SYNCHRONOUS FIFO VERIFICATION (UVM)

Kiereia Ayman

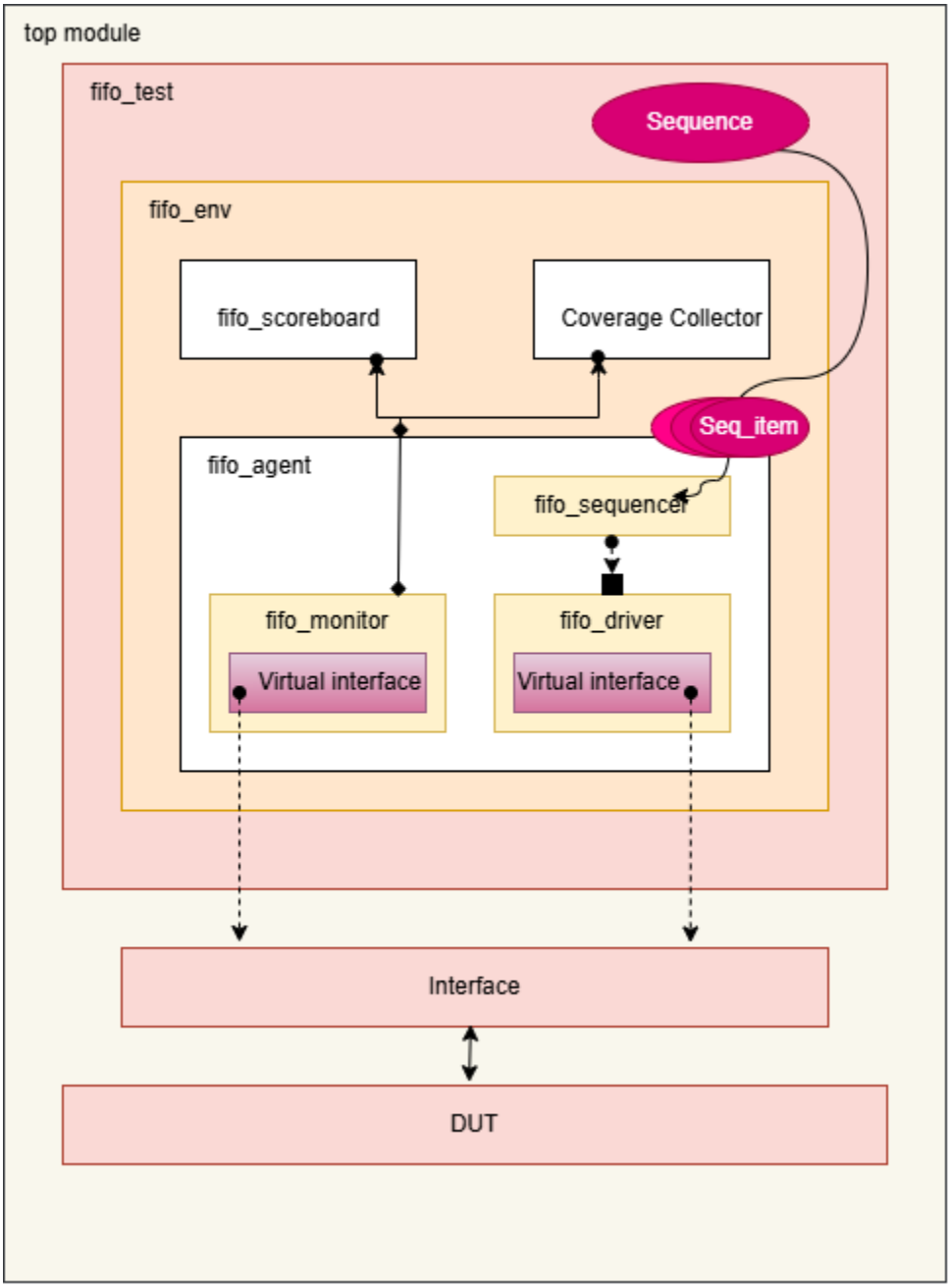
FIFO Port Description Table

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

Verification Plan

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_0	when reset is asserted, the DUT resets both pointers and counters	Directed at the start of randomization	-	Immediate assertion in the DUT
FIFO_1	write operation --> when write enable is asserted and read enable is disasserted	Directed at the start of randomization	-	Checker in the testbench (scoreboard)
FIFO_2	Read operation--> when read enable is asserted and write enable is disasserted	Directed at the start of randomization	-	Checker in the testbench (scoreboard)
FIFO_3	write and read operations are done simultansly but when empty only write is done and when full only read is done	Directed at the start of randomization	_	Checker in the testbench (scoreboard)
FIFO_4	Randomzing all the inputs to see how the DUT reacts	Randomized with constraints on write, read and reset	_	Checker in the testbench (scoreboard)
write ack	When a write enable signal (wr_en) is active and the FIFO is not full, wr_ack should be asserted to confirm the write operation.	Write enable is Randomized with constraint to be 70% of the time active	Cross coverage between wr_ack signal and read and write enables combinations	concurrent assertion
overflow	If a write is attempted when the FIFO is full, overflow should be asserted.	Write enable is Randomized with constraint to be 70% of the time active	Cross coverage between overflow signal and read and write enables combinations	concurrent assertion
underflow	If a read is attempted when the FIFO is empty, underflow should be asserted.	Read enable is randomized with constraint to be 30% of the time active	Cross coverage between underflow signal and read and write enables combinations	concurrent assertion
Empty flag	When the internal count is zero, the empty flag should be asserted.	Read enable is randomized with constraint to be 30% of the time active	Cross coverage between empty signal and read and write enables combinations	Immediate assertion
Full Flag	When the internal count equals the FIFO depth, the full flag should be asserted.	Write enable is Randomized with constraint to be 70% of the time active	Cross coverage between full signal and read and write enables combinations	Immediate assertion
Almost Full	When the count reaches FIFO depth - 1, almostfull should be asserted.	Write enable is Randomized with constraint to be 70% of the time active	Cross coverage between almostfull signal and read and write enables combinations	Immediate assertion
Almost empty	When the count equals 1, the almostempty signal should be asserted.	Read enable is randomized with constraint to be 30% of the time active	Cross coverage between almostempty signal and read and write enables combinations	Immediate assertion
Pointer Wraparound	After writing or reading FIFO_DEPTH entries (0 to 7), the write or read pointer should eventually wrap around back to 0. Same applies for the counter (0 to 8).	_	-	concurrent assertion
Pointer Threshold	Internal pointers cannot exceed the FIFO_DEPTH entries in any given time. Same applies for the counter.	_	-	concurrent assertion

UVM Drawing


top module
fifo_test
Sequence
fifo_env
fifo_scoreboard
Coverage Collector
Seq_item
fifo_agent
fifo_sequencer
fifo_monitor
Virtual interface
fifo_driver
Virtual interface
Interface
DUT

How the testbench works

• The top module instantiates the interface & DUT and binds the SVA (checks for the outputs correctness) then run uvm_test.

• A uvm_test starts execution.

• It creates the uvm_env, which contains everything else (agents, scoreboard, etc.).

• There are 4 different sequences[reset, write-only, read-only, write-read] that generate sequence_items (transactions).

• These transactions are passed to the uvm_sequencer, then to the uvm_driver.

• The uvm_driver converts the sequence items into pin-level signals and drives them to the DUT through a virtual interface.

• The uvm_monitor observes the DUT outputs, translates them into transactions, and forwards them to the scoreboard and coverage collector.

• The uvm_scoreboard checks data_out correctness, and the coverage collector tracks test coverage.

Detected bugs

1- Underflow was combinational. It must be sequential.
2- Overflow must be set to low when wr_en is high and count is less that FIFO_DEPTH.
3- Overflow must be set to high when rd_en is high and count isn't zero.
4- Wr_ack, overflow and underflow must be set to zero at reset.
5- almostfull becomes high when the count equals FIFO_DEPTH – 1 not FIFO_DEPTH – 2 .
6- count isn't handled when both the wr_en and rd_en are high.

Transcript

```
# ----------------------------------------------------------------
# UVM_INFO fifo_test.sv(47) @ 2: uvm_test_top [run_phase] reset deasserted
# UVM_INFO fifo_test.sv(48) @ 2: uvm_test_top [run_phase] stim gen started
# UVM_INFO fifo_test.sv(50) @ 30002: uvm_test_top [run_phase] stim gen ended
# UVM_INFO fifo_test.sv(51) @ 30002: uvm_test_top [run_phase] reset asserted
# UVM_INFO fifo_test.sv(53) @ 50002: uvm_test_top [run_phase] stim gen ended
# UVM_INFO fifo_test.sv(54) @ 50002: uvm_test_top [run_phase] reset asserted
# UVM_INFO fifo_test.sv(56) @ 70002: uvm_test_top [run_phase] stim gen ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 70002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO fifo_scoreboard.sv(160) @ 70002: uvm_test_top.env.sb [report_phase] Total number of errors:0
# UVM_INFO fifo_scoreboard.sv(161) @ 70002: uvm_test_top.env.sb [report_phase] Total number of correct transactions:35001
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   14
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 70002 ns  Iteration: 61  Instance: /top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

Code Coverage

Statement Coverage:

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      29        29         0   100.00%
```

I removed the wr_ptr, rd_ptr and count from the design and added them to a shared package to been visible to the assertions that is why the statements seem less.

Branch Coverage:

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        25        25         0   100.00%
```

Condition Coverage:

```
Condition Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----       ----    ------  --------
    Conditions                      20         20         0   100.00%
```

2 conditions are excluded as one of their possiblities cant be reached.

In write mode: when Full is low and wr_en is high so it will not enter the else branch any time it already had entered the else if branch.

In read mode: when empty is low and rd_en is high so it will not enter the else branch any time as it already had entered the else if branch.

Toggle Coverage:

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%
```

Assertion Coverage

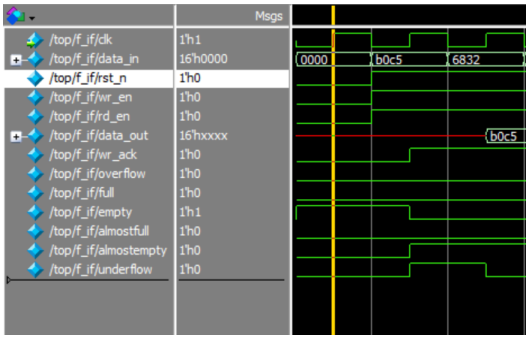
```
ASSERTION RESULTS:
---------------------------------------------------------------------
Name                File(Line)                   Failure      Pass
                                                 Count        Count
---------------------------------------------------------------------
/top/DUT/assertions_inst/reset
                    assertions.sv(15)                 0          1
/top/DUT/assertions_inst/wr_ack1
                    assertions.sv(21)                 0          1
/top/DUT/assertions_inst/overflow1
                    assertions.sv(25)                 0          1
/top/DUT/assertions_inst/underflow1
                    assertions.sv(29)                 0          1
/top/DUT/assertions_inst/wr_ptr1
                    assertions.sv(33)                 0          1
/top/DUT/assertions_inst/rd_ptr1
                    assertions.sv(37)                 0          1
/top/DUT/assertions_inst/count1
                    assertions.sv(42)                 0          1
/top/DUT/assertions_inst/wr_ptr2
                    assertions.sv(45)                 0          1
/top/DUT/assertions_inst/rd_ptr2
                    assertions.sv(48)                 0          1
/top/DUT/assertions_inst/almostempty1
                    assertions.sv(51)                 0          1
/top/DUT/assertions_inst/almostfull1
                    assertions.sv(54)                 0          1
/top/DUT/assertions_inst/empty1
                    assertions.sv(57)                 0          1
/top/DUT/assertions_inst/full1
                    assertions.sv(60)                 0          1
/fifo_sequence_pkg/write_only_sequence/body/#ublk#40571367#17/immed__20
                    fifo_sequence.sv(20)              0          1
/fifo_sequence_pkg/read_only_sequence/body/#ublk#40571367#37/immed__40
                    fifo_sequence.sv(40)              0          1
/fifo_sequence_pkg/write_read_sequence/body/#ublk#40571367#57/immed__60
                    fifo_sequence.sv(60)              0          1
/fifo_sequence_pkg/write_read_sequence/body/#ublk#40571367#65/immed__67
                    fifo_sequence.sv(67)              0          1
```

Functional Coverage

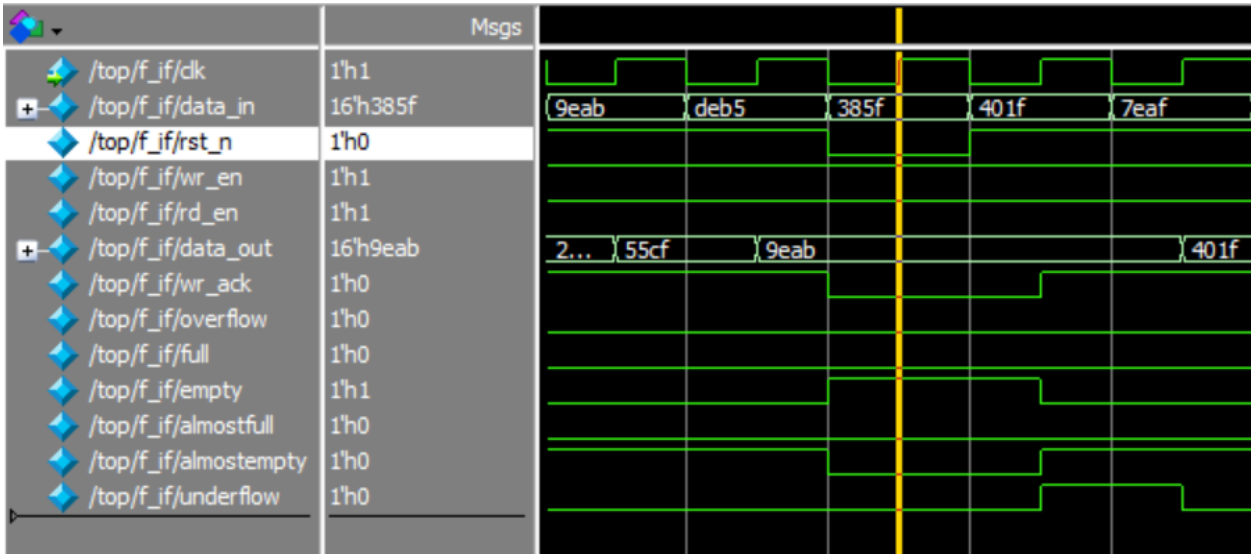
```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1
```

Waveform

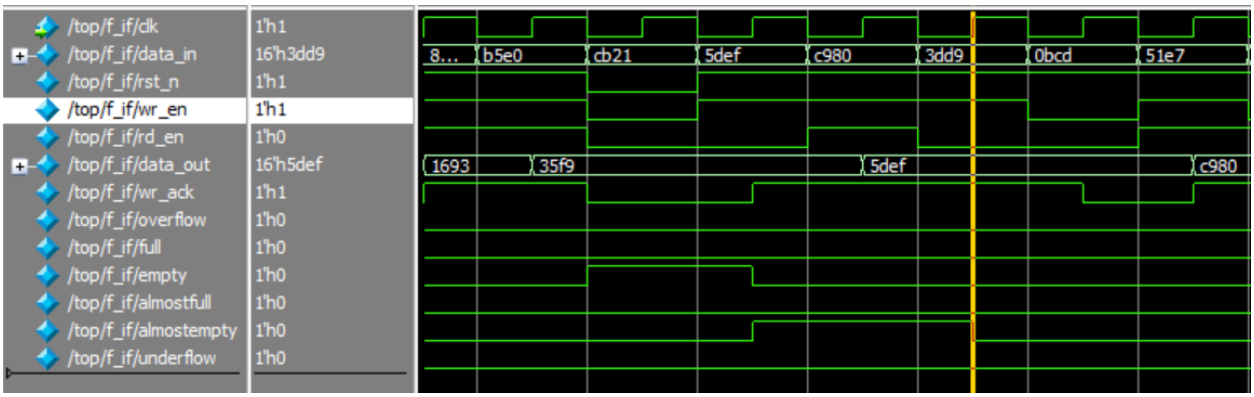
Reset sequence:



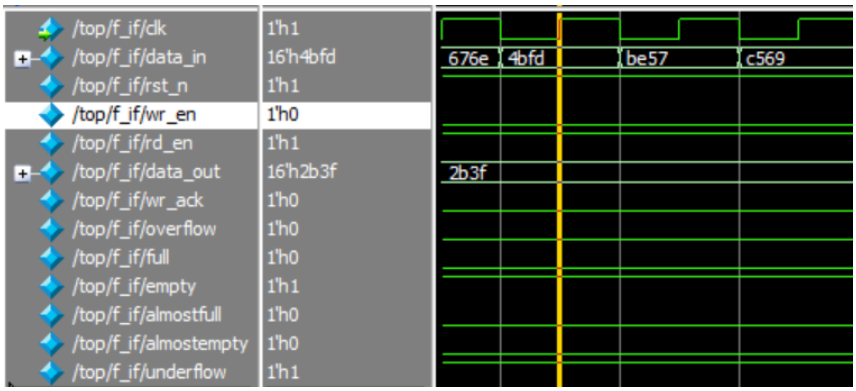
Write-read sequence:



Write sequence:



Read sequence:



FIFO Assertions Summary Table

Feature	Assertion
FIFO should be reset properly	(!rst_n) -> (wr_ptr == 0 && rd_ptr == 0 && count == 0 && wr_ack == 0 && overflow == 0 && underflow == 0)
Write acknowledged only when FIFO is not full	@(posedge clk) disable iff (!rst_n) (wr_en && !full) |=> wr_ack
Overflow occurs when writing to a full FIFO	@(posedge clk) disable iff (!rst_n) (wr_en && full) |=> overflow
Underflow occurs when reading from an empty FIFO	@(posedge clk) disable iff (!rst_n) (rd_en && empty) |=> underflow
Write pointer increments on valid write	@(posedge clk) disable iff (!rst_n) (wr_en && !full) |=> (wr_ptr == ($past(wr_ptr) + 1) % FIFO_DEPTH))
Read pointer increments on valid read	@(posedge clk) disable iff (!rst_n) (rd_en && !empty) |=> (rd_ptr == ($past(rd_ptr) + 1) % FIFO_DEPTH))
FIFO count does not exceed depth	count <= FIFO_DEPTH
Write pointer does not exceed depth	wr_ptr < FIFO_DEPTH
Read pointer does not exceed depth	rd_ptr < FIFO_DEPTH
Almost empty flag active when count == 1	(count == 1) == almostempty
Almost full flag active when count == FIFO_DEPTH - 1	(count == FIFO_DEPTH - 1) == almostfull
Empty flag active when count == 0	(count == 0) == empty
Full flag active when count == FIFO_DEPTH	(count == FIFO_DEPTH) == full