Com Sci 152B Digital Design
Lab 0: ISE Introduction

Michael Hale, 004-620-459
Matthew Nuesca
Shilin

Overview

The main purpose for this lab was to get students familiar with the ISE that will be used for future projects. We were tasked with starting a new project and copying Verilog code for a counting circuit.

Implementation

The counter module accepted three inputs and generated one output:

- `clock` (input): Pulse signal for which our count increases on a positive edge.
- `enable` (input): The counter will only increment while this signal is high.
- `reset` (input): The counter will reset to zero with the next clock pulse.
- `counter_out` (output): The 4-bit unsigned integer representing the current count. After 1111, the counter will overflow to 0000.

The circuit is essentially an always block with the sensitivity set to execute on a positive `clock` edge. Inside the always block we investigate three possible cases:

1. `reset = 1`: The counter is immediately set to 0 with no further action.
2. `enable = 1, reset = 0`: The counter is incremented by one.
3. `enable = 0, reset = 0`: No action is taken whatsoever.

Obstacles and Solutions

While familiarizing ourselves with the code, we discovered that none of us knew the purpose of the `timescale` keyword. With a quick google search, we learned that the `timescale` keyword is a compiler directive used to specify the time unit and precision when performing simulations of synthesized code. The first value after the keyword indicates the time unit for the delay. For example, If the timescale is set to 1ns then #2 would delay for 2 ns. The second value after the forward slash indicates the precision for a delay with decimals. For example if the timescale is set to 1ns / 1ps and you have a delay of 1.1234, the delay would result in 1.123 ns. We were also asked to copy the testbench module and simulate the counter.

Our next task was to map the `counter_out` signal to a set of LEDs and map the `reset` and `enable` signals to a set of switches. One issue that we encountered was not being able to find instructions on what pin to map for the enable signal. As a result, we just decided to use the switch next to the reset in order to control the enable signal. Finally we were tasked with downloading the design onto the FPGA. At first, it seemed like it wasnt working because all LEDs that corresponded to the count register were on, but we came to the conclusion that the clock was running too fast to notice the LEDs turn on and off. The solution was to downsample the clock signal by adding a clock divider implemented using a separate counter which we will denote `divider`. `divider` would increment at the positive edge of the clock and the main counter would increment at the positive edge of the most significant bit of `divider`. After playing around with different bit sizes for divider, we concluded that 26 bits is enough bits to notice a difference in the count signal through the LED pins.