

EECS 373 – Fall 1998

Lab 7: Bus Interfacing for I/O Devices

Introduction:

At this point you should be very familiar with the low-level software aspects of communicating with I/O devices. In this lab, you will look at some of the hardware aspects of interfacing a microprocessor to an I/O device.

Goals:

1. To gain experience designing logic to interface to a microprocessor bus.
2. To explore tradeoffs between hardware and software implementation of features.

Specifications:

This lab will expand on Lab 6 by moving the control of the I/O devices (dip switches, pushbuttons, seven-segment display, bar-graph display, and DAC) to the microprocessor. Software running on the processor, rather than hardwired logic, will control the flow of information from the input devices to the output devices.

You will implement four device registers in the Xilinx chip:

1. A halfword-size read-only register that reflects the state of the eight DIP switches and the two pushbutton switches. The bit positions and polarities within the halfword are up to you.
2. A byte-size write-only register that controls the seven-segment display. The display should always show the value last written to this register in hexadecimal.
3. A byte-size write-only register that controls the bar-graph display. The display should always show the value last written to this register in binary.
4. A byte-size write-only register that controls the DAC output. The DAC should continuously output a voltage proportional to the value last written to this register.

As in Lab 6, pressing S1 will take the value on the DIP switches and update the seven-segment display, while pressing S2 will take the value on the DIP switches and update the bar-graph display. Unlike Lab 6, the DAC should output a voltage proportional to the **product** of these two values. (The product of two eight-bit values is not an eight-bit value, but you have only an eight-bit DAC. Think about it; it's not hard.)

Details:

Read Chapter 13 (External Bus Interface) for this lab. You will not be using the internal memory controller, so the MPC823 will consider your registers to be 32-bit ports (see Section 13.4.5). However, you only need to support correct accesses (i.e. byte writes to the output registers and halfword reads of the input register). You do not need to be concerned with burst transfers, bus arbitration, or transfer errors.

All of your I/O device registers should be mapped into the address range 0x02400000-0x0240000F. The specific addresses you use are up to you. In order to reduce the amount of logic required to decode the address bus, you should only use address lines A6-A11 and A26-A31 when determining if the address of the current cycle is intended for your devices. Remember that A0-A5 (the six most significant bits of the 32-bit address) are not present on the external address bus.

Use an interrupt to read the switches every 100 ms. Use your prime-number generator from Lab 5 as the main body of your program.

Read the “Lab Report” section for some opportunities for extra credit on this lab.

Prelab Questions:

1. **PRIOR TO COMING TO LAB**, you must have a schematic of your design and an initial draft of your program.
2. Since you are only checking a subset of the external address bits, there are a number of different addresses that will actually be able to access each of your devices. (Assume that the internal address bits A0-A5 are all 0.) How many different addresses will access the switch register? What are they? (These additional addresses are commonly referred to as “ghost” or “shadow” locations for the register.)
3. Write a simple program (or set of programs), separate from your main program, to thoroughly test each of the device registers. If a program tests more than one register, test each register thoroughly before moving on to the next. You may assume that you will be running this program under the debugger, so it does not need to do any explicit output (i.e., you can check that things are working by looking at register values).

Lab Procedure:

1. Thoroughly test your hardware design using the simulator. Be sure that you only drive the data and control signals when the appropriate addresses appear on the bus in conjunction with the TS* signal. If you drive the bus at any other time, the entire system may not work.
2. Download your design to the Xilinx board. Use the ‘read’ and ‘write’ commands in the command window to access the registers to verify that things are not completely broken. If there is a problem at this point, go back to the simulator; if you can’t find an error there, use the test points in conjunction with the small logic analyzer to debug your error.
3. Download your test program (from prelab question 3) to the processor and trace through it using the debugger.
4. If your test program finds no errors, download your main program and run it.
5. Once you are certain that your design is complete, notify a TA. The TA will help you to download your design onto a board at the lab station with the HP16601 logic analyzer. Observe and draw the bus transactions for reading the switch values and writing to one of the output devices.

Lab Report:

You must do at least a paper design for questions 7 and 8. Extra credit is for those who go the extra step of implementing, debugging, and demonstrating that their design works.

1. Briefly summarize the operation of your circuit and how you arrived at your design.
2. Include a printout of your final schematic and a commented listing of your program.
3. Discuss any difficulties you had in designing, implementing, or debugging the circuit.
4. How many CLBs and IOBs did your design require? What fraction of the total number of available CLBs and IOBs does this represent, respectively?
5. Compare the complexity of your hardware to your design from Lab 6. What are the advantages and disadvantages of using a microprocessor instead of hardwired control logic?
6. Draw the two bus transactions you observed on the HP16601 logic analyzer. Use “cause/effect” arrows to indicate exactly what is happening on the bus.
7. On a read of the switch register, how much time is there from the point where the addressed register is identified (i.e., the address is decoded and the appropriate enable signal(s) are generated) to the point where the data must be stable for the processor?

8. Recall that interfacing a slower device to the bus may require the addition of wait states. Design a circuit that would add two wait states on every read of the switch register (but *not* on any other register accesses). **For 10 points extra credit**, implement this circuit, demonstrate it to the TA on the HP16601 analyzer, and include your modified schematic.
9. Notice on the board schematic that the data signals connecting the Xilinx to the DAC are actually a “local” data bus shared with the ADC and the SRAM. If you wanted to use the ADC or the SRAM, you could no longer just place the DAC value on the bus and disable the DAC latches. Instead, you would have to send the DAC value over this local bus and latch it in an internal DAC latch. Describe in detail how you would interface the microprocessor to the DAC in this case. Pay particular attention to the timing requirements for the control signals of the DAC. Draw timing diagrams for the interface, including all relevant microprocessor signals and the bus clock. **For 10 points extra credit**, implement this circuit, demonstrate it to the TA on the HP16601 analyzer, and include your modified schematic.

Hints & Tips:

1. Use the TS* signal to latch the address (and other control signals) for use by the decoding circuitry.
2. Use the test points available as the signals IC[31:17] on the BOARD_IO component to bring important internal signals out for debugging.