467 / Project 2-b

Purpose:

- To understand the concepts behind system interfacing
- To understand how to communicate with an asynchronous device (sram)

Overall goal:

The overall goal of this project (project 2-b) is to build on project 2-a. In 2-a you build a RAMDAC, but simply took the address and placed as the input to the DAC. In reality this address needs to go to the framebuffer, fetch a byte and that byte is placed out to the RAMDAC.

To begin this project I suggest you build the serial interface. The serial interface is taking the place of the PCI bus in our display card. I suggest you do the serial interface first because you will need it in order to properly test the memory interface.

Serial interface:

When you interface between two different systems you must keep in mind that those systems have completely different ideas about ground (i.e. logical zero). Although a lot of times in the lab ground will be the same for different components this is merely an artifact of proximity and chance. To interface two systems being driven by separate power supplies you must interface them as if they are different analog devices. This is nicely achieved with an op-amp – a device that will "measure" the difference between two voltages (such as ground and signal from one system) and produce a new voltage related to this difference from another set of voltages. For serial communication an op-amp/interface chip that is commonly used for this is a DS275 (there are several chips that people often used, including something called a MAX232. All of them pretty much do the same thing. We are using DS275's because they only require a single power supply voltage).

Use a DS275 chip and build a serial interface. Hook this up to a PC and using Visual C++ (or anything else you want to, Java, whatever) write some client side code to talk to this serial interface. Send some data to your device. For testing purposes you can use this data to alter something simple about what is being rendered by the RAMDAC (for example, have your serial interface send a byte and use this byte to set the color of all pixels being drawn by the RAMDAC). Ultimately what you will need to do is design a simple protocol that allows you to send (command,x,y,c) to set individual pixel values in your frame buffer, but delay this part for a bit.

Framebuffer:

The frame buffer you will be building is actually two frame buffers. An "on-screen" one which is being read by the RAMDAC and an "off-screen" one which is being sent draw commands from the serial interface. This switching of frame buffers is accomplished with a simple mux on the interface.

Although you can declare a large memory inside of the FPGA there is not enough memory inside the FPGA for the framebuffers, Z-buffers and triangle lists. Hence you will build your frame buffer from external components. You will need to build two 128k framebuffers/Z-buffers. The lab will supply you with 2 128kx8 SRAM chips. You will need to build these into two different banks (one bank for each frame buffer).

Note that a large part of this component of the assignment is simply wiring up everything well. There will be a lot of wires (at least 60, probably closer to 70 after you do all of the control lines). Be neat with the wiring otherwise this will become a nightmare to assemble!

NOTE THAT YOU CAN FRY THE FPGA IF YOU PROGRAM THE MEMORY PORT WRONG. This occurs because the bus between the SRAM and the FPGA is bidirectional. At times the memory chip will drive this bus, and at other times the FPGA will drive the bus (we are talking about the data bus here). Please test your SRAM interface in ActiveHDL against the SRAM chip model (we will supply one), and make sure you and the SRAM model are not driving the bus at the same time.

Final assembly:

Wire up the RAMDAC to the framebuffer and use the serial interface to control everything. You should accept two different types of commands [0,f,0,0] which sets the currently on-screen framebuffer to f (where f equals 0 or 1). This should also set the off-screen frame buffer to 1 – f. Next you should accept the command [1,x,y,c] which sets the pixel at (x,y) to gray-scale shade c. Write a program that sends a pixel over to your frame buffer (off screen) and then switches the on-screen frame to it. Send a whole picture.

Notes

This project will become progressively less defined as we go on. Minor details such as the interface specification between the PC and the board is entirely up to you. In general, so long as your project accomplishes the same basic thing and meets the higher order goals of the project (such as in future projects we will want to build a pipelined design), then all of the internal stuff is up to you. Note that this does mean one groups PC interface will not be compatible with anothers unless you coordinate, but se la ve.

It may be simpler to use a fixed size protocol between the PC and the display card. The serial port sends only a byte at a time, and using a fixed size protocol could simplify the recombining process. Alternatively you could use a protocol like: [length, data]. You probably also want a back channel acknowledgement. All of this, though is for you to decide.

Suggestions

I suggest you create a number of modules. One module (RAMDAC) is most of project 2-a. Another module SRAMInterface provides 2 read and 1 write port, and some control ports. Another module SerialInterface reads packets of information from the serial interface and hands them off (without much higher level control) to PixelDrawControl which handles the drawing of pixels and switching of frame-buffers. The reason for this modularity is that in Project 3 we will toss out PixelDrawControl and replace it with a rudimentary 3D pipe, but largely keep SRAMInterface, SerialInterface, and RAMDAC in place. So, when you are working on this project, keep a keen eye on the ability to re-use those components.

Finally, I suggest that between your components you do not use bidirectional ports (i.e. at the SRAMInterface, expose 2 read and 1 write port (or 2 write ports if it matters), but make these unidirectional. The SRAMInterface will need to take these ports and route them to the bidirectional ports used to talk to the actual SRAM). The reason for this is bidirectional ports are a pain, and really only useful for limited pin configurations (as in external pins, which are generally a limited resource). If you use them internally it is more likely you will fry the FPGA and have a heck of a time debugging your design.

Grading

As always, your grade depends on making it work. The monitor should have a picture on it!