COVERAGE REPORT

This project examined two forms of coverage: functional coverage and code coverage. Code coverage quantifies the extent to which the simulation exercised the RTL structure. This include measurements like FSM (Finite State Machine) coverage, condition and expression coverage, line coverage, and toggle coverage. Conversely, functional coverage shows the proportion of planned actions and scenarios that were actually triggered during testbench execution, as defined by SystemVerilog covergroups.

**Coverage Types and Tools:**

Code Coverage**:** Includes line, toggle, condition, expression, and FSM coverage.

Functional Coverage**:** Measures whether all defined functional scenarios (via covergroups) have been exercised.

Code coverage approached 100% but functional coverage did not. In order to observe situations including mode transitions, input combinations, error conditions, and configuration settings, a number of covergroups were established in order to capture functional intent. Not all coverpoints and bins were hit, though, according to an examination of the coverage reports (such as cvg.html and cvgBins.html). This suggests that some functional scenarios were not tested throughout the simulation; they could be corner instances or particular sequences.

The amount of source code that the testbench exercised during simulation is measured by code coverage. The analysis's findings show that all of the design's source code was used. Line, toggle, condition, branch, and expression coverage are among the metrics that make up the code coverage. The logic in charge of write and read pointer operations, data storage through memory, and status signal generation for full and empty states were all exercised, as were all executable lines within the FIFO RTL. Every bit in important data and control signals, including the write and read pointers, the FIFO data bus (wr\_data, rd\_data), and the memory array, alternated between 0 and 1 at least once during the simulation, according to the toggle coverage.

SystemVerilog covergroups that were instantiated within the UVM agent's monitor component were used to define and gather functional coverage. Throughout the simulation, these covergroups monitored significant combinations of functional conditions. In particular, distinct covergroups were established for the FIFO's write and read interfaces, enabling the observation of important control signal exchanges.  
  
Two coverpoints were part of the write covergroup: full, which is the FIFO full flag, and wr\_en, which is the write enable signal. All four significant scenarios—write attempted while full, write attempted while not full, write not attempted while full, and write not attempted while not full—were seen, according to cross-coverage between these two coverpoints. This cross verified that both safe and error-prone write scenarios were investigated by the testbench.

The pointer logic uses Gray-coded addressing, where some bits—especially the higher-order ones—only toggle after multiple full FIFO cycles. For example, the second MSB changes only once every 128 writes per full wrap. To capture both rising and falling edges and ensure toggle coverage on that bit, the testbench must continuously assert wr\_en for at least 256 cycles even after the FIFO reaches full. However, the current tests are configured to stop shortly after reaching the full condition, preventing this transition from occurring. As a result, some toggle bins remain uncovered despite the design functioning correctly.

In conclusion, the 100% code coverage shows that the testbench is architecturally complete. To guarantee complete validation of the behavior of the design, it is necessary to handle the missing bins from a functional standpoint. It is advised to examine the unhit bins and develop targeted test cases or upgrade the current constrained-random stimulus to exercise the missing scenarios in order to increase functional coverage. A more reliable and validated design should come from rerunning the simulation after those issues have been resolved, which should assist boost functional coverage closer to 100%.