

## 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.

Note: If a read and write enables were high and the FIFO was empty, only writing will take place and vice versa if the FIFO was full.

#### 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 in the document.
2. It will have an initial block and inside it a forever loop that waits for negedge clock at the start of the loop and then sample the data of the interface and assign it to the data variables 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 method named sample\_data of the object of class FIFO\_coverage and the second process is calling a method 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.

#### Steps:

1. Adjust the design to take an interface and change the file extension to sv.
2. Create a package in a new file that will have a class named FIFO\_transaction
  - a. Inside of this class add the FIFO inputs and outputs as class variables of the class as well as adding 2 integers (RD\_EN\_ON\_DIST & WR\_EN\_ON\_DIST)
  - b. Add a constructor that takes 2 inputs and override the values of RD\_EN\_ON\_DIST and WR\_EN\_ON\_DIST, let the default of RD\_EN\_ON\_DIST be 30 and WR\_EN\_ON\_DIST be 70
  - c. Add the following 3 constraint blocks
    1. Assert reset less often
    2. Constraint the write enable to be high with distribution of the value WR\_EN\_ON\_DIST and to be low with 100-WR\_EN\_ON\_DIST
    3. Constraint the read enable the same as write enable but using RD\_EN\_ON\_DIST
3. Create a package in a new file that will have a class for functional coverage collection named FIFO\_coverage
  - a. Import the previous package (Add the import statement after the package declaration)
  - b. The class will have an object of the class FIFO\_transaction named F\_cvg\_txn.
  - c. Create a covergroup. The coverage needed is cross coverage between 3 signals which are write enable, read enable and each output control signals (outputs except data\_out) to make sure that all combinations of write and read enable took place in all state of the FIFO.
  - d. Create a void function inside it named sample\_data that takes one input named F\_txn. This input is an object of class FIFO\_transaction. This function will do the following

1. Assign F\_txn to F\_cvg\_txn
  2. Trigger the sampling of the covergroup using the .sample method
4. Create a package in a new file named FIFO\_scoreboard
- a. Import the FIFO\_transaction package.
  - b. Add variables for the data\_out\_ref, full\_ref, etc. to be used in the reference\_model function
  - c. Create a function named check\_data that takes one input which of type FIFO\_transaction
    1. Inside this function, call another function named reference\_model that you will create and pass to it the same object that you have received
    2. Reference\_model function will check the input values from the input object and assign values to the class properties data\_out\_ref, full\_ref, etc.
    3. After the reference\_model function returns, you will compare the reference outputs calculated with the outputs of the object received. Increment the error\_count or correct\_count. Also, display a message if error occurs.
5. Open the design file and add assertions to the FIFO inside the design file.
- a. Add assertions to all the FIFO output flags (outputs except data\_out) as well as the internal counters of the FIFO.
  - b. Extra part to be done: Guard the assertions using conditional compilation with the `ifdef directive with macro named SIM, and then include the macro in the vlog command +define+SIM option in the vlog command of your do file. Refer to [this](#) link to learn more about conditional compilation.

Note: since you have added assertions to the flags of the FIFO inside of the FIFO design, then the check\_result will check only for the data\_out

## Requirements

1. Verification Plan, where you will list your verification plan, an example of the document can be found in the link [here](#). In the functionality check column, you can specify whether the requirements are being checked only by the golden/reference model, assertion or both.
2. You are welcome to create your own constraints, coverpoints, and assertions. If you would like feedback or ideas, feel free to share your additions with me or your assigned teaching assistant.
3. Use Do file to run the top and make sure to generate the coverage report and check that the code coverage, functional coverage, and sequential domain (assertions) coverage are 100%
4. QuestaSim snippets, report any bugs detected and modify the RTL. Report the before and after to show the changes made to the RTL.

### Submission file:

.rar file containing the following:

- PDF file having the requirements with the bugs detected for the FIFO
- Do file to run the top
- SystemVerilog files & also add snippets to the code in the PDF, it will be easier for the grading process