Alberto Diaz - UID: 604967268

Evan He - UID: 705619333

Brandon Truong - UID: 705326387

Lab 0 Report

**Introduction**

This project served as an introduction to Vivado 2018.2 software and using Basys 3 FPGA board. We followed the provided tutorial to go through the workflow of creating a new project, adding source files, writing Verilog code and a testbench, testing through Vivado’s simulation software, and finally demonstrating the finished product on a Basys 3 board. Our Verilog code implemented a simple 4-bit counter that would increment in each clock cycle so long as the “enable” signal was high. If a “reset” signal was sent, the counter would restart at 0. We demonstrated this 4-bit counter using LEDs found on the Basys 3 board where each LED represented a single bit. We also assigned two switches to control the “reset” and “enable” signals while using the internal clock as our clock signal for the counter. The internal clock however proved to be too fast for us to verify our counter, so we decided to implement a clock divider to slow down the clock signal for our counter module..

**Design**

*Counter Module* -

`timescale 1ns / 1ps

module my\_counter(

input clock,

input reset,

input enable,

output reg [3:0] counter\_out

);

reg [25:0] clock\_reg;

wire clock\_en;

assign clock\_en = clock\_reg[25];

always @ (posedge clock)

clock\_reg <= clock\_reg + 1;

always @ (posedge clock\_en)

if (reset)

counter\_out <= 0;

else if (enable)

counter\_out <= counter\_out + 1;

endmodule

Our Counter Module is very simple. We keep a 4-bit “counter” register to keep track of how many clock cycles have passed. At the positive edge of each clock edge, we increment the counter. When testing on the FPGA board, the internal clock was too fast for the human eye, so we implemented a clock divider. The clock divider is essentially another counter that increments at every positive clock edge. Once the clock divider counter passes a threshold, it flips another clock we called clock\_en to control the counter output.

*Testbench* -

module test\_bench(

);

reg clock, reset, enable;

wire[3:0] counter\_out;

my\_counter UUT(.clock(clock), .reset(reset), .enable(enable), .counter\_out(counter\_out));

initial begin

clock = 1;

reset = 0;

enable = 0;

#5;

reset = 1;

clock = 0;

#5;

clock = 1;

#5;

reset = 0;

clock = 0;

#5

clock = 1;

#5;

clock = 0;

enable = 1;

for (integer i = 0; i < 20; i = i + 1)

begin

#5;

clock = ~clock;

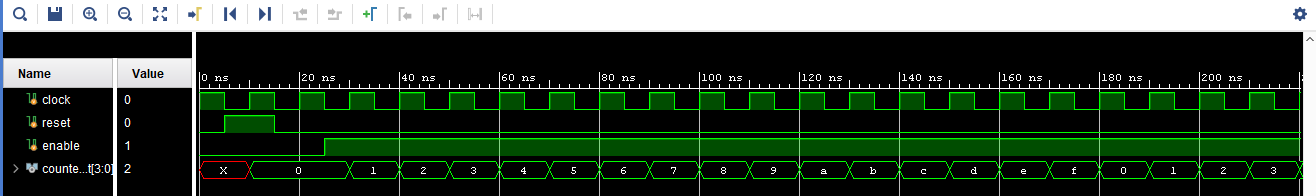
end

end

endmodule

In our testbench, we create an instance of our Counter module to test and set up “clock”, “reset”, and “enable” signals for input. At first, we manually control the clock, reset, and enable signals to test different test cases including: (1) “reset” signal resets our counter, (2) while “enable” is low, the counter does not increment at positive edges of the “clock” signal, and (3) while “enable” is set to high, the counter will increment at positive edges of the “clock” signal for 20 cycles.

**Results**

**Figure 1:** Screenshot of simulation waveform. Shows different cases including “reset” high, “enable” high and low, and 20 clock cycles while “enable” is high.

Running our simulation, we get the waveform shown in Figure 1. As expected, when “reset” gets set to high, our counter will get set to 0 at the next posedge of the clock so long as “reset” is still high. When “reset” is set low and “enable” is also low, we see that the counter does not increment despite clock cycles passing by. When “enable” gets set to high, we finally see the counter increment at every posedge of the clock. In the above figure, we see the counter reach at least 20 clock cycles.

**Problems and Solutions**

One such problem that we ran into was the internal clock of the board being too fast. The board’s default clock pin seems to be under a 100MHz oscillator, meaning that the 4-bit counter when given the internal clock as the base input will run too fast for leds to update properly. This can be fixed by using a clock divider instead to create a slower clock cycle. We did this by assigning the MSB of a 26-bit register to be the high state of a clock, which gave us a slower clock cycle. Given the reference internal clock of 100 MHz, making the MSB the high state gives us about

which gives us a clock cycle for our 4-bit counter that can be properly updated by the leds.

We also ran into a problem where our reset no longer worked when we re-implemented some functionality. This was due to us assigning reset in two places at once, which could not be compiled. The solution for us was to move all the functionality out of the always block running under the internal clock and into the clock divider’s always block, allowing us to compile our code.

**Contributions**

Alberto - Wrote introduction. Contributed to code and testbench.

Evan - Wrote design section. Contributed to code and testbench.

Brandon - Wrote Problems and Solutions section. Contributed to code and testbench.