

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	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

		can be performed before the FIFO is full.			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

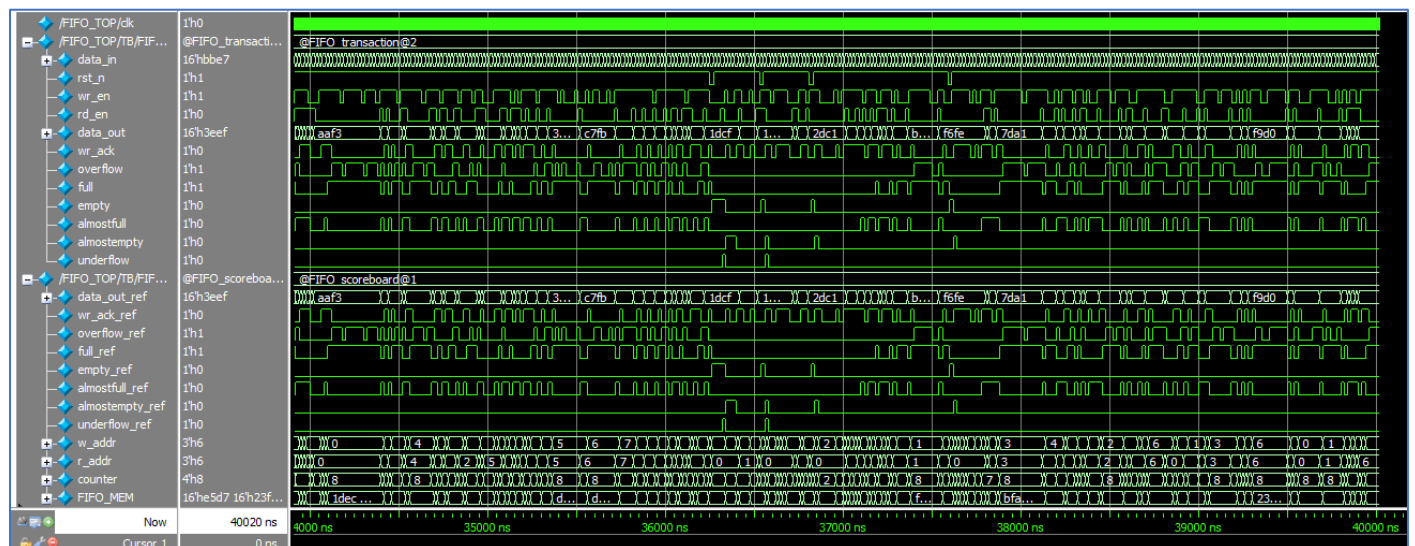
- Designed reference model that behaves as the required design by its own specs in FIFO_scoreboard_pkg. This package has function that takes the input given by the testbench and checks the outputs by calculating it using the reference model function and takes the output of the design and then compare 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.






































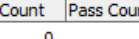

We used 2000 test case to verify the design:











```
# /*****/
# ERRORS:          0
# CORRECT:         2000
# /*****/
```



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'
@ (posedge F_if.clk) (!F_if.rst_n) => ((wr_ptr == 0) && (rd_ptr == 0) && (count == 0));	
full_flag	When the count equals 8, at the same time full must be '1'
@ (posedge F_if.clk) disable iff (!F_if.rst_n) (count == (FIFO_DEPTH)) -> F_if.full;	
empty_flag	When the count equals 0, at the same time empty must be '1'
@ (posedge F_if.clk) disable iff (!F_if.rst_n) (count == 0) -> F_if.empty;	
almostfull_flag	When the count equals 7, at the same time almostfull must be '1'
@ (posedge F_if.clk) disable iff (!F_if.rst_n) (count == FIFO_DEPTH-1) -> F_if.almostfull;	
almostempty_flag	When the count equals 1, at the same time almostempty must be '1'
@ (posedge F_if.clk) disable iff (!F_if.rst_n) (count == 1) -> F_if.almostempty;	
overflow_flag	When the full is '1' and the wr_en is '1', at the following clock edge overflow must be '1'
@ (posedge F_if.clk) disable iff (!F_if.rst_n) (!F_if.wr_en && count < (FIFO_DEPTH)) && (F_if.full & F_if.wr_en) => F_if.overflow;	
underflow_flag	When the empty is '1' and the rd_en is '1', at the following clock edge underflow must be '1'
@ (posedge F_if.clk) disable iff (!F_if.rst_n) (!F_if.rd_en && count != 0) && (F_if.empty && F_if.rd_en) => F_if.underflow;	
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'
@ (posedge F_if.clk) disable iff (!F_if.rst_n) (F_if.wr_en && count < (FIFO_DEPTH)) => F_if.wr_ack;	
counter_up	Checks the count increments after every write operation
@ (posedge F_if.clk) disable iff (!F_if.rst_n) ((F_if.wr_en, F_if.rd_en) == 2'b10) && !F_if.full) => ((\$past(count) + 1) == count);	
counter_down	Checks the count decrements after every read operation
@ (posedge F_if.clk) disable iff (!F_if.rst_n) ((F_if.wr_en, F_if.rd_en) == 2'b01) && !F_if.empty) => ((\$past(count) - 1) == count);	

 /FIFO_TOP/DUT/rst_check_cp	SVA		Off	45	1 Unli...	1	100%		
 /FIFO_TOP/DUT/full_flag_cp	SVA		Off	1960	1 Unli...	1	100%		
 /FIFO_TOP/DUT/empty_flag_cp	SVA		Off	51	1 Unli...	1	100%		
 /FIFO_TOP/DUT/almostfull_flag_cp	SVA		Off	1229	1 Unli...	1	100%		
 /FIFO_TOP/DUT/almostempty_flag_cp	SVA		Off	102	1 Unli...	1	100%		
 /FIFO_TOP/DUT/overflow_flag_cp	SVA		Off	680	1 Unli...	1	100%		
 /FIFO_TOP/DUT/underflow_flag_cp	SVA		Off	9	1 Unli...	1	100%		
 /FIFO_TOP/DUT/wr_ack_flag_cp	SVA		Off	719	1 Unli...	1	100%		
 /FIFO_TOP/DUT/counter_up_cp	SVA		Off	525	1 Unli...	1	100%		
 /FIFO_TOP/DUT/counter_down_cp	SVA		Off	170	1 Unli...	1	100%		

Name	Assertion Type	Language	Enable	Failure Count	Pass Count
 /FIFO_TOP/DUT/rst_check_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/full_flag_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/empty_flag_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/almostfull_flag_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/almostempty_flag_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/overflow_flag_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/underflow_flag_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/wr_ack_flag_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/counter_up_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/DUT/counter_down_ap	Concurrent	SVA	on	0	1
 /FIFO_TOP/TB/#anonblk#182146242#17#4#/#ublk#182146242#18/immed__19	Immediate	SVA	on	0	1

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

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/FIFO_coverage_pkg/FIFO_coverage		100.00%				
TYPE cg		100.00%	100	100.00...		✓
+ CVP cg::wr_en_cp		100.00%	100	100.00...		✓
+ CVP cg::rd_en_cp		100.00%	100	100.00...		✓
+ CVP cg::wr_ack_cp		100.00%	100	100.00...		✓
+ CVP cg::overflow_cp		100.00%	100	100.00...		✓
+ CVP cg::full_cp		100.00%	100	100.00...		✓
+ CVP cg::empty_cp		100.00%	100	100.00...		✓
+ CVP cg::almostfull_cp		100.00%	100	100.00...		✓
+ CVP cg::almostempty_cp		100.00%	100	100.00...		✓
+ CVP cg::underflow_cp		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_rd_en_cross		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_wr_ack_cross		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_overflow_cross		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_full_cross		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_empty_cross		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_almostfull_cross		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_almostempty_cross		100.00%	100	100.00...		✓
+ CROSS cg::wr_en_underflow_cross		100.00%	100	100.00...		✓
+ CROSS cg::rd_en_wr_ack_cross		100.00%	100	100.00...		✓
+ CROSS cg::rd_en_overflow_cross		100.00%	100	100.00...		✓
+ CROSS cg::rd_en_empty_cross		100.00%	100	100.00...		✓
+ CROSS cg::rd_en_almostfull_cross		100.00%	100	100.00...		✓
+ CROSS cg::rd_en_almostempty_cross		100.00%	100	100.00...		✓
+ CROSS cg::rd_en_underflow_cross		100.00%	100	100.00...		✓