Lab 8 – ADCs and Acceleration

Overview

In this lab you will use the built in accelerometers to build a program that lets the user move a red LED dot around the array based on tilting the Wunderboard. A random green LED will be turned on and the user will try to get the dots to be the same place. Once they are at the same place, the green dot is randomly moved to another location.

Prelab

1. What is acceleration compared to velocity?

2. When talking about Analog to Digital Converters (ADC) what is meant by the 'number of bits?'

3. Looking at the skeleton files supplied for the lab, how many bits of resolution does the **`read_adc()`** function return? (it is in the adc.h and adc.c files)

4. Subtle changes (shaking) of the Wunderboard will cause unwanted accelerations to occur possibly making the LED 'jump around.' How might you remove these random accelerations using software?

Procedure

Task 1

The first task is to learn to use the read_adc() function. The read_adc() function is contained in two separate file, adc.h and adc.c The '.h' file is a header file. in C, header files contain constants, #defines, comments, and function prototypes. The .c file is the actual source code for the adc functions. These files need to be compiled just like your main file as your main file is 'dependent' on them. Luckily, the Makefile provided checks these dependencies as complies as needed. You only have to type **make all**. To understand the functionality of the read_adc() function, review the comments in the .h and .c files.

Your first task will be to display the accelerometer data from the X axis on one of the columns on the LED array. You get to choose the column number and the LED colors. The data should be displayed in binary. Once completed, show your TA. An example image is shown in figure 26.

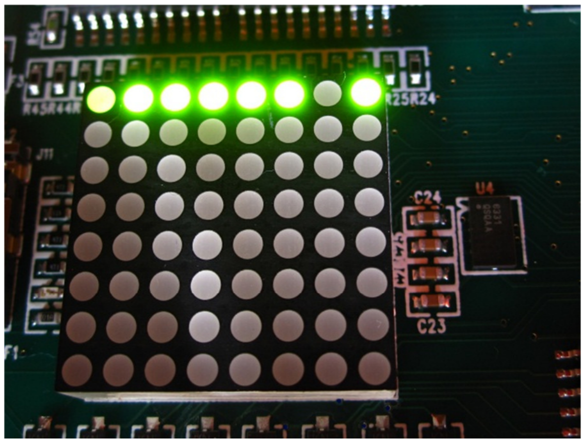
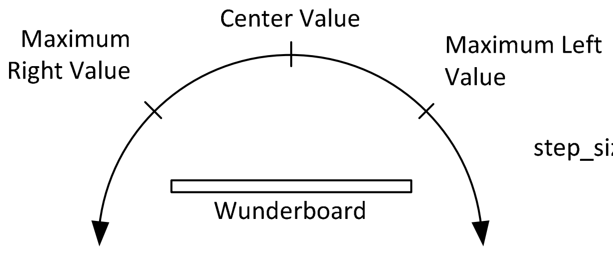


Figure 26: Example Acceleration Data

Task 2

For this task, you will need to make a single dot illuminate based on how much tilt is applied to the Wunderboard. Start by setting your Wunderboard on a flat level surface while it is running the program from task 1. Record the number it is displaying, this will be your 'center.' Now tip the Wunderboard in the X direction some comfortable amount (no more than 45 degrees). The farther you tip it the easier your coding will be, but don't go more than 45 degrees. Record this new number. This will be the maximum deviation.

Once you have these two numbers, come up with how big each step will be mathematically. A step is how much the accelerometer must change before the LED that is illuminated is changed. Figure 27 shows a number line of a hypothetical case. Once you have figured out the size of each step, write code that will display only one LED at a time based on the tilt of the Wunderboard. An example video link can be found on the lab webpage. Once completed, show your TA.



$$\text{step\_size} = \frac{\text{(Maximum Right Value)} - \text{(Maximum Left Value)}}{8}$$

Figure 27: Computing the Steps

Task 3

This final task is to build the game. You have practiced working with the X axis, how you need to develop the Y axis and have it change which column is turned on rather than which LED is turned on in a row. You may want to calibrate the Y axis as well (like you did in task 2).

The video on the course webpage shows the game you should build. You need to randomly turn on one of the LEDs on the display. The user then uses the accelerometer to move the 'player' to where the random LED is. Once there, there should be a short delay and the random LED that is on should change and the user must move the player again.

Demonstrate and Submit code

When your code is submitted, it will be processed both to ensure it compiles and runs correctly and to evaluate its comments.

When ready, submit your source code to TEACH. Your c file should be named main.c

TEACH Assignment Name: **Lab8**

Study Questions

1. In your own words, describe what the following assignments statements would do and give the value assigned.

    a. X = (1<<4);

    b. Y = !(1<<0);

    c. Z = ~(1<<6);

Extended Learning

The extended learning for this week is to build a VU meter that uses the Wunderboard, microphone to measure the intensity of sound. To do this, you will again use the **`read_adc()`** function but instead of reading the accelerometer, you need to read the channel that is connected to the microphone. The microphone is on channel 4 (this happens to be pin 4 on port F, PF4). Similar to the accelerometer, you just find the value from the microphone for a quiet room and for a very loud room. The VU meter should have one LED on, and as the volume of the noise increases, more LEDs turn on in order until all 8 would be on for the loudest noise. The lowest 4 bits of the VU meter should be green, the next 2 LEDs should be yellow, and the final two (when it is loudest) should be red.

Once completed, show your TA. An example video link can be found on the lab webpage.

Demonstrate and Submit code

When your code is submitted, it will be processed both to ensure it compiles and runs correctly and to evaluate its comments.

When ready, submit your source code to TEACH. Your c file should be named main.c

TEACH Assignment Name: **Lab8extra**

Lab Summary

Task	**Completed?**
Prelab	4pts.
Task 1	1pts.
Task 2	2pts.
Task 3	7pts.
Study Questions	4pts.
Extended Learning demo and upload file (Optional)	