Decrementing Counter

Ben Heard

Abstract

The second project implements a size counter that accepts a new size and then tracks the number of bytes that have been written before indicating that the checksum computation is complete. This lab is the start of the implementation for that counter.

An important difference between the lab and the project is that this lab does not require you to implement a loading mechanism for the counter. Instead, each time the counter returns to its initial state it will load with the value 87 (decimal).

1 Getting the Content

This has been covered a number of times in the previous labs. Please refer to them for reference. Once you pull the latest from your repository you will have a new folder called lab_007 and, in it, there will be a README file, modelsim.ini file, and this PDF.

2 Input/Output

NOTE: This won't be a direct drop in to your project. You will still need to provide a mechanism to load a size value.

The above being said, the signals to the block are shown below.

1. **rst_n**: Active low synchronous reset
2. **clock**: Free-running clock
3. **data_valid**: Indication that you should decrement

4. **last**: Output indicating that this clock edge is the last byte

The output, last, is a signal that may be used by your controller to know when the correct number of bytes has been received. For this lab you will just drive the **last** signal as a top level output.

3 Requirements

The system must conform to the following requirements.

1. The module shall be called **size_count**. The name of the file that contains the module may be of a different name but the name of the module must be **size_count**.

2. The module shall have only the ports listed above.

3. Out of reset, the **last** output shall be low.

4. Out of reset, the initial value of your internal **count** shall be decimal 87, binary 32'b0000_0000_0000_0000_0000_0000_0101_0111.

5. Following reset, the counter will decrement at the clock edge each time the input **data_valid** is asserted.

6. On the clock edge that the 87^{th} **data_valid** pulse occurs, the system will assert **last** and clear itself back to the value decimal 87.

7. The process then, repeats, tracking each time **data_valid** asserts and driving a pulse on **last** on the 87^{th} byte seen.

That requirement about last is important. Since the project requirement is that you drive a valid signal along with the computed checksum in the clock cycle following the last data byte your project will need to perform some action as it captures the final byte. Specifically, it will need to move to a state where it can drive valid. That is why we specify that **last** assert along with capturing the final byte of the message.

More practically, this means that the **last** output must be a Mealy type output dependent on both the state (the value in the counter) and the input **data_valid**. Note, that there is no requirement for a seperate state machine, the implementation may be a controlled counter. Recall from lecture that

when we implement counters the controller ended up being a single state with Mealy outputs meaning that the machine wasn't necessary and we could connect the inputs directly to the counter datapath.

4 Timing

The testbench provided with the encrypted solution produces the following timing diagram. Notice the relationship of **last**.

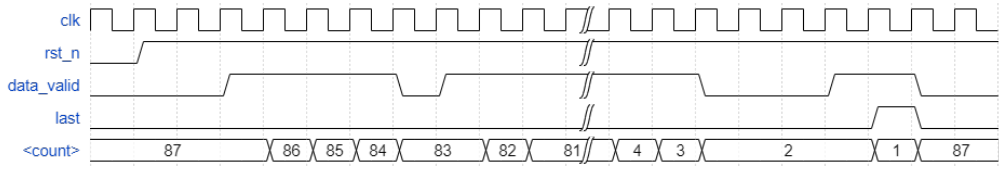

Figure 1: Timing Diagram

5 Implementation

There is nothing particularly special about the implementation and you are free to choose any style of implementation (structural, dataflow, procedural) for your Verilog. As always, I highly recommend drawing out your schematics and before beginning to write any Verilog.

6 What to Do

6.1 Git

Push your implementation to GitHub for grading. Provide the SHA that you would like graded in the Quiz for the assignment. If none is provided, we will grade the latest commit before the due date for the assignment.

7 Grading

Grading will be based on the following criteria.

- Was anything submitted?
- Did it compile with vlog *.v without errors or warnings?
- Did it load into the simulator with the provided example testbench?
- Did it run with the provided example testbench?
- Did it produce correct results with the provided example testbench?

7.1 Grading Notes

There are a few things to keep in mind when submitting your files for grading. Follow these rules to ensure that your assignment is graded.

1. All Verilog files necessary for the assignment shall be in the lab 7 directory, not a subdirectory.
2. All files needed for your implementation shall match the glob *.v. This just means that if there are Verilog files needed for your design you will name them with the .v extension.
3. The module shall be named **size_count**. The filename may be whatever you choose but the module name has been specified.
4. The module ports shall be as above. The order doesn't matter but the names and directions must match.