**CME 435: Verification of Digital Systems**

**Final Project**

**Ryan Trampe**

**December 20, 2021**

**ryt805**

**Student # 1125 4939**

**Testbench Usage Instructions**

*Usage Instruction:*

To run each phase, simply use command run\_phase#.csh, where # is the number phase that you want to run. Additionally, there is a run\_phase.csh that takes an integer 1-8 as an argument and runs that phase of the testbench.

Additionally, there is a regression\_test.csh script that runs the regression test of the testbench.

The script run\_phase9.csh supports the listing of all available script through the “-l” argument or the running of individual testcases through the “-t” option.

*Additional features:*

* Code to toggle DUT bugs are inside dut/dut\_top.sv
* Device outputs are written to a file that is placed in the *report* subdirectory named phase#\_out.txt
* Scoreboard outputs (including error details) are placed in the file phase#\_err.txt
* HTML output are placed inside the subdirectory *report/html\_output*
* When running the regression test, a text version of the ucdb is made under report named *xswitch\_cc.rpt*
* A more detailed version of the coverage report is made named *detailed\_xswitch\_cc.rpt*

*Writing testcases:*

To write a testcase, you would need to create the .sv file inside verification/phase9\_testcases/testcases directory. Make sure to name the file the same as the name of the testcase. Afterwards, you need to include the testcase .sv file inside the xswitch\_test\_pkg.sv and write the name of the file to both the testbench program (testbench.sv) and as an added element to the list of testcases in the run\_phase9 script (script/run\_phase9.csh).

**Observed Bugs**

1) Output Port 0 FIFO not being written to

One of the bugs caught by the testbench in the xswitch is that when all ports are valid to be written into the xswitch and there is a stream of valid writes to device, the FIFO storage associated with output port 0 will not accept several values. Looking at the results from the testcase fifo\_case\_1, the device will ignore inputs from a stream 256 valid writes of transaction starting at the 192nd transaction until the 216th transaction, totalling 21 clock cycles of data that was not stored in FIFO storage of output port 0.

2) Data\_rcv for Ports deasserting

Another bug caught by the testbench is that the data\_rcv deasserts even when the port has already received the signal. This bug was found on the crv test case. Examining the outputs, there are cases where data\_rcv is deasserted when there is still room in the target FIFO for the data and address pair to be written to. When reading the data after the transaction, the device still outputs the original data that was written to the FIFO. In short, data\_rcv deasserts even if there is still space to take data and still takes the data as input.

3) Data\_rdy for ports deasserting

The final bug that is observed in the testbench is that data\_rdy will deassert itself even when the FIFO associated with that port is non-empty. The data\_rdy port will correct itself and reassert on the next clock if there is data on the associated port FIFO. When reading from a transaction with an invalid data\_rdy that was asserted, the associated port would have their data-address pair change as if a new pair of data was dequeued from the device fifo. This bug was found on the crv testcase when errors saying that the data\_rdy is deasserting were showing up on the error messages.

**Functional Coverage**

The testbench’s total functional coverage on the xswitch is 93.75%.

The coverage space of the testbench makes sure that each value coming in and out of the xswitch is hit at least. once. There are also a few crosses that help with covering more functionality of the xswitch. The first cross is between the wr\_en and data\_rcv ports to cover the functionality of the source port. This cross is meant to provide coverage for all the possible outcomes that data\_in has and the different responses that data\_rcv can have due to the state of the inner port FIFO. The second cross is between the rd\_en and data\_rdy ports to cover the functionality of the destination port. This cross is meant to provide coverage for all the possible machine states that the device is in by reading the data\_rdy port while crossing with the possible requests that can be made in the rd\_en inputs.

There are a few holes in functional coverage space for this testbench. One of the holes with this testbench is related to the FIFO ports. The holes that are present are the different configuration that each of the fifo ports can have. The testcases that are currently in this testbench all test the fifo operations port by port, so cases where a specific bit is asserted are covered. This leaves cases where multiple bits are asserted, such as when 3 out of 4 fifos are full, not being covered by the direct testing. The general constrained random verification fails to meet these coverage space too because they do not have the transaction count to be able to fill up the fifos, given the probabilities that determine if a transaction is reading or writing.

**Testbench Performance**

The verification of this testbench is able overlap the verification of multiple features by distributing them across multiple cases. In this way, multiple features are being verified with each testcase. However, each testcase is written with the idea that the testcase is trying to confirm a single feature and that feature is the first thing to examine if there were any errors to come up. There are only 2 cases of general testcases that try to cover as much as it can: the sanity\_check testcase and the crv testcase. The sanity\_check testcase tries to confirm that the basic functionality of the xswitch of begin able to take in signal and outputting to the destination port is still functioning. On the other hand, the crv testcase is a general constrained random case that tries to be as random as possible while still having some constraints that are adhering to the limitations of the device.

This method of overlapping feature checks does leave some features out. The most glaring feature to be left out is when multiple ports are being written to and read from. Each direct case from this testbench is looking at a specific port which leaves the interaction between the ports and their fifos to not be covered.

Overall, this testbench can cover all the features that are built into every port individually but unable to cover features that involve ports interacting with each other. For example, the testcases will not be able to specifically check how writing to fifo 0 affects the data that are already in the other fifo. In terms of execution time, this testbench is relatively fast as it can output the error messages and device outputs int a separate file instead of writing the results to standard out, which would significantly slow down the completion of the regression test. As a result, the regression test of this testbench is relatively fast though it comes with the cost of not being able to see the output results of all the cases that are being run by the regression test. To see the outputs of a specific testcase, the user would need to run the testbench with that specific testcase as an argument using the run\_phase9 script.

**Testbench Improvements**

One improvement is to have a separate directory where you can place the files that Questasim generates i.e. work folder. From here you would be able to delete the folder contents and make them with each compiler to ensure that the newest version of the testbench and the device.

The addition of more testcases that can deal with interactions between each port, especially when testing fifo port outputs to be able to get 100% functional coverage.