Lab 2
Sequential Verilog

Prof. Kredo

Due: By 23:59 Friday, February 27

Introduction

In this lab you will accomplish several goals:

- Develop sequential Verilog modules
- Verify module operation by writing testbenches
- Test your design on the DE1-SoC board

Lab 2 must be done individually.

Do not divide clock.

Requirements

Your objective is to build a sequential circuit that allows the user to specify the contents of a 128-bit (16 byte) register through the use of the switches, buttons, and 7-segment LEDs on the DE1-SoC board. A user modifies and controls the display of the register by moving a virtual window across the register. The system only displays data and only changes data within the window.

Data Display

The system displays the portion of the register in the window on the 7-segment LEDs. Since there are six 7-segment LEDs, the system can display a total of 24 bits (3 bytes) of the register at a time. For any window, HEX 5 and HEX 4 display the most significant (highest numbered byte) of the window and HEX1 and HEX0 display the least significant byte of the window. Upon reset, the window should start in the right-most position and display bytes 2, 1, and 0.

To adjust the data display, the user presses the Left or Right buttons, which causes the window to move over the register to the left or right, respectively. For example, if the current window displays bytes 7, 6, and 5, then the window should change to display bytes 8, 7, and 6 when the user presses the Left button or change to display bytes 6, 5, and 4 when the user presses the Right button.

If the user presses the Right button when the window is at the right-most position (currently displaying bytes 2, 1, and 0), then the button press is ignored.

The left-most position corresponds to the case when the most significant byte of the register (byte 15) is displayed on HEX1 and HEX0. If the user presses the Left button in this position, then the button is ignored. When the window does not overlap with the register, then it should display a blank (all segments off). For example, if byte 14 is displayed on HEX1 and HEX0 and byte 15 is displayed on HEX3 and HEX2, then blank should be displayed on HEX5 and HEX4 since they do not overlap with the register.

Additionally, any time the contents of the register have not changed since reset, the display should read `--` (two minus signs). Thus, the display should read `--` for every **byte** in the register until that **byte** has changed. Each byte changes independently, so you will have some bytes that display a value and some bytes that remain `--` since they have not changed.

Data Change

The user changes the data in the register by using the switches and the Capture button. When the Capture button is pressed, the value currently on switches 7 through 0 is stored into the right-most (lowest number) byte in the window. For example, if the current window displays bytes 12, 11, and 10 as `45 E2 09`, the switches contain the value 0x11, and the user presses the Capture button, then the value 0x11 should be saved to byte 10 of the register and the display should change to `45 E2 11`. As another example, if the current window displays blank (the window is past the register), byte 15, and byte 14 as `|| -- --`, where `||` are blank displays, the switches contain the value 0xC2, and the user presses the Capture button, then the value 0xC2 should be saved to byte 14 of the register and the display should change to `|| -- C2`.

I/O

Use the following buttons for the system operations.

Action	**Key**
Capture	KEY3
Left	KEY2
Right	KEY1
Reset	KEY0

The DE1-SoC board contains a 50 MHz clock for use in your system. **Do not divide or modify the clock.** Your system should run with a single clock running at 50 MHz.

Testbench

For each module in this lab, you must write a testbench that verifies the functionality of the module. You do not need to test every possible input combination (for example, testing all 256 combinations of data from the switches), but your testbenches should thoroughly test your modules. When writing your testbenches, follow the guidelines below.

- Include an error signal that your testbench raises whenever an error occurs.
- Clearly comment your testbench code to document the exact behavior you are testing.
- Ensure your tests cover all paths through your module description.

Your top-level module should contain very little or no logic. For this lab, you do not need a testbench for your top-level module.

Evaluation

Submit your assignment by creating a project archive in Quartus II by selecting 'Archive Project...' from the Project Menu. Submit only your project archive (qar) file. Before archiving your project, ensure the 'File set' is 'Source Control' by clicking the 'Advanced...' button in the Archive Project window.

Your lab will be evaluated based on:

- 20 points: Display of register on 7-segment displays
- 20 points: Data changes with Capture button and switches
- 20 points: Movement of window with Left/Right buttons
- 10 points: Blank displayed when window leaves register
- 10 points: `--` displayed unless byte changed
- 20 points: Testbenches

Hints

- Test and debug in steps. Start with a subset of the lab requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts. In general, the Evaluation requirements are ordered in a way to ease this implementation process.
- Write your testbenches early and verify each module works before integrating it into the system and before programming your board.
- Altera has many free online video tutorials to help learn the Quartus II software suite.

Synopsis Constraints File

Put the lines below in a text file with the extension sdc, such as lab2.sdc, to inform Quartus about the clock in your system. Replace `YOUR_CLOCK_NAME` with the name of your clock signal in your top-level module.

```
create_clock -period 20 [get_ports YOUR_CLOCK_NAME]
derive_pll_clocks
derive_clock_uncertainty
```