Project

Synchronous FIFO

Parameters

- FIFO_WIDTH: DATA in/out and memory word width (default: 16)
- FIFO_DEPTH: Memory depth (default: 8)

Ports

Port	Direction	Function
data_in	Input	Write Data: The input data bus used when writing the FIFO.
wr_en		Write Enable: If the FIFO is not full, asserting this signal causes data (on data_in) to be written into the FIFO
rd_en		Read Enable: If the FIFO is not empty, asserting this signal causes data (on data_out) to be read from the FIFO
clk		Clock signal
rst_n		Active low asynchronous reset
data_out	Output	Read Data: The sequential output data bus used when reading from the FIFO.
full		Full Flag: When asserted, 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.
almostfull		Almost Full: When asserted, this combinational output signal indicates that only one more write can be performed before the FIFO is full.
empty		Empty Flag: When asserted, 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.
almostempty		Almost Empty: When asserted, this output combinational signal indicates that only one more read can be performed before the FIFO goes to empty.
overflow		Overflow: 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.
underflow		Underflow: 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.
wr_ack		Write Acknowledge: This sequential output signal indicates that a write request (wr_en) has succeeded.

Overview of the testbench flow:

The top module will generate the clock, pass it to the interface, and the interface will be passed to the DUT, tb, and monitor modules. The tb will reset the DUT and then randomize the inputs. At the end of the test, the tb will assert a signal named test_finished. The signal will be defined as well as the error_count and correct_count in a shared package that you will create named shared_pkg.

The monitor module will do the following:

1. Create objects of 3 different classes (FIFO_transaction, FIFO_scoreboard, FIFO_coverage). These classes will be discussed later.
2. It will have an initial block and inside it a forever loop that has a negedge clock sample. With each negedge clock, the monitor will sample the data of the interface and assign it to the data of the object of class FIFO_transaction. And then after that there will be fork join, where 2 processes will run, the first one is calling a function named sample_data of the object of class FIFO_coverage and the second process is calling a function named check_data of the object of class FIFO_scoreboard.
3. So, in summary the monitor will sample the interface ports, and then pass these values to be sampled for functional coverage and to be checked if the output ports are correct or not.
4. After the fork join ends, you will check for the signal test_finished if it is high or not. If it high, then stop the simulation and display a message with summary of correct and error counts.