

TB Implementation Report - MS2

The testbench for the async FIFO both instantiates the DUT and defines every class-based verification component. Stimulus originates in the Generator and travels through the Drivers to the DUT interface. Monitors watch the interface signals and forwards observations to the scoreboard. The scoreboard maintains a model of the DUT and checks reads are equal to writes and occur in FIFO order. All components communicate through mailboxes.

The `fifo_transaction` class is used to enable stimulus and observation in the testbench. Every other component either creates, consumes, or inspects a `fifo_transaction` object. The class body contains a one-bit enum `op_t` and an 8 bit value `data`. `op_t` distinguishes a write transaction from a read transaction, and `data` carries the read/write payload. If the transaction is a read, `data` is left blank at transaction creation.

The `fifo_generator` class creates all read/write transactions and distributes them to the corresponding driver. Mailbox handles for both drivers are contained in the `fifo_generator` class definition. This means the generator does not own the mailboxes but only has a reference for them. In the `run()` task, the generator first produces write transactions and sends them to the write driver before producing read transactions and sending them to the read driver.

The write driver operates in the `wclk` domain. It is initialized by `reset`, and `run()` begins on the falling edge of `wrst` to make sure the driver does not try to drive the DUT before system has exited `reset`. `Get()` blocks until the generator has placed a transaction into the driver. The handshake protocol then begins. `W_valid` is driven high on a rising edge of `wclk`, placing `tr.data` onto `w_data` with non-blocking assignments. When `w_ready` is high (FIFO is ready to accept data), the data is transferred and `w_valid` is set to low. This returns the driver to idle. The read driver functions the same as the write driver except operating in the `rlck` domain and on the DUT's read port.

The input monitor passively observes the DUT's write port, detects when a wire transfer has occurred, and forwards transferred data to the scoreboard. It samples `w_valid` and `w_ready`, making sure a write transfer has occurred when both are high. It runs entirely on `wclk` and sends the value of `w_data` through a mailbox for the scoreboard to consume. The output monitor functions the exact same as the input monitor except operating in the `rlck` domain and observing the DUT's read port.

The scoreboard serves three distinct roles: it serves as a reference model for the DUT's functional behavior, compares actual output against expected output, and defines the

functional coverage. Run() contains two threads running inside of a fork-join_none block. This allows the scoreboard to run the entire duration of the simulation while receiving data from both monitors at unpredictable times. The functional coverage contains one covergroup with two coverpoints, one for direction of FIFO transfer and one for FIFO fill range. The coverpoints are crossed to ensure both writes and reads have been observed at every fill level.