Shape

Description automatically generated with medium confidence

CME 435 – Project Report

Electrical & Computer Engineering

University of Saskatchewan

**CME435 Project Report​**

**​​Xswitch Testbench​**

|  |  |  |
| --- | --- | --- |
| Date: |  | ​​2022-12-12 |
| Document: |  | Xswitch Testbench |
| Authors: |  | Vu Nguyen (vun786),  Brix Marcellana (bem346) |

The information contained in this document is confidential, privileged and only for the information of the intended recipient and may not be used, published, or redistributed without the prior written consent of the University of Saskatchewan.

1. **Embedded Bugs**

**A.1 -** The fifo\_af and fifo\_full toggled incorrectly and indicate the wrong values for the actual size of the fifo. The fifo\_af and fifo\_full are specifically wrong for the 3rd least significant bit and the 2nd most significant bit. This indicates that both the indicators are wrong for port 2. Due to this bug, data\_rcv is also indicating wrong outputs based on the current state of the previous write since data\_rcv turns to 0 when the fifo is full. This bug can lead to multiple types of errors: either the buffer for port 2 could be smaller compared to the other buffers, or the fifos are just indicating the wrong outputs. Nevertheless, the bug was caught using the write\_full\_test which writes into the fifo until the buffer is full. The testcase caught all three errors for fifo\_af, fifo\_full, and data\_rcv.

**A.2 -** addr\_out[47:32] and data\_out[47:32] were both outputting the wrong values for the read. Based on the bugs caught on A.1, it could be suspected that the values were somehow discarded when the fifo for port 2 indicated that it was full. The simulation output indicates that the values were still stored in the fifo since the data\_rdy was correct for the entire reading sequence until the fifo was empty. Although the values were stored in the fifo, the values read in the output were completely different. The output address and data for the port was altered near the start of the writting process in the switch. This bug was seen using the read\_and\_write test where it caught both addr\_out and data\_out near the end of the reading sequence.

**A.3 -** data\_out[31:16] for port 1 was outputting the wrong values near the end of the reading sequence for read\_and\_write\_test. Unlike port 2, the fifos for port 1 were indicating the correct values for both reading and writting. The reading sequence, however, caught the data\_out for port 1 to be wrong for the latter half of the simulation. The address going along with the data for port 1 was correct, but the data value was somehow altered along the way.

1. **Coverage Space and Holes**

**B.1 -** Functional Coverage

Covergroup cg\_reset (66.66% coverage):

* Contains the bins that cover all 2 combinations of reset. The reason we did not hit 100% is because we did not include the case when reset value turns from 1 to 0.

Covergroup cg\_data\_in (92.50% coverage):

* Contains the coverage for data\_in where the 64 bits are divided into 4 16-bit groups
* Considering its size, the 16-bit groups divided into 8 bins
* Covers both max and min values

Covergroup cg\_addr\_in(90.00%):

* Contains the coverage for addr\_in where the 64 bits are divided into 4 16-bit groups
* Considering its size, the 16-bit groups divided into 8 bins
* Covers both max and min values

Covergroup cg\_data\_input\_indicators(77.86%):

* Contains the coverage for the 16 possibilities of of rd\_en
* Contains the coverage for the 16 possibilities of of wr\_en

Covergroup cg\_wr\_en\_toggle(100.00%):

* Contains the coverage for the toggle of the 4 least significant bits of wr\_en

Covergroup cg\_rd\_en\_toggle(100%):

* Contains the coverage for the toggle of the 4 least significant bits of rd\_en

Covergroup cg\_config(96.15%):

* Contains the coverage of the 1-bit port\_en and its toggle
* Contains the coverage of the 1-bit port\_wr and its toggle
* Contains the coverage of the port\_sel for all 4 cases
* Contains the toggle coverage of port\_sel for both of the least significant bits
* Contains the coverage for port address where the 16 bit is split into 8 bins
* Covers the max and min value of port address

Covergroup cg\_fifo(43.05%):

* Contains the bins that cover fifo\_empty, fifo\_full, fifo\_ae, and fifo\_af. Covers the 16 combinations of each fifo.

Covergroup cg\_fifo\_empty\_toggle(100%):

* Contains the toggle coverage of the 4 least significant bits of fifo\_empty.

Covergroup cg\_fifo\_almost\_empty\_toggle(1.46%):

* Contains the toggle coverage of the 4 least significant bits of fifo\_ae.

Covergroup cg\_fifo\_full\_toggle(93.75%):

* Contains the toggle coverage of the 4 least significant bits of fifo\_full.

Covergroup cg\_fifo\_almost\_full\_toggle(1.26%):

* Contains the toggle coverage of the 4 least significant bits of fifo\_af.

Covergroup cg\_data\_output\_indicators(68.75%):

* Contains the coverage for data\_rcv for all 16 possibilities.
* Contains the coverage for data\_rdy for all 16 possibilities.

Covergroup cg\_data\_rcv\_toggle(100%):

* Contains the toggle coverage for data\_rcv for with its 4 least significant bits.

Covergroup cg\_data\_rdy\_toggle(100%):

* Contains the toggle coverage for data\_rdy for with its 4 least significant bits.

Covergroup cg\_data\_out(90.0%):

* Contains the coverage for data\_out where the 64 bits are divided into 4 16-bit groups
* Considering its size, the 16-bit groups divided into 8 bins
* Covers both max and min values

Covergroup cg\_addr\_out(20.0%):

* Contains the coverage for addr\_out where the 64 bits are divided into 4 16-bit groups
* Considering its size, the 16-bit groups divided into 8 bins
* Covers both max and min values

The functionalities that are not covered in the functional coverage are the bits that are insignificant to the features of the dut. This involves:

* the 6 most significant bits of port select
* the 4 most significant bits of data\_rcv
* the 4 most significant bits of data\_rdy
* the 4 most significant bits of wr\_en
* the 4 most significant bits of rd\_en

1. **Testbench Performance**

Does the testbench verify the same set of design features in multiple testcases feature while ignoring other features?

The testbench was able to cover most if not all of the design features. The testcases intersect between each especially the read and write since testing these features would have prerequirements such as configurations and premodified buffer states. The testbench ignores features that are not required for the dut, this includes specific bits of the wr\_en, rd\_en, data\_rcv, data\_rdy, and port\_sel. Features heavily tested are the configuration mechanism of the xswitch, the storage of the data into the fifo, and the reading capability from the ports.

Does the testbench cover all design features effectively?

Since the testbench has multiple test cases for each feature, all design features are covered effectively while running the regression test. However, the testbench is controlled to have a low number of samples while doing coverage, keeping the coverage low while having a faster simulation time.