Lab Assignment 3
General purpose memory mapped I/O on the ZedBoard

The goal of this lab is to provide an introduction to memory-mapped I/O and learn how to use general purpose memory mapped I/O on the ZedBoard. You will learn how to read/write values from/to memory-mapped I/O and see how to work with LEDs, switches, and push buttons on the ZedBoard. This lab may take multiple lab sessions.

Instructions

- Each lab assignment consists of a set of pre-lab questions, the actual time in the lab, and a lab report.
- The pre-lab assignments prepare you for the challenges you will be facing in the actual lab. So that each lab member gets the same benefits, pre-lab assignments have to be solved individually. Feel free to discuss problems with class and lab mates, but complete the assignment on your own.

Reading List

The following reading list will help you to complete the pre-lab assignment. The readings will also help you in subsequent lab assignments. Please complete the readings that are marked ***[Required]*** before attempting the pre-lab assignments.

[1] Materials on memory mapped I/O including
  - http://en.wikipedia.org/wiki/Memory-mapped_I/O
  - http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/IO/mapped.html

[2] Materials on ‘sudo’ used in linux
  - http://www.tutorialspoint.com/unix_commands/sudo.htm

Pre-Lab Assignment

We have only very limited time for the lab available. To make efficient use of the lab time, you will need to prepare for it. First, go through the assigned reading above and get an overview of the schematics and the manuals mentioned. Second, approach the questions below to apply your knowledge. Please submit the solution to the pre-lab assignments in PDF format via blackboard.

Sudo command in Linux

Pre 3.1) What is the purpose of the command sudo? What is the advantage of using sudo, instead of working as the root user?

Memory Mapped I/O

Pre 3.2) In your own words, describe what is memory mapped I/O?

Zedboard's Memory mapped I/O

Pre 3.3) Review the example C program provided in Blackboard.

a. Write a short description of the purpose of this code, and how it achieves this goals.
b. In particular, describe the following functions/macros:
    i. open
    ii. REG_WRITE (pay particular attention to the casting)
    iii. mmap
    iv. munmap

   Please use online resources (e.g. man pages) to guide you in the exploration.

Programming Practice

It is proper programming practice to abstract low-level memory mapped I/O accesses in meaningful functions that ease access to peripherals. Such functions hide addresses and provide a functional meaningful interface.
One example is the function *userio_ledSetAll* that displays the lower 8 bits of an integer value on Zedboard LEDs. Note that the user of this function does not need to worry about addresses and offsets, only the desired value to be displayed.

Pre 3.4) Analogous to *userio_ledSetAll* (which controls all LEDs at once), write a function for controlling each LED individually. The function should have the following interface:

```
void userio_ledSet(unsigned char *pBase, unsigned int
ledNr, unsigned int state);
```

In this function *ledNr* selects LED to control (0 … 7) and *state* controls if LED is off (0), or on (1). For example:

```
userio_ledSet(pMemBase, 0, 1);
```

Should turn on LED 0.
Hints: Look into *userio_ledSetAll,* it writes to the memory location of each LED individually. See the definition of each LEDs offset address (#define at top of the file). What is the address distance between LED offsets, are they constant? Can you

compute an offset based on an LED number (0 … 7)? Compute first the offset for the desired LED and store it in variable `unsigned int ledOffset`.

Note, since you do not have the ZedBoard available during the preLab, we cannot call the REG_WRITE right now. For simplicity, just debug your code and validate that it produces the correct offset address. Just to give you a heads up, later in the lab you will need to add a call to REG_WRITE, setting the memory mapped register as follows:

```
REG_WRITE(pBase, ledOffset, state);
```

Add comments to the function. Describe the arguments and operation performed.

Pre 3.5) Write a function prototype for a function that abstracts access to the switches. This function should work analogous to the `userio_ledSet` function above. Name the function `userio_switchGet().` Show the function prototype with arguments and return type. Document the function prototype.

Lab Assignments

This lab continues to familiarize you with the Zedboard as an embedded computing device and C programming. In this lab you will get more accustomed to working in Linux, write more complex C programs and learn how to call the compiler gcc with some options. Moreover, you will learn different data type manipulation in C and use functions to modularize your code and make it easier to debug.

Connecting to ZedBoard

1. Login into the Windows desktop PC (we will refer to this system as the host) using your myneu credentials.
2. Make sure your Zedboard is powered (plugged in and the power switch on the board is in the ON position), and has been given time to boot up (generally takes a few minutes). Then connect the Zedboard to the PC host via Ethernet to the RJ45 connector.
3. Make an SSH connection to the Zedboard, where the IP address is 192.168.1.10, and port number of 22.
   Don't work as root, you already created your user accounts, use that!

Lab 3.0 Enable your user id to run sudo command

Before starting the lab, you will need to have root privilege to access the registers on the system. Instead of running your assignment as root. Let's enable your group userid to run commands using sudo:

4. Log into the ZedBoard as *root*, password *root*.
5. Add your user to the sudo group to allow sudo access. Issue the command (replace <userName> with our group's user name):
   ```
   adduser <userName> sudo
   ```
6. Now log off and log in with your non-root account.
7. During this lab, to run any of the programs, you will need to prefix them with sudo:
   ```
   sudo <programName>
   ```
   This executes <programName> with root privileges.

   If for some reason your run into a permission problem with sudo, login again as root and issue the following command: `chmod 4755 /usr/bin/sudo`.

Lab 3.1 Controlling LEDS and complete the implementation for `userio_ledSet()`

8. Start with the skeleton code (Lab3-skeleton.c). Compile and run it. Validate the number entered is displayed on the LEDs. Record in your lab report, which numbers you have tested, with the expected and observed outcome.
9. Expand the skeleton code with the function for setting an individual LED. Name the C file "lab3-singleLed.c". Add the function userio_ledSet() from prelab 3.4. Add the call to REG_WRITE setting the memory mapped register like:
   ```
   REG_WRITE(pBase, ledOffset, state);
   ```
   Note: the LEDs are only writable – their values are not readable since they are only output devices. Similarly, the switches and pushbuttons you will use in the next section are only readable – they are defined as input devices.

10. Modify the main function to ask the user for an individual LED to be turned on/off, and call userio_ledSet() accordingly.
11. Document your code. In particular, document the `REG_WRITE` macro. What type of conversions happen?

Lab 3.2 Reading the switches and reflecting the values on the LEDs

12. Copy your program “lab3-singleLed.c” into “lab3-SwitchToLed.c”.
13. Implement the function `userio_switchGet()` you defined in the pre-lab assignment. A switch that is pushed toward the LED is treated as “ON”. Note that the address offsets of the switches are defined in the program. You can read the value of each switch using its offset and REG_READ.
14. Expand the sample program such that it will read the present value of the switches on the ZedBoard and show the value on the LEDs. Use the two new access functions userio_switchGet and userio_ledSet to realize the functionality. Note, write the main as such that the program repeatedly reads the switches and outputs to the LEDs.

Lab 3.3 Push button controlled step counter

15. Copy your program “lab3-SwitchToLed.c” to “lab3-pushButton.c”.
16. Implement the function `userio_pushButtonGet()`. The function should read the five pushbuttons. Assuming you are viewing the ZedBoard with the switches in the lower right corner, the top and bottom buttons are identified as PBTNU (“up”) and PBTND (“down”), respectively. The right and left buttons are identified as PBTNR (“right”) and PBTNL (“left”), respectively. The center button is identified as PBTNC (“center”). `userio_pushButtonGet()` should return 0, if no button is pushed. 1 “up”, 2 “right”, 4 “donw”, 8 “left”, 16 “center”. Check the push button based on their numerical order. Return from the function as soon as one pushed button is found. The memory location for a push button reads 1, if the button is pushed. When you release the button, the value will be 0 again. Provide new #defines (PBTN_UP, PBTN_DWN …) to simplify access to these values.
17. Validate the implementation of `userio_pushButtonGet()` by outputting the value to the LEDs (see previous step).
18. Implement a new C program called PushButton.c that will begin by reading the value of switches, and then use them as an initial counter value. Display the counter value on the LEDs. Then, then use the five push buttons to control the value shown on the LEDs:
    - PBTN_UP increases the counter
    - PBTN_DWN decreases the counter
    - PBTN_CNTR loads the counter with the current switch values.

    An overflow will rollover to zero or 255 for the values 255 or 0, for increment or decrement, respectively.

    ➢ Note that reading from the push buttons will be tricky. Only increment / decrement once per button push (e.g., pushed to not pushed). Hint: store the last value for the button status and only go through the action evaluation once per change of the button status.

Lab 3.4 Left/Right shift of push buttons

19. Next, copy PushButton.c to a new program named Shift.c. Expand the capabilities of PushButton.c In this new program, pushing PBTN_RGHT will shift the value to the right one position, and PBTN_LEFT will shift it to the left one position.

Lab 3.5 Automatic Counter controlled by push buttons

20. Copy your program "lab3-pushButton.c" to "lab3-pushCounter.c"
21. Use the framework developed in 3.3 and copy it into a new program Counter.c. The initial value of the counter will be assigned by switches. The push buttons will control the direction and speed of counting. PBTN_CNTR starts or stops counting. PBTN_UP and PBTN_DWN define the direction of counting. PBTN_RGHT increases the counting speed, and PBTN_LEFT decreases it. Note that the speed should be relatively slow at the beginning (maybe 2 second delay between each value, you can generate the delay using the sleep() function).

Extra credit

22. Discuss a procedure to handle when multiple buttons are pushed simultaneously. What is the desired behavior? Reason about your design choice. What needs to change to realize the desired behavior?

Laboratory Report

Your report should be developed on a wordprocessor (e.g., OpenOffice or LaTeX), and should include graphics when trying to present a large amount of data. Include the output of compiling and running your programs. Upload the lab report on blackboard. Include the programs you develop in the appendix of your lab report.