

Laboratory V

LCD Framebuffer

Introduction

In this laboratory you will explore different ways of creating video images. There are four projects. In the first one you will create an image “on the fly” using PAL macros to tell your code which pixel is about to be drawn. The second one will examine the values of some of the parameters that will affect the next two projects. The third one will create an image on the fly, but will perform its own synchronization with the video refresh cycle. The last project will create an image by reading pixel values from RAM in real time, thus implementing a classic Framebuffer design. All four projects will be written so they operate both in simulation and as FPGA configurations.

Lab Activities

1. Draw an Image Using PAL Macros for Synchronization
2. Display Video Parameter Values
3. Draw a Test Pattern by Synchronizing with HBlank
4. Draw an Image From a Framebuffer
5. Submit a Report of Your Lab Activities

Draw an Image Using PAL Macros for Synchronization

Create a workspace named “Laboratory V” and create a project in it named “SevenSeg Display.” Delete all configurations for the project except Debug and EDIF. Set up those configurations in the usual way. You won’t need anything but the standard header files and libraries for this project. You won’t need any conditional compilation in your code for this project, or any of the other projects in this Laboratory, either.

For this project you are to write a Handel-C program named *sevenseg.hcc* that reads from the keyboard and writes their ASCII codes, in hexadecimal, on two seven segment displays. The special feature about this project is that you are to draw the seven segment displays on the LCD screen instead of using real seven segment displays.

One thread is to read characters and to use their hex values to assign values to two seven-bit variables that tell which segments to illuminate. You can do this by table lookup from a ROM: `rom unsigned 7 decoder[16] = { 0b1111110, 0b0110000, ... };` The idea is that you can do two table lookups from *decoder* to determine what segments to turn on or off each time you read a character from the keyboard.

The heart of the project is to code an endless loop that continuously determines the coordinates being refreshed on the LCD and decides what color to display there. The macros, *PalVideoOutGetX(handle)* and *PalVideoOutGetY(handle)* return the current X and Y coordinate values, and you have to decide what color to draw (using *PalVideoOutWrite(handle, pixel)*) on the next clock cycle. You can do this with one big *if* statement that tests if the coordinates are inside one of the currently “on” segments of either seven segment display or not.

LCD Framebuffer

You will need to make the sides of each segment vertical or horizontal to make it easy to test whether a point is inside a particular segment or not. With proper macro expressions defined, your *if* statement could include code like, “ ... ((left_seg_a == 1) && (x > left_seg_a_left) && (x < left_seg_a_right) && (y > left_seg_a_top) && y < (left_seg_a_bottom)) || ... ”

Be sure your code works correctly both for simulation and on the RC200E.

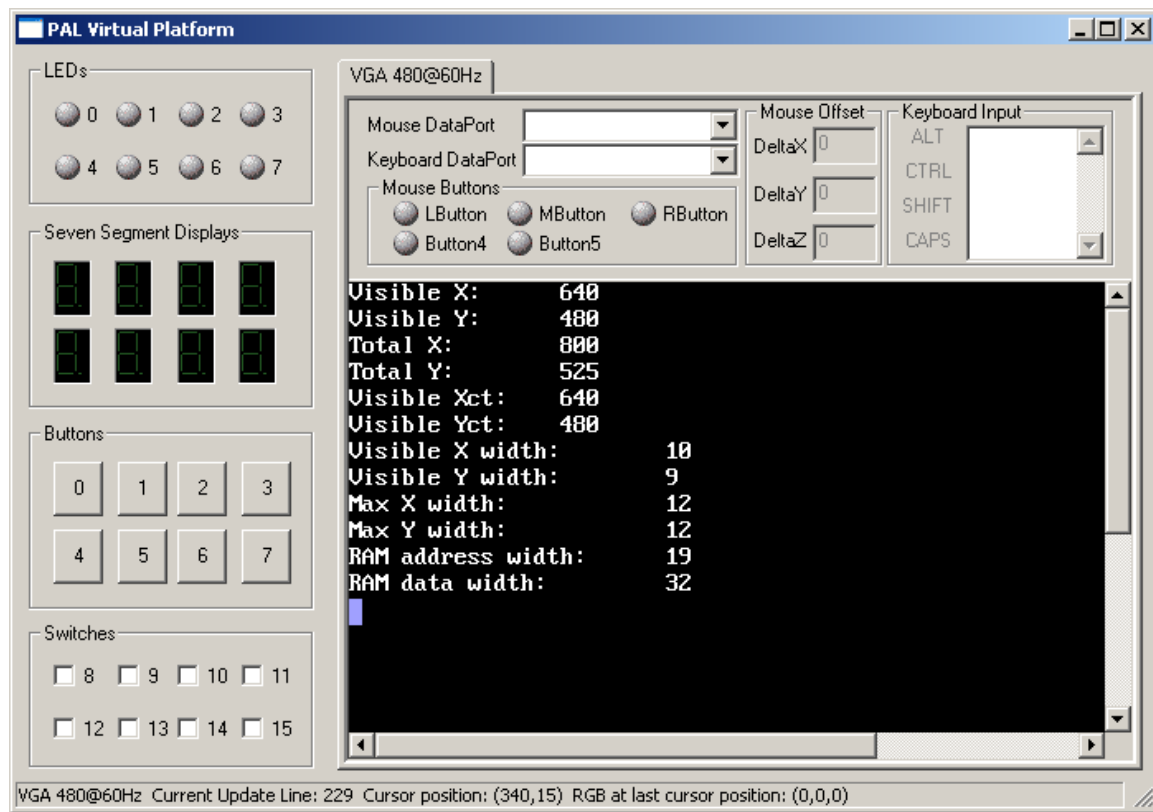
Display Video Parameter Values

Create a second project called Parameters and configure it for EDIF and for Simulation. You will need to include the *pal_console.hch* header file in your source code and the *pal_console.hcl* library file in your Linker list. Write a Handel-C program called *parameters.hcc* that displays the following parameters on the console at run time:

- The number of visible pixels per scan line, determined at compile time.
- The number of visible scan lines, determined at compile time.
- The number of visible pixels per scan line, determined at run time.
- The number of visible scan lines, determined at run time.
- The total number of pixels per scan line.
- The total number of scan lines.
- The width (number of bits) needed for a variable that holds the X coordinate of a visible pixel.
- The width needed for a variable that holds the Y coordinate of a visible pixel.
- The width needed for a variable that ranges over all X coordinates on a scan line.
- The width needed for a variable that ranges over all Y coordinates on the display.
- The width needed for a variable that holds a RAM memory address.
- The width needed for a variable that holds a word of RAM data.

Be sure to use the appropriate PAL macros to determine all these parameter values. For example, some of the widths are not what you might expect them to be. Run the program both in simulation mode and on the RC200, and note the differences between the two sets of values. Be sure to report these results in the Discussion section of your lab report, perhaps as tabular data.

LCD Framebuffer

Sample Output**Draw a Test Pattern by Synchronizing with HBlank**

Your third project is to be named *Unbuffered*, and the Handel-C source file should be called *unbuffered.hcc*. This project is a standard one that does not need the PAL Console.

Use the *PalVideoOutGetHBlank(handle)* to synchronize your code with the beginning of each scan line. Draw a white pixel in the first column of each scan line, and draw vertical bars of alternating colors across the remainder of each line. The requirements for this project can be accomplished quite easily because (1) It's all right to write during the invisible portions of the refresh cycle, and (2) if you write white pixels to during all the clock cycles when HBlank is true, the last one you write will show up in column one of the next scan line.

An easy way to draw alternating colors on a scan line is simply to test one bit position of the register you use to keep track of the current position on the scan line. If you test the rightmost bit, the bars will be one pixel wide; if you test bit 1, the bars will be two pixels wide, etc.

Be sure your program works equally well when simulating and when running on the RC200.

Draw an Image From a Framebuffer

Use one of the RC200E's PL1 memory banks as a framebuffer. Decide how to map pixel coordinates to memory addresses, and write a test pattern into RAM. The test pattern is

LCD Framebuffer

to consist of a one-pixel wide white border along all four edges of the visible part of the display. Inside the border, draw vertical bars of alternating colors. Use the simulated version to verify that the correct pixel values are being stored in the correct memory locations. (*Note: The PAL virtual console shows byte addresses, not word addresses.*)

Now write code that continuously reads from the framebuffer and draws the pixels to their proper locations on the screen. Because of the delays involved in reading from the memory and writing to the screen, you will have to start processing each scan line during the end of the HBlank period of the previous scan line.

Use macro expressions to parameterize your code so that it does not directly use any of the numerical values you looked at in the Parameters project. But be sure your code will work correctly regardless of whether it is being simulated or configured for the FPGA. You are to meet this requirement without using conditional compilation.

You will need to pipeline your operations, starting 3 clocks before the beginning of a scan line. If $t(0)$ refers to the clock period during which a pixel is displayed in the leftmost column of the display (*i.e.*, the first clock cycle after HBlank goes false), the first few stages of the pipeline can be represented as follows:

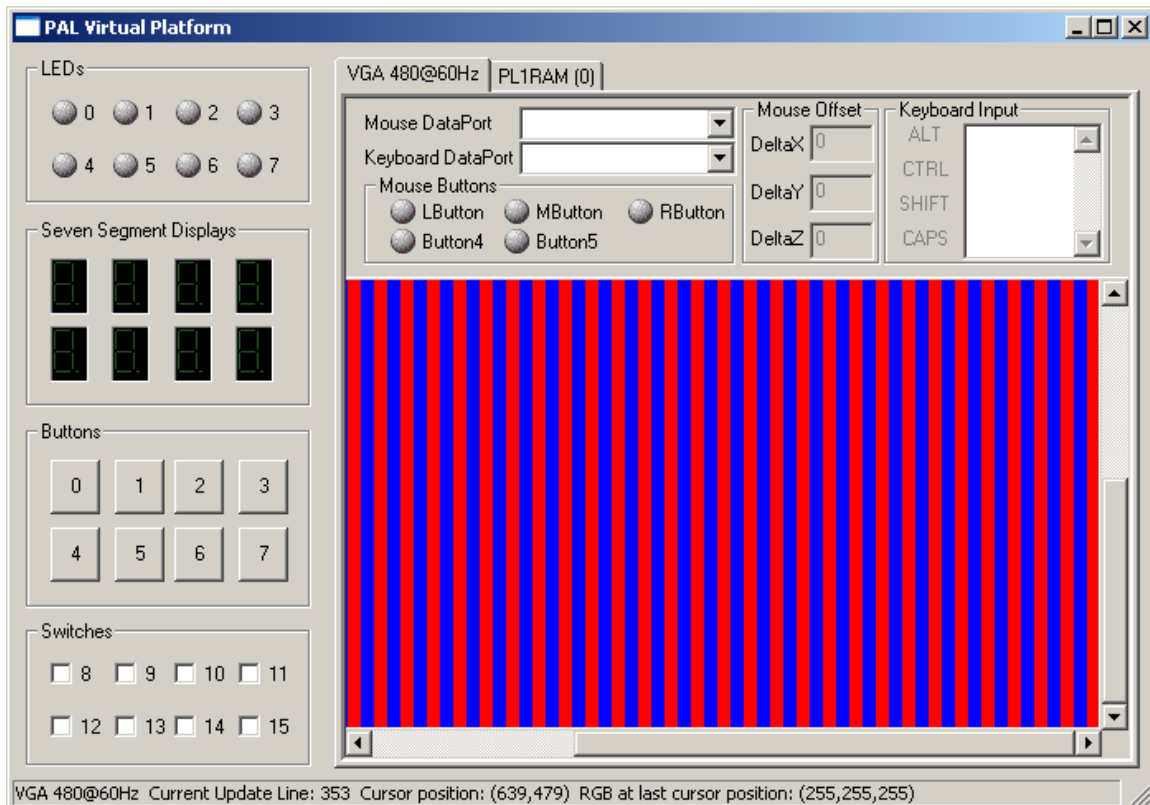
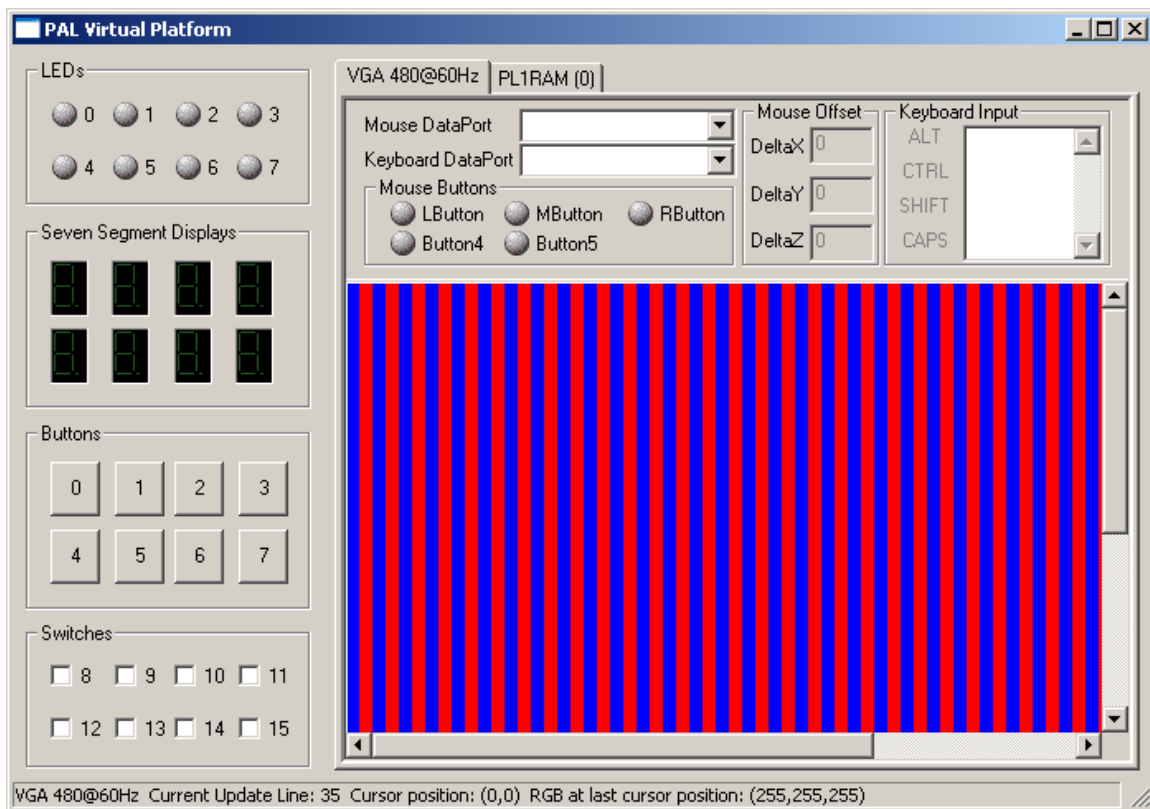
$t(-3)$	Set address for pixel 0			
$t(-2)$	Set address for pixel 1	Read pixel 0		
$t(-1)$	Set address for pixel 2	Read pixel 1	Write pixel 0	
$t(0)$	Set address for pixel 3	Read pixel 2	Write pixel 1	Pixel 0 displays
$t(1)$	Set address for pixel 4	Read pixel 3	Write pixel 2	Pixel 1 displays

When you test your code on the simulator, you can position the mouse over the simulated display to see what pixel value has been written to each position. Verify that all the pixels in the first and last rows and first and last columns -- and no others -- are white.

There is a bug in the simulator. If you make the Pal Virtual Console big enough to see the entire screen at once, it chops off the right and bottom edges so you can't see those borders. However, if leave the window small and use the scroll bars to look at those edges, you can see them.

Be sure your code generates “pixel perfect” images on both the simulated console and on the actual RC200E!

LCD Framebuffer

Sample Output

LCD Framebuffer

Optional: Write a macro expression or function named *write_pixel* (*x*, *y*, *pixel*) that writes pixel values into the framebuffer. This code must block during periods when the memory is being used to update the display to avoid mangling the contents of the RAM. Use *write_pixel*() to show the ASCII codes for keyboard characters by drawing the images of two seven-segment displays on the console, as in the SevenSeg project.

Submit a Report of Your Lab Activities

Use a word processor to write a report of your lab activities that follows the format of the Lab Report Guidelines for this course, and email it to me by midnight of the due date.