**Signals description and modifications**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Signal | Direction | Description | Bugs | Fix | Check |
| data\_in | Input | The input data bus used when writing the FIFO. | No bugs founded | No fix needed | Using the reference model implemented in FIFO\_scoreboard\_pkg |
| wr\_en | Input | If the FIFO is not full, asserting this signal causes data (on data\_in) to be written into the FIFO | When the fifo memory is empty and wr\_en and rd\_en are high the internal counter doesn’t increment. So, the data stored will be overwritten by the following write operation | Added condition if the wr\_en and rd\_en are high at the same time and the fifo is empty, the internal counter increments | Using the reference model implemented in FIFO\_scoreboard\_pkg |
| rd\_en | Input | If the FIFO is not empty, asserting this signal causes data (on data\_out) to be read from the FIFO | When the fifo memory is full and wr\_en and rd\_en are high the internal counter doesn’t decrement. So, the data stored will be readed again by the following read operation | Added condition if the wr\_en and rd\_en are high at the same time and the fifo is full, the internal counter decrements | Using the reference model implemented in FIFO\_scoreboard\_pkg |
| clk | Input | Clock signal | No bugs founded | No fix needed | - |
| rst\_n | Input | Active low asynchronous reset | No bugs founded | No fix needed | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (rst\_check) |
| data\_out | Output | The sequential output data bus used when reading from the FIFO. | No bugs founded | No fix needed | Using the reference model implemented in FIFO\_scoreboard\_pkg |
| full | Output | This combinational output signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO. | No bugs founded | No fix needed | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (full\_flag) |
| almostfull | Output | This combinational output signal indicates that only one more write can be performed before the FIFO is full. | Asserted when the fifo still have two empty locations to store | Modified the condition | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (almostfull\_flag) |
| empty | Output | This combinational output signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO. | No bugs founded | No fix needed | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (empty\_flag) |
| almostempty | Output | This output combinational signal indicates that only one more read can be performed before the FIFO goes to empty. | No bugs founded | No fix needed | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (almostempty\_flag) |
| overflow | Output | This sequential output signal indicates that a write request (wr\_en) was rejected because the FIFO is full. Overflowing the FIFO is not destructive to the contents of the FIFO. | No bugs founded | No fix needed | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (overflow\_flag) |
| underflow | Output | This sequential output signal Indicates that the read request (rd\_en) was rejected because the FIFO is empty. Under flowing the FIFO is not destructive to the FIFO. | Designed as combination signal | Modified to be sequential | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (underflow\_flag) |
| wr\_ack | Output | This sequential output signal indicates that a write request (wr\_en) has succeeded. | Designed as combination signal | Modified to be sequential | Using the reference model implemented in FIFO\_scoreboard\_pkg, and inserted assertion in the design file called (wr\_ack\_flag) |

**The verification plans**

* Designed reference model that behaves as the required design by its own specs in FIFO\_scoreboard\_pkg. This package has function that takes the input given by the testbench and checks the outputs by calculating it using the reference model function and takes the output of the design and then compare it to the reference model signals.

If the outputs are correct, increments the (correct\_count) counter in shared\_pkg package. If the outputs have any wrong signal, increments (error\_count) counter in shared\_pkg package and display message by the time of simulation that the error occurred at.

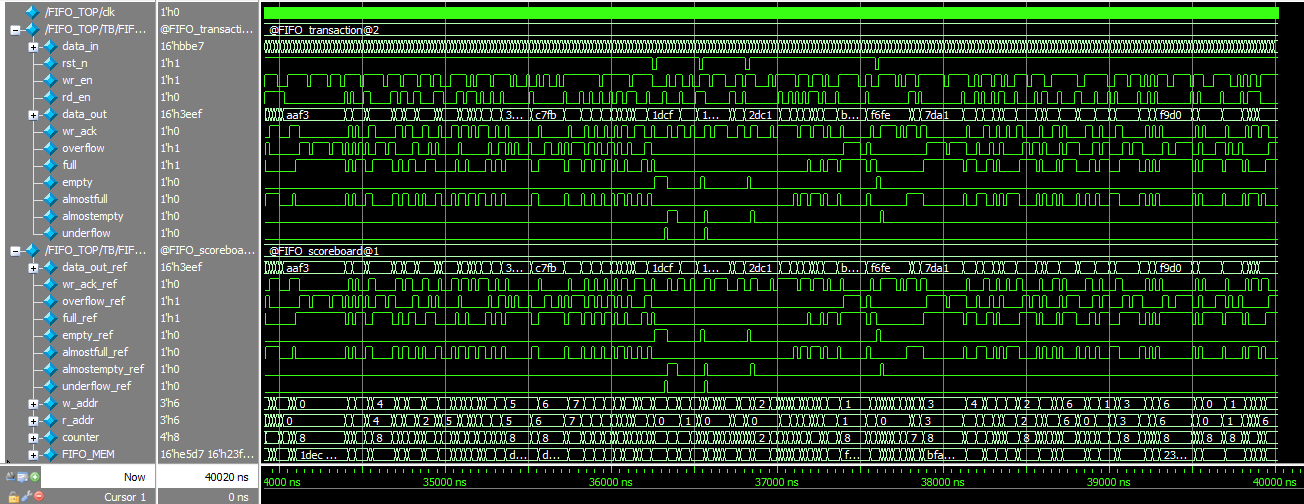
* There is object declared at FIFO\_transaction\_pkg that simulates the signals (inputs and outputs of the design), at the testbench we determine the number of tests required. And then:

1. Initialize the system by initializer task.
2. Randomize the transaction object and assign the inputs to the specified signals of the object.
3. After one clock cycle it must to force the wr\_en and rd\_en to be zero; because the test takes two clock cycles and if wr\_en was one it will write the same data two times, and assign the outputs to the object.
4. Call sample\_data method and pass the object to it; to check the functional coverage.
5. Call check\_data method and pass the object to it; to check the functionality of the design. Does it work as required or not?
6. After finishing the required number of tests, the TB assigns test\_finished to ‘1’ in shared\_pkg to inform the monitor that the test finished and required now to print the correct\_count and error\_count counters.

* The functional coverage detected by the cover group cg declared at FIFO\_coverage\_pkg.

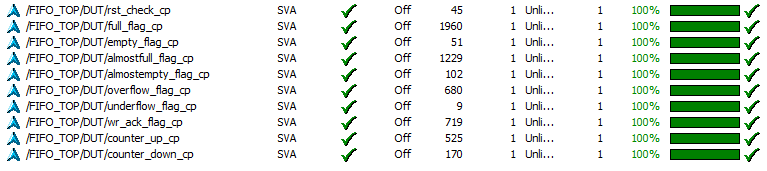
We used 2000 test case to verify the design:

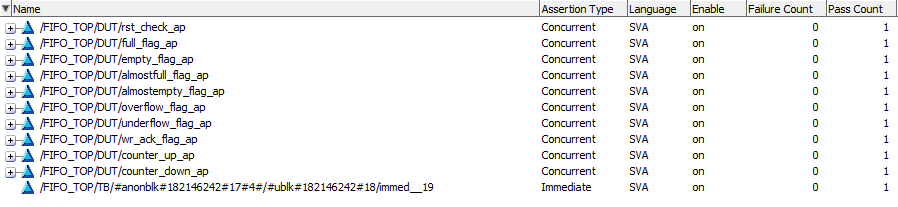




**Embedded assertions**

|  |  |
| --- | --- |
| property name | Description |
| rst\_check | When the rst\_n is low, at the following clock edge wr\_ptr, rd\_ptr and count must be ‘0’ |
|  | |
| full\_flag | When the count equals 8, at the same time full must be ‘1’ |
|  | |
| empty\_flag | When the count equals 0, at the same time empty must be ‘1’ |
|  | |
| almostfull\_flag | When the count equals 7, at the same time almostfull must be ‘1’ |
|  | |
| almostempty\_flag | When the count equals 1, at the same time almostempty must be ‘1’ |
|  | |
| overflow\_flag | When the full is ‘1’ and the wr\_en is ‘1’, at the following clock edge overflow must be ‘1’ |
|  | |
| underflow\_flag | When the empty is ‘1’ and the rd\_en is ‘1’, at the following clock edge underflow must be ‘1’ |
|  | |
| wr\_ack\_flag | When the wr\_en is ‘1’ and the count less than 8, at the following clock edge wr\_ack must be ‘1’ |
|  | |
| counter\_up | Checks the count increments after every write operation |
|  | |
| counter\_down | Checks the count decrements after every read operation |
|  | |





**The cover-group**

The detected cross coverage:

|  |  |  |
| --- | --- | --- |
| Cover point name | Signal one | Signal two |
| wr\_en\_rd\_en\_cross | wr\_en | rd\_en |
| wr\_en\_wr\_ack\_cross | wr\_en | wr\_ack |
| wr\_en\_overflow\_cross | overflow |
| wr\_en\_full\_cross | full |
| wr\_en\_empty\_cross | empty |
| wr\_en\_almostfull\_cross | almostfull |
| wr\_en\_almostempty\_cross | almostempty |
| wr\_en\_underflow\_cross | underflow |
| rd\_en\_wr\_ack\_cross | rd\_en | wr\_ack |
| rd\_en\_overflow\_cross | overflow |
| rd\_en\_empty\_cross | empty |
| rd\_en\_almostfull\_cross | almostfull |
| rd\_en\_almostempty\_cross | almostempty |
| rd\_en\_underflow\_cross | underflow |

