

# Lab Assignment 6

## Programming the FPGA Using Simulink

### Lab 6.0 Introduction

The goal of this lab is to develop a digital design and download the design on to the FPGA using Simulink. You will be using the Simulink HDL Coder to program the FPGA on the ZedBoard. Mimicking the way we controlled different devices on the FPGA by software, you will now connect the switches to LEDs by hardware, and then control the LEDs with a simple counter.

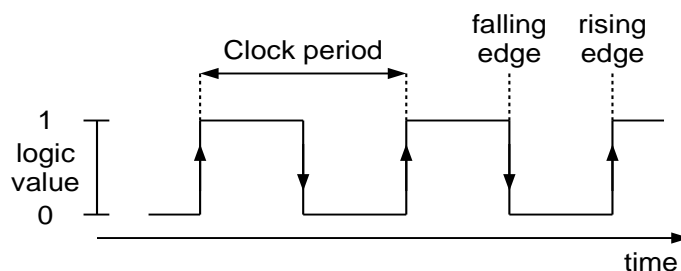
### Lab 6.1 Clock signals

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  $0.002\mu\text{s}$  or 20ns.
- The clock frequency is defined as the reciprocal of the clock period. The clock frequency of the ZedBoard's FPGA is 50MHz.

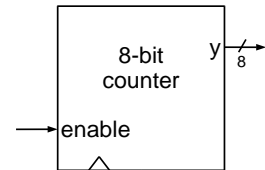
Clock signals are typically connected to all storage components in a logic design. Their purpose is synchronizing the transition of these components into their next state. This transition will typically happen at the falling edge of each clock cycle. In other words, whenever the clock signal moves from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



## Lab 6.2 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.

The following logic block is an example of an 8-bit counter:



This 8-bit counter 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.

### Pre-Lab Assignment

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


*Require Reading:* Simulink Tutorial in blackboard - Chapter 2

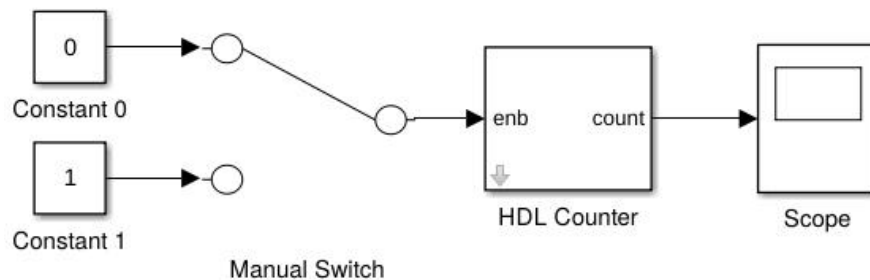
<https://blackboard.neu.edu/>

**You don't need to turn in anything for this pre-lab.**

## Lab 6.3 Design of a free-running counter

1. Open Simulink.
2. Create a new Simulink model. Click 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. 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

4. Click on the *Library Browser* icon (the 4 little red and blue squares).
5. Insert component *Simulink* → *Signal Routing* → *Manual Switch* into your design, by dragging and dropping it.
6. 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.
7. Copy and paste the constant input to get a second input. Change the value to 1.
8. Connect constant 0 to the top input of the switch, and constant 1 to its bottom input.
9. 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.
10. 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.
11. By default, the scope limits the history to a certain number of cycles. We need to change this setting to allow the timing diagram to record the scope's output for the entire simulation. On the pop-up dialog for the scope, click on the *Settings*  icon, click on the *History* tab, and uncheck option *Limit data points to last*.
12. Save the Simulink model in a file called *counter\_simple.slx*. Your design should look like this:

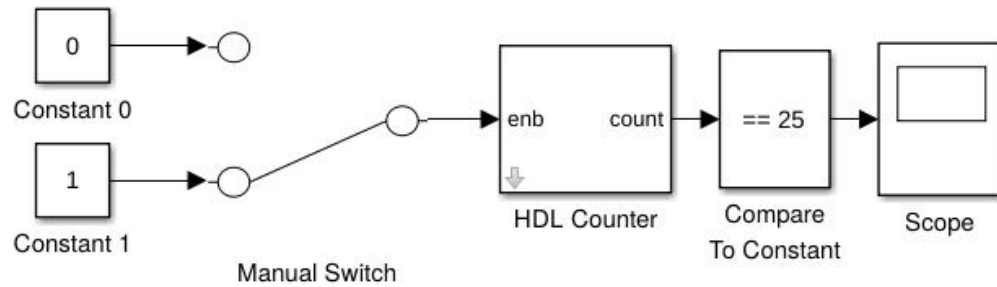


### Assignment 1

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 6.4 Design of a counter with comparator**

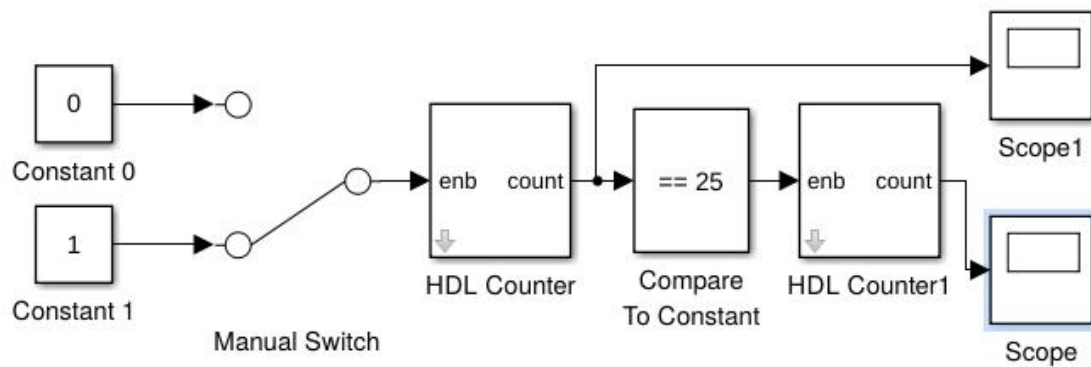
1. Make a copy of file *counter\_simple.slx* and save it as *counter\_compare.slx*.
2. Insert component *Simulink* → *Logic and Bit Operations* → *Compare to Constant*. Place it in between the counter and the scope, and change the value of the compared constant to 25. Your design should look like this:

**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 6.5 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. Your 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 6.6 Connecting the switches to the LEDs**

Our first attempt to program the FPGA will make it connect the switches to the LEDs, in the same way you did in previous lab sessions by software. When a switch is activated, its associated LED will light up. The LED will switch off otherwise.

1. In the Windows *Start* menu, open *All Programs* → *Xilinx Design Tools* → *ISE Design Suite 14.4* → *System Generator* → *System Generator*. Make sure that you open MATLAB from here, since some of the HDL coder components may not be available otherwise.
2. Open MATLAB, and inside MATLAB, open the Simulink library browser by clicking on the corresponding icon.
3. Create a new Simulink model by clicking on the *New* icon and selecting *Simulink Model*. Wait for the model to get created. The bottom status bar tells you if the system is busy. You need to be patient, this process may be long sometimes. Save your new design in a file named *switches\_to\_leds.slx*.
4. Insert component *Simulink* → *Sources* → *In1* into your design. Double-click on it, select the *Signal Attributes* tab, and change its data type to *boolean*. Also change the *Sample time* value to 1.
5. Insert component *Simulink* → *Sinks* → *Out1*. Change its data type to *boolean*, too.
6. Connect component *In1* directly to component *Out1*.
7. Select both *In1* and *Out1* components and create 7 additional copies of them.
8. Select all components (Ctrl+A), right-click on them and select option *Create Subsystem from Selection*.
9. Right-click on the subsystem you have created in the previous step, and click on *Block Parameters*. Select the option *Treat as atomic unit*.
10. Next, click on the *Code* menu. Select option *HDL Code* → *HDL Workflow Advisor*.
11. From the new opened window, titled *System Selector*, select the name of your subsystem, as listed, and click *OK*. The HDL Workflow Advisor tool will now open.
12. On the tree view on the left, select item *Set Target* → *Set Target Device and Synthesis Tool*. Select *IP Core Generation* as the *Target workflow*. Select the ZedBoard as the Target platform. Select the system's Desktop for *Project Folder*, and click on *Run This Task*.
13. Also on the tree view on the left, select item *Set Target* → *Set Target Interface* tab. You will see a list with the inputs and outputs of your design. Assign each input to *DIP Switches*, and each output to *LEDs General Purpose*. Click on *Run This Task*.
14. On the tree view, select item *Prepare Model For HDL Code Generation* → *Check Global Settings*, and click on *Run This Task*. When this finishes, click on *Modify All*, and then click on *Run This Task* again.
15. On the tree view, right-click on *HDL Code Generation* → *Generate RTL Code and IP Core*. In the pop-up menu, click on *Run to Selected Task*. When done, click *OK* in the *Code Generation Report*.
16. On the tree view, select item *Embedded System Integration* → *Create Project*, then click on *Run This Task*.
17. On the tree view, select *Embedded System Integration* → *Generate Software Interface Model*, activate checkbox *Skip this task*, and then click on *Run This Task*.
18. On the tree view, select *Embedded System Integration* → *Build FPGA Bitstream*, and click on *Run This Task*. In this step, Simulink will run the Xilinx tool in the background to generate a bitstream of your design for the FPGA on the ZedBoard. You need to wait for this step to finish. It takes around 10 minutes.

19. Next, use the USB cable to connect your PC to the ZedBoard's microUSB port, located next to the power plug. Power up the ZedBoard, and wait for it to boot.
20. On the tree view, select *Embedded System Integration* → *Program Target Device*, and click on *Run This Task*.
21. Wait about 2 minutes. The FPGA will be programmed, and the ZedBoard will be restarted.
22. Play with the switches on the ZedBoard and see the result.

**Assignment 4**

Take a screenshot of your final Simulink design and include it your report. Use different positions for the switches that show how the corresponding LEDs light up. Record what you see.

**Advice**

The process of uploading a design into the FPGA can take around 15 minutes in total. Always simulate your designs on Simulink first to make sure that they work correctly before uploading them to the platform.

### Lab 6.7 Controlling the LEDs with a cascaded counter

Our next objective is creating a design with a free-running 8-bit counter whose output is connected to the LEDs on the ZedBoard. Every time the counter transitions to a new state, the LEDs will reflect the value of the new count.

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.
7. Use the HDL Workflow Advisor as presented in the previous section in order to upload your design into the FPGA. The *Out1* components should be now connected to the LEDs.

#### Assignment 5

Validate that the design works correctly on the FPGA, making sure that the speed of the counter is as intended. Include a screenshot of your Simulink design on your report, together with a picture of the working counter, while all switches are in the OFF position.

### Lab Report

No formal lab report required for this lab. 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 blocks wherever necessary. Submit all these in a single PDF file (one per team).