# 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 SparkFun RED-V development board.

## 1. System Requirements

Your system should have two LEDs and two pushbuttons. It should flash the LEDs in a hard-to-remember 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 sequence.

Begin by sketching a schematic of your circuit on a piece of paper. The schematic should include two LEDs and two pushbutton switches on a breadboard wired to your RED-V board with appropriate resistors. Remember that floating is not the same as logic 0. Test the pushbutton with an ohmmeter to determine which pins get connected when it is pressed. Use 3V3 from the RED-V board to power your circuits so that your circuits automatically power off when the RED-V board is unplugged.

Before doing any wiring, remove any 5V power to the red power busses that you might have wired in lab 1, and connect the 3V3 to the red power busses on the breadboard. 5V could damage the Freedom E310 pins, so never connect anything to 5V.

Write your code using pointer reads and writes directly to memory-mapped GPIO. Do NOT use higher level libraries such as EasyREDVIO.h in this lab, and do not use digitalRead or digitalWrite. Consult the FE310 User Manual on the class web page for GPIO addresses.

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

```
#define LED0 4      // LED0 on GPIO 4
#define LED1 2      // LED0 on GPIO 2

// Output enable register
volatile unsigned long * GPIO0_output_en = (unsigned long*) 0x10012008;
// Note |=, not =, to avoid disturbing other GPIO pins
*GPIO0_output_en |= (1 << LED0) | (1 << LED1);
```

The user should be able to hold a pushbutton down for any length of time, then release it. Remember that a switch sometimes 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.

If you think a RED-V board is damaged, please put a note on it with the symptoms (so nobody else has your same problems) and report it to the teaching team.

### 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 RED-V board 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.