Πανεπιστήμιο Κύπρου
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών
Ηλεκτρονικών Υπολογιστών

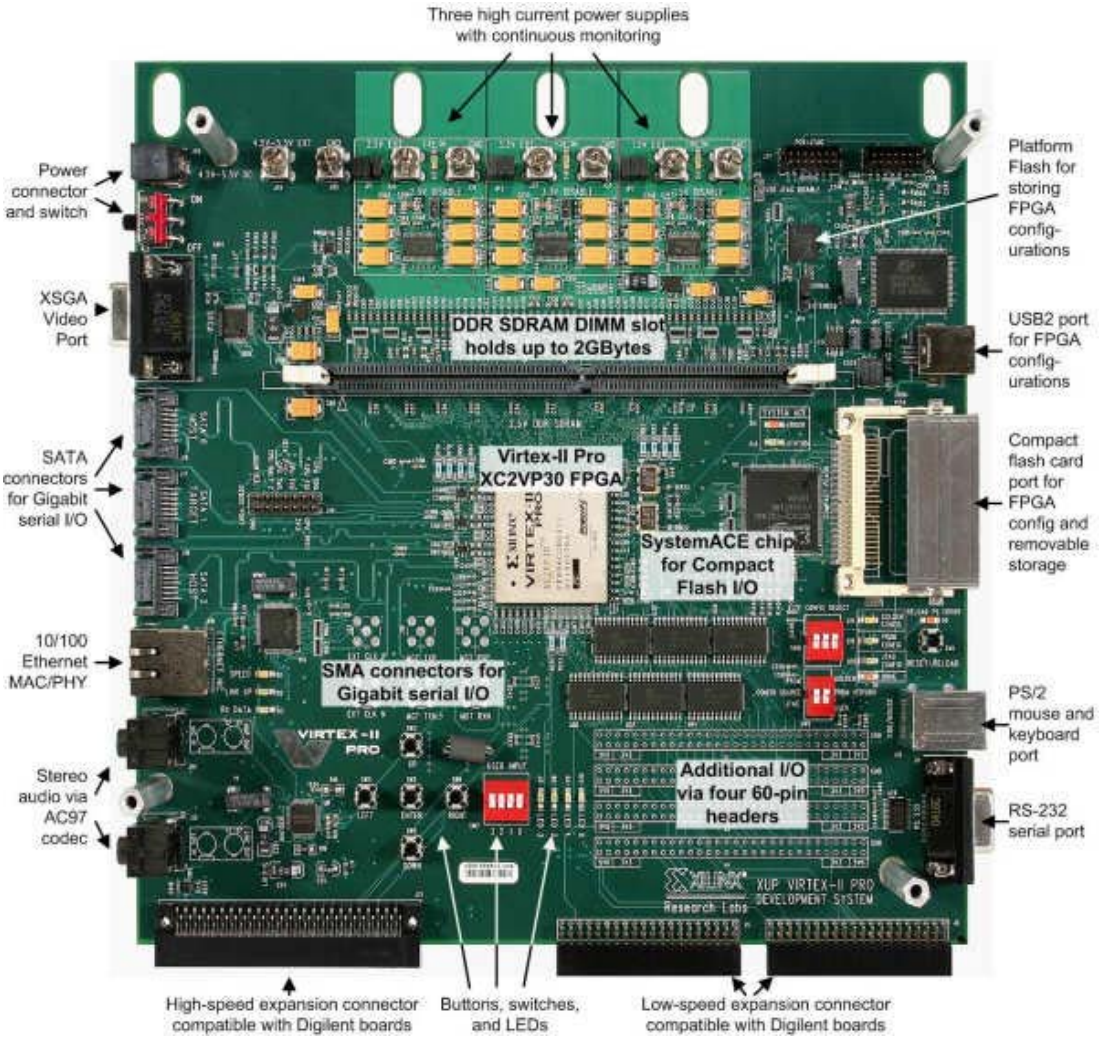
HMY 664
Digital Design with FPGAs

Lab Assignment 4 – Memory and I/O

12 Οκτωβρίου 2010

Due Date: October 25th, 2010

The programmable logic boards used for ECE 408/664 are Xilinx Virtex-II Pro development systems. The centerpiece of the board is a Virtex-II Pro XC2VP30 FPGA (field-progammable gate array), which can be programmed via a USB cable or compact flash card. The board also features PS/2, serial, Ethernet, stereo audio and XSGA video ports, user buttons, switches and LEDS, and expansion ports for connecting to other boards.



The Xilinx Virtex-II Pro Development System.

To see some of the features in action, you can run the board's built-in self test.

<u>Caution!</u>

The boards contain many exposed components that are sensitive to static electricity. Before touching the boards, try to remember to discharge any static electricity you may have built up by touching a grounded piece of metal (i.e. part of the desk). Especially remember to do this after you have been walking around the room on a carpeted floor (please keep your shoes on).

Part 1: Memory Controller Design (5 points)

In this lab, you are requested to design and implement a memory controller, to read and write data from an on-chip RAM that you will also create.

The memory controller works by receiving request signals from a component, along with an associated address. The controller serves two types of requests, read and write. The memory controller also interfaces to the world with a data bus, which contains either data that is to be written on memory (when a write request is issued) or data that the memory will deliver in the event of a read request. **You are to design your own memory controller**.

The memory controller should operate at least at 100Mhz (System Clock), and your data bus should be 48 bits wide. You may use Core Generator to generate any IP cores that you deem essential towards your design. The I/O interface to the controller consists of an address field (32 bits), the data field (48 bits), and the handshake signals you deem necessary for your assignment to work. You are free to design either a fully synchronous controller or a fully asynchronous controller, or even a hybrid version; however you need to show your work completely and in clear manner to receive full credit. Recall that a memory address can be partitioned into Row-Column bits, as explained in the undergraduate computer organization courses. An address field is separated into two sub-fields; the row selector and the column selector. For your assignment purposes, you need to be able to address **at least 2^{32} Bytes** of memory.

In order to test your controller, you need to write an ON-BOARD testbench; a design that reads and writes to a memory, checking that the data written is the data received. In contrast to previous testbenches, your testbench needs to be synthesizable so that it can be loaded to the FPGA itself. Therefore, you need to create a memory unit as well. You can generate on-chip ram using Xilinx's Core Generator – we will learn how in the lectures.

Start by designing a counter that generates memory addresses. Use a test value of interchangeable 1's and 0's (i.e. 01010101…….01) to first write in even memory locations, and reverse the order of 1's and 0's for the odd locations (i.e. 101010……10). When the addressable memory is full, you need to start reading the values. You can either read from all memory locations, but that will be time consuming, hence you can choose randomly the addresses to read from. Consequently, you need to compare both odd and even addresses, to verify that data was written correctly in memory. You may use the LED's onboard for testing purposes. A significant portion of your assignment is your testing strategy. You have to document your testing strategy very well, and explain why you chose to test your controller in the manner you tested it, and how the test proceeded. Document these steps in your logbook, as you will report on them later on. For full credit, you need to show to your instructor your complete testing strategy and be prepared to answer questions about your choices.

Part 2: Memory Application (5 points)

In this part, you will put your memory controller to use. You will instantiate an FPGA memory block with data given to you individually from your instructor (when you complete PART A you need to email your instructor and he will email you your test data), and you are to use that data to utilize your microcontroller. You will first implement a simple matrix multiplication. You will be given a 4x100 matrix consisting of 8-bit positive integer values by your instructor/TA once you report that you completed Part 1. You are to instantiate the matrix in a READ-ONLY ROM memory block on your FPGA using Core Generator. Using those values, you are going to first transpose the matrix and store the Transposed matrix in the destination RAM. Recall that the **transpose** of a matrix A is another matrix A^{T} (also written A^{tr}, ${}^{\mathrm{t}}A$, or A') created by any one of the following equivalent actions:

- write the rows of A as the columns of A^{T}
- write the columns of A as the rows of A^{T}
- reflect A by its main diagonal (which starts from the top left) to obtain A^{T}

Formally, the transpose of an $m \times n$ matrix A is the $n \times m$ matrix

$$A^{\mathrm{T}}_{ij} = A_{ji} \text{ for } 1 \leq i \leq n, 1 \leq j \leq m.$$

If you stored the matrix in a raster scan fashion (i.e. from left to right, row-wise) in your ROM, you need to design a controller which reads data from the ROM in that fashion, and determines based on the location of the incoming value, the address which needs to be activated to store the transposed value in the destination RAM memory. At this point, your memory controller can be used to write the data in the destination RAM. When you have successfully completed the transpose operation, you have two matrices; the original stored in the ROM, and the Transpose, stored in the DRAM. Multiply the two, by bringing data from the memory and by reading the data from the ROM. Recall that when you multiply a 4x100 matrix with a 100x4, the result is a 4x4 matrix. The resulting 4x4 matrix can be stored on the FPGA, in a new RAM location defined by you. When the result is calculated, you are to turn on all four LEDs on the board, and have them on for 5 seconds. You are required to use any bits necessary for each matrix value, so take into consideration the possibility of having to add a sequence of several 16-bit values that result from each multiplication.

You are to design a small system to shift the output sixteen 16-bit values (**note that in case an individual value is more than 16-bits long, the low-order bits can be ignored in THIS step of this Part**) on the board LEDs, in groups of four, based on the input combination that the user provides to the system using the 4 switches. For example, if the user wishes to see the value #14 (see below), then the LEDs start to display the value in groups of four bits at a time (from LSB to MSB), every 1 second. Hence, to display a 1001 0011 1010 0010, the first value displayed would be 0010, the second 1010, etc. When each value finishes, the LEDs go back to stay on for five seconds.
Location Chart:

0	***1***	***2***	***3***
4	***5***	***6***	***7***
8	***9***	***10***	***11***
12	***13***	***14***	***15***

Hence, to display location 0, the user would need to input 0000, and so on. When the five seconds pass, and the user does not change the input, the LEDs display the location that the switches are found. Note that in all cases, when the 5 seconds expire, the value displayed is for the location that the switches are set at that very moment. You will receive more information in class for this part.

Part 3: I/O Application (15 points)

A. VGA Controller Implementation (7 points)

In this part, you will design and implement a VGA controller to be used in the lab. A VGA controller receives as inputs from the system the video signal and system clock, and generates the Red/Green/Blue signals and the two synchronization signals, Horizontal and Vertical, which are sent to a VGA Monitor. A VGA Monitor can be thought of as a grid of pixels where each pixel is a picture element that can be set to a specific color. There are 480 rows and each row consists of 640 pixels. The VGA interface works serially, that is, color information for each respective pixel is sent one after the other, as opposed to all at once. The VGA timings are standardized, and are provided here for your reference:

VGA Timings

The following table lists timing values for several popular resolutions.

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

Source: Rick Ballantyne, Xilinx Inc.

We will be using the first row parameters for this assignment, which implies that your Pixel clock needs to be at 25.175Mhz (which you will generate from the system clock of 100Mhz) and the resolution will be at 640 Columns x 480 Rows. You can use the Digital Clock Management modules (DCM) provided with the Virtex II Pro FPGA to accurately you're your signals. Colors can be represented using a triplet consisting of the intensities of each fundamental color (Red, Green, Blue). The monitor expects these values to be analog, and thus a DAC (Digital to Analog Converter) is used. Depending on the type of monitor and video card you use at home, your computer uses 16, 24, or 32 bits to encode this color information. For the assignment purposes, we will assume that we will be using 8 bits per color component of each pixel; hence the pixel will be total of 24 bits (8 for Red, 8 for Blue, and 8 for Green).

An important fact to realize is that the VGA monitor does not have memory and thus will not store the pixel information being written to it. Instead, the pixels must be continuously sent to the display to achieve a stable image. The VGA controller needs to be receiving from the memory pixels and will be responsible for constantly sending out the pixel information. There are two signals that are used to maintain synchronization with the monitor: horizontal and vertical sync. These signals are active low (activated when the signal is logic 0). The timing for these signals is defined by the standards for VGA monitors.

During a horizontal sync cycle, the pixel information is sent for all row (in our case 640) pixels followed by the lowering of the horizontal sync signal. The signal is used to coordinate the start of new lines.

A vertical sync is achieved by lowering the vertical sync signal. The vertical sync instructs the monitor to return to the top of the screen (0,0). The horizontal sync cycle that follows becomes the first row on the screen.

To familiarize yourself with VGA monitors, download and read the "VGA Adapter" reference article from the assignment website. **I will provide a VGA tutorial in the class on Tuesday October 19th.** There are also a couple of really handy websites that talk in detail about VGA adapters, and all provide code (both VHDL and Verilog) snippets which you may use in your assignment provided you give credit to the authors:

http://www.eecg.utoronto.ca/~jayar/ece241_06F/vga/
http://www.cs.unc.edu/~stewart/comp290-ghw/vga.html
http://www.stanford.edu/class/ee183/vga.shtml
http://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml

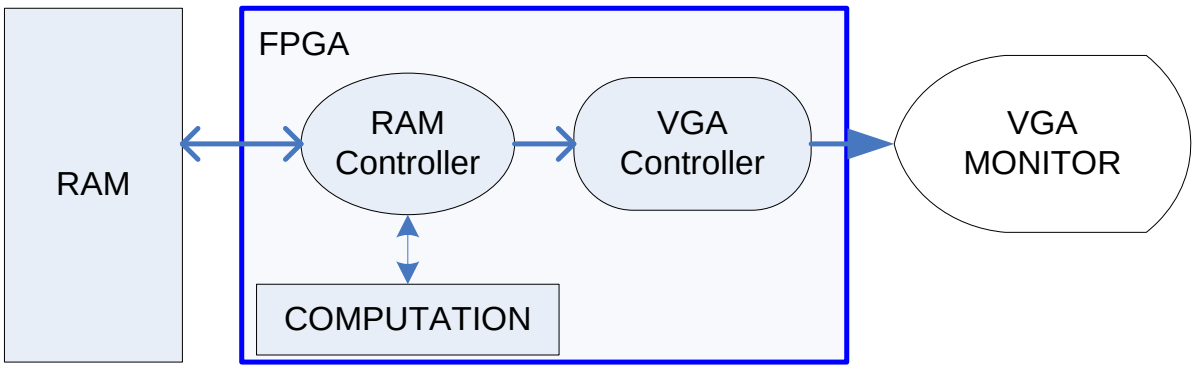
The controller must take the pixels and the system clock as inputs from the FPGA, and generate the VGA output signals to drive a common VGA interface (i.e. the LCD monitors in the lab).

B. Video Display (8 points)

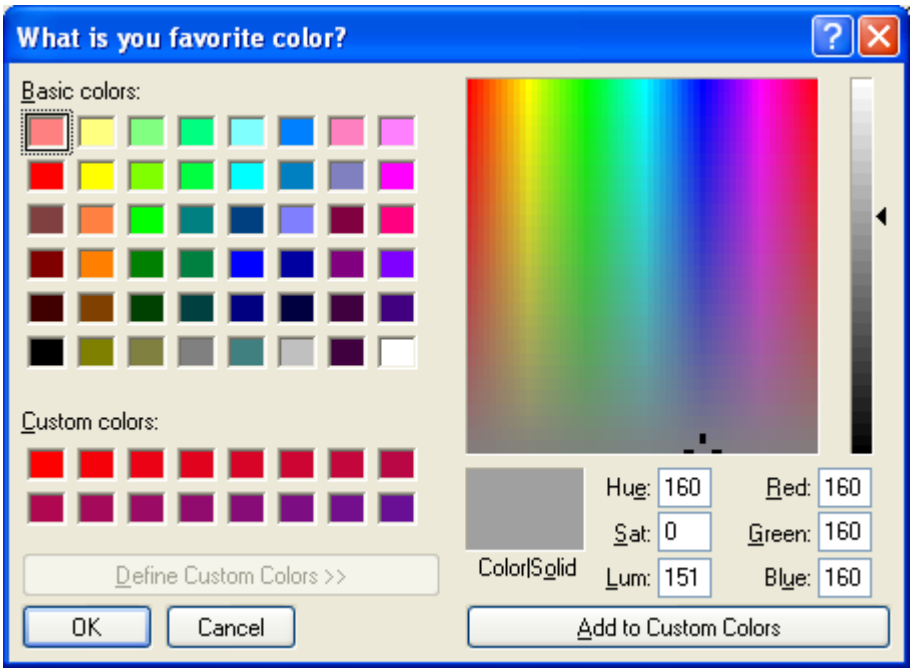
In this part, you will use the RAM memory controller from Part A and the VGA controller together to output data from a memory to the VGA output. Start by generating a randomized 320x240 array of 24 bits per value. Each value represents a pixel, but in this part, we are not

interested so much to the value, as getting our design to work. You can play with the Windows Color Scheme on any Microsoft Windows machine to perhaps get some ideas for interesting colors. Your array then needs to be stored in the RAM (use the controller from Part A) and then can be displayed with your VGA controller in a loop mode, by constantly incrementing the numbers in your array by a constant number (i.e. when the picture ends, you can loop back again to maintain a constant video output, and add one more intensity value to your array). You will be given more details about this in the lab.

The overall diagram of your system is shown below:



The Windows color scheme (see below) can be used to give you suggested values for your testing. Implement the integrated design and using a start button (place it in one of your switches) and see the output on the VGA. You can start by placing a single color in the entire picture, and then you can split the picture in two colors, three colors, and so on.



Extra Credit (1-5 points)

The top three designs in terms of the speed that the system operates, receive 5, 3 and 1 points extra credit. In order to speed-up your design, you can devise several tricks. You may explore pipelining, Core Generator parameters, etc. You can also map FPGA resources to the system, such as the multiplication to the on-chip multipliers, etc. Note that your system also contains several design exploration opportunities in the memory controller domain – you have 64 bits in the databus, which you are asked to utilize wisely for speedup improvements. To EARN those points you must indicate CLEARLY the speedup contributions of your design. Pure luck will not be considered "extra credit".

WARNING – If there are two or more designs that have the same speed, and are within the top three, there will be NO extra credit. In other words, I want to see three CLEAR winners to give them points.

Turn-In Instructions

You are to submit a test strategy report for your Memory Controller testing strategy. The report should indicate clearly what testing methodologies you have chosen, and why they were selected, how they were applied and how you ensured that your design was functionally correct.

To turn your lab to your instructor, please use compression software such as Winzip or Winrar and put your project report and all your design files (including board reference designs such as UCF files) in a Zip file. Email the zip file to your instructor by October 25th, 2010

References

Digilent Virtex-II Pro Development system

XUP Virtex-II Pro User Guide

Xilinx ISE Online Help

Xilinx iMPACT Online Help