EECS 373

Lab 4: Bus Interfacing for Byte-Addressable Memory

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 mechanisms required for handling multiple data transfer sizes, including transfers smaller than the bus width.
2. Help you to understand the overheads involved in handling unaligned accesses.
3. Introduce you to how a simple memory structure interfaces to a bus.

Overview

This lab expands on Lab 3, adding a small memory module to your hardware design. For simplicity, in Lab 3 you assumed that any access to your I/O registers was a 32-bit access. Unlike an I/O register, a general-purpose memory module must handle any access size the CPU may generate—byte, half-word, or word, in the case of the MPC823. Also unlike your I/O registers, your memory module will occupy a range of bus locations. As a result, your design will use different portions of the address supplied by the MPC823 for different purposes. Some of the address bits will determine whether or not your memory module is being accessed; some will determine which word within the module is being accessed; and, if the access size is less than a word, some of the address bits will determine which byte or bytes within the word are being accessed.

To exercise your memory module, you will write a small program to read and write the memory using the DIP switches and LED displays. You will reuse the I/O registers from your Lab 3 design to interface with the switches and LEDs.

Design Specifications

Hardware

Implement a 256-byte memory module on the Xilinx device (using 32x8 memory parts from the Xilinx library) and interface it to the MPC823 bus. This memory should occupy 256 consecutive byte addresses starting at address 0x3100000. The module should correctly handle aligned byte, halfword, and word reads and writes. (Recall that the MPC823 will only generate aligned accesses on the bus.) Your memory module should be integrated with your hardware design from Lab 3, so that your Lab 3 I/O registers can be used in conjunction with the memory.

Software

Write an assembly-language program that lets you use the switches and LED displays on the expansion board to direct the processor to read and write your memory one byte at a time. The resulting operation is reminiscent of the front panel of several old computer models (like the DEC PDP-11 series), where a bank of toggle switches and LEDs could be used to read and write memory locations. On these old machines, the front panel switches controlled hardware that directly performed read and write transactions: the purpose of the front panel was to let users access memory even when the CPU couldn't. For example, the front panel could be used to manually enter an instruction sequence (in binary!) to boot the

machine, or to examine memory locations to debug the system when it had crashed so hard that the CPU was not responding.

Your program will be emulating this front panel operation in software by reading the switches, performing any appropriate memory accesses, and displaying the results on the LEDs. Note that, unlike the old-time front panels, the switches and LEDs will not talk directly to the memory module; although they share the same data bus, all interaction between the switches, LEDs, and memory is performed in software by the CPU.

Your program must operate as follows:

- Your program must maintain a “current memory location”, which is a byte address in the range 0-255 indicating one of the 256 bytes in your memory module. (0 is 0x3100000, 255 is 0x31000FF) The current memory location should be set to 0 initially.
- At all times, the bar-graph display indicates the address of the current memory location in binary, and the seven-segment display indicates the byte value stored at the current memory location.
- When pushbutton S1 is pressed, the binary value entered on the DIP switches should be written to the current memory location.
- When pushbutton S2 is pressed, the current memory location address should be changed. The new current memory location is determined by the position of S3 and S4, as specified in Table 1.

Table 1: Effect of S3 & S4 on current memory location.

S3 position	S4 position	New current location
(any)	up	value on DIP switches
down	down	(current location – 1) mod 256
up	down	(current location + 1) mod 256

Design Notes and Hints

- Review your lecture notes on bus transfer sizing. Read Section 13.4.5 of the PowerPC MPC823 User’s Manual.You may assume you are using a 32 bit data port. Pay particular attention to Figure 13-18, Table 13-2, and Table 13-4 on page 13-33.
- As in Lab 3, you should check only the upper six external address lines (A[6:11]) to determine whether your memory module is being accessed. Of course, you will also have to examine some low-order address lines to determine which location in your memory module is being accessed.
- Use the RAM32x8S part from the Xilinx library. Each part is equivalent to 256 D flip-flops in an array of 32 rows and 8 columns. The five address inputs select one of the 32 rows. The eight Q outputs reflect the values stored in the eight flipflops of the selected row.
- You should construct a 128-byte (32-word) memory module using four of the RAM32x8S parts. This module should use four independent byte-enable inputs to control access size, as discussed in lecture. Define this module as a Xilinx macro.
- Construct your 256-byte module by combining two of your 128-byte modules. You may want to sketch this top-level design first to help you figure out exactly which inputs and outputs you need to provide on the 128-byte module design.

- You will need to generate the byte enable signals yourself from the MPC823's TSIZ control signals and low-order address bits. You could put this logic in a separate macro outside the two 128-byte memory modules.
- Because you will now have multiple devices that can respond to a read request on your Xilinx chip (the memory module and the switches), you will need separate tri-state buffers for each device to prevent bus contention. If you did not use a separate tri-state buffer with the switches in Lab 3, you will have to add one. To reduce the complexity of your schematic, consider making another macro containing the switches and the tristate buffer. Add an enable line as an input.
- In general, don't hesitate to create new Xilinx macros to encapsulate portions of your design. Macros improve reliability by allowing you to reuse known functionality and improving the organization and readability of your schematic.
- You might find that when you increment or decrement the memory address erratic changes occur. Consider that when the software reads the switch position, it does so many times over. This will cause the action associated with the switch to occur an unpredictable number of times. To avoid this, design your software or hardware to detect the transition of the push button switch from open (not pushed) to closed (pushed).
- You may find it necessary to observe signals with the testpoints macro in more then one macro and the top level schematic. It is not possible to have the testpoints macro applied more then once within a project hierarchy. The best way to handle this problem is to port the signals you wish to observe in your macro with hierarchal connectors. Then simply connect these signals to the testpoints macro on the top level schematic. This way, signals from several macros or the top level schematic can be observed.

Pre-lab Assignment

Questions 1 through 2 are to be done individually. Questions 3 through 5 are to be done with your partner/s.

1. For each of the following data types, write the instruction or instruction sequence that reads a value of that type from memory and places it in a register. Assume that the memory address is in r5 and load the value into r9. (**6 points**)
   a. Signed 32-bit integer
   b. Unsigned 32-bit integer
   c. Signed 16-bit integer
   d. Unsigned 16-bit integer
   e. Signed 8-bit integer
   f. Unsigned 8-bit integer
2. Consider the following program fragment.

```
        .data
        .align 2
data:   .byte 0x99,0x88,0x77,0x66,0x55,0x44,0x33,0x22
        .text
        .align 2
        .global _start
```

```
_start: lis  r3, data@h
        ori  r3, r3, data@l
        lbz  r4, 0(r3)
        lbz  r5, 5(r3)
        stb  r5, 1(r3)
        lhz  r6, 0(r3)
        lhz  r7, 1(r3)
        lhz  r8, 3(r3)
        lwz  r9, 0(r3)
        lwz  r10, 1(r3)
        stw  r10, 3(r3)
        lwz  r11, 2(r3)
```

a. List the values of registers r4 through r11 after executing the fragment. (**8 points**)

b. Which (if any) of these instructions cause unaligned accesses? (**3 points**)

c. For the unaligned accesses, list the sequence of aligned accesses required to fetch the data. You may use generalized notation. For example, a three step access: byte 0x88--> r3+1, 1/2 word 0x7766-->r3+2, byte 0x55-->r3+4 (**3 points)**

d. What values are placed on the data bus for each of the accesses generated by the instruction "stw r10, 3(r3)"? See Table 13-3 on page 13-27 of the white book. (**3 points)**

3. **With your lab partner/s**, fill out Table 3 at the end of the lab document. (**3 points)**

4. **With your lab partner/s**, submit either a hand drawn or sufficiently large Xilinx schematic showing all your data paths between the memory, switches and LEDs and the PROC macro. Label all data paths with bus names. All devices may be shown as macros or boxes. It is not necessary to show control signal paths. On the data paths, indicate where TriState buffers are required. (**3 points)**

5. **With your lab partner/s,** design a memory write decoder that provides the appropriate byte enables for byte 0-3, half word and word writes for a 32 bit data port. Table 13-3 of the White Book shows the external bus contents for the various write combinations. Be careful, Table 13-3 shows the contents of the external data bus for 8, 16 and 32 bit data ports. Submit a simulation of your decoder showing the combinations for TSIZE, A30 and A31 listed in Table 13-3 with the appropriate byte enables activated. Include a schematic or HDL listing of your decoder. (**3 points)**

6. Submit one copy of the documents from steps 3 - 5 with all the partners listed.

In-Lab Procedure

Part I: Debugging the Memory Module

1. Thoroughly test your 128-byte memory module design (not the full 256-byte module) 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. Interface your memory module to the processor and implement your design.

3. Download your design to the Xilinx board. Use the '**read**' and '**write**' commands in the SDS command window to verify byte, half word and word accesses. The '**-b**', '**-**

w', and '**-l**' switches will force read and write to perform byte, half-word, and word accesses respectively. Debug using aligned accesses.

4. 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 logic analyzer to debug your error. Since the logic analyzer is configured to trigger on the falling edge of TS*, you should be able to capture the whole transaction on the screen.

Demonstration 1.1: Using the Logic Analyzer, show your lab instructor that the appropriate data paths are selected in the 128 byte macro for an aligned byte, half word and word write. All byte enable control lines should be mapped to test points. Similarly, demonstrate a read access on the logic analyzer including the appropriate read enable lines or line.

5. Add the second 128-byte module and any additional logic required to interface to the bus. Repeat steps 1-5.

6. You have now several components internal to the Xilinx chip sharing the bus. Make sure that no more than one device drives the internal buses at any one time, and that the external bus is driven (i.e., PD_OUT_EN is asserted) if and only if one of the internal components is driving the internal bus.

Demonstration 1.2: Using the Logic Analyzer, show your lab instructor a read access of the memory with the memory tristate enable, the switches tristate enable and the PD_OUT_EN mapped to test points.

Part II: Debugging the Software

1. Thoroughly test your program using the simulator. As in Lab 3, you can emulate the effects of changing the switches by changing the contents of location 0x02900000 in the memory window. You can then check the value written to 0x02900000 to see if your program is behaving as you expect.

2. Once your program appears to work on the simulator, download it to the processor and trace through it using the debugger.

3. 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. After you modify several locations using the DIP switches, stop your program and read the locations as words from the command window to verify them.

Demonstration 1.3: Demonstrate the software working with your hardware to the lab instructor.

Part III: Observing Aligned and Unaligned Bus Transactions

1. Type the program from pre-lab question 2. Call the file "lab4b.s". Copy a linker command file from a previous lab to lab4b-lnk.txt, edit it to change the address of the data segment to 0x3100000. This will locate the data segment in your memory module. Assemble and link your program.

2. Load the TS_ADDR_PAT Logic Analyzer configuration file. This configuration file is setup to trigger on TS* and address pattern. Make sure the address pattern is set to the memory base address 0x3100000. So that the analyzer will trigger on any address in the memory, set the last two digits in the address pattern to don't cares or XX.

3. Download lab4b.elf to the target system. Verify that your memory module was initialized properly by the download. Observe all the bus transactions that occur as you single-step through the program. Record the number of transactions for each access.

Record the address, size, and full 32-bit values shown on the bus for each transaction. These values will be used to answer questions in the post lab.

4. Single-step through the program. Did the final results match your answers in the pre-lab question 2a?

Demonstration 1.4: Using the Logic Analyzer, show the bus transactions for the write command write -lx 0x3100001 = 0x01234567. How many accesses are required? What are the 32 bit values on the data bus for each access? Do these values correspond to the specifications described in Table 13-3 on page 13-27 of the white book?

Post-lab

1. Summarize the operation of your circuit and describe how you arrived at the design. Include a printout of your final schematic. You may lose points if your schematic is disorganized or unclear. (**6 points**)

2. 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. (**6 points**)

3. As you add more devices and memory modules to your system, it is important to keep track of where everything is and which addresses are available for additional devices. A *memory map* is a diagram that depicts the system address space and indicates the devices or memory modules occupying each address range. Table 2 is a memory map for the existing memory and devices on the target board. All address decoding for these components is done on the MPC823, so all 32 address bits are significant.

   Redraw this memory map, including two new regions: one for your LED and switch registers and another for your memory module. These regions should cover all of the addresses for these components (i.e., including the shadow locations), assuming that A0-A5 are 0. (**6 points**)

Table 2: MPC823FADS Address Map

Address Range	Contents
FFE00000 – FFFFFFFF	Boot memory (Flash)
FF004000 – FFDFFFFF	(not used)
FF000000 – FF003FFF	MPC823 (on-chip) memory and device registers
F0008000 – FEFFFFFF	(not used)
F0000000 – F0007FFF	Board device registers
00400000 – EFFFFFFF	(not used)
00000000 – 003FFFFF	Main memory (DRAM)

4. Report your observations from the HP16601A. What data values are shown on the bus for the stores? Describe how these values relate to the value that is being stored. Does the number of bus transactions that you observed for each access agree with your answers for pre-lab question 2c? (**4 points**)

5. The MPC823 reduces every unaligned memory access to a sequence of aligned bus transactions. If we remove the restriction that bus transactions must be aligned, some

unaligned memory accesses could be completed in fewer transactions (*without* modifying your memory module to read multiple words in one transaction). (**8 points**)

a. Which (if any) unaligned half-word accesses could be performed in fewer transactions?

b. Which (if any) unaligned word accesses could be performed in fewer transactions. What changes would be required to the MPC823 bus to make this feasible?

I/O Control Table 3

Enable Control **Action**	MW B0 EN	MW B1 EN	MW B2 EN	MW B3 EN	MR B0 EN	MR B1 EN	MR B2 EN	MR B3 EN	Switches-TriState Enable	LEDs Register Enable (CE)	PDOUT_EN
Memory Write Byte 0											
Memory Write Byte 1											
Memory Write Byte 2											
Memory Write Byte 3											
Memory Write 1/2 Word B0-B1											
Memory Write 1/2 Word B2-B3											
Memory Write Word											
Memory Read Byte 0											
Memory Read 1/2 Word B0-B1											
Memory Read Word											
Memory Read Byte 3											
Switches Read											
LEDs Write											

Indicate which control signals should be activated for each action listed in column 1. Use the letter '**E**' to indicate Enable. Assume empty cells indicate the correponding control signal is not enabled. MW B0 EN means Memory Write Byte 0 Enable, etc. Assume the memory write enable signals enables a Clock Enable or Memory Write Enable signal. MR B0 EN means Memory Read Byte 0 enable.Assume the memory read enable signals enables a tristate enable for the respective byte wide data path.Assume that BAR and 7Segment Led registers are enabled by LEDs Register Enable.

Lab 4 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 the Memory Module

D1.1 Demonstrate appropriate data paths are selected for byte, half word and word writes.

Lab instructors initials:

D1.2 Verify that there is not bus conflict between the switches and the memory module by observing the tristate enables for memory module, switches and PD_OUT_EN on logic analyzer.

Lab instructors initials:

Part II: Debugging the Software

D1.3 Demonstrate your software and hardware working together.

Lab instructors initials:

Part III: Observing Aligned and Unaligned Bus Transactions

D1.4 Show the unaligned word access on logic analyzer identifying size, data bus values and address bus values for each access.