Project 2: Size Counter

Ben Heard

April 13, 2021

Abstract

The second project requires accepting a size that indicates the number of bytes that your offload engine will receive. Once the data start signal asserts you will have to track how many bytes you've received to know when you have received the entire message and can provide the computed checksum.

In this lab you will implement the size counter for project 2.

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. Although, you should be able to integrate it with you other implementation easily.

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

1. **rst_n**: Active low synchronous reset
2. **clock**: Freerunnning clock
3. **size**: A 32-bit value representing the number of bytes in the message.

4. **size_valid**: Indication that the size should be captured

5. **data_start**: Indication that you should begin decrementing

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

The inputs behave the the same as in the project. The output, however, 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. At the rising clock edge when **size_valid** is asserted, the system shall capture the **size** input as the number of bytes to capture.

5. Once a **size** has been captured the system shall wait until **data_start** asserts. The system shall begin counting rising clock edges in the cycle after **data_start** has asserted.

6. The system shall track the number of rising clock edges that have occurred since **data_start** and assert the **last** signal such that it is high in the same cycle in which the last data byte will be received.

7. Once the system has received the correct number of bytes and asserted the **last** output it shall return to a beginning state where it will be able to received another **size** and start the process over.

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.

4 Timing

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

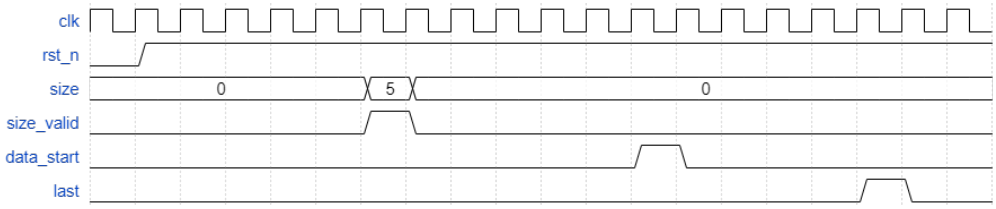

Figure 1: Timing Diagram

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 state diagrams 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?
- Did it run to completion with the hidden testbench(es)?
- Did it produce correct results for the hidden testbench(es)?

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.