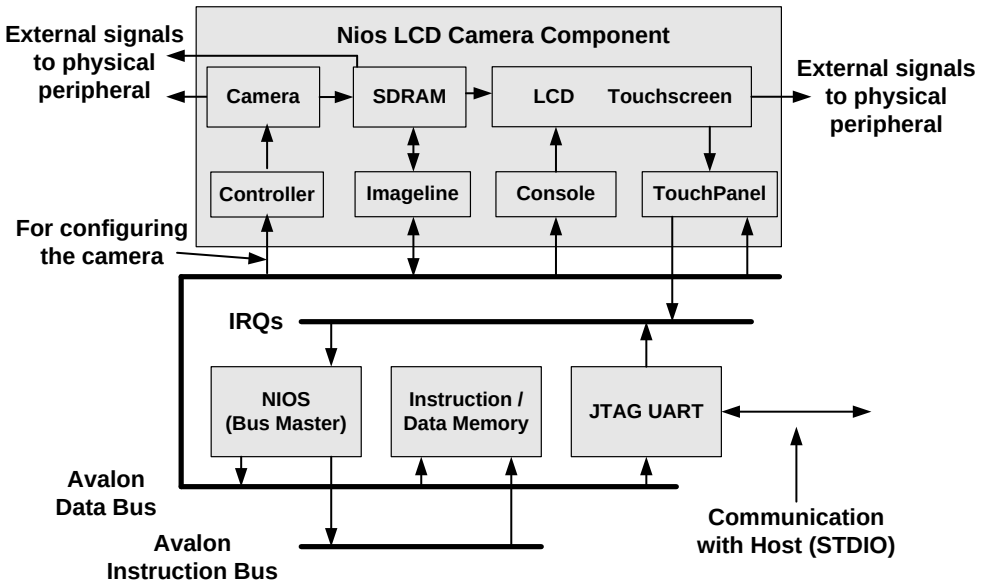

Figure 2: Four peripherals integrated into a single component connected to the NIOS processor.

Currently, there are six keys being displayed on the touchpanel/LCD console. Key 0 is used to start/stop streaming; key 1 is used to implement the color space conversion on an image stored in the SDRAM and keys 4 and 5 are used to adjust the exposure time of the camera (and hence brightness and frame rate). In the reference source code you are given functions that can read an entire line of RGB data from the SDRAM (through the custom LCD component) into the NIOS memory and write an entire line of RGB data from the NIOS memory back to the SDRAM. These functions are essential for performing embedded image processing functions on the frame buffers stored in the SDRAM. Note however, although the processing is *embedded,* as the experiment clearly shows, it is by no means *real-time*.

You have to perform the following tasks in the lab:

- When key 2 is pressed, implement a simple low-pass filter on the grayscale image. Note, the behavior of key 1 should be kept in the design. If Y[i][j] is the pixel value in row “i” and column “j” in the grayscale image, then the low-pass filter equation is Y’[i][j] = (1/4)*Y[i][j-1] + (1/2)*Y[i][j] + (1/4)*Y[i][j+1], where Y’ are the new values in the low-pass filtered image. Note, for the border conditions, the first and the last columns are replicated.

Experiment 3

In this experiment you will process an image received from the digital camera in hardware and learn about its performance advantages.

In this experiment, the data from the SDRAM is buffered on-chip, i.e., into embedded memories in the field-programmable gate array (FPGA), and the processing of images is done in hardware in *real-time*. The meaning of keys 0, 4 and 5 is the same as in *experiment2*. For this experiment, however, key 1 is used to switch between the color, color-negative, grayscale and grayscale-negative images.

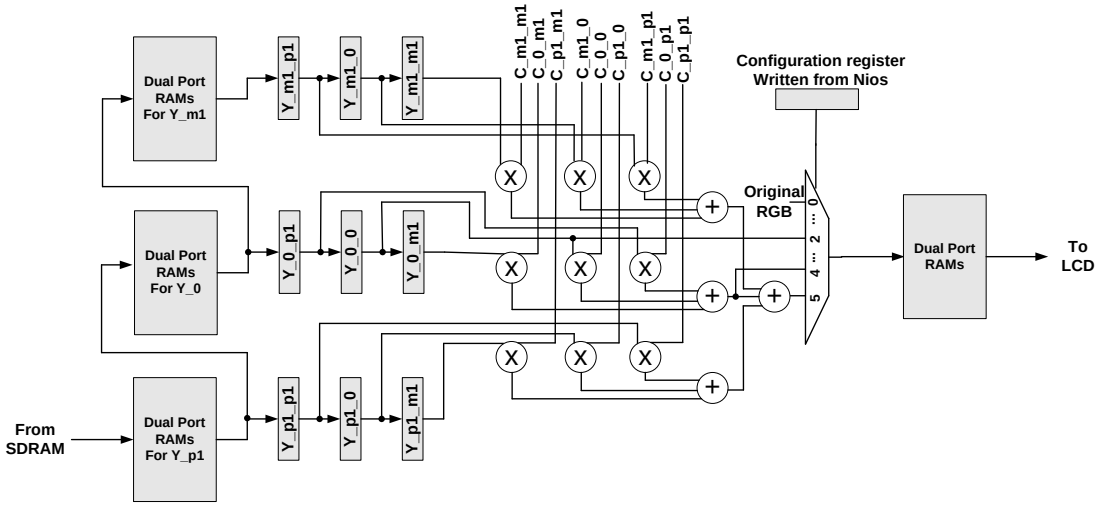

Figure 3: The filter placement between two sets of streaming dual-port RAMs.

Figure 3 shows the location of the real-time two-dimensional (2D) filtering circuitry, "insulated" from the SDRAM and LCD by Dual Port RAMs. Note that in each of the Dual Port RAM block in the figure, three embedded memories are used to buffer the RGB data and to compute the grayscale value. In each clock cycle, 3 consecutive values from the columns in the grayscale image are used to implement the one-dimensional (1D) filter; as for the 2D filter, a neighborhood of 9 values is used to compute the output. The specific action of the filtering circuitry is controlled using a configuration register accessible through NIOS. As for the notation, for each row "i" and column "j", the Y_m1_p1, Y_m1_0 and Y_m1_m1 are Y[i-1][j+1], Y[i-1][j] and Y[i-1][j-1] respectively; Y_0_p1, Y_0_0 and Y_0_m1 are Y[i][j+1], Y[i][j] and Y[i][j-1] respectively; and Y_p1_p1, Y_p1_0 and Y_p1_m1 are Y[i+1][j+1], Y[i+1][j] and Y[i+1][j-1] respectively. The registers for holding the three consecutive Y values for each of the three consecutive lines have already been implemented in the source code provided. The datapath for the 1D filter is also given, however the extension of the datapath to support 2D filtering must be done in-lab, as specified below:

- First note that when key 2 is pressed, the low-pass 1D filter from experiment 2 will be activated;
- When key 3 is pressed, implement the following 2D low-pass filter on the grayscale image: Y'[i][j]= (1/2)*Y[i][j]+(1/16)*(Y[i-1][j-1]+Y[i-1][j]+Y[i-1][j+1]+Y[i][j-1]+Y[i][j+1]+Y[i+1][j-1]+Y[i+1][j]+Y[i+1][j+1]);
- Note, the datapath shown in Figure 3 is not fully implemented in the source code; only the 1D filter and the shift registers for the Y values from the 3 consecutive columns for each of the 3 consecutive rows are given; you will need to extend the datapath to accommodate 9 multipliers and the associated adders; for each of the 9 multipliers, one of the operands will be the Y value shown in the figure; the second operand of the multipliers must be driven by coefficient signals to be defined by you with the labeling convention from Figure 3; the values for these signals will be constants in the lab experiment (as per the filter equation given above), however they will be made programmable in the take-home exercise 2;
- Note, the low-pass filter is implemented in hardware in *real-time*; hence after the Verilog file(s) has (have) been modified, the Quartus synthesis/place-and-route must be completed and the FPGA device must be re-programmed before the software configuration can be done through NIOS; note, however, a Qsys rebuild is not necessary, because the system config will not change.

Write-up Template

COE4DS4 – Lab #4 Report
Group Number
Student names and IDs
McMaster email addresses
Date

There are 2 take-home exercises that you have to complete within one week. Label the projects as exercise1 and exercise2.

Exercise 1 (3 marks) – Modify the experiment1 as follows. The display area is partitioned into 10x15 rectangles of size 64x32 pixels each. After loading the program onto the FPGA (power-up) each rectangle will have its own color, starting with RGB=000 in the top-left corner and decrementing the RGB code by 1, as you move from the left to the right. For a new row of rectangles, the color from the rightmost rectangle in the previous row will be decremented. Use the top two keys from the touch panel to move up and down from one rectangle to another. Use the next two keys from the touch panel (starting from the top) to move left and right from one rectangle to another. Note, the rectangle on which you are currently positioned is marked by displaying a small cursor of size 32x16 pixels in the middle of the rectangle. This cursor, to be made visible, must be the negative of the color of the rectangle on which you are positioned. On power-up the cursor is in the top-left corner. Note also, when you hit the borders of the display you will roll over, e.g., if you hit a rectangle in the top row and you press the move up key then you will move to the bottom row. Use the bottom two keys from the touch panel to change the color of the rectangle on which you are currently positioned. When incrementing/decrementing a color the 3-bit RGB code will be increased/decreased by 1. Note, as the color changes also the cursor color (which is the negative of the rectangle color) will also change. If a color change causes all the rectangles in a row to have the same color, then display a message in the terminal "All rectangles from row <row_id> have color <color_id>". Submit your sources and in your report write a third-of-a-page paragraph describing your reasoning.

Exercise 2 (3 marks) – Modify *experiment3* as follows. In the lab, the filter coefficients for the 2D filter were constants. In the take-home exercise, whenever you press key 3 on the touchpanel console, you will iterate through seven filtered images implemented using the following 7 sets of filter coefficients:

```
Set 0 = {1, 1, 1, 0.5, 1, 0.5, 1, 1, 1, 8}     // first 3 values are C_m1_m1, C_m1_0, C_m1_p1
Set 1 = {1, 1, 1, 1, 8, 1, 1, 1, 1, 16}        // next 3 values are C_0_m1, C_0_0, C_0_p1
Set 2 = {1, 1, 2, 2, 4, 2, 2, 1, 1, 16}        // next 3 values are C_p1_m1, C_p1_0, C_p1_p1
Set 3 = {1, -2, 1, -2, 5, -2, 1, -2, 1, 1}     // last value is the scale factor which is
Set 4 = {-1, -1, -1, -1, 9, -1, -1, -1, -1, 1} // used to normalize the weighted sum
Set 5 = {0, -1, 0, -1, 5, -1, 0, -1, 0, 1}     // for example, the in-lab filter is Set 1
Set 6 = {-1, -2, -1, -2, 28, -2, -1, -2, -1, 16}
```

It is essential to note that all the coefficient values, as well as the scale factor, must be programmable through NIOS. This implies that the onus is on you to modify both the Verilog and the C code, in order to provide a new set of registers that can be accessed from NIOS using `IOWR(…);` one suggestion is to include these registers as offset registers in the ImageLine interface and to access them in the same way as the filter config is currently loaded, i.e. `IOWR(NIOS_LCD_CAMERA_COMPONENT_0_IMAGELINE_BASE, 4, config);` needless to say, the value `4` from the above statement stands for the offset register `4` used to load the filter config word (see *readme.component.txt* and *Nios_Imageline_Interface.v*); you obviously must define your own offset registers, and then modify both the software and hardware accordingly. Submit your sources and in your report write a third-of-a-page paragraph describing your reasoning.

VERY IMPORTANT NOTE:
This lab has a weight of 6% of your final grade. The report has no value without the source files, where requested. Your report must be in "pdf" format and together with the requested source files it should be included in a directory called "coe4ds4_group_xx_takehome4" (where xx is your group number). Archive this directory (in "zip" format) and upload it through Avenue to Learn before noon one week after the day you have done the in-lab experiments for lab 4. Late submissions will be penalized.