ECE 243S - Computer Organization
February 2024
Lab 6

Coding in C and Audio Input and Output

Due date/time: During your scheduled lab period in the week of February 26, 2024. [Due date is not when Quercus indicates, as usual.]

Learning Objectives

The goal of this lab is to become comfortable with using the C programming language to do "embedded system" input/output, which you have been doing up to now using the Nios II Assembly Language. We will begin my doing something very similar to previous labs, except in C. Then we'll move on to a more complex input/output device: sound input and output. You will see a relationship between concepts needed here and those in two other courses in second year ECE: ECE 216 Signals and Systems, and ECE 231 Introductory Electronics. You'll be exposed to hardware-level detail required to link audio input and output in software - in lectures and in this lab.

What To Submit

You should hand in the following files prior to the end of your lab period. However, note that the grading takes place in-person, with your TA:

- The assembly code for Parts I,II, III and IV in the files `part1.c`, `part2.c`, `part3.c` and `part4.c`.

Part I

Recent lectures have described how to access memory-mapped input/output devices in the C language, using C pointers. The only other thing you'll need to know to link your use of memory-mapped registers in Nios II assembly language and the C language, are the The C bit-level logical operators, which relate to the use of the assembly `and, or, xor, srl, sll` instructions:

- logical `AND` is done using the operator
- logical `OR` is done using the `|` operator
- logical `XOR` is done using the `^` operator
- logical `NOT` is done using the `~` operator
- shift left is done using the « operator
- shift left is done using the » operator

You can see examples of how these are used here: https://www.programiz.com/c-programming/bitwise-operators

Also, one other thing: recall that pointer arithmetic in C takes into account the byte-addressability of the item being pointed to, and its size in bytes. If you want to de-reference a pointer to an integer with an offset, the offset must be in words, not bytes. For example, in the following code we add 3 to the base pointer of KEYs to access the memory mapped edge capture register at address `0xFF20005C`, rather than 12_{10} as we did in the related assembly code.

```
volatile int *KEYs = 0xff200050
int edge_cap;

// getting the KEYs edge capture register into the variable edge_cap:

edge_cap = *(KEYs + 3);
```

The task of Part I: Write a C-language program that turns on all ten Red LEDs when button KEY_0 is pressed and released, and turns them off when KEY_1 is pressed and released. The program should run continuously, responding to either one of those Keys being pressed and released. The program should use the edge capture register of the Key pushbuttons parallel port to determine when one of those Keys is activated. You should use your knowledge from the previous labs to make sure this works. **Do not** use interrupts, just use polling on the edge capture register.

Create a new folder to hold your solution for this part and put your code into a file called `part1.c`. You will need to make a Monitor Program project for running your code on a DE1-SoC board; but you can also debug your program at home using CPUlator. Show it working to your TA for grading in the lab. Submit `part1.c` to Quercus before the end of your lab period.

Part II

Write a C-Language program that takes the input from the microphone jack on the DE1-SoC board, and connects it to the output speaker jack on the board, using the audio input and output system described in class. From class, you will know that this involves collecting samples of the input from the input FIFO of the audio interface and sending them to the output FIFO.

You will want to test this program using CPUlator, no doubt! However, this is one of those times where CPUlator is not a perfect replication of the hardware. In particular, it isn't capable of using the microphone on your computer to act as the input microphone. So, instead, when you access the samples from the CPUlator-based input FIFOs, you will get a simple sawtooth wave, as described here: https://cpulator.01xz.net/doc/, which says "Audio recording is functional, but will record a sawtooth wave with period 32 samples (250 Hz at 8 kHz sampling rate)." One other thing to make sure you get right here is to use the right version of CPUlator, which uses the 8khz sampling rate, and it is here: https://cpulator.01xz.net/?sys=nios-de1soc.

Create a new folder to hold your solution for this part and put your code into a file called `part2.c`. Show it working to your TA for grading in the lab. Submit `part2.c` to Quercus before the end of your lab period.

Part III

Write a C-language program that will generate a square wave to go to the audio output device (no input). Make it so that the frequency of the audio output is changeable (across the audible range from about 100Hz to 2KHz) using the 10 switches on the DE1-Soc to create a fine-grained selection of the frequency.

Create a new folder to hold your solution for this part and put your code into a file called `part3.c`. Show it working to your TA for grading in the lab. Submit `part3.c` to Quercus before the end of your lab period.

Part IV

Write a C-language program, similar to the one in part II, except that it acts as an echo - as if you were speaking/yelling into the a large canyon, as shown in this video: https://www.youtube.com/watch?v=SCUUu96PYjo which shows the distinction between "reverb" and "echo." (We want echo in this case). See the lecture notes that describe, in general, how this might be done. We suggest a delay of roughly 0.4 seconds. You should experiment

to find a good damping amount.

Create a new folder to hold your solution for this part and put your code into a file called `part4.c`. Show it working to your TA for grading in the lab. Submit `part4.c` to Quercus before the end of your lab period. Show your TA that you have this program working.