**CS M152B Lab 0 Report**

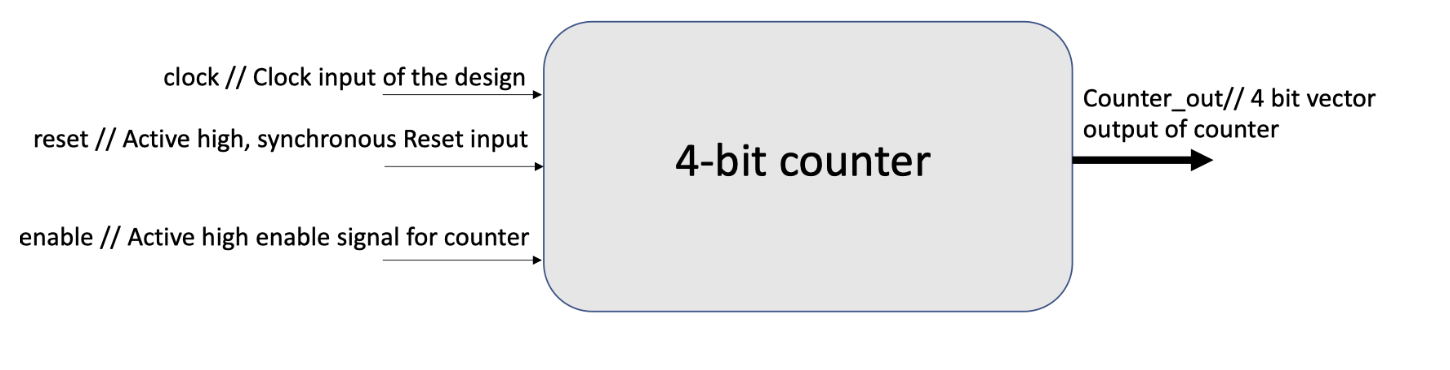
**By: Rodrigo Gonzalez and Chris Baker**

**UIDs: 905161154 and 105180929**

**TA: Ghazal Ershadi**

1. **Introduction**

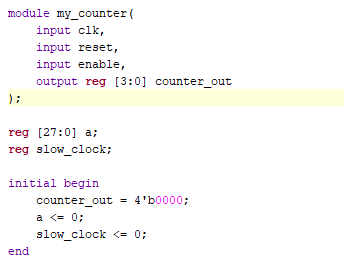
The purpose of this lab is to provide a brief introduction into both the software tools that will be used to work on our designs (Vivado and Verilog) as well as the hardware that will synthesize and implement the designs (Basys 3 FPGA) throughout the course. This introduction included a simple 4-bit counter and simulation testbench, along with a functioning counter on the FPGA with LEDs indicating the counter and switches controlling an enable and reset signal. Below is a brief diagram illustrating the requirements for this counter design.



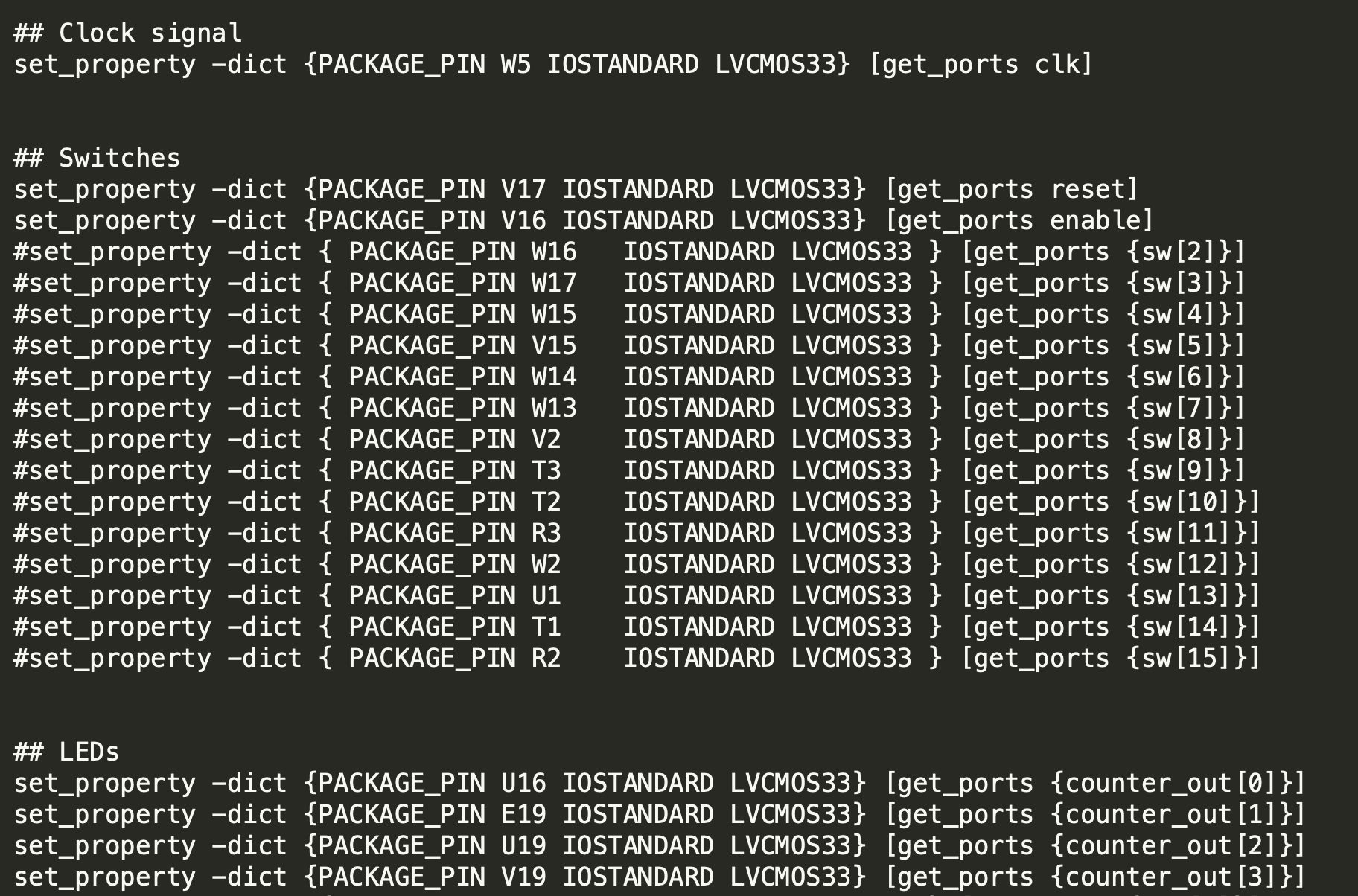
1. **Design and Implementation**
   1. **4-bit Counter**

The main design task for this lab is creating a functioning 4-bit counter in Verilog and demonstrating the functionality through simulation waveforms produced by a robust testbench. Then, this 4-bit counter must synthesize and run on the Basys3 FPGA, showing the incrementing counter using LEDs. The two screenshots below show the Verilog code for both the original 4-bit counter design that runs on the system clock, as well as the modified code for the FPGA. The next section has a screenshot of the testbench code, and its results are shown in the Simulation Waveforms section.

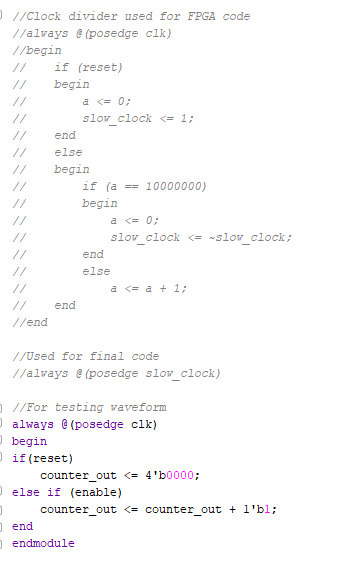
The original 4-bit counter design task required inputs *clock* for clock input, *reset* for synchronous reset signal, and *enable* for the enable signal for counter, as well as 4-bit output *counter\_out*. The task is to produce a 4-bit counter to output *counter\_out* that runs on the positive edges of the *clock* input when *enable* is high. When *reset* is high, *counter\_out* is reset to 0. The first screenshot shows the aforementioned input and output variables in the module definition, along with local registers for clock division that will be discussed further. There's also an *initial* statement that sets *counter\_out* and the local registers to 0 to control the initial state of the module.

****

As previously mentioned and seen above, there are registers and code segments dedicated to generating appropriate results on the FPGA board LEDs. To map the pins and signals on the board to those of the 4-bit counter, there must be edits to the *.xdc* constraints file where we uncomment and rename the ports to our specifications, seen below. The system clock is called *clk* in the constraints file, and is thus named as such in the module code. The first two switches on the board, originally mapped to *sw[0]* and *sw[1]* are renamed to *reset* and *enable* to function as those signals. Then, each of the 4 bits of the *counter\_out* are mapped to the first 4 LEDs, as all these lines are uncommented. However, the system *clk* signal produced a clock that was too fast to see state changes on the LEDs. The originally specified 4-bit counter ran on this fast clock, which in the case of the Basys3 FPGA board is around 100 MHz. To counteract this for the demo, we implemented a clock divider large enough to reduce the clock to a frequency traceable by the human eye.

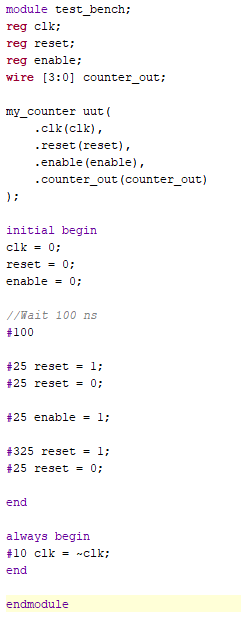


The code screenshot below shows both the commented clock divider and the original 4-bit counter code. As described above, we needed to implement a clock divider to allow the demo to be seen on the FPGA. The always block runs on the system *clk* and produces the signal *slow\_clock*, which would then be fed into the uncommented always block at the bottom. The current always block is the original design used in the testbench, where the 4 bit counter output increments on the system *clk* only when *enable* is high, otherwise resetting to 0 when *reset* is high. Small side note: notice that in the commented code *slow\_clock* is set to 1 on the *reset* check. This is because the always block on the *slow\_clock* would not reach the *reset* stage on the second block, and is thus set to high so the *reset* is read and *counter\_out* returns to 0.

****

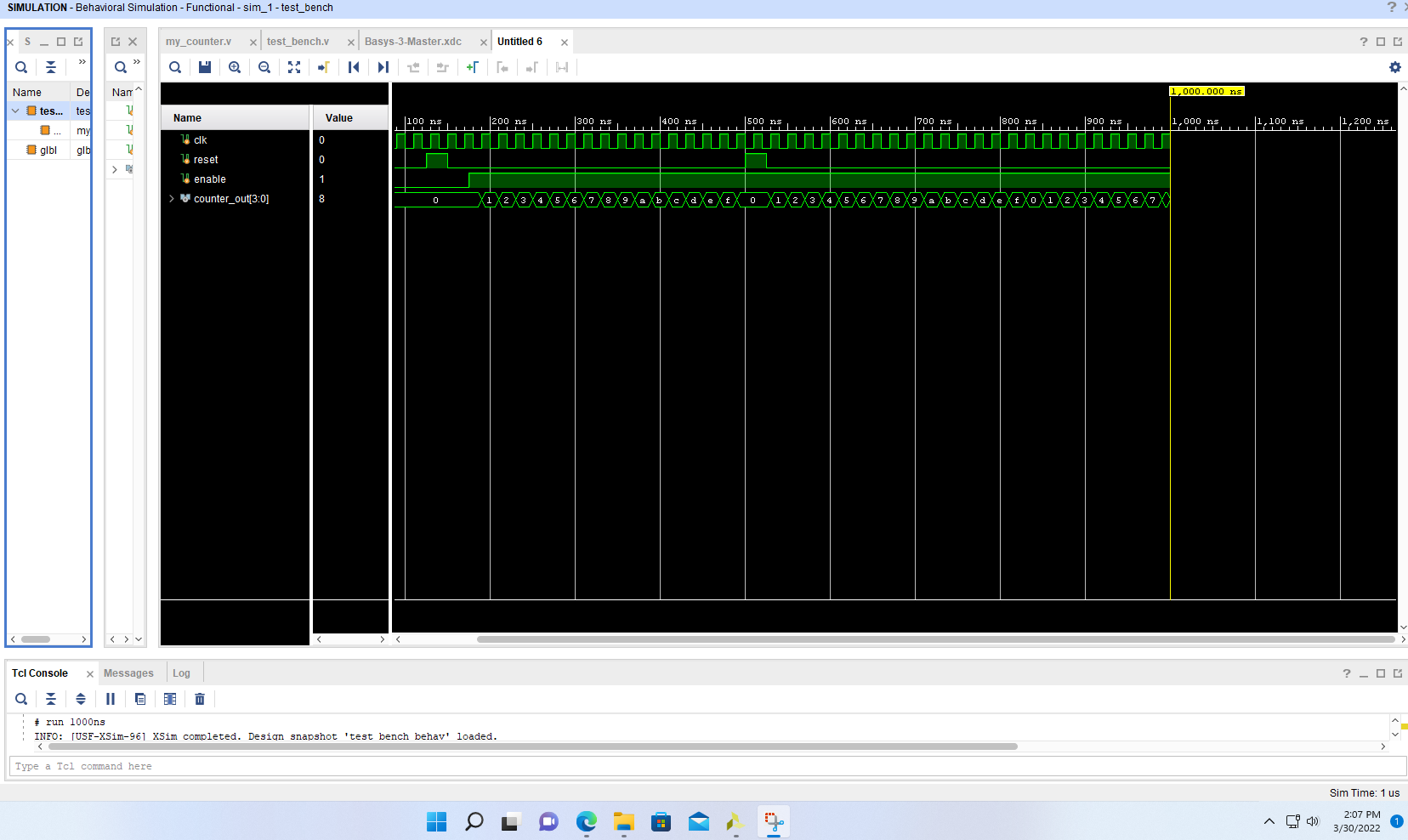
* 1. **Testbench**

The screenshot below is the code for the testbench that thoroughly tests the 4-bit counter module described above, titled *my\_counter*. There are the appropriate local registers and wires that will be plotted in the simulation waveforms in the next section. The *my\_counter* instance is declared as *uut*. Initially, all inputs are set to 0. Then, after 100 nanoseconds, *reset* is set to high and low after two sets of 25 nanoseconds to show that this does not affect output until *enable* is high. Once *enable* is high, the test waits for 325 nanoseconds before setting *reset* to high, which should bring *counter\_out* back to 0 synchronously. The always block creates the oscillating clock signal, which flips its signal every 10 nanoseconds throughout the runtime of the testbench. The resulting waveforms are seen in the next section.

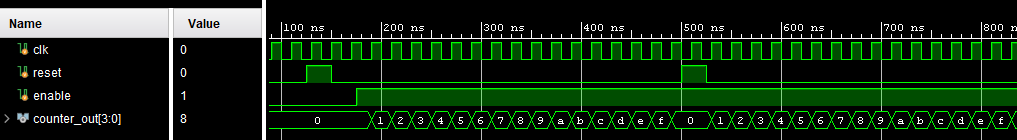
****

1. **Simulation Waveforms**

Below are screenshots of the Vivado Simulation waveforms that are produced by the robust testbench implemented by the team. The waveforms show the possible variations of the input signals *clk, reset,* and *enable* and how they appropriately change the 4 bit output *counter\_out*.



As seen above, when *reset* is low and *enable* is high, the output *counter\_out* is incremented on each positive edge of the system clock *clk* input. In the middle of the simulation, the testbench sets *reset* to high while *enable* is high to demonstrate a reset to the output value of *counter\_out* back to 0. In the beginning of the simulation, *enable* is low and no changes are made to the output, regardless of *reset*'s value so *counter\_out remains* 0 until the enable signal goes high and the positive clock edge occurs (counting up to 1 at about 190ns). Below is a closeup image of the previous complete waveform for reference and easier visualization.



Using the results from the above waveforms, the team implemented the necessary output assignments and constraints to produce traceable LED output on the Basys3 FPGA board. This consisted of selecting 4 consecutive LEDs on the board and setting each to a different bit of the 4-bit counter (i.e. LED PACKAGE\_PIN U16 => get\_ports {counter\_out[0]} … ). In addition, reset and enable were controlled by a couple of switches on the board (Package Pins V16 and V17) Since the system clock runs at a very high frequency, the LEDs would be lit up at a pace that is indiscernible by the human eye. Thus, a clock divider allowed the LEDs to light up at the right pace for viewers to notice the incrementing binary values in the LEDs. This visual output on the FPGA was checked off by the TA.

1. **Conclusion**
   1. **Challenges**

The main challenges in this lab process were produced from the FPGA implementation of the 4 bit counter for the demo, many stemming from the learning curve of translating functioning Verilog code to functioning output on the hardware. First, there were issues with the constraints file, where we had uncommented unused pins on the FPGA board and were not passing viable ports for these pins. After learning the necessary pins and passing the counter output into the LEDs, as well as naming the input signals appropriately to match the switches on the board, the constraints stopped producing errors and the bitstream completed successfully. Second, the LEDs were finally lighting up, but would light up all at once and at a lower brightness. This is because the 4-bit counter ran on the system clock, which has an extremely high frequency. The LEDs were updating at a speed that was too fast to see changes. After working with different clock dividers, the team found a clock that produced appropriate and human visible LED updates whenever the *enable* switch was on. However, our final challenge came when the *reset* signal did not reset the *counter\_out* output (all LEDs were not off). When producing the divided clock signal, the reset signal did not carry through the correct state updates. Thus, by detecting the reset signal on the fast system clock edges and enabling the slow clock when detected, *counter\_out* receives the reset signal and returns to 0.

* 1. **Contributions**

Both team members were active and enthusiastic throughout the implementation, debug, and simulation process. The team utilized knowledge from the pre-req class to work through the issues and challenges described above collaboratively. In terms of work division, Rodrigo focused more on the implementation of the 4-bit counter, while Chris focused on the robust testbench and simulation process. One of the lab periods required a bit of extra time to stay over, and both members willingly stayed to work through remaining issues. In the end, it was a positive and collaborative group experience.