EECS 373

Lab 3: Bus Interfacing for I/O Devices

Schedule

See posted lab schedule for pre-lab, in-lab and post-lab due dates.

Objectives

The purpose of this lab is to:

1. Introduce you to the basic logic required to interface a device to the MPC823 bus.
2. Observe the interaction of hardware and software in a very simple environment.
3. Gain very basic experience with system design involving hardware, software, and integration.

Overview

Interfacing hardware I/O devices to a microprocessor is a simple matter of observing the processor's bus signals and responding accordingly. Once done, the hardware device can be controlled by software running on the processor. In this lab, you will interface the switches and LEDs on the expansion board (which you used in Lab 1) to the MPC823 bus. You will also write a simple assembly-language program to read the state of the switches and update the LED display accordingly.

Design Specifications

Your hardware design will consist of two I/O device registers: an input pathway to read the state of the switches and an output register to control the state of the LED displays. Specifically:

- Create a 32-bit input pathway at address 0x02900000 that returns the state of the eight DIP switches in bits 24-31 and the toggle and pushbutton switches in bits 20-23.
- Create at 32-bit output register at address 0x02900004 that sets the pattern displayed on the LEDs. After writing to this register, the seven-segment display must show the hexadecimal value contained in bits 24-31 of the value written, and the bar-graph display should show the binary value contained in bits 16-23 (0 = off, 1 = on). Bits 0-15 are unused and should be ignored.

The software component consists of an assembly-language program that continuously reads the state of the switches and sets the LED displays accordingly. Your program should have the following features:

- When pushbutton S1 is pressed, the seven-segment display should be updated to show the binary value entered on the DIP switches.
- When pushbutton S2 is pressed, the bar-graph display should be updated to show the binary value entered on the DIP switches.
- The state or position of toggle switch S3 determines how values are to be displayed on the bar-graph LED and seven-segment display when S1 or S2 are depressed. If switch S3 is in the "normal" position, the displays show the actual binary value on the DIP switches. If switch is S3 is in the other ("complemented") position, the value displayed on the bar graph must be the one's complement of the DIP-switch value, and the value displayed on the seven-segment display must be the two's complement (neg-

ative) of the DIP-switch value. You may choose which S3 switch position is “normal” and which is “complemented”.

- Toggle switch S4 is unused.

Design Notes and Hints

- Review pages 13-9 through 13-14 of the Motorola PowerPC MPC823 User’s Manual.
- The PROC macro from the EECS373 library represents the basic bus signal connections for the Xilinx chip on the expansion board. Signals on the left side of the macro block are processor inputs (Xilinx outputs), while signals on the right side are processor outputs (Xilinx inputs). The data bus is handled specially: although the pins are bi-directional, the Xilinx software requires splitting them up into two parts on the Xilinx chip. PD_IN is the data bus input (from the processor to the Xilinx), while PD_OUT is the data bus output (from the Xilinx to the processor). The PD_OUT_EN signal enables the data bus drivers, which will drive the value on the PD_OUT bus onto the Xilinx pins. Use the Hierarchy tool to open up the PROC macro and see how these two busses and the enable signal are connected to the physical data bus pins. In this lab, the tri-state drivers inside the PROC macro will act as the tristates which allow your device to return read data to the CPU.
- Low-true signals are denoted in Xilinx by a “_BAR” suffix (e.g., $\overline{\text{TA}}$ is TA_BAR).
- The open-collector wired-OR Xilinx output signals (TA_BAR, TEA_BAR, IRQ1_BAR, and IRQ7_BAR) use a special output circuit that only drives the output pin low when the input signal is low, and leaves it tri-stated (floating) otherwise. You do not need to drive it with a tristate device.
- You should make sure to drive all unused Xilinx output signals, i.e. for this lab, tie TEA_BAR, IRQ1_BAR, and IRQ7_BAR to VCC.
- The MPC823 will sample its input signals (TA_BAR and, on a read, the data bus) on each rising edge of the clock. You need to provide non-zero setup and hold times for these inputs relative to the clock edge. An easy way to do this is to drive the signals from one falling clock edge to the next, providing roughly one-half of a clock cycle of setup and hold each.
- To save on address decoding logic, you should check only the upper six external address lines (A[6:11]—recall that A[0:5] are not available off the processor chip) to determine whether your registers are being accessed. and one low-order address line (which one?) to determine which of your two registers is being accessed. The remaining address bits are don’t-cares.
- $\overline{\text{TA}}$ must be asserted for exactly one rising edge of the bus clock.
- There is a macro in the EECS 373 library (Dual7sdsp) that takes an 8-bit binary input, converts it to the appropriate hexadecimal seven-segment on/off patterns, and multiplexes it onto the dual seven-segment display. You need only supply a clock (i.e., gclock from the PROC macro).
- You will need to select a device to implement your LED register. Consider a clocked device such as the FD8CE. This is a macro consisting of 8 D flip-flops in parallel and a CE (clock enable) control. . The clock enable is used to clock data into the register at the appropriate time. Use the hierarchy key to examine the contents.

Pre-lab assignment

Pre-lab questions 1 - 4 must be done **individually.**

Pre-lab questions 5 - 6 can be worked on with your **partner**.

Also, with your partner come up with an initial design for your hardware and software. Chances are it will not work. That's Ok. In this lab you will work on debugging techniques for intergrateing software and hardware.

For all the questions use the following constants:

```
.equ zero, 0
.equ one, 1
```

1. Write an instruction or instruction sequence that will perform each of the following single-bit operations on the value in register r4. All other bits in r4 should remain unchanged. In this problem and in all your programs, use the C arithmetic operators (for example, +, <<, >>, ~, |, &) to derive the constant values you may need. Your code will be much clearer and (as long as all the operands are constants) the assembler will evaluate these expressions for you, eliminating potential calculation errors. For example, a 32-bit value with a single 1 bit in bit position 21 can be written as:

   ```
   (1 << (31 - 21))
   ```

   (The "31 – *x*" is required because, in the PowerPC world, bit position 0 is the *most* significant bit.) Your intent is much clearer and errors are much less likely than if you did the calculation in your head and just used the equivalent constant value (e.g., 0x400). (***14 points, 2 points each***)

   a. Set bit 29.

   b. Set bit 18.

   c. Set bit 4.

   d. Toggle (complement) bit 15.

   e. Toggle bit 6.

   f. Clear bit 21.

   g. Set the bit indicated by the value in r6. Assume that r6 holds a value between 0 and 31.

2. Give an instruction sequence that will execute the lwz in the code fragment below if and only if bit 21 of r4 is 1. (***4 points***)

   ```
               (your code goes here)
               lwz r5, 0(r6)
   skipload:   (more code)
   ```

3. Why is the LED register at 0x02900004 and not 0x02900001? (***4 points***)

4. Besides the address you designed your registers to be accessed from, how many other **word**-aligned addresses (shadow locations) will they respond to? See appendix A on shadow locations. What are these addresses? You can specify a range. Remember, the MPC823 external bus consists of address lines A6 – A31, so you can assume A0 – A5 are zero. (***4 points***)

5. Sketch the timing for a zero wait read cycle on the page provided in appendix B.Assume that the MPC823 data and TA setup and hold times are 1/4 clock cycle. (***2 points***)

6. Sketch the timing for a zero wait write cycle on the page provided in appendix C. Assume that the led register data setup and hold times are 1/4 clock cycle. (***2 points***)

In-Lab Procedure

Part I: Debugging your hardware

Step 1. Basic functionality

Implement the interface circuitry in a macro and test the macro using the simulator. Although this is not a prelab requirement, your time in lab will be better utilized if you simulate your design before coming to lab. Since it is difficult to completely simulate MPC read and write cycles, just focus on address decode correctness and $\overline{TA}$ generation. Remember when using buses to always label them.

Configure the logic analyzer with the configuration file, 'TS_trig'. This configuration maps the MPC external address bus, data bus, control bus and 8 testpoints for observation. The MPC external bus signals are physically connected to the logic analyzer via the ribbon cables connected to the MPC processor board. The configuration also sets the trigger to capture on an active low $\overline{TS}$ transition. Triggering on *transfer start* captures a complete read or write cycle.

In addition to the MPC external bus signals, it will also be very helpful to observe the control signals associated with reading the switches and writing the LEDs. For now, connect the switches tri-state buffer enable signal, the LED register latch (clock) and enable signal to test points. It is also very useful to observe the PD_OUT_EN signal since it enables data onto the MPC823 data bus via PROC macro. Make sure the appropriate test point wires from the logic analyzer are physically connected to the EECS373 IO board.

Power on the target board, then start SingleStep. In the initial SingleStep dialog box, check "Debug without a file", then click OK. This causes SingleStep to initialize the target board without downloading a program. *Remember, you must initialize the target board using SingleStep before downloading your Xilinx design*.

Download your design to the Xilinx board with the suggested test point signals connected. Use the 'read' and 'write' commands in the SingleStep command window to test the I/O device registers. Specifically, 'read –rlx 0x02900000' will read your switches, and 'write –l 0x02900004= 0x1234' will write the 32-bit value 0x00001234 to your LED register. (The read and write arguments are lowercase Ls, not ones.)

Initiate a read and write cycle with the SingleStep 'read' and 'write' commands and observe the transactions with the Logic Analyzer.

- Verify that $\overline{TA}$ occurs at the correct time for both read and write cycles. If it does not, you must correct your design before going any further. Do not expect to debug your logic by simply observing your schematic. If you are suspicious about some facet of your circuits functionality, connect test points to the signals and verify the operation with the logic analyzer.
- Verify that the enable control for the switches tri-state buffer and PD_OUT_EN signal occurs at the correct time during a read cycle. If they do not, you must correct your design before going any further. Again, use the logic analyzer to observe and verify the operation of your interface logic.
- Verify that the enable and latching (clock) signal to the led register occur at the correct time for a write cycle. If they do not, you must correct your design before going any further. Use the logic analyzer to verify your design assumptions.

The 'read' command reads several adjacent locations in addition to the one you specify. Do the values in the adjacent locations confirm or contradict your answer to pre-lab question 4?

Try writing the LEDs via one of the "shadow" locations. What happens?

Step 2. Measure setup & hold times

Capture a read transaction to your input register on the logic analyzer. What are the setup and hold times of $\overline{TA}$ and the data bus relative to the clock edge?

When a read transaction is initiated with the SingleStep read command, just one bus transaction occurs. The data bus is constructed such that values on the bus are held until another transaction changes the state of the bus. Consequently, when the data bus is observed with just one transaction, the data hold time appears to be artificially long. Since PDOUTEN defines the time that the data bus is guaranteed to be driven, measuring the hold time with respect to this signal should give a good indication of the hold time.

Demonstration 1.1: Show your captured read transaction to the lab instructor and indicate that you can measure the $\overline{TA}$ and the data setup and hold times.

Capture a write transaction to your output register on the logic analyzer. What are the $\overline{TA}$ and data setup and hold times relative to the clock edge?

As previously stated, the data bus is constructed such that values on the bus are held until another transaction changes the state of the bus. Consequently, the data hold time may appear to by artificially long. During a write cycle, the data bus is not driven after $\overline{TA}$ is recognized by the processor. According to the timing specifications, the data is guaranteed to be on the data bus a minimum of 5ns after the rising clock during $\overline{TA}$. Measure and verify that this minimum occurs.

Demonstration 1.2: Show your captured write transaction to the lab instructor and indicate that you can measure the $\overline{TA}$ and the data setup and hold times.

Step 3. Experiment with access size

Access your I/O registers using byte rather than word operations. You can do this from the SingleStep command line by using '-b' instead of '-l' (i.e., 'read –rbx' and 'write –b'). Record what happens for questions 7 and 8 of the post-lab.

Part II: Debugging your software

Thoroughly test your program using the SingleStep simulator. The simulator will assume that 0x02900000 and 0x02900004 are regular memory locations. You can emulate the effects of changing the switches by changing the contents of location 0x02900000 in the memory window. You can observe changes to LEDS by checking the values written to location 0x02900004.

Part III: Integration

Once your hardware and software appear to work in isolation, download your program to the processor and trace through it using the debugger. Once you are satisfied that your program works, click the green "go" icon and let your program run. Adjust the switches on the target board to test your system.

Demonstration 2.1: Demonstrate your system for the lab instructor.

Post-Lab

1. Summarize the operation of your circuit and describe how you arrived at the design. Include a printout of your final schematic, including schematics of any macros you defined. **Itemize** in a systematic fashion the different steps involved in your design. Make sure that the schematic printouts are big enough that all the signals are legible. (***4 points***)

2. Sketch the bus activity for a read of the switch register and a write of the LED register. Clearly indicate causality using arrows (e.g., from clock edges or input control

transitions to output transitions). (***4 points***)

3. What are the setup and hold times you measured at the HP logic analyzer in Part I? (***3 points***)

4. Include a well-commented listing of your program. Comments should include register usage (which values are kept in which registers), descriptions of all symbols, and explanations of all derived expressions. (***4 points***)

5. In Lab 1, the contents of the DIP switches are displayed directly on the BAR LEDs. In this Lab, we used generalized hardware and software to essentially perform the same function. What are the advantages and disadvantages of using a microprocessor instead of hard-wired control logic? (***3 points***)

6. Try to read your LED register. You should discover that you cannot. Why not? Your answer should include data path and bus transaction issues.(***2 points***)

7. Sketch the logic required to make it do so. (***2 points***)

8. How does your input register handle byte reads? Describe what happened when you attempted to use byte reads in Part I. What address would you use to have a byte read return the settings of the eight DIP switches? (***4 points***)

9. How does your output register handle byte writes? Describe what happened when you attempted to use byte writes in Part I. What address would you use to have a single byte write set the bar-graph display value without affecting the seven-segment display? Sketch the additional logic needed to make this work. (***4 points***)

Lab 3 Demonstration Sheet

Print this page and present it to your lab instructor when demonstrating the various lab sections. Turn this sheet in with your post lab or when your in lab demonstration is due. You are required to turn in only one demonstration sheet per group.

List Partners Names

_____

_____

_____

Part I: Debugging Your Hardware

D1.1 Show your captured read transaction to the lab instructor and verify measurement of setup and hold times.

Lab instructors initials:

D1.2 Show your captured write transaction to the lab instructor and verify measurement of setup and hold times.

Lab instructors initials:

Part III: Integration

D2.1 Demonstrate your system for the lab instructor.

Lab instructors initials:

Appendix A, Shadow Locations

When implementing address decoders for memory mapped I/O, it is common practice to decode just a subset of the address bus. It is not necessary to uniquely decode every address combination if there are not conflicts with other memory mapped I/O. The external address bus of the MPC823 consists of address signals A7 - A31 or there are 2^25 possible byte addressable locations. It is not likely that this many I/O locations are needed. So, the address space is partitioned into ranges. A map of these address ranges is often called a memory map. It will show over what ranges I/O devices are decoded.

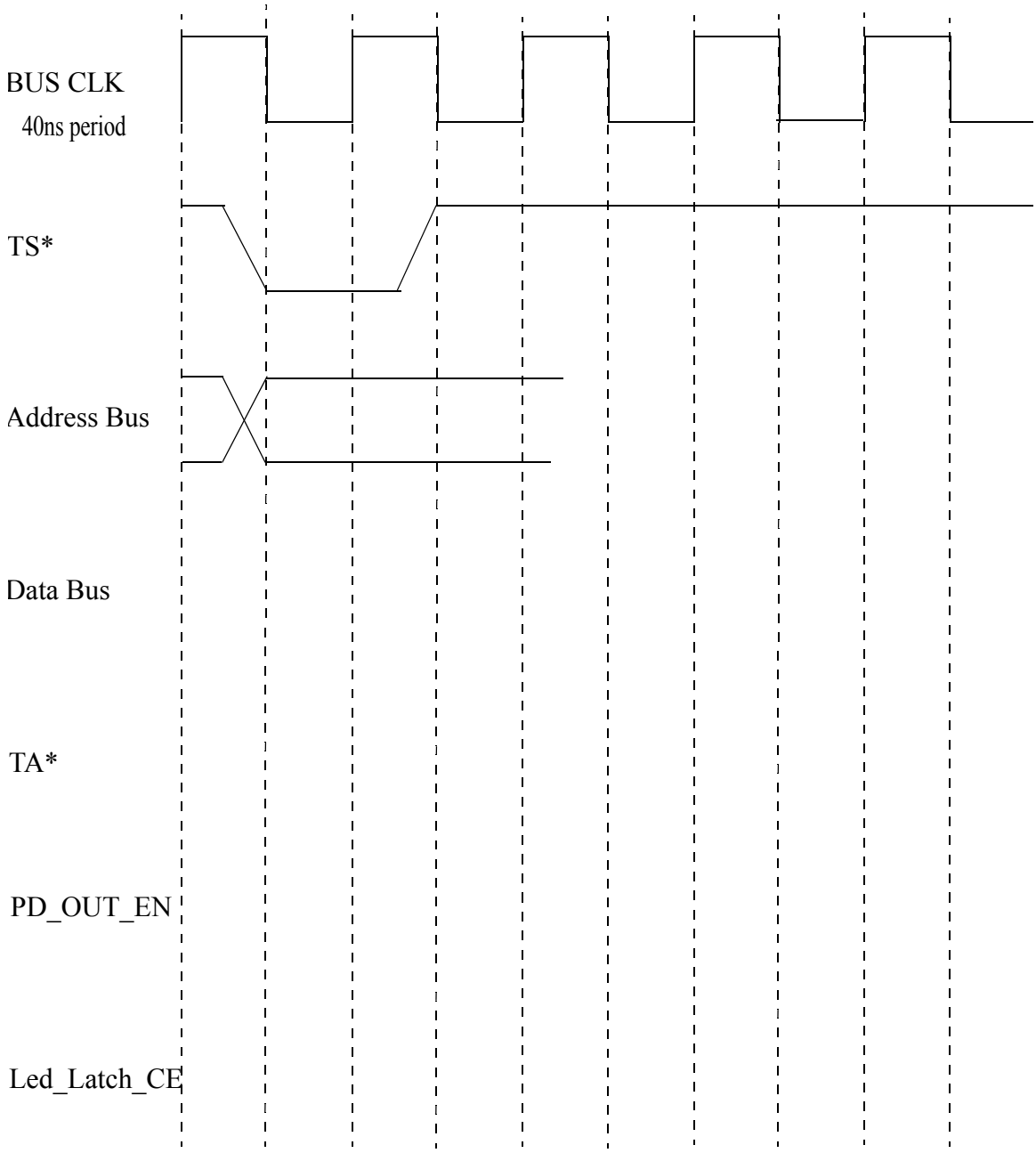
Consider a simple example. Assume that you are working over a very limited address space, say just three signals. So, there are 2^3 or 8 possible address locations. Lets say that we need to address two devices. Although the cost of implementing a unique address decode for these two devices is minimal, lets say for the sake of illustration that we want to use minimal decoding logic. In this case, we could just use address line A2 as the address decode. When A2 is low, it could be used to enable device 1 and when A2 is high it could be used to enable device 2.

Table 1:

A2	A1	A0	device enable
0	0	0	device 1
0	0	1	device 1
0	1	0	device 1
0	1	1	device 1
1	0	0	device 2
1	0	1	device 2
1	1	0	device 2
1	1	1	device 2

Device 1 responds to addresses 000 - 011 and device 2 responds to addresses 100 - 111. The first address of the address range that the device responds to is it's designated address. For device 1, this is 000 and for device 2, it is 100 shown in red. The remaining addresses are called shadow locations shown in blue. The total locations that devices 1 and 2 respond to are 2^2 = 4. The total number of shadow locations devices 1 and 2 respond to are 2^2 - 1 = 3. The range of shadow locations for device 1 is 001 - 011 or 1 - 3. The range of shadow locations for device 2 is 101 - 111 or 5 - 7.

Appendix B, Read Timing



Appendix C, Write Timing