**ECE324 Lab 5: Waterfall to Bouncing Ball**

Name(s):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **Exercise** | **Course outcome** | **Grade** |
| Lab5 Demo | 2.a, 2.d, 5.c, 7.b | /15 |
| Lab5 Extra Credit | 2.a, 2.d, 5.c, 7.b | /3 |
| Lab5 Report | 2.a, 5.c, 7.b | /25 |
|  | **TOTAL:** | /40 |

2.a. Define engineering problems from specified needs for digital systems including implementation on FPGAs using HDL programming.

2.d. Produce FPGA designs that meet specified needs.

5.c. Collaborate with individuals with diverse backgrounds, skills and perspectives.

7.b. Employ appropriate learning strategies such as communicating with an expert, using external resources, experimentation, simulation, etc.

# Learning objectives:

1. Improve SystemVerilog HDL skills by using behavioral modeling to design sequential logic.
2. Improve FPGA design skills by using Xilinx Vivado software to implement a sequential logic function and demonstrate it on an FPGA prototype board.
3. Understand FPGA synthesis and mapping by producing a detailed report of utilization of FPGA logic resources.

# Functional specification:

Using the 16 LEDs on a Nexys4DDR, model a ball constantly bouncing 4 feet high.

# Procedure:

1. Implement on a Nexys4DDR the fully-functional waterfall code you are given, and study how it works.
2. Change the logic to simulate drips from a 4 foot high faucet. Do this by replacing the free-running counter that generates *timeBaseCntr* with a mod-m counter, and use a calculator (available on a PC) to produce the modulo value M (and based on M, generate the appropriate value of N). Verify your change works in hardware, but you don’t need to demonstrate it to the professor.
3. Model a ball bouncing 4 feet high by changing the sawtooth waveform generated by the up-only counter for *t*, into the triangle waveform shown below, doing the following:
   1. Replace the *always* block generating *t[-1:-17]* with an instantiation of univ\_bin\_counter, with *syn\_clr*, *load*,and *d* driven to 0’s to make it just an up/down counter with outputs *t[-1:-17]*, *max\_tick*, and *min\_tick*. Note that asynchronous initialization of *q* to 0 is done inside your univ\_bin\_counter instance, so no initialization should be done when declaring *t* to be logic in your top level module.
   2. To control the direction of counting, infer in your top level module a single flip-flop called *up*, initialized to 1 in its declaration. Use the *max\_tick* and *min\_tick* outputs of the univ\_bin\_counter to determine the next state of *up*.

/| /| /\

/ | / | / \

/ | / | / / \ /

/ |/ |/ / \/

Sawtooth Triangle

# New Code provided

**Lab5\_Waterfall.sv –** Module that simulates a waterfall.

**free\_run\_bin\_counter.sv** (simplified from Chu listing 4.11)

**mod\_m\_counter.sv** (simplified from Chu listing 4.13)

**univ\_bin\_counter.sv** (simplified from Chu listing 4.12)

**Lab5\_Waterfall.xdc**

# Deliverables:

1. Demonstrate operation of your completed FPGA design to the professor or lab assistant.
2. Submit a written lab report including the following items:
   1. A written description of your modifications to the SystemVerilog code.
   2. Did your design satisfy all aspects of the functional specification? If not then describe and comment on any functions not implemented.
   3. In a Xilinx Vivado implementation *Utilization Report*, find and report the number and percentage of each of the following FPGA device logic resources used in your design: Look Up Tables, Flip-Flops, and DSP’s.
3. Submit your changed SystemVerilog code with your lab report (no testbench is necessary for this lab). Your SystemVerilog code must include header comments stating your names, date and class number. Your Verilog code must also include comments explaining the operation of the code.

# Extra Credit (up to 3 points):

Make the ball fall time programmable with the 16 switches, and demonstrate to your professor or T.A. a simulation of a ball falling the height of the 185 foot high Leaning Tower of Pisa (<https://en.wikipedia.org/wiki/Galileo's_Leaning_Tower_of_Pisa_experiment>). Suggestion: to set the switches, use a PC’s calculator app in programmer mode to convert decimal to binary.

# Food for Thought:

For your final project, you might want to simulate gravity starting at a particular velocity, such as may be done in a game of Artillery, or in a lunar lander simulator. A different set of equations can be used to update at constant time intervals the vertical and horizontal components of velocity and location:

Vyc = g\*t + Vyp

Vyc is the current y-axis (vertical) velocity

g is the gravitational acceleration

t is the incremental time interval

Vyp is the previous y-axis (vertical) velocity

Lyc = (Vyc\*\*2 – Vyp\*\*2) \* (1/(2\*g)) + Lyp,

Lyc is the current y-axis location

Vyc is the current y-axis velocity

Vyp is the previous y-axis velocity

g is the gravitational acceleration constant, so 1/(2\*g) is a constant multiplicand

Lyp is the previous y-axis location

Lxc = Vx\*t + Lxp

Lxc is the current x-axis location

Vx is the x-axis velocity (which is a constant unless the object hits something)

t is the incremental time interval

Lxp is the previous x-axis location

When using these formulas, velocities Vyc and Vyp are negative when the object is moving away from the earth, and locations Lyc and Lyp have lower values when the object is farther away from the earth (upside down compared to the y axis on a Cartesian coordinate system).