Lab Assignment 7
Designing Counters Using Simulink

Lab 7.0 Introduction

The goal of this lab is to develop a digital design (a counter) using Simulink.

Lab 7.1 Clock signals

A clock signal is a periodic signal, constantly transitioning between logic values 0 and 1 with symmetric pulses, as shown in the following timing diagram:

The following properties of a clock signal can be highlighted:

- A falling edge of the clock is a transition from a value of 1 to a value of 0.
- A rising edge is a transition from a value of 0 to a value of 1.
- The clock period is defined as the time between one rising edge and the next (or one falling edge and the next). The clock period of the ZedBoard's FPGA is 0.02µs or 20ns.
- The clock frequency is defined as the reciprocal of the clock period. The clock frequency of the ZedBoard's FPGA is 50MHz.

Clock signals are typically connected to all storage components in a logic design. Their purpose is synchronizing the transition of these components into their next state. This transition will typically happen at the falling edge of each clock cycle. In other words, whenever the clock signal moves from 1 to 0, all bits stored in logic blocks will begin storing their new values (should they actually need to change).

Lab 7.2 Counters

An n-bit counter is a logic component storing an n-bit binary number that is incremented on every falling edge of a clock signal. A free-running counter is a counter that takes all possible n-bit values incrementally, and is reset to 0 only at the time it overflows. Its counterpart is a limited-range counter, which is reset after it takes a value lower than the highest one representable with n bits. An 8-bit counter, for example, has the following interface:

- The triangle-shaped port on the bottom left of the logic block indicates a connection from a centralized clock signal. The counter will react to falling edges in this input signal.
- An optional enable input allows us to decide when the counter should run or stop. If enable is set, the counter increments its value on the falling edge of the clock. If enable is not set, the counter just remembers its previous value.

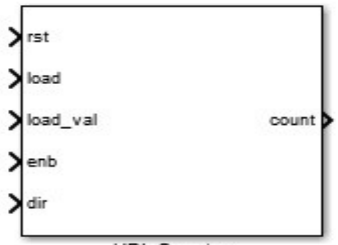
  An 8-bit output provides the current value of the counter at all times.

The following reading list will help you to complete the assignment. The readings will also help you in subsequent lab assignments.

Assignment 0

[1] Follow Simulink Tutorial in blackboard - Chapter 2
https://blackboard.neu.edu/

[2] The following is a HDL Counter. Explain what each of the 5 inputs and 1 output ports are (what they do), and what data type they accept.

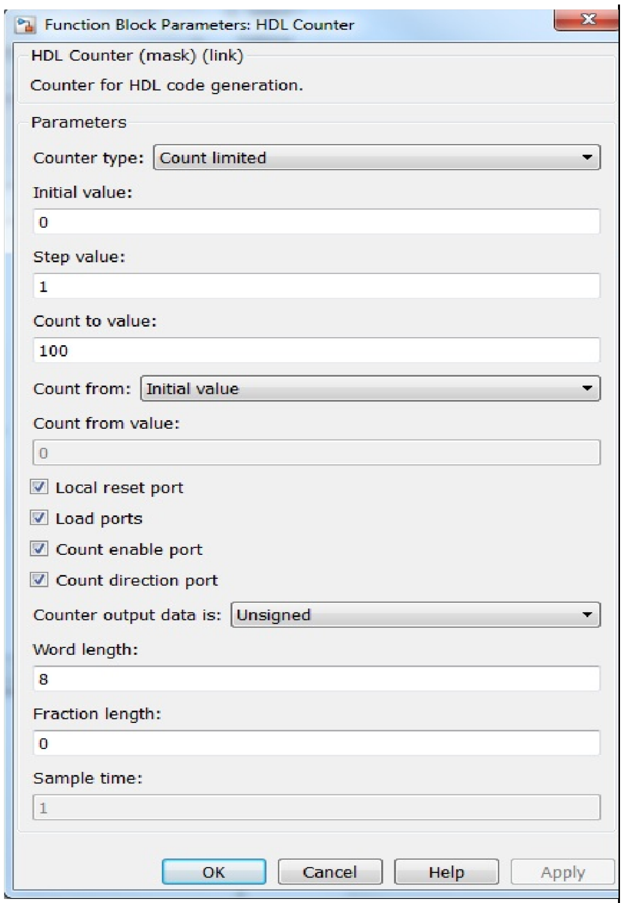


HDL Counter

[3] In the HDL Counter configuration window shown, explain what the following mean: "Initial value", "Step value", " Count to value" and "Word length"
(Hint: In Simulink, right click on a HDL Counter and select "Help")

[4] Contact Bounce

https://www.allaboutcircuits.com/textbook/digital/chpt-4/contact-bounce/


Function Block Parameters: HDL Counter

HDL Counter (mask) (link)

Counter for HDL code generation.

Parameters

Counter type: Count limited

Initial value: 0

Step value: 1

Count to value: 100

Count from: Initial value

Count from value: 0

[✓] Local reset port

[✓] Load ports

[✓] Count enable port

[✓] Count direction port

Counter output data is: Unsigned

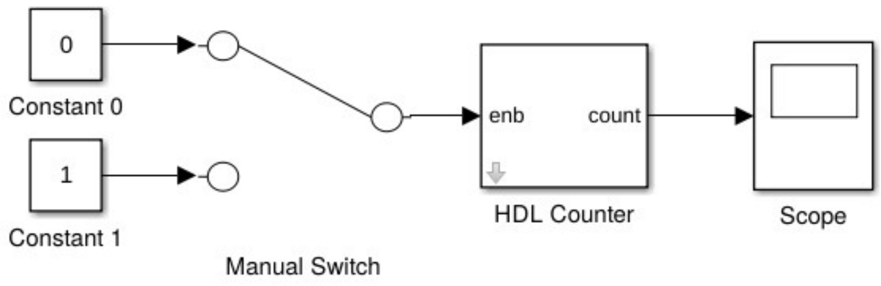
Word length: 8

Fraction length: 0

Sample time: 1

Lab 7.3 Design of a free-running counter

1. Open Simulink.
2. Create a new Simulink model. Clink on the New icon and select Simulink Model. Wait for the model to get created. The bottom-left status bar on MATLAB shows messages indicating when the system is busy.
3. Create a Lab7 folder on your Desktop and save the Simulink design in that folder. Switch to MATLAB and set the MATLAB workspace to the same folder
4. Simulink offers different solvers to cover everything from continuous models (e.g., for control theory) to discrete HDL. For your Simulink model to operate with counters and discrete logic, we need to configure a discrete solver. Open the configuration settings in menu entry Simulation → Model Configuration Parameters. In the pop-up dialog, select:
    a. Type: Fixed-step
    b. Solver: Discrete (no continuous states)
    c. Fixed-step size: 1
5. Click on the Library Browser icon (the 4 little red and blue squares).
6. Insert component Simulink → Signal Routing → Manual Switch into your design, by dragging and dropping it.
7. Insert component Simulink → Sources → Constant, and double-click on its properties. In Signal Attributes, change Output data type to boolean. In the main tab, set the constant value to 0.
8. Copy and paste the constant input to get a second input. Change the value to 1.
9. Connect constant 0 to the top input of the switch, and constant 1 to its bottom input.
10. Insert component HDL Coder → HDL Operations → HDL Counter into your design. Double-click on the counter to open its properties and set the Counter type to Free running. Make sure that Word
length is set to 8 bits, and that Count enable port is checked. Connect the switch output into the enable input of the counter.
11. Insert component Simulink → Sinks → Scope, and connect the output of the counter into the scope's input. Double-click on the scope, and observe the new window popping up. Leave this window open, as it will be automatically updated once you run your simulation.
12. To record the scope's output for the entire simulation, click on the Set⚙gs icon, click on the Logging tab, and ensure that the Limit data points to last option is unchecked.
13. Save the Simulink model in a file called counter_simple.slx. Your design should look like this:

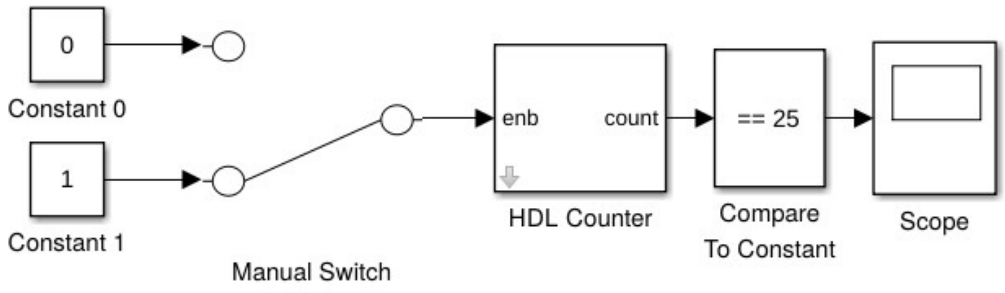


Assignment 1

Simulate the design for 1000 cycles by editing the simulation stop time in the top text block located near the Run icon. Describe what you observe on the scope component depending on the switch position. You can change the switch position by double-clicking on it, and re-running the simulation. Explain how different values assigned to input enable through the manual switch affect the behavior of the counter. Add a screenshot of your Simulink design, including the output of the scope.

Lab 7.4 Design of a counter with comparator

1. Make a copy of file counter_simple.slx and save it as counter_compare.slx.
2. Insert component Simulink → Logic and Bit Operations → Compare to Constant. Place it in between the counter and the scope, and change the value of the compared constant to 25. Y our design should look like this:

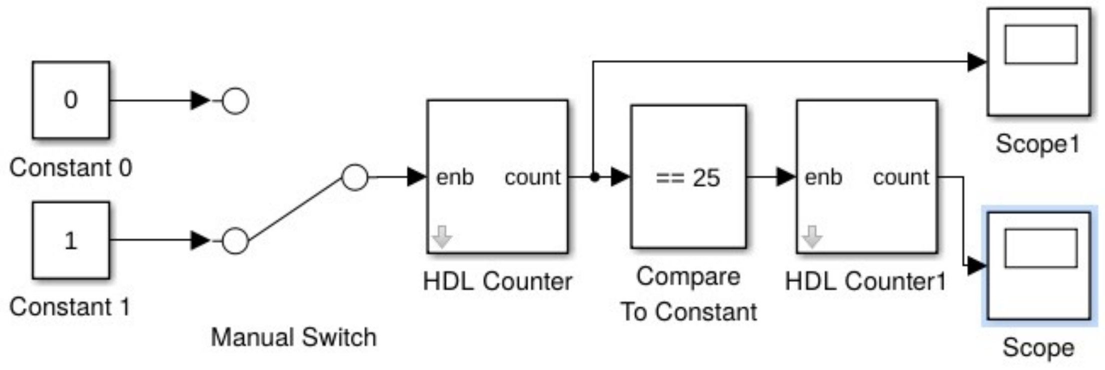


Assignment 2

Simulate your design with the switch value set to 1. Take a screenshot of your simulation, including the output of the scope, and describe it. What is the behavior of the circuit you just created?

Lab 7.5 Design of a cascaded counter

1. Make a copy of counter_compare.slx from the previous assignment and save it as counter_cascaded.slx.
2. Copy the counter, and insert it between the scope and the comparator.
3. Add another scope after the first counter. Y our model should look like this:



Assignment 3

Simulate your design and observe the output of the scopes with the switch set to 1. What is the relationship between the scopes? How can you describe the behavior of the new circuit after connecting the two counters in cascade?

Discuss how the counting speed of the second counter can be controlled. What components would you add to the circuit and with what configurations in order to make the second counter slower or faster?

Lab 7.6 Controlling the LEDs with a cascaded counter

Our next objective is creating a design with a free-running 8-bit counter whose output can be connected to the LEDs on the ZedBoard.

The ZedBoard has a default clock frequency of 50MHz. This means that synthesizing an HDL counter on the FPGA will make it change its state to the next count at this specific frequency. The state transitions would happen so fast that the intermediate values of the LEDs wouldn't even be observable by the human eye. We will leverage cascaded counters to reduce the counting speed.

1. Make a copy of file counter_cascaded.slx and save it as counter_to_leds.slx. Configure the second counter in the design as an 8-bit free-running counter.
2. Remove the Manual Switch block and both Constant Blocks. Double-click on the first counter and uncheck option Count Enable Port. The first counter will now be always active, and increment its value every cycle.
3. Configure the comparator with an equality comparison to constant 1.
4. Configure the first counter, currently counting at 50MHz, such that it resets to zero twice per second, that is, once every 500ms. Change the counter type to Count limited and adjust the Count to value field accordingly. You will also need to adjust field Word length if the current size of the counter cannot represent its new maximum value.
5. Use the bit-slice module developed in the previous lab session in order to split the output of the second counter into 8 individual boolean values. Then, connect each of these values into an Out1 component.
6. Validate the correct functionality of the design through simulation. You can either use a scope or a display block to observe the outputs.

Assignment 4

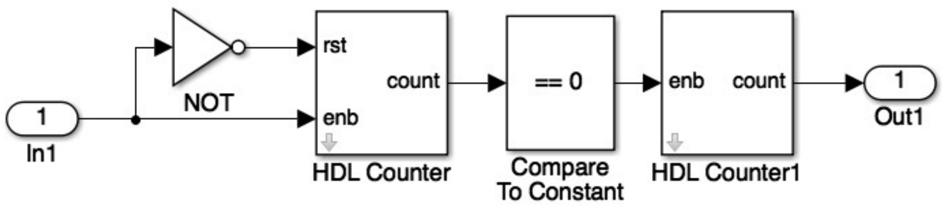
Test the design in Simulink

. a) Include a screenshot of your Simulink design on your report.

Lab 7.7 Designing a de-bounce circuit to manage the Push Buttons

Using push buttons can be tricky to implement, since the physical imperfection of the contacts within a push button typically causes signal bouncing, as mentioned in http://www.allaboutcircuits.com/textbook/digital/chpt-4/contact-bounce/. Every time a button is pressed, its output signal will quickly transition between 0 and 1 before it takes a final stable value of 1. To solve this problem, we will design a cascaded counter in order to guarantee that a signal coming from a push button has settled with a value of 1 for long enough before we consider the button to have been effectively pressed.

1. Create a cascaded counter design and save it as button_to_led.slx (similar to Lab 7.6 design).
2. Configure the first counter to have a Local reset port and a Count enable port. Connect the enable port to an In1 component, which will later be mapped to the center push button (PBTNC). This makes the first counter increment its value for as long as the button is pressed.
3. Add an inverter (NOT gate). Connect its input to the push button signal (PBTNC) and its output to the reset port of the counter. This makes the counter go back to 0 when the bush button is released.
4. Configure the comparator in such a way that it yields 1 only after the button signal has been set to 1 for 250ms straight. Remember that the HDL counter is running at a frequency of 50MHz (50,000,000 cycles per second), which will determine the value for the comparator's constant. Calculate this value for the comparator (currently set to 0).
5. Configure the first counter with type Count Limited. As you observed in the previous lab assignment, the maximum value that the counter can take determines the frequency at which the Compare to Constant block is sending a pulse set to 1. In our case, this will determine the frequency at which the LED will blink if we just keep the push button pressed for a longer period of time. Configure the Count to Value property of the first counter to make the LED blink at 1 frequency of 1Hz (once per second) when the push button is kept pressed.
6. Configure the second counter as a 1-bit free-running counter. Connect the output of the counter to an Out1 component, which will be mapped into an LED. Your design should look like this (except for the value shown in the Compare to Constant block):



7. For simulation purposes, insert a Scope component in your design. Configure the scope to have three inputs: double-click on the scope, click on the Settings icon in the pop-up window, select tab General, and enter “3” in field Number of axes. The scope now has three inputs, which you can connect to the output of the first counter, the magnitude comparator, and the second counter, respectively.
8. By default, the scope limits its history to the last 5,000 cycles. In order to allow the timing diagram to record a signal for the entire simulation, double-click on the scope, click on the

Settings icon in the pop-up window, select the History tab, and uncheck option Limit data points to last.

9. Also for simulation purposes, you can temporarily replace the In1 port by a Step logic block followed by a Data Type Conversion block (set to output boolean). You can configure the Step block to mimic a user pressing the push button, by generating a pulse of 500ms—this period should be given in number of cycles. You will also need to set the Initial value to 1, Final value to 0, and Sample Time property of the Step block to 1.

Running the simulation

10. Simulate your design during 10s of virtual time, using the appropriate amount of simulation cycles. In order to keep reasonable simulation speeds, you can assume that the FPGA clock runs at a frequency of 50kHz, instead of the actual 50MHz. This means that you can divide the counter limits, constant values, and number of cycles by 1000.
    a) Take a screenshot of your design and add it to your pre-lab turn in. Explain the numeric values chosen for the Compare to Constant component and the counter limits during the simulation.
    b) Add another screenshot for the simulation of your design, showing the complete output of the Scope component, appropriately zoomed in. You can use the Autoscale button for this purpose. Describe the output you observe, and justify its correctness.

Lab 7.8 Change Counting Speed

Extend your design to control the counting speed of the 8-bit counter. The PBTNU pushbutton should increase the counting speed, and PBTND pushbutton should decrease the speed. Save this model as LedCount_speed.slx.

Hint 1: Expose the reset port of the first counter in the cascaded counter design. Implement a circuit to trigger a reset when the counter reaches a certain value even if the counter is set to “free running mode”. The reset value can be set by an additional HDL counter that changes output value by an appropriate step value when one of the push buttons is pressed. PBTNU should decrement the value to increase speed and PBTND should do the reverse to decrease counting speed.

Hint 2: How can you use two push buttons to control one input?

Simulate your design in Simulink and make sure it works.

Lab Report

Submit all your responses to the questions asked, screen captures of all your subsystems and the contents of the subsystems, and explain any settings for the various counters and blocks wherever necessary. Submit all these in a single PDF file.