The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 541 Digital Logic and Computer Design
Prof. Montek Singh
Spring 2020

Final Project (PART A): Attaching I/O Devices to the Processor

Issued Mon 4/6/2020; ***Due Wed Apr 15, 2020, 11:59pm (see Note)***

NOTE: Please aim to complete this part of the project as early as you can. If you can complete it by Wed Apr 15, you will be in good shape to start working on your demo app. Any later than that and you will be quite pressed in time to implement a reasonable app in time for demos (which start Apr 20), given the usual semester-end crunch with all the other courses. Note that there is no lab session on Fri Apr 10 because of a University holiday.

You will learn the following in this lab:

- Designing a memory-mapped I/O system
- Integrating your I/O devices from an earlier lab with the MIPS CPU and memories

---

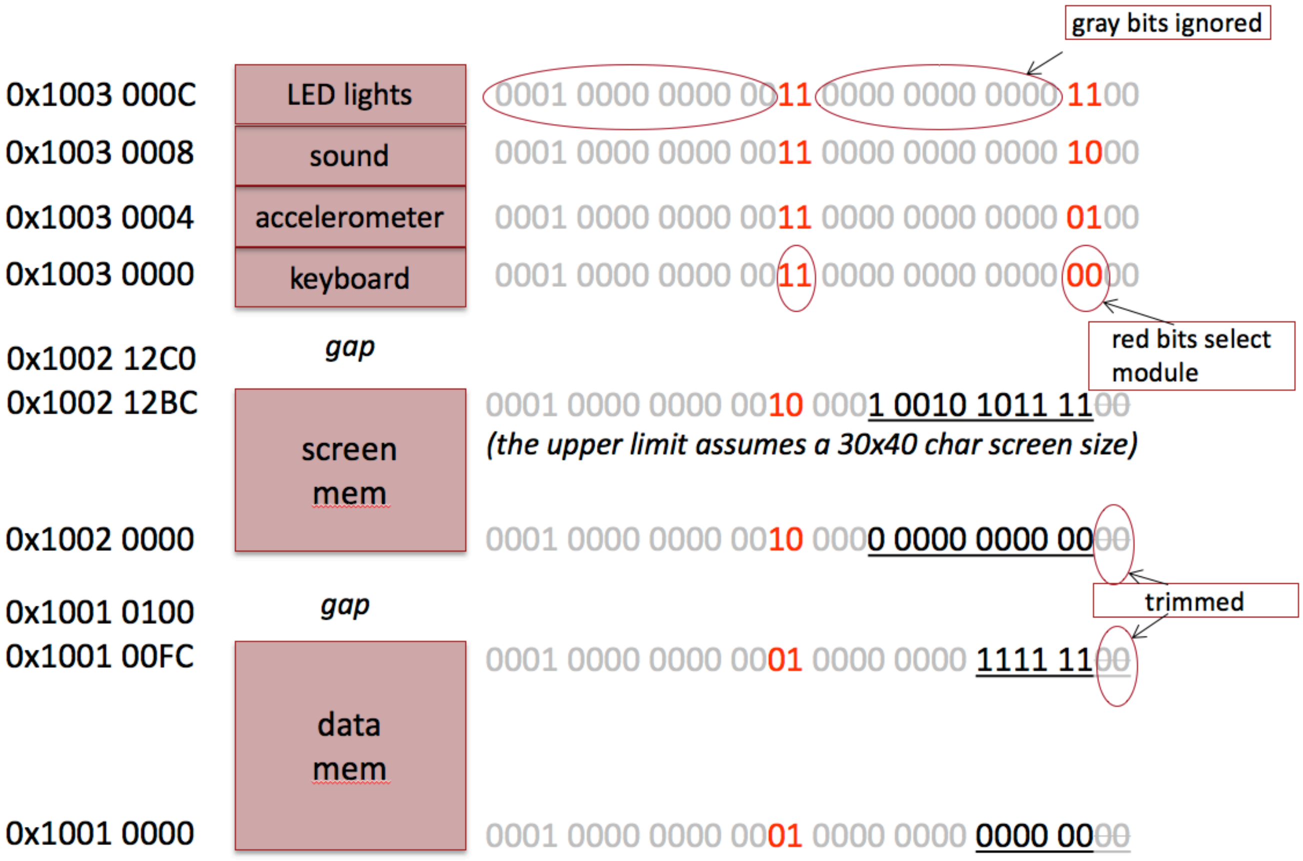
Part 0: Thoroughly test your design of the MIPS CPU from Lab 8

Before you proceed with the tasks below, make sure that your MIPS CPU from Lab 8 is working correctly. The tester provided for Lab 8 should result in all correct outputs (i.e., all ERROR signals should be green).

Part 1: Integrate the CPU and the display unit using memory mapping

As discussed in Lecture 15 (figures reproduced below), you will integrate the CPU and the display using memory-mapped I/O. The desired memory mapping scheme was discussed in class (see next page).

-- over --



gray bits ignored

0x1003 000C LED lights 0001 0000 0000 0011 0000 0000 0000 1100

0x1003 0008 sound 0001 0000 0000 0011 0000 0000 0000 1000

0x1003 0004 accelerometer 0001 0000 0000 0011 0000 0000 0000 0100

0x1003 0000 keyboard 0001 0000 0000 0011 0000 0000 0000 0000

red bits select module

0x1002 12C0 *gap*

0x1002 12BC screen mem 0001 0000 0000 0010 0001 0010 1011 11~~00~~

(the upper limit assumes a 30x40 char screen size)

0x1002 0000 0001 0000 0000 0010 0000 0000 0000 00~~00~~

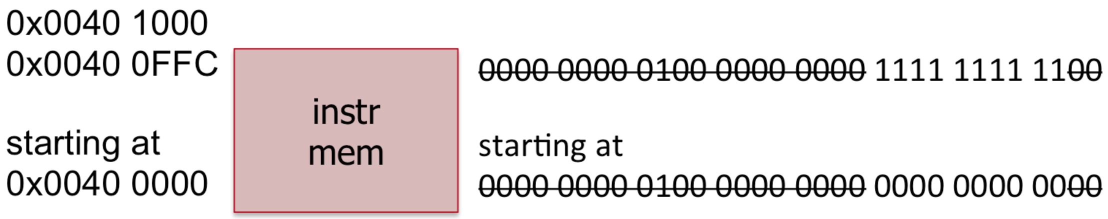
trimmed

0x1001 0100 *gap*

0x1001 00FC data mem 0001 0000 0000 0001 0000 0000 1111 11~~00~~

0x1001 0000 0001 0000 0000 0001 0000 0000 0000 00~~00~~

Assigning the data memory to start at address 0x1001_0000 allows you to use the MARS assembler with the “Default” configuration, which places code at 0x0040_0000.



0x0040 1000

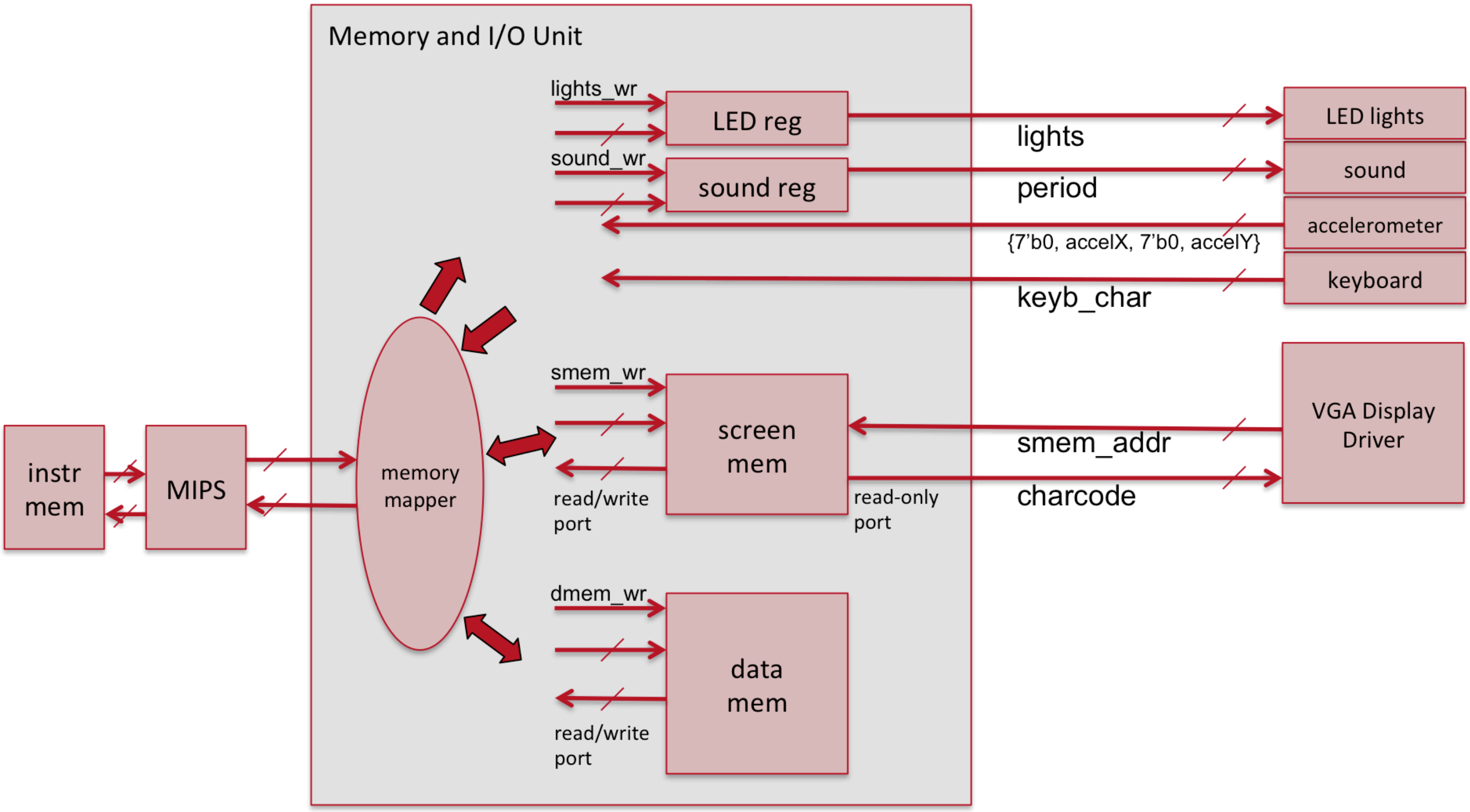
0x0040 0FFC instr mem ~~0000 0000 0100 0000 0000~~ 1111 1111 11~~00~~

starting at
0x0040 0000

starting at
~~0000 0000 0100 0000 0000~~ 0000 0000 00~~00~~

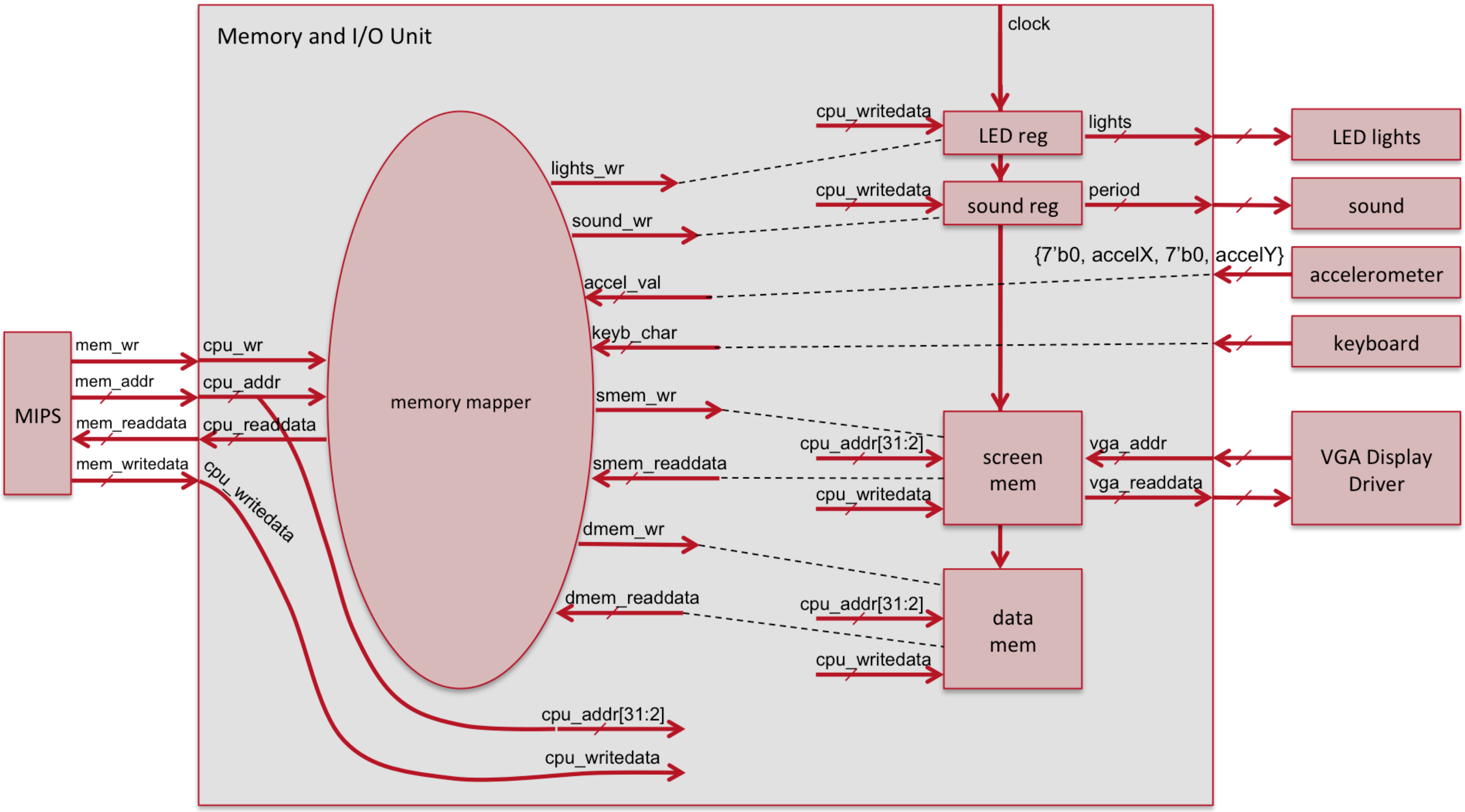
We will simply drop the upper bits of the address coming into instruction, data and screen memories. This happens *automatically* if you use the memory templates provided earlier, which have the number of address bits automatically calculated using the number of memory locations, i.e., *$clog2(NLoc)*.

To implement this memory map, use the block diagram below. (A more detailed figure appears on the next page.) Put the "Memory and I/O unit" in a module called **memIO,** and name the file **`memIO.sv`**. You should start with integrating the screen memory into **memIO,** and then proceed to also integrate the keyboard, accelerometer, sound and LED lights modules (from Lab 5).



NOTE: The screen memory from Lab 7 Part 3 has now been modified to have TWO PORTS (a read-write port on its left side, and a read-only port on its right side), and it is now placed inside the "Memory and I/O unit". You may see "Verilog for Memories" slides for a template for such a 2-port memory. In more detail, the address for the screen memory that is generated by the display driver now goes into the memory-I/O unit through a read-only port shown on the right side in the figure above. This port is distinct from the read-write port used by the MIPS processor. Thus, the screen memory now has two separate interfaces, one to the display driver and another to the MIPS CPU. The idea is that a program running on the CPU can modify the screen memory, which automatically changes what is displayed on the screen by the display driver.

Below is a more detailed version of the figure showing all of the connections between the modules.



Carefully design your *memIO* unit by adhering to the hierarchy and connectivity shown in the detailed figure. You may, of course, choose different names for the wires inside the *memIO* module, as long as you use them consistently.

Note that the name of a wire may be different inside a module compared with outside. For example, the memory address generated by the MIPS CPU is labeled `mem_addr` outside the *memIO* unit, but called `cpu_addr` inside. The name used inside, `cpu_addr`, was conveniently chosen to distinguish it from the other address input to the *memIO* unit, called `vga_addr`, which is the generated by the VGA display driver for looking up the current character code for display. Outside the *memIO* unit, `vga_addr` also has a different name; coming out of the display driver, it is called `smem_addr` (see picture on previous page).

Implement this part on the boards! Write a short program to have your MIPS write characters to *Screen Memory* and see if they show up on the monitor! Sample programs are provided on the website as well. If all goes well, you will see some activity on the screen. And you will have a fully functional computer with I/O.

Good luck!

Start working on your final project demo idea. As explained in class, submit a short proposal on what you would like to implement for your final project demo via Sakai by Wed Apr 8 (before start of class) so we can give you timely feedback.

What to submit:

- **The Verilog sources for the top module (top.sv) and the memIO module (memIO.sv).**
- **In a couple of sentences, state if everything works as you expect, or if there are some problems you still need to resolve.**
- **Show a working demo of your design by Wed Apr 15. Either show it to an LA during office hours, or attach a video to your email submission.**

How to submit: **Please submit your work by email by Wed Apr 15, 11:59pm (see Note on front page):**

- **Send email to: `comp541-submit-s20@cs.unc.edu`**
- **Use subject line: Project PART A**
- **Include the Verilog file as attachment as specified above, and your statement about any unresolved bugs.**
- **Include a video of the demonstration if you have not showed it to an LA yet.**