Fall 2022 / EC311
Lab 2: Design a counter, debouncer and 7-segment display driver
Due Date: November 4, 2022 for Part 1, November 18, 2022 for Part 2

1. Goals

- Design of simple sequential circuits.
- Implementation of your designs on an FPGA

2. Overview

This lab will introduce you to sequential logic and it's Verilog implementation. Sequential logic, unlike combinational logic, is dependent on an external trigger. A clock is a commonly used as external trigger as that allows all parts of the system to be synchronized.

3. Part 1 – Counter + Debouncer (15 points for debouncer + 15 points for counter)

Design and implement an 8-bit counter in behavioral Verilog using push-buttons on the Nexys-A7 FPGA board. Everytime you push a button, the counter should increment by 1 unit. Pushing the button may last several clock cycles, thus the counter may count several times for a single push. To prevent this, you will need to write Verilog code to debounce the push-buttons of the FPGA board. The debouncer should be used to count 1 unit no matter how long the button is pressed.

3.1. I/O Requirements

The module containing the counter and debouncer must have a clock, active-low reset and increment as input signals, and it should output the 8-bit counter value. The increment and reset inputs must be mapped to push-buttons. The output should be displayed on the board LEDs. Use binary encoding when displaying the numbers using LEDs.

3.2. Components

Your design should have 3 modules:

1. debouncer - The decouncer module is used to clean noisy signals from the push button inputs. Pressing a push button may not lead to clean transitions from 0 to 1 or 1 to 0. The switch may mechanically bounce between the two states. A debouncer circuit will detect transitions and prevent rapid switching between 1 and 0.

   Implement a basic debouncer using a debouncer-counter (which is dedicated to the debouncer module) that increments at every clock cycle after the push button was pressed. Once the debouncer-counter has reached its maximum value, the state of the push-button should be changed and we reset the debouncer-counter. This technique utilizes the fact that our global clock is significantly faster than the time it takes a person to press the push-button repetitively. The debouncer module has three input signals – global clock, reset and button_in. The output of the module is the button_out signal. A sample algorithm for implementing the debouncer is shown in Figure 1 below. Test your design using test bench.

2. counter - The counter module has three input signals: global clock, reset and increment. The output of the counter module should be an 8-bit count value. At the positive edge of the clock, if the increment signal is high then the counter is incremented. At the negative edge of reset, the counter should be reset to zero. Test your design using test bench.

3. top - In the top module instantiate the debouncer and counter modules. Connect the modules as shown in Figure 2. Demonstrate the working top module on the FPGA to the TAs. Refer to Pre-Lab B and the Vivado tutorial for directions on implementing the design on the FPGA.

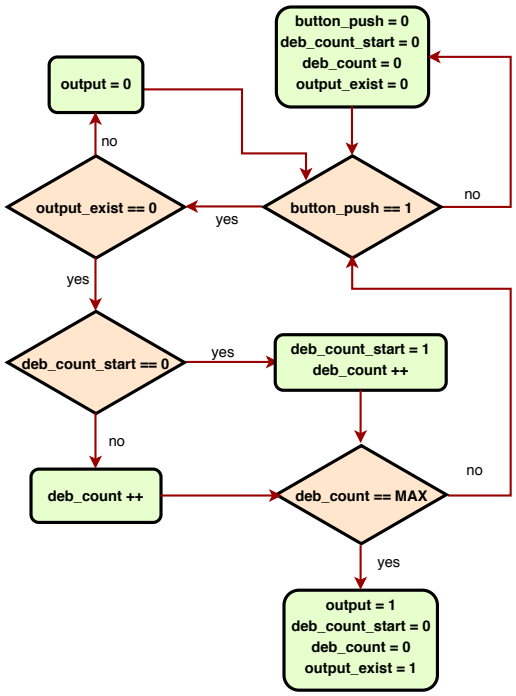

Figure 1: Sample debounce algorithm.

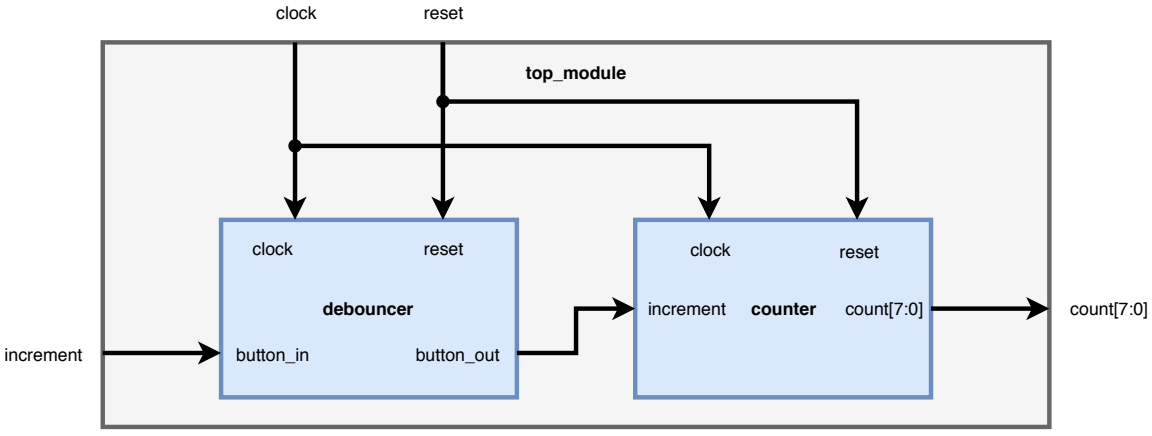

Figure 2: The overview of top module.

4. Part 2 – Counter + Debouncer + 7-segment Display Driver (50 points)

Design a 7-segment display driver to display the results of the counter from part 1 on the FPGA board's 7-segment display[1]. The design will have two modes:

1. Automatic counting: the counter should increment automatically every second.
2. Manual counting: the counter should increment on the push of a button.

You will need a control switch and a multiplexer to enable selection and usage of one of the two modes. Your design will use clock dividers to create multiple clock domains:

1. Slow clock (1Hz): to use as an increment signal in the automatic counting mode.
2. Fast clock (1kHz): to iterate through different 7-Segment displays.
3. Normal clock (100MHz): to control everything else (debouncer, clock divider, etc).

4.1. I/O Requirements

The top level module must have clock, reset, mode_select and increment signals as input signals. The top level module must output a 4-bit digit_select signal to select the digit to light up (only 1 digit can be lit at a time) and the 7-bit encoded digit display value. The increment and reset inputs must be mapped to push buttons. The mode select must be mapped to a switch. The encoded digit value must be displayed on the board's digit display.

4.2. Components

Your design should have 6 modules:

1. seven_segment_decoder - A 7-Segment Display digit is composed of 7 LEDs (marked CA, CB, ..., CG on your board). The four 7-segment displays on the Nexys-A7 board are common anode, meaning that the LEDs will light up when you set the input to 0 and the LEDs will turn off when you set the input to 1 (active low inputs). The four 7-segment displays are controlled by the corresponding active low AN line (AN0, ..., AN3). This AN line is named display_select in the block diagram below. Only one AN line should be set to 0 at a time, such that only one of the four 7-segment displays is ON at any given time. If several display_select bits are low at the same time, multiple 7-segment displays will be lit up with the current encoded value because all four 7-segment displays are connected to the same 7 encoded outputs (named "seven" in the block diagram) from binary to 7-segment decoder.

   For this module you must implement a binary to 7-segment display decoder. The input signal is a 4-bit binary number (representing one hexadecimal digit), and the output is the seven segments of the display. Your Verilog code can include a case statement with one-to-one mapping of input number to output pattern.

   For more detail on the 7-Segment Display, refer to pages 23 and 24 of the Nexys-A7 Reference Manual. You will need to consult the manual to determine which FPGA pins are connected to the hex display.

[1] See section 9 at https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual for more details about the 7-segment display

2. counter16 - We would like to use the four 7-segment displays to display a 4-digit hexadecimal number, which is 16-bit binary number. That is, the 4 least significant bits will be displayed on the right-most 7-segment display, and the 4 most significant bits will be displayed on the left-most 7-segment display. Modify your counter from part 1 so that it will now output a 16-bit number. This should be a simple change.

3. display_control - The binary to 7-segment display decoder can drive one 7-segment display at a time, while we would like to display a different digit on each of the four 7-segment displays. We will use the decoder to drive each of the four displays in a round-robin fashion. To do so, we will make the displays flash each digit very fast (1kHz) such that the user would not notice that the digits are not being displayed concurrently.

   In order to do this, you need to apply the codes 4'b1110, 4'b1101, 4'b1011, and 4'b0111 to the display_select (AN) line. This will make sure that only one digit (one 7-segment display) out of four digits is ON at a time. Make sure to synchronize your digit value output with the display alternation.

4. clock_divider - A clock divider receives a clock signal as input, and using an internal counter divides the clock by some number in order to achieve the desired lower frequency and duty cycle clock signal. For example, a simple divide-by-2 clock divider can be used to convert a 2 KHz clock signal to a 1 KHz clock signal. Note that your duty cycle for the 1Hz clock may not need to be 50% depending on your implementation.

   Your design will have two clock dividers. The first one will convert the 100MHz clock into a 1Hz clock. The second will convert the 100MHz clock into a 1kHz clock. You may use separate modules for each clock divider or you may use Verilog parameters to create a parameterized module (If necessary, ask TAs about parameters).

5. debouncer - This module can be reused from part 1 without any modifications.

6. top - The Top Level module should instantiate display_control, counter16, clock_divider, seven_segment_decoder and debouncer modules.

   Connect the modules as shown in Figure 3. Demonstrate the working top module on the FPGA to the TA. Refer to Pre-Lab B for directions on implementing the design on the FPGA. Refer to the Nexys-A7 Reference Manual to determine appropriate pin assignments for the I/O pins.

   Note that your design must have a global active-low reset input (omitted from the figure for clarity), which will reset all modules in the design.

   Note that your simulations do not need to use clock dividers as simulations with 1Hz and 1kHz clocks would be very long. For simulations, you can generate these clock signals in the test bench.

5. Deliverables

You need to show your design hierarchy (under Project Manager → Sources → Design Sources), your Verilog code, and FPGA implementation (on the FPGA board) to the TA.

Your design has to abide by the following requirements. Failure to follow the instructions will result in a grade reduction.

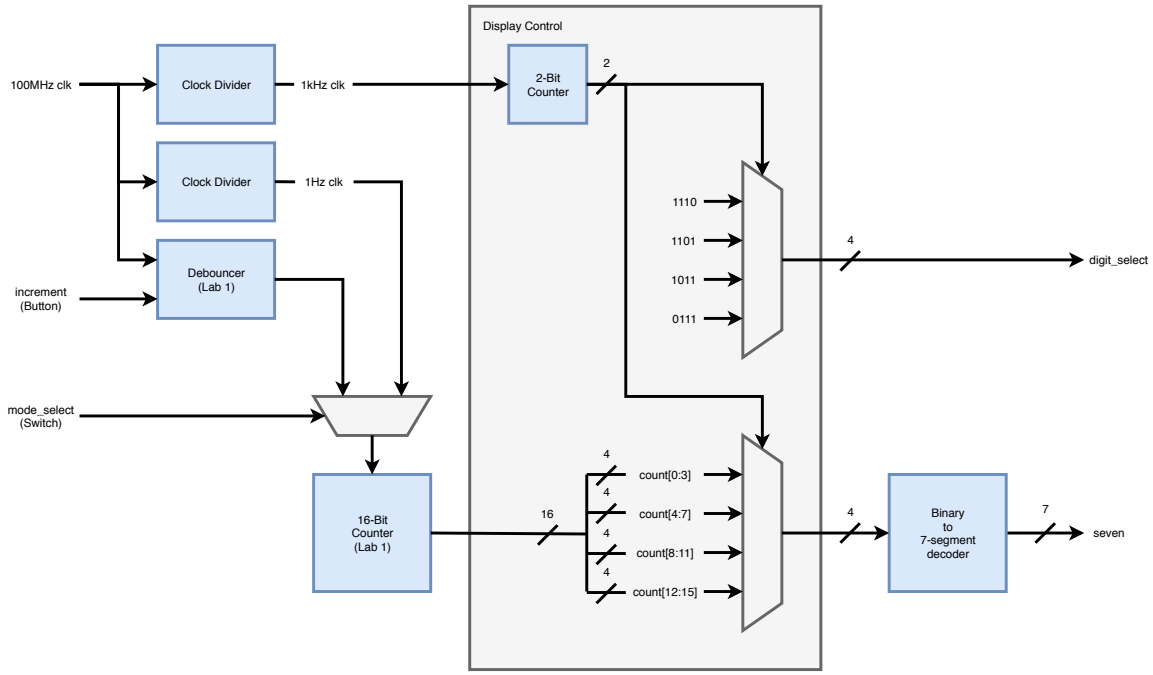

Figure 3: The overview of top level module for part 2. Note that reset is not shown on the diagram but it must be included in your design.

1. Use multi-bit vector/bus/array (just different ways of naming) instead of 1-bit signals for all multi-bit signals
2. Follow the input/output specifications.
3. For each test bench, you need to cover all the typical and corner combinations of input values to prove your logic is right.
4. Be hierarchical (use sub modules), and use separate file for each module. The test benches should be thorough but do not need to provide complete coverage. Complete test coverage in sequential designs can be difficult to achieve.

For part 1 of the lab, the TA will test your design for correct functionality. Correct debouncer operation is worth 15 points, and correct counter increment functionality is worth 15 points. For part 2 of the lab, the TA will test the two modes of operation of your design. Each mode is worth 25 points. To pass all the test cases in part 1 and part 2, make sure to think of all the corner cases while implementing your design.

You must turn in a lab report via Blackboard. The lab report should be broken up into these sections: Objective, Design Description (can include screenshots timing diagrams, schematics, etc.), Implementation Details, and Summary. Your report should contain enough detail such that any other engineer can replicate your experiments after reading your lab report. Please include all the required details in your lab report (including test benches). Your lab report is worth **20 points**.

You must submit all your code via Github Classroom. Detailed instructions for submiting the code will be provided at a later date.