E85: Digital Electronics and Computer Engineering
Lab 7: Simon

Objective

The purpose of this lab is to write embedded software using general-purpose I/Os. Specifically, you will write a C program to play the game of Simon on a Nucleo.

1. System Requirements

Your system should have two LEDs and two pushbutton switches on a breadboard wired to your Nucleo with appropriate resistors. Test the pushbutton with an ohmmeter to determine which pins get connected when it is pressed. Use 3V3 from the Nucleo board to power your circuits so that your circuits automatically power off when the Nucleo is unplugged. You may pick which pins you want to use, but avoid D4 and D5, which are bridged to other pins on the board.

Your system should flash the LEDs in an interesting predetermined sequence at a reasonable speed, and then check that you press the switches in the corresponding sequence. The sequence will start with length of 2, and increase in length by 1 each time you correctly play back the sequence until the length reaches a maximum of 12. If you play the sequence incorrectly, the game will revert to the length 2.

The user should be able to hold a pushbutton down for any length of time, then release it. Remember that a switch bounces open and closed for a few milliseconds when it is pushed or released. If you sample the switch too fast, you may capture the bounce and misconstrue it as a very fast button push. You can avoid this problem by sampling slowly enough that you never take multiple readings during the bounce interval, yet fast enough to never miss a real push.

Write your code using pointer reads and writes directly to memory-mapped GPIO. Do not use higher level libraries such as EasyNucleoIO in this lab.

Here's some sample code to configure the LED pins as outputs

```
// LED0 on pin D0, which is GPIO A10
#define LED0 10
// LED1 on pin D1, which is GPIO A9
#define LED1 9
// PB0 on pin D2, which is GPIO A12
volatile unsigned long *GPIOA_MODER = (unsigned long*)0x48000000;
*GPIOA_MODER |= (1 << (LED0*2)) | (1 << (LED1 * 2)); // note |=, not =, to not disturb other PORTA pins
```

Note that the Nucleo board uses certain GPIO pins to talk with the debugger. If you change the pin modes for these pins, you'll not only break communication with the debugger, but remarkably also brick the board so that it ceases to function. Therefore, never change the pin mode on a pin you aren't specifically using. Use the |= command to make a particular pin an output without disabling the modes of other pins.

If you are unable to communicate with an STM board using the debugger over the USB cable, you may be able to restore communication with the following procedure:

In "Options for Target," click on the Debug tab, then next to the ST-Link Debugger, click on Settings. In the settings, look for the Debug pane, and change Connect to "under reset" and Reset to "sysresetreq." For more information,

http://www.keil.com/support/docs/3774.htm

2. Extra Credit

Add a feature to make your game more fun. For example, vary your sequence unpredictably each time the game starts, or add LEDs for winning and losing.

What to Turn In

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. Schematic of the circuit on your breadboard, including which Nucleo pins are connected.
3. C code for your Simon program.
4. Does your Simon program work? Can you play it all the way to the length 12 sequence?
5. Extra credit, if applicable.

Please indicate any bugs you found in this lab manual, or any suggestions you would have to improve the lab.