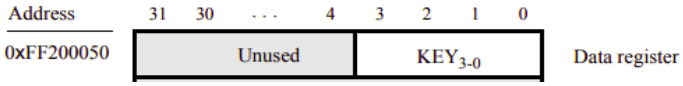
Lab 2: Memory-Mapped I/O Device Programming in C

The purpose of this lab is to interface with memory-mapped input/output (I/O) peripheral devices, such as the LEDs, Keys, and 7-segment displays on the DE10-standard development board. You will write and test C programs for the ARM Cortex-A9 processor on the DE10-standard board. Before starting Part I of the lab assignment, you should have finished the pre-lab reading from Lab 1: DE10-Standard Computer System User Manual Section 2.1 & 2.10.1-4.

Part I

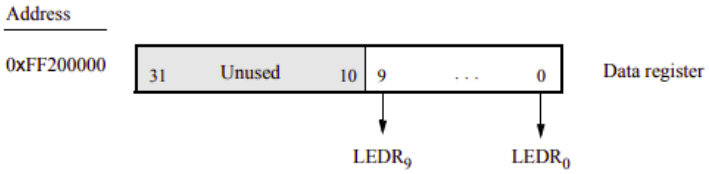
Start the Intel Monitor Program and open your Lab1 project. Study getting_started.c carefully to understand the program. In a text or word file, write your answers to the following questions.

a. How would you check the status of KEY0 (i.e. BIT0 of the following register)?



b. How would you slow down the speed of rotating the LEDRs?

c. How would you turn off all the LEDRs when the program starts?



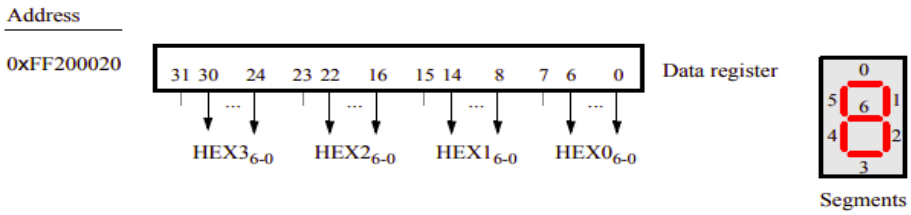
Part II

It is highly recommended that a new folder and Intel Monitor project are created for each part of the assignment. Make a new folder for this portion of the lab (e.g. Lab2_2) and copy getting_started.c and address_map_arm.h from your Lab1 folder to the newly created folder. Rename getting_started.c to Lab2_2.c. Make a new Intel Monitor Program project in the folder where you stored the copied file. During the process of creating a new project, select "C Program" as the Program Type and add Lab2_2.c to your project (only C code should be added to project; other supplementary files (.h) stay in the project folder). On the top of your program, write a comment line "Lab 2 Part2 by <your name>". You may also add other things, e.g., functions of the program, etc. Please do so for all your future programs. It is common practice in industry to have a header section (comments) at the top of every program. All your programs must be compiled and tested on the DE10 board before submitting.

Modify Lab2_2.c to perform the following. Refer to section 2.10 in the *DE10-Standard Computer System User Manual* for register information for the 7-segment LED HEXs, LEDRs, and switches.

a. Instead of setting the LED pattern when any key is pressed, make it do so only when KEY0 is pressed.

b. Add a volatile int pointer to the HEX0 base register (Check address_map_arm.h for the name of the register), similar to the SW and LED pointers.
   Use an array, e.g. char seg_bit[16], to store the corresponding bit patterns of a 7-segment LED display HEX for the binary value represented by SW3-0. See below for the bit pattern of the data register. Setting a bit to '1' will light the corresponding LED segment. For example, the bit pattern when SW3-0 is set to 0011 is 0b1001111, which is saved in seg_bit[3]. HEX0 uses the first 7 bits (bits 0-6) of the data register.
c. When KEY1 is pressed, display the HEX value of the SW3-0 on the 7-segment LED HEX0. For example, if the binary value of SW3-0 is 1010, HEX0 should display A. All other HEXs, i.e. HEX5-1 should be off. HEX values stay when KEY1 is released.



Part III

Using getting_started.c as a reference, write a C program to sweep through the LEDRs, with only one LEDR turned on at any time. First, the rightmost light LEDR0 should be on, then LEDR1, then LEDR2, and so on. Turn on one LEDR light at a time. When you get to the leftmost light LEDR9, the direction should be reversed. Use a variable to keep track of the direction. Only one LEDR light is ever on at one time. The effect should be a single light sweeping from right-to-left, then left-to-right, and so on. Use a proper delay (e.g. delay_count = 1,000,000) so that the light moves at a reasonable speed.

Prelab reading: None

What to submit:

For Part I, submit the text file with your answers. For the other portions of the lab, zip the entire folder of each step and submit the zip file in Blackboard.