CSEE W4840 Embedded System Design Lab 2

Stephen A. Edwards

Due February 12, 2004

Abstract

Use your newfound C programming skills to write a program that displays “Hello World!” on the video display using the simple framebuffer provided for you.

1 Introduction

One of the niftier peripherals on the XSB–300E board is the THS8133B video DAC. This is a triple 10-bit DAC that can operate as fast as 80 MHz and is connected directly to the FPGA and a VGA connector. As such, it is perfectly suited to driving one of the flat-panel displays in the lab.

We have designed a video frame buffer that uses one byte per pixel. This is actually fairly tricky. It uses the onboard SRAM to store the frame, and a memory interface arbitrates between processor access to the memory and video access (video access gets priority). For this lab, though, you can simply ignore all of this and treat the frame buffer as a section of memory that just happens to be displayed on a monitor.

2 Hello Video

We have provided a simple canned project that exercises the video display. Make sure it compiles, downloads, runs, and displays video before tackling the rest of this lab.

1. Create a directory called, say, `lab2` and `cd` into it.
2. Unpack the project from my home directory:

   ```
   $ tar zxvf ˜sedwards/4840/lab2.tar.gz
   ```

3. Invoke `make download`. After a while, this will generate and download a `.bit` file that runs a program (`c_source_files/hello_video.c`) that clears the screen, draws a red border, a circle, square, triangle, and some lines.

3 The Framebuffer

The 640 × 480 framebuffer uses one byte per pixel. The base address is 0x00800000 and is arranged as a sequence of rows in memory, i.e., the address of a pixel at coordinates (x,y) is $\mathrm{0x00800000} + x + 640y$. The bits in each byte represent brightness levels as shown in Figure 1.

7							0
r2	r1	r0	g2	g1	g0	b1	b0

Figure 1: The layout of bits in each byte of the framebuffer.

4 The Assignment

Write a program to display “Hello World!” on the video display.

Write it as a character-printing function called repeatedly by the main function; you will need such a character generator for the next lab. (You may want to look at the next lab to get a hint of where you are going to save you time.)

Memory is at a premium: encode your font as a bitmap, i.e., by using one byte for each row of each character. Creating a font is tedious: you only need enough for “Hello World” in this lab, but you’ll need the full alphabet for the next lab, so there’s no harm in starting early. An 8 × 8 font is fine. Feel free to locate an existing font and adapt it for your program. If you do this, document where you got the font. Do not copy it from another group, however.

Make your character printing function take three arguments: the character to print, the row at which to display it, and the column.

Show your working graphical “Hello World!” to a TA, have him sign a printout of your solution (i.e., the C program including the font data), and hand that it.

Again, shorter, elegant, readable solutions will be scored higher than those that merely work.