CSEE W4840 Embedded System Design Lab 5

Stephen A. Edwards

Due March 4, 2004

Abstract

Modify the simple framebuffer we provided to make it display characters instead of a bitmap. Couple this with a simple C program that display "Hello World."

1 Introduction

Engineering rarely starts with a blank slate. A bridge builder starts with a site; a computer engineer often starts with existing pieces of a system—a programming language, an operating system, pre-built peripherals, and whatnot. As a result, *system integration*—the act of assembling and adapting pieces to form systems—is often the most important part of an engineer's job. In this lab, you will reverse-engineer the framebuffer we have provided and modify it to directly generate characters, effectively moving the responsibility for displaying character from software to hardware.

In labs two and three, you used a bitmapped framebuffer to display characters. While most modern computers take this approach to support GUIs and other graphical applications, it puts more of a strain on the memory and video system than a character-based display. Historically most video displays were character-based, meaning memory held information about characters rather than pixels, and the video hardware was responsible for translating this information on-the-fly into a suitable bitmap. Character displays have two advantages and one main disadvantage. Since a single byte can represent a, say 8×8 character matrix, a character display requires far less memory than a bitmapped one. Furthermore, since only one byte is necessary per character, the memory bandwidth is also reduced. The main disadvantage, obviously, is that a character display cannot display arbitrary images.

For this lab, you will modify the framebuffer we have provided to make it directly display characters, i.e., a byte in video memory will contain a code for an ASCII character, not pixels. To do this, you will modify the VGA memory address generator and video data output circuitry to convert in-memory data to a character bitmap before sending it out to the screen.

2 The opb_xsb300 Peripheral

The VHDL files for the custom peripheral we designed are in myip/opb_xsb300_v1_00_a/hdl/vhdl in the lab5.tar.gz tarball and described in Table 1.

This peripheral performs three important functions. First, it generates video by generating the necessary timing signals (horizontal sync, vertical sync, and blanking, which blanks the video output during horizontal and vertical refresh), memory addresses for each pixel, and eventually formats and ships out this video data to the triple 8-bit video DAC on the board. The vga_timing.vhd file generates the control signals and vga_.vhd does the rest.

file	purpose
opb_xsb300.vhd	The top-level module: instances of the memory controller and video circuitry
pad_io.vhd	I/O pads for the off-chip memory bus
memoryctrl.vhd	A complex state machine that arbitrates between processor and video accesses
vga_timing.vhd	The video timing and address generator. Produces synchronization and blanking signals along with video memory addresses
vga.vhd	The video generator: uses the video timing generator and the memory controller to fetch bytes from memory and send them to the video DAC.

Table 1: Files in the opb_xsb300 peripheral

Second, it controls the off-chip SRAM through address and data busses, chip selects, and so forth. The glue logic in the main module, opb_xsb300.vhd, generates the signals and the off-chip drivers for these signals are in pad_io.vhd.

Finally, it arbitrates access to this memory between the video controller and the processor. In each cycle, the processor, video, or both may want access to the memory. Since the video absolutely needs the memory when it asks for it (otherwise, the display would flicker), the memory controller (in memoryctrl.vhd) gives priority to the video system, possibly making the processor wait its turn.

For this assignment, you will need to modify the vga and vga_timing files to adapt them to work as a character display. It should not be necessary to modify the others.

The vga_timing block generates four key signals: horizontal sync, vertical sync, blanking, and the RAM address for the currently-displayed pixel. These are generated from three counters: a pixel counter that tracks the horizontal position of the current pixel, a line counter, and an address counter that is reset at the beginning of each field and advanced every time a visible pixel is displayed (i.e., the address counter does not count during horizontal and vertical blanking). The horizontal and vertical sync signals turn on for a fixed range of pixels and lines respectively. The blanking signal is essentially the logical OR of slightly fatter versions of these signals.

The vga block contains a single instance of vga_timing. It slightly modifies the RAM address before passing it along to the memory controller. It stores the video data from memory in a very simple shift register, since memory returns 16 bits at a

time that encode two pixels. In alternate pixels, the shift register loads a 16-bit value from memory and shifts one byte left to get the second pixel in a 16-bit word.

3 The Assignment

Adapt the provided video controller to display characters and demostrate it works by creating a simple C program that displays "Hello character world" using your new display controller.

There are three main tasks in VHDL:

- Create a character set ROM using distributed RAM blocks. See the VHDL handout or the XST manual for templates for specifying such blocks. You may want to write a little program that generates the VHDL for initializing this ROM. You may use the same font you used for Lab 2.

  If you put your ROM in a separate VHDL file, make sure you add its filename to the "pao" file in myip/opb_xsb300_v1_00_a/data. This is a list of VHDL files that comprise the peripheral.

- Modify the video shift register in vga.vhd so that it loads itself from the character set ROM, shifts one bit left every pixel, and uses one bit per pixel. Make sure to change the frequency of the "video request" signal coming from the vga block so that it fetches a new word every two characters instead of every two pixels.

- Modify the video address generator in vga_timing.vhd to advance only one byte every eight pixels (I suggest using an 8×8 character grid) and repeat the same set of addresses for eight lines before advancing.

As usual, turn in a printout of *every* file you personally create for this assignment (e.g., if you do not modify memoryctrl.vhd, do not bother to print it out). Make sure your names are on it. Demonstrate it to a TA, have him sign off on it, and turn in the listing.

VHDL is a terribly verbose language that can quickly become unreadable. Again, we are looking for style in addition to substance.