UVM Project - Synchronous FIFO

By: Amr Ayman Mohamed Abdo

Team name(solo): Silicon Maestro

Contents

Verification Plan

1. Test writing operation by performing writes more often than reads. **(Only_Write)**
2. Test reading operation by performing reads more often than writes. **(Only_Read)**
3. Test the FIFO with multiple concurrent writers and readers to ensure data integrity and proper flag behavior. **(Concurrent_Write_Read)**
4. Test simultaneous write-read operation by performing reads and writes with equal distribution probability. **(Simultaneous_Read_Write)**
5. Fill the FIFO to its maximum capacity and attempt more writes to ensure the `full` and `almsotfull` flags are set and `overflow` is triggered. **(Write_Full)**
6. Completely empty the FIFO and attempt more reads to check if the `empty` and `almostempty` flags are set and `underflow` is triggered. **(Read_Empty)**
7. With a full FIFO, perform a simultaneous write and read to check if the FIFO correctly updates the `full` flag and does not set the `overflow`. **(Simultaneous_Read_Write_Full)**
8. With an empty FIFO, perform a simultaneous read and write and verify the correct behavior of the `empty` flag and does not set the `underflow`. **(Simultaneous_Read_Write_Full)**
9. On reset, verify that the FIFO is cleared and all flags are in the correct initial state. **(Reset)**
10. Perform random sequences of reads and writes to test the FIFO's robustness. **(Read_Write_Randomized)**
11. Test flags stability when neither reading nor writing is taking place. **(Flags_Stability)**

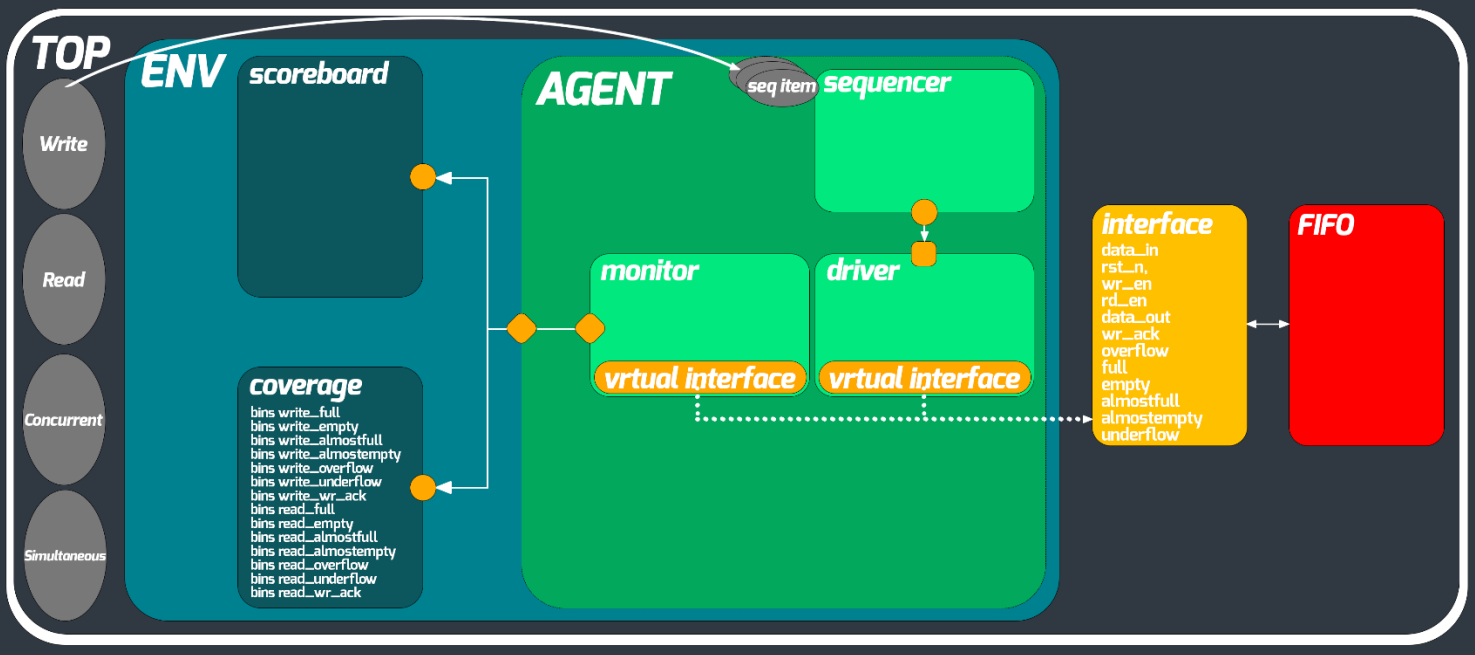
Verification Requirements Document Snippet

Verification Requirements				
Label	**Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
Only_Write	Write unique values to the FIFO while reading is disabled	constraint rd_en to be zero while randomizing wr_en to be high most of the time. Constraint data_in to have unique values.	Included in **write_seq** that ensures that 8 consecutive writes have taken place	Outputs are checked against reference model + **assert_full, assert_almostfull & assert_wr_ack**
Only_Read	Read the previously written values to the FIFO while writing is disabled	constraint wr_en to be zero while randomizing rd_en to be high most of the time.	Included in **read_seq** that ensures that 8 consecutive reads have taken place	Outputs are checked against reference model + **assert_empty & assert_almostempty**
Concurrent_Write_Read	Loop containing a sequence of writes followed by another sequence of reads	constraint rd_en to be zero while randomizing wr_en to be high most of the time while writing and vice versa	Included in **empty_to_not** and **full_to_not** that ensure that a writing sequence took place after a reading sequence and vice versa	Outputs are checked against reference model
Simultaneous_Read_Write	Read and Write simultaneously and check for the flags remaining low.	constraint wr_en and rd_en to be high simultaneously most of the time.	Included in **write_read_simlt** cross bin that covers read cross write	Outputs are checked against reference model
Write_Full	Write to a full FIFO and check for the overflow to be deasserted when rd_en is asserted	When the FIFO is full constraint wr_en to be high most of the time and rd_en to be low most of the time	Included in **write_full** cross coverage bin that covers write cross full	Outputs are checked against reference model + **assert_overflow**
Read_Empty	Read from an empty FIFO and check for the underflow to be deasserted when rd_en is asserted	When the FIFO is empty constraint rd_en to be high most of the time and wr_en to be low most of the time	Included in **read_empty** cross coverage bin that covers read cross empty	Outputs are checked against reference model + **assert_underflow**
Reset	In reset cases check for the FIFO to be cleared and empty is asserted and full is deasserted	Constraint reset to be activated more frequently while randomizing the other inputs	Included in **full_to_empty** bin that covers the transition from a full FIFO to an empty FIFO (possible only if reset is activated)	Outputs are checked against reference model + **assert_count_rst, assert_wr_ptr_rst & assert_rd_ptr_rst**
Simultaneous_Read_Write_Full	When the FIFO is full and reading and writing are taking place simultaneously, then overflow must remain low	When the FIFO is full, constraint wr_en and rd_en to be high simultaneously most of the time.	Included in **write_read_simlt_full** cross coverage between **write_read_simlt** and **full**	Outputs are checked against reference model + **assert_overflow**
Simultaneous_Read_Write_Empty	When the FIFO is empty and reading and writing are taking place simultaneously, then underflow must remain low	When the FIFO is empty, constraint wr_en and rd_en to be high simultaneously most of the time.	Included in **write_read_simlt_empty** cross coverage between **write_read_simlt** and **empty**	Outputs are checked against reference model + **assert_underflow**
Read_Write_Randomized	Perform random sequences of reads and writes to test the FIFO's robustness	Randomize inputs without constraints		Outputs are checked against reference model
Flags_Stability	When neither reading nor writing is taking place, flags must remain stable	Constraint rd_en and wr_en to be low most of the time	Included in **write_nor_read** that covers that write and read are deactivated simultaneously	Outputs are checked against reference model + **assert_full, assert_empty, assert_almostfull, assert_almostempty, assert_overflow, assert_underflow, assert_wr_ack**

Assertions Table

Assertions		
Assertion location	**Feature**	**Assertion**
FIFO_SVA	overfllow should fall when a successful writing operation occurs	**assert_overflow_fall:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full |=> (FIFOif.wr_en && FIFOif.rd_en) |-> ##2 (!FIFOif.overflow)));
FIFO_SVA	underfllow should fall when a successful reading operation occurs	**assert_underflow_fall:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.empty |=> (FIFOif.wr_en && FIFOif.rd_en) |-> ##2 (!FIFOif.underflow)));
FIFO_SVA	wr_ack must be zero whenever the FIFO is full	**assert_wr_ack_full:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full |=> (!FIFOif.wr_ack)));
FIFO_SVA	almostfull and almostempty are mutually exclusive events	**assert_almostfe:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.almostfull |-> !FIFOif.almostempty))
FIFO_SVA	almostfull and almostempty are mutually exclusive events	**assert_almostef:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.almostempty |-> !FIFOif.almostfull));
FIFO_SVA	full and empty are mutually exclusive events	**assert_full_empty:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (!(FIFOif.full && FIFOif.empty)));
FIFO_SVA	wr_ack must be zero whenever the a write attempt is taking place to a full FIFO	**assert_write_full:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full && FIFOif.wr_en) |=> !FIFOif.wr_ack);
FIFO (DUT)	overflow should be asserted when a write attempt is taking place to a full FIFO	**assert_overflow:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFOif.full) |=> (FIFOif.overflow == 1));
FIFO (DUT)	underflow should be asserted when a read attempt is taking place to an empty FIFO	**assert_underflow:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFOif.empty) |=> (FIFOif.underflow == 1));
FIFO (DUT)	wr_ack should be asserted whenever a succesfull writing operation occurs	**assert_wr_ack:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |=> (FIFOif.wr_ack == 1));
FIFO (DUT)	count should increment whenever a succesfull writing operation occurs and no reading operation occurs	**assert_count_incr:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full && !FIFOif.rd_en) || (FIFOif.wr_en && !FIFOif.full && FIFOif.rd_en && FIFOif.empty) |=> (count == $past(count) + 1));
FIFO (DUT)	count should decrement whenever a succesfull reading operation occurs and no writing operation occurs	**assert_count_decr:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && !FIFOif.empty && !FIFOif.wr_en) || (FIFOif.rd_en && !FIFOif.empty && FIFOif.wr_en && FIFOif.full) |=> (count == $past(count) - 1));
FIFO (DUT)	count should increment whenever a succesfull writing operation occurs	**assert_wr_ptr_incr:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |=> (wr_ptr == $past(wr_ptr) + 1) || (wr_ptr == 0));
FIFO (DUT)	count should decrement whenever a succesfull reading operation occurs	**assert_rd_ptr_incr:** assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && !FIFOif.empty) |=> (rd_ptr == $past(rd_ptr) + 1) || (rd_ptr == 0));
FIFO (DUT)	wr_ptr should be stable whenever wr_en is high, rd_en is low and the FIFO is full	**assert_write_pointer_stable:** assert property (@(posedge FIFOif.clk) disable iff(!FIFOif.rst_n) FIFOif.full && FIFOif.wr_en && !FIFOif.rd_en |=> $stable(wr_ptr));
FIFO (DUT)	rd_ptr should be stable whenever rd_en is high, wr_en is low and the FIFO is empty	**assert_read_pointer_stable:** assert property (@(posedge FIFOif.clk) disable iff(!FIFOif.rst_n) FIFOif.empty && FIFOif.rd_en && !FIFOif.wr_en |=> $stable(rd_ptr));
FIFO (DUT)	count should be asynchronously resseted upon the negative edge of rst_n	**assert_count_rst:** assert final (count == 0);
FIFO (DUT)	wr_ptr should be asynchronously resseted upon the negative edge of rst_n	**assert_wr_ptr_rst:** assert final (wr_ptr == 0);
FIFO (DUT)	rd_ptr should be asynchronously resseted upon the negative edge of rst_n	**assert_rd_ptr_rst:** assert final (rd_ptr == 0);
FIFO (DUT)	data_out should be asynchronously resseted upon the negative edge of rst_n	**assert_data_out_rst:** assert final (FIFOif.data_out == 0);
FIFO (DUT)	overflow should be asynchronously resseted upon the negative edge of rst_n	**assert_overflow_rst:** assert final (FIFOif.overflow == 0);
FIFO (DUT)	underflow should be asynchronously resseted upon the negative edge of rst_n	**assert_underflow_rst:** assert final (FIFOif.underflow == 0);
FIFO (DUT)	wr_ack should be asynchronously resseted upon the negative edge of rst_n	**assert_wr_ack_rst:** assert final (FIFOif.wr_ack == 0);

UVM Structure



Code Snippets

FIFO_config.sv

```
1  package FIFO_config_pkg;
2      import uvm_pkg::*;
3      `include "uvm_macros.svh"
4
5      class FIFO_config extends uvm_object;
6          `uvm_object_utils(FIFO_config)
7
8          virtual FIFO_if FIFO_vif;
9
10         function new(string name = "FIFO_config");
11             super.new(name);
12         endfunction : new
13
14     endclass : FIFO_config
15
16 endpackage : FIFO_config_pkg
```

FIFO_seq_item.sv

```
1   package FIFO_seq_item_pkg;
2       import uvm_pkg::*;
3       `include "uvm_macros.svh"
4
5       parameter VALID_OP = 6;
6       typedef enum bit [2:0] {OR, XOR, ADD, MULT, SHIFT, ROTATE, INVALID_6, INVALID_7} opcode_e;
7       typedef enum {MAXPOS = 3, ZERO = 0, MAXNEG = -4} reg_e;
8
9
10      class FIFO_seq_item extends uvm_sequence_item;
11          `uvm_object_utils(FIFO_seq_item)
12
13          parameter FIFO_WIDTH = 16;
14          parameter FIFO_DEPTH = 8;
15
16          //inputs
17          rand logic [FIFO_WIDTH-1:0] data_in;
18          rand logic rst_n, wr_en, rd_en;
19
20          //outputs
21          rand logic [FIFO_WIDTH-1:0] data_out;
22          rand logic wr_ack, overflow;
23          rand logic full, empty, almostfull, almostempty, underflow;
24
25          //reference model outputs
26          rand logic [FIFO_WIDTH-1:0] data_out_ref;
27          rand logic wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
28
29          function new(string name = "FIFO_seq_item");
30              super.new(name);
31          endfunction : new
32
33          function string convert2string();
34              return $sformatf("%s rst_n = 0b%0b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b\n
35                  , data_out = 0b%0b, wr_ack = 0b%0b, overflow = 0b%0b, full = 0b%0b\n
36                  , empty = 0x%0h, almostfull = 0x%0h, almostempty = 0x%0h, underflow = 0x%0h", super.convert2string, rst_n, wr_en, r
37          endfunction : convert2string
38
39          function string convert2string_stimulus();
40              return $sformatf("rst_n = 0b%0b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b", rst_n, wr_en, rd_en, data_in);
41          endfunction : convert2string_stimulus
42
43          int WR_EN_ON_DIST, RD_EN_ON_DIST, RST_ON_DIST;
44
45          constraint Reset_Constraint {
46              rst_n dist {1:=(100-RST_ON_DIST), 0:=RST_ON_DIST};
47          }
48
49          constraint General_Constraints {
50              wr_en dist {1:=WR_EN_ON_DIST, 0:=(100-WR_EN_ON_DIST)};
51              rd_en dist {1:=RD_EN_ON_DIST, 0:=(100-RD_EN_ON_DIST)};
52              (full == 1)  -> wr_en dist {1:=80, 0:=20};
53              (empty == 1) -> rd_en dist {1:=80, 0:=20};
54              unique {data_in};
55          }
56
57          constraint Simultaneous_Read_Write {
58              (wr_en && rd_en) dist {1:=80, 0:=20};
59              (full == 1)  -> wr_en dist {1:=80, 0:=20};
60              (empty == 1) -> rd_en dist {1:=80, 0:=20};
61              unique {data_in};
62          }
63
64      endclass : FIFO_seq_item
65
66  endpackage : FIFO_seq_item_pkg
```

FIFO_reset_sequence.sv

```
1   package FIFO_reset_sequence_pkg;
2       import uvm_pkg::*;
3       import FIFO_seq_item_pkg::*;
4       `include "uvm_macros.svh"
5
6       class FIFO_reset_sequence extends uvm_sequence #(FIFO_seq_item);
7           `uvm_object_utils(FIFO_reset_sequence)
8
9           FIFO_seq_item seq_item;
10
11          function new(string name = "FIFO_reset_sequence");
12              super.new(name);
13          endfunction : new
14
15          task body();
16              seq_item = FIFO_seq_item::type_id::create("seq_item");
17              seq_item.Simultaneous_Read_Write.constraint_mode(0);
18              start_item(seq_item);
19              seq_item.rst_n = 0;
20              seq_item.wr_en = 0;
21              seq_item.rd_en = 0;
22              seq_item.data_in = 0;
23              finish_item(seq_item);
24          endtask : body
25      endclass : FIFO_reset_sequence
26
27  endpackage : FIFO_reset_sequence_pkg
```

FIFO_write_sequence.sv

```
1   package FIFO_write_sequence_pkg;
2       import uvm_pkg::*;
3       import FIFO_seq_item_pkg::*;
4       `include "uvm_macros.svh"
5
6       class FIFO_write_sequence extends uvm_sequence #(FIFO_seq_item);
7           `uvm_object_utils(FIFO_write_sequence)
8
9           FIFO_seq_item seq_item;
10
11          function new(string name = "FIFO_write_sequence");
12              super.new(name);
13          endfunction : new
14
15          task body();
16              repeat(500) begin
17                  seq_item = FIFO_seq_item::type_id::create("seq_item");
18                  seq_item.Simultaneous_Read_Write.constraint_mode(0);
19                  seq_item.WR_EN_ON_DIST = 90; seq_item.RD_EN_ON_DIST = 10; seq_item.RST_ON_DIST = 5;
20                  start_item(seq_item);
21                  assert(seq_item.randomize());
22                  finish_item(seq_item);
23              end
24          endtask : body
25      endclass : FIFO_write_sequence
26
27  endpackage : FIFO_write_sequence_pkg
```

FIFO_read_sequence.sv

```
package FIFO_read_sequence_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_read_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_read_sequence)

        FIFO_seq_item seq_item;

        function new(string name = "FIFO_read_sequence");
            super.new(name);
        endfunction : new

        task body();
            repeat(500) begin
                seq_item = FIFO_seq_item::type_id::create("seq_item");
                seq_item.Simultaneous_Read_Write.constraint_mode(0);
                seq_item.WR_EN_ON_DIST = 10; seq_item.RD_EN_ON_DIST = 90; seq_item.RST_ON_DIST = 5;
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask : body
    endclass : FIFO_read_sequence

endpackage : FIFO_read_sequence_pkg
```

FIFO_concurrent_sequence.sv

```
package FIFO_concurrent_sequence_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_concurrent_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_concurrent_sequence)

        FIFO_seq_item seq_item1;
        FIFO_seq_item seq_item2;

        function new(string name = "FIFO_concurrent_sequence");
            super.new(name);
        endfunction : new

        task body();
            repeat(100) begin
                repeat(10) begin
                    seq_item1 = FIFO_seq_item::type_id::create("seq_item1");
                    seq_item1.Simultaneous_Read_Write.constraint_mode(0);
                    seq_item1.WR_EN_ON_DIST = 90; seq_item1.RD_EN_ON_DIST = 10; seq_item1.RST_ON_DIST = 2;
                    start_item(seq_item1);
                    assert(seq_item1.randomize());
                    finish_item(seq_item1);
                end

                repeat(10) begin
                    seq_item2 = FIFO_seq_item::type_id::create("seq_item2");
                    seq_item2.Simultaneous_Read_Write.constraint_mode(0);
                    seq_item2.WR_EN_ON_DIST = 10; seq_item2.RD_EN_ON_DIST = 90; seq_item2.RST_ON_DIST = 2;
                    start_item(seq_item2);
                    assert(seq_item2.randomize());
                    finish_item(seq_item2);
                end
            end
        endtask : body
    endclass : FIFO_concurrent_sequence

endpackage : FIFO_concurrent_sequence_pkg
```

FIFO_simult_sequence.sv

```
1   package FIFO_simult_sequence_pkg;
2       import uvm_pkg::*;
3       import FIFO_seq_item_pkg::*;
4       `include "uvm_macros.svh"
5
6       class FIFO_simult_sequence extends uvm_sequence #(FIFO_seq_item);
7           `uvm_object_utils(FIFO_simult_sequence)
8
9           FIFO_seq_item seq_item;
10
11          function new(string name = "FIFO_simult_sequence");
12              super.new(name);
13          endfunction : new
14
15          task body();
16              repeat(500) begin
17                  seq_item = FIFO_seq_item::type_id::create("seq_item");
18                  seq_item.constraint_mode(0);
19                  seq_item.Simultaneous_Read_Write.constraint_mode(1);
20                  seq_item.WR_EN_ON_DIST = 50; seq_item.RD_EN_ON_DIST = 50; seq_item.RST_ON_DIST = 5;
21                  start_item(seq_item);
22                  assert(seq_item.randomize());
23                  finish_item(seq_item);
24              end
25          endtask : body
26      endclass : FIFO_simult_sequence
27
28  endpackage : FIFO_simult_sequence_pkg
```

FIFO_sequencer.sv

```
1   package FIFO_sequencer_pkg;
2       import uvm_pkg::*;
3       import FIFO_seq_item_pkg::*;
4       `include "uvm_macros.svh"
5
6       class FIFO_sequencer extends uvm_sequencer #(FIFO_seq_item);
7           `uvm_component_utils(FIFO_sequencer)
8
9           function new(string name = "FIFO_seq_item", uvm_component parent = null);
10              super.new(name, parent);
11          endfunction : new
12
13      endclass : FIFO_sequencer
14
15  endpackage : FIFO_sequencer_pkg
```

FIFO_driver.sv

```
1  package FIFO_driver_pkg;
2      import uvm_pkg::*;
3      import FIFO_seq_item_pkg::*;
4      `include "uvm_macros.svh"
5
6      class FIFO_driver extends uvm_driver #(FIFO_seq_item);
7          `uvm_component_utils(FIFO_driver);
8
9          virtual FIFO_if FIFO_vif;
10         FIFO_seq_item stim_seq_item;
11
12         function new(string name = "FIFO_driver", uvm_component parent = null);
13             super.new(name, parent);
14         endfunction : new
15
16         task run_phase(uvm_phase phase);
17             super.run_phase(phase);
18             forever begin
19                 stim_seq_item = FIFO_seq_item::type_id::create("stim_seq_item");
20                 seq_item_port.get_next_item(stim_seq_item);
21                 FIFO_vif.data_in = stim_seq_item.data_in;
22                 FIFO_vif.rst_n = stim_seq_item.rst_n;
23                 FIFO_vif.wr_en = stim_seq_item.wr_en;
24                 FIFO_vif.rd_en = stim_seq_item.rd_en;
25                 @(negedge FIFO_vif.clk);
26                 seq_item_port.item_done();
27                 `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
28             end
29         endtask : run_phase
30
31     endclass : FIFO_driver
32
33 endpackage : FIFO_driver_pkg
```

FIFO_monitor.sv

```
package FIFO_monitor_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_monitor extends uvm_monitor;
        `uvm_component_utils(FIFO_monitor)
        virtual FIFO_if FIFO_vif;
        FIFO_seq_item rsp_seq_item;
        uvm_analysis_port #(FIFO_seq_item) mon_ap;

        function new(string name = "FIFO_monitor", uvm_component parent = null);
            super.new(name, parent);
        endfunction : new

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            mon_ap = new("mon_ap", this);
        endfunction : build_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                rsp_seq_item = FIFO_seq_item::type_id::create("rsp_seq_item");
                @(negedge FIFO_vif.clk);
                rsp_seq_item.data_in = FIFO_vif.data_in;
                rsp_seq_item.rst_n = FIFO_vif.rst_n;
                rsp_seq_item.wr_en = FIFO_vif.wr_en;
                rsp_seq_item.rd_en = FIFO_vif.rd_en;
                rsp_seq_item.data_out = FIFO_vif.data_out;
                rsp_seq_item.wr_ack = FIFO_vif.wr_ack;
                rsp_seq_item.overflow = FIFO_vif.overflow;
                rsp_seq_item.full = FIFO_vif.full;
                rsp_seq_item.empty = FIFO_vif.empty;
                rsp_seq_item.almostfull = FIFO_vif.almostfull;
                rsp_seq_item.almostempty = FIFO_vif.almostempty;
                rsp_seq_item.underflow = FIFO_vif.underflow;
                rsp_seq_item.data_out_ref = FIFO_vif.data_out_ref;
                rsp_seq_item.wr_ack_ref = FIFO_vif.wr_ack_ref;
                rsp_seq_item.overflow_ref = FIFO_vif.overflow_ref;
                rsp_seq_item.full_ref = FIFO_vif.full_ref;
                rsp_seq_item.empty_ref = FIFO_vif.empty_ref;
                rsp_seq_item.almostfull_ref = FIFO_vif.almostfull_ref;
                rsp_seq_item.almostempty_ref = FIFO_vif.almostempty_ref;
                rsp_seq_item.underflow_ref = FIFO_vif.underflow_ref;
                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH)
            end
        endtask : run_phase
    endclass : FIFO_monitor

endpackage : FIFO_monitor_pkg
```

FIFO_scoreboard.sv

```
package FIFO_scoreboard_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(FIFO_scoreboard)
        uvm_analysis_export #(FIFO_seq_item) sb_export;
        uvm_tlm_analysis_fifo #(FIFO_seq_item) sb_fifo;
        FIFO_seq_item sb_seq_item;

        int error_count;
        int correct_count;

        function new(string name = "FIFO_scoreboard", uvm_component parent = null);
            super.new(name, parent);
        endfunction : new

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export = new("sb_export", this);
            sb_fifo = new("sb_fifo", this);
        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction : connect_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(sb_seq_item);

                if (sb_seq_item.data_out == sb_seq_item.data_out_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the data_out_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.data_out_ref))
                    error_count++;
                    `uvm_error("run_phase", "data_out")
                end

                if (sb_seq_item.full === sb_seq_item.full_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the full_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.full_ref))
                    `uvm_error("run_phase", "full")
                    error_count++;
                end

                if (sb_seq_item.empty === sb_seq_item.empty_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the empty_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.empty_ref))
                    error_count++;
                    `uvm_error("run_phase", "empty")
                end

                if (sb_seq_item.almostfull === sb_seq_item.almostfull_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the almostfull_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.almostfull_ref))
                    error_count++;
                    `uvm_error("run_phase", "almostfull")
                end

                if (sb_seq_item.almostempty === sb_seq_item.almostempty_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the almostempty_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.almostempty_ref))
                    error_count++;
                end

                if (sb_seq_item.overflow === sb_seq_item.overflow_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the overflow_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.overflow_ref))
                    error_count++;
                end

                if (sb_seq_item.underflow === sb_seq_item.underflow_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the underflow_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.underflow_ref))
                    error_count++;
                end

                if (sb_seq_item.wr_ack === sb_seq_item.wr_ack_ref)
                    correct_count++;
                else begin
                    `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s
                        While the wr_ack_ref:0x%0h", sb_seq_item.convert2string(), sb_seq_item.wr_ack_ref))
                    error_count++;
                end
            end
        endtask : run_phase

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("Total successful transactions: %0d", correct_count), UVM_LOW);
            `uvm_info("report_phase", $sformatf("Total failed transactions: %0d", error_count), UVM_LOW);
        endfunction : report_phase
    endclass : FIFO_scoreboard

endpackage : FIFO_scoreboard_pkg
```

FIFO_coverage.sv

```
package FIFO_coverage_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_coverage extends uvm_component;
        `uvm_component_utils(FIFO_coverage)
        uvm_analysis_export #(FIFO_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(FIFO_seq_item) cov_fifo;
        FIFO_seq_item cov_seq_item;

        covergroup FIFO_cvg_gp;
            //Labeling coverpoints to be visible in the functional coverage report
            write_cp:                coverpoint cov_seq_item.wr_en              {bins wr_en[]              = {[0:1]};}
            read_cp:                 coverpoint cov_seq_item.rd_en              {bins rd_en[]              = {[0:1]};}
            full_cp:                 coverpoint cov_seq_item.full               {bins full[]               = {[0:1]};}
            empty_cp:                coverpoint cov_seq_item.empty              {bins empty[]              = {[0:1]};}
            almostfull_cp:           coverpoint cov_seq_item.almostfull         {bins almostfull[]         = {[0:1]};}
            almostempty_cp:          coverpoint cov_seq_item.almostempty        {bins almostempty[]        = {[0:1]};}
            overflow_cp:             coverpoint cov_seq_item.overflow           {bins overflow[]           = {[0:1]};}
            underflow_cp:            coverpoint cov_seq_item.underflow          {bins underflow[]          = {[0:1]};}
            wr_ack_cp:               coverpoint cov_seq_item.wr_ack             {bins wr_ack[]             = {[0:1]};}
            not_to_empty_cp:         coverpoint cov_seq_item.empty              {bins not_to_empty         = (0=>1);}

            write_seq_cp:            coverpoint cov_seq_item.wr_en              {bins write_seq            = (0=>1[*8]=>0);}
            read_seq_cp:             coverpoint cov_seq_item.rd_en              {bins read_seq             = (0=>1[*8]=>0);}
            empty_to_not_cp:         coverpoint cov_seq_item.empty              {bins empty_to_not         = (1=>0);}
            full_to_not_cp:          coverpoint cov_seq_item.full               {bins full_to_not          = (1=>0);}
            write_read_simlt_cp:     cross write_cp, read_cp                    {bins write_read_simlt     = binsof(write_cp.wr_en[1]) && binsof(read_cp.rd_en[1]); option.cross_auto_bin_max = 0;}
            full_to_empty_cp:        cross not_to_empty_cp, full_to_not_cp      {bins full_to_empty        = binsof(full_to_not_cp) && binsof(not_to_empty_cp); option.cross_auto_bin_max = 0;}
            write_read_simlt_full_cp:  cross write_read_simlt_cp, full_cp       {bins write_read_simlt_full  = binsof(write_read_simlt_cp) && binsof(full_cp.full[1]); option.cross_auto_bin_max = 0;}
            write_read_simlt_empty_cp: cross write_read_simlt_cp, empty_cp      {bins write_read_simlt_empty = binsof(write_read_simlt_cp) && binsof(empty_cp.empty[1]); option.cross_auto_bin_max = 0;}
            write_nor_read_cp:       cross write_cp, read_cp                    {bins write_read_simlt     = binsof(write_cp.wr_en[0]) && binsof(read_cp.rd_en[0]); option.cross_auto_bin_max = 0;}

            write_cross_states: cross write_cp, full_cp, empty_cp, almostfull_cp, almostempty_cp, overflow_cp, underflow_cp, wr_ack_cp {
                bins write_full =       binsof(write_cp) && binsof(full_cp);
                bins write_empty =      binsof(write_cp) && binsof(empty_cp);
                bins write_almostfull =  binsof(write_cp) && binsof(almostfull_cp);
                bins write_almostempty = binsof(write_cp) && binsof(almostempty_cp);
                bins write_overflow =    binsof(write_cp) && binsof(overflow_cp);
                bins write_underflow =   binsof(write_cp) && binsof(underflow_cp);
                bins write_wr_ack =      binsof(write_cp) && binsof(wr_ack_cp);
            }

            read_cross_states: cross read_cp, full_cp, empty_cp, almostfull_cp, almostempty_cp, overflow_cp, underflow_cp, wr_ack_cp {
                bins read_full =         binsof(read_cp) && binsof(full_cp);
                bins read_empty =        binsof(read_cp) && binsof(empty_cp);
                bins read_almostfull =   binsof(read_cp) && binsof(almostfull_cp);
                bins read_almostempty =  binsof(read_cp) && binsof(almostempty_cp);
                bins read_overflow =     binsof(read_cp) && binsof(overflow_cp);
                bins read_underflow =    binsof(read_cp) && binsof(underflow_cp);
                bins read_wr_ack =       binsof(read_cp) && binsof(wr_ack_cp);
            }

        endgroup : FIFO_cvg_gp

        function new(string name = "FIFO_coverage", uvm_component parent = null);
            super.new(name, parent);
            FIFO_cvg_gp = new();
        endfunction : new

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            cov_export = new("cov_export", this);
            cov_fifo = new("cov_fifo", this);
        endfunction : build_phase


        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
        endfunction : connect_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(cov_seq_item);
                if(cov_seq_item.rst_n)
                    FIFO_cvg_gp.sample();
            end
        endtask : run_phase
    endclass : FIFO_coverage

endpackage : FIFO_coverage_pkg
```

FIFO_agent.sv

```
package FIFO_agent_pkg;
    import uvm_pkg::*;
    import FIFO_sequencer_pkg::*;
    import FIFO_driver_pkg::*;
    import FIFO_monitor_pkg::*;
    import FIFO_config_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_agent extends uvm_agent;
        `uvm_component_utils(FIFO_agent)
        FIFO_sequencer sqr;
        FIFO_driver driver;
        FIFO_monitor monitor;
        FIFO_config FIFO_cfg;
        uvm_analysis_port #(FIFO_seq_item) agt_ap;

        function new(string name = "FIFO_agent", uvm_component parent = null);
            super.new(name, parent);
        endfunction : new

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            if(!uvm_config_db#(FIFO_config)::get(this, "", "CFG", FIFO_cfg)) //get the cfg from the db and assign it to the cfg of the driver
                `uvm_fatal("build_phase", "Agent - Unable to get the configuration object")

            sqr = FIFO_sequencer::type_id::create("sqr", this);
            driver = FIFO_driver::type_id::create("driver", this);
            monitor = FIFO_monitor::type_id::create("monitor", this);

            agt_ap = new("agt_ap", this);
        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            driver.FIFO_vif = FIFO_cfg.FIFO_vif;
            monitor.FIFO_vif = FIFO_cfg.FIFO_vif;
            driver.seq_item_port.connect(sqr.seq_item_export);
            monitor.mon_ap.connect(agt_ap);
        endfunction : connect_phase

    endclass : FIFO_agent

endpackage : FIFO_agent_pkg
```

FIFO_env.sv

```
1  package FIFO_env_pkg;
2      import uvm_pkg::*;
3      import FIFO_agent_pkg::*;
4      import FIFO_scoreboard_pkg::*;
5      import FIFO_coverage_pkg::*;
6      `include "uvm_macros.svh"
7
8      class FIFO_env extends uvm_env;
9          `uvm_component_utils(FIFO_env)
10
11         FIFO_agent agt;
12         FIFO_scoreboard sb;
13         FIFO_coverage cov;
14
15         function new(string name = "FIFO_env", uvm_component parent = null);
16             super.new(name, parent);
17         endfunction : new
18
19         function void build_phase(uvm_phase phase);
20             super.build_phase(phase);
21             agt = FIFO_agent::type_id::create("agt", this);
22             sb = FIFO_scoreboard::type_id::create("sb", this);
23             cov = FIFO_coverage::type_id::create("cov", this);
24         endfunction : build_phase
25
26         function void connect_phase(uvm_phase phase);
27             super.connect_phase(phase);
28             agt.agt_ap.connect(sb.sb_export);
29             agt.agt_ap.connect(cov.cov_export);
30         endfunction : connect_phase
31
32     endclass : FIFO_env
33
34 endpackage : FIFO_env_pkg
```

FIFO_test.sv

```
package FIFO_test_pkg;
    import uvm_pkg::*;
    import FIFO_env_pkg::*;
    import FIFO_config_pkg::*;
    import FIFO_reset_sequence_pkg::*;
    import FIFO_write_sequence_pkg::*;
    import FIFO_read_sequence_pkg::*;
    import FIFO_concurrent_sequence_pkg::*;
    import FIFO_simult_sequence_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_test extends uvm_test;
        `uvm_component_utils(FIFO_test)

        FIFO_env env;
        FIFO_config FIFO_cfg;
        FIFO_reset_sequence reset_seq;
        FIFO_write_sequence write_seq;
        FIFO_read_sequence read_seq;
        FIFO_concurrent_sequence concurrent_seq;
        FIFO_simult_sequence simult_seq;

        function new(string name = "FIFO_test", uvm_component parent = null);
            super.new(name, parent);
        endfunction : new

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env = FIFO_env::type_id::create("env", this);
            FIFO_cfg = FIFO_config::type_id::create("FIFO_cfg", this);
            reset_seq = FIFO_reset_sequence::type_id::create("reset_seq", this);
            write_seq = FIFO_write_sequence::type_id::create("write_seq", this);
            read_seq = FIFO_read_sequence::type_id::create("read_seq", this);
            concurrent_seq = FIFO_concurrent_sequence::type_id::create("concurrent_seq", this);
            simult_seq = FIFO_simult_sequence::type_id::create("simult_seq", this);

            if(!uvm_config_db#(virtual FIFO_if)::get(this, "", "FIFO_IF", FIFO_cfg.FIFO_vif)) //get the vif and assign it to the vif of the cfg
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the FIFO from the uvm_config_db")

            uvm_config_db#(FIFO_config)::set(this, "*", "CFG", FIFO_cfg);
        endfunction : build_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            //reset sequence
            `uvm_info("run_phase", "Reset Asserted", UVM_LOW)
            reset_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "Reset Deasserted", UVM_LOW)

            //write sequence
            `uvm_info("run_phase", "Write Sequence Started", UVM_LOW)
            write_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "Write Sequence Ended", UVM_LOW)

            //read sequence
            `uvm_info("run_phase", "Read Sequence Started", UVM_LOW)
            read_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "Read Sequence Ended", UVM_LOW)

            //concurrent sequence
            `uvm_info("run_phase", "Concurrent Sequence Started", UVM_LOW)
            concurrent_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "Concurrent Sequence Ended", UVM_LOW)

            //simult sequence
            `uvm_info("run_phase", "Simultaneous Sequence Started", UVM_LOW)
            simult_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "Simultaneous Sequence Ended", UVM_LOW)
            phase.drop_objection(this);
        endtask : run_phase

    endclass : FIFO_test

endpackage : FIFO_test_pkg
```

FIFO_top.sv

```
import uvm_pkg::*;
import FIFO_test_pkg::*;
`include "uvm_macros.svh"

module FIFO_top ();
    bit clk;

    initial begin
        forever #1 clk = ~clk;
    end

    FIFO_if FIFOif(clk);
    FIFO DUT(FIFOif);
    FIFO_ref REF(FIFOif);
    bind FIFO FIFO_SVA FIFO_SVA_inst(FIFOif);


    initial begin
        uvm_config_db#(virtual FIFO_if)::set(null, "uvm_test_top", "FIFO_IF", FIFOif);
        run_test("FIFO_test");
    end
endmodule : FIFO_top
```

FIFO_if.sv

```
interface FIFO_if #(
    parameter FIFO_WIDTH = 16,
    parameter FIFO_DEPTH = 8)(input bit clk);

    logic [FIFO_WIDTH-1:0] data_in;
    logic rst_n, wr_en, rd_en;
    logic [FIFO_WIDTH-1:0] data_out;
    logic wr_ack, overflow;
    logic full, empty, almostfull, almostempty, underflow;

    logic [FIFO_WIDTH-1:0] data_out_ref;
    logic wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

    modport DUT (
        input data_in, clk, rst_n, wr_en, rd_en,
        output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
        );

    modport REF (
        input data_in, clk, rst_n, wr_en, rd_en,
        output data_out_ref, wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref
        );

    modport SVA (
        input data_in, clk, rst_n, wr_en, rd_en, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
        );

endinterface : FIFO_if
```

FIFO_SVA.sv

```
module FIFO_SVA (FIFO_if.SVA FIFOif);

    assert_overflow_fall: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full |=> (FIFOif.wr_en && FIFOif.rd_en) |-> ##2 (!FIFOif.overflow)));
    cover_overflow_fall: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full |=> (FIFOif.wr_en && FIFOif.rd_en) |-> ##2 (!FIFOif.overflow)));

    assert_underflow_fall: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.empty |=> (FIFOif.wr_en && FIFOif.rd_en) |-> ##2 (!FIFOif.underflow)));
    cover_underflow_fall: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.empty |=> (FIFOif.wr_en && FIFOif.rd_en) |-> ##2 (!FIFOif.underflow)));

    assert_wr_ack_full: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full |=> (!FIFOif.wr_ack)));
    cover_wr_ack_full: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full |=> (!FIFOif.wr_ack)));

    assert_almostfe: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.almostfull |-> !FIFOif.almostempty));
    cover_almostfe: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.almostfull |-> !FIFOif.almostempty));

    assert_almostef: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.almostempty |-> !FIFOif.almostfull));
    cover_almostef: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.almostempty |-> !FIFOif.almostfull));

    assert_full_empty: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (!(FIFOif.full && FIFOif.empty)));
    cover_full_empty: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (!(FIFOif.full && FIFOif.empty)));

    assert_write_full: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full && FIFOif.wr_en) |=> !FIFOif.wr_ack);
    cover_write_full: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.full && FIFOif.wr_en) |=> !FIFOif.wr_ack);

endmodule : FIFO_SVA
```

FIFO_ref.sv (reference model)

```
module FIFO_ref(FIFO_if.REF FIFOif);
    logic [FIFOif.FIFO_WIDTH-1:0] mem[$];
    bit wr_ack_next, wr_ack_next1;

    always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
        if(~FIFOif.rst_n) begin
            mem.delete();
            FIFOif.overflow_ref <= 0;
            FIFOif.wr_ack_ref <= 0;
        end
        else begin
            if(FIFOif.wr_en) begin
                if(!FIFOif.full_ref) begin
                    mem.push_front(FIFOif.data_in);
                    FIFOif.wr_ack_ref <= 1;
                    FIFOif.overflow_ref <= 0;
                end else begin
                    FIFOif.wr_ack_ref <= 0;
                    FIFOif.overflow_ref <= 1;
                end
            end else begin
                FIFOif.wr_ack_ref <= 0;
                FIFOif.overflow_ref <= 0;
            end
        end
    end

    always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
        if(~FIFOif.rst_n) begin
            mem.delete();
            FIFOif.data_out_ref <= 0;
            FIFOif.underflow_ref <= 0;
        end
        else begin
            if(FIFOif.rd_en) begin
                if(!FIFOif.empty_ref) begin
                    FIFOif.data_out_ref <= mem.pop_back();
                    FIFOif.underflow_ref <= 0;
                end else begin
                    FIFOif.underflow_ref <= 1;
                end
            end else
                FIFOif.underflow_ref <= 0;
        end
    end

    always_comb begin
        case(mem.size())
            0:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 1;
                FIFOif.almostempty_ref = 0;
            end
            1:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 1;
            end
            FIFOif.FIFO_DEPTH-1:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 1;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 0;
            end
            FIFOif.FIFO_DEPTH:
            begin
                FIFOif.full_ref = 1;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 0;
            end
            default:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 0;
            end
        endcase // mem.size()
    end

endmodule
```

Detected Bugs

1. **Extended count bus by 1-bit to allow it to reach FIFO_DEPTH**

Before:

```
22   reg [max_fifo_addr:0] count;
```

After:

```
15   reg [max_fifo_addr+1:0] count;
```

2. **Added this line to deassert overflow incase of writing**

Before:

```
28       else if (wr_en && count < FIFO_DEPTH) begin
29           mem[wr_ptr] <= data_in;
30           wr_ack <= 1;
31           wr_ptr <= wr_ptr + 1;
32       end
```

After:

```
21       else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
22           mem[wr_ptr] <= FIFOif.data_in;
23           FIFOif.wr_ack <= 1;
24           wr_ptr <= wr_ptr + 1;
25           FIFOif.overflow <= 0;
26       end
```

3. **Added this line to deassert underflow incase of reading**

Before:

```
46       else if (rd_en && count != 0) begin
47           data_out <= mem[rd_ptr];
48           rd_ptr <= rd_ptr + 1;
49       end
```

After:

```
40       else if (FIFOif.rd_en && count != 0) begin
41           FIFOif.data_out <= mem[rd_ptr];
42           rd_ptr <= rd_ptr + 1;
43           FIFOif.underflow <= 0;
44       end
```

4. **Removed this line since underflow is a sequential output not combinational**

Before:

```
66   assign underflow = (empty && rd_en)? 1 : 0;
```

After:

```
73   //assign FIFOif.underflow = (FIFOif.empty && FIFOif.rd_en)? 1 : 0;
```

5. **Added the sequential implementation of the underflow**

Before:

```
42    always @(posedge clk or negedge rst_n) begin
43        if (!rst_n) begin
44            rd_ptr <= 0;
45        end
46        else if (rd_en && count != 0) begin
47            data_out <= mem[rd_ptr];
48            rd_ptr <= rd_ptr + 1;
49        end
50    end
```

After:

```
36    always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
37        if (!FIFOif.rst_n) begin
38            rd_ptr <= 0;
39        end
40        else if (FIFOif.rd_en && count != 0) begin
41            FIFOif.data_out <= mem[rd_ptr];
42            rd_ptr <= rd_ptr + 1;
43            FIFOif.underflow <= 0;
44        end
45        else begin
46            if (FIFOif.empty & FIFOif.rd_en)
47                FIFOif.underflow <= 1;
48            else
49                FIFOif.underflow <= 0;
50        end
51    end
```

6. **Modified this line as FIFO is almostfull when count reaches FIFO_DEPTH-1**

Before:

```
67    assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
```

After:

```
74    assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH-1)? 1 : 0;
```

7. **Added this line as data_out must asynchronously be resetted (sequential)**

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        rd_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;
        rd_ptr <= 0;
        FIFOif.underflow <= 0;
```

8. Added this line as overflow must asynchronously be resetted (sequential)

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
        FIFOif.overflow <= 0;
```

9. Added this line as underflow must asynchronously be resetted (sequential)

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        rd_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;
        rd_ptr <= 0;
        FIFOif.underflow <= 0;
```

10. Added this line as wr_ack must asynchronously be resetted (sequential)

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
        FIFOif.overflow <= 0;
        FIFOif.wr_ack <= 0;
    end
```

DUT after fixing bugs and adding assertions

```
module FIFO(FIFO_if.DUT FIFOif);

localparam max_fifo_addr = $clog2(FIFOif.FIFO_DEPTH);

reg [FIFOif.FIFO_WIDTH-1:0] mem [FIFOif.FIFO_DEPTH-1:0];

reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr+1:0] count;                                      //modified_code(1)      /*to allow count to reach FIFO_DEPTH*/

always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
        FIFOif.overflow <= 0;                                       //added_code (1)        /*overflow must asynchronously be resetted (sequential)*/
        FIFOif.wr_ack <= 0;                                         //added_code (2)        /*overflow must asynchronously be resetted (sequential)*/
    end
    else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
        mem[wr_ptr] <= FIFOif.data_in;
        FIFOif.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        FIFOif.overflow <= 0;                                       //added_code (3)        /*overflow should be deasserted in the case of writing*/
    end
    else begin
        FIFOif.wr_ack <= 0;
        if (FIFOif.full & FIFOif.wr_en)
            FIFOif.overflow <= 1;
        else
            FIFOif.overflow <= 0;
    end
end

always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;                                       //added_code (4)        /*data_out must asynchronously be resetted (sequential)*/
        rd_ptr <= 0;
        FIFOif.underflow <= 0;                                      //added_code (5)        /*underflow must asynchronously be resetted (sequential)*/
    end
    else if (FIFOif.rd_en && count != 0) begin
        FIFOif.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        FIFOif.underflow <= 0;                                      //added_code (6)        /*underflow should be deasserted in the case of reading*/
    end
    else begin
        if (FIFOif.empty & FIFOif.rd_en)                            //added_code (7) begin  /*added the sequential implementation of underflow*/
            FIFOif.underflow <= 1;                                  //********************//
        else                                                        //********************//
            FIFOif.underflow <= 0;                                  //********************//
    end                                                             //added_code (7) end
end

always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({FIFOif.wr_en, FIFOif.rd_en} == 2'b10) && !FIFOif.full)
            count <= count + 1;
        else if ( ({FIFOif.wr_en, FIFOif.rd_en} == 2'b01) && !FIFOif.empty)
            count <= count - 1;
        else if ({FIFOif.wr_en, FIFOif.rd_en} == 2'b11) begin       //added_code (8) begin  /*added the part that handles simultaneous read-write*/
            case ({FIFOif.full, FIFOif.empty})                      //********************//
                2'b01: count <= count + 1;                          //********************//
                2'b10: count <= count - 1;                          //********************//
            endcase                                                 //********************//
        end                                                         //added_code (8) end
    end
end

assign FIFOif.full = (count == FIFOif.FIFO_DEPTH)? 1 : 0;
assign FIFOif.empty = (count == 0)? 1 : 0;
//assign FIFOif.underflow = (FIFOif.empty && FIFOif.rd_en)? 1 : 0;  //removed_code          /*underflow is sequential and not combinational*/
assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH-1)? 1 : 0;   //modified_code (2)     /*FIFO is almostfull when count reaches FIFO_DEPTH-1 not FIFO_DEPTH-2 as 0 is not included*/
assign FIFOif.almostempty = (count == 1)? 1 : 0;

`ifdef SIM

    //Assertions

    //Assertions to the combinational flags
    always_comb begin
        if(count == FIFOif.FIFO_DEPTH) begin
            assert_full:  assert final (FIFOif.full == 1) else $error("Mismatch: full_tb (%0d) != full_ref (1)", FIFOif.full);
            cover_full:  cover final (FIFOif.full == 1);
        end

        if(count == 0) begin
            assert_empty:  assert final (FIFOif.empty == 1) else $error("Mismatch: empty_tb (%0d) != empty_ref (1)", FIFOif.empty);
            cover_empty:  cover final (FIFOif.empty == 1);
        end

        if(count == FIFOif.FIFO_DEPTH-1) begin
            assert_almostfull:  assert final (FIFOif.almostfull == 1) else $error("Mismatch: almostfull_tb (%0d) != almostfull_ref (1)", FIFOif.almostfull);
            cover_almostfull:  cover final (FIFOif.almostfull == 1);
        end

        if(count == 1) begin
            assert_almostempty:  assert final (FIFOif.almostempty == 1) else $error("Mismatch: almostempty_tb (%0d) != almostempty_ref (1)", FIFOif.almostempty);
            cover_almostempty:  cover final (FIFOif.almostempty == 1);
        end
    end


    //Assertions to the sequential flags
    assert_overflow: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFOif.full) |=> (FIFOif.overflow == 1));
    cover_overflow: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFOif.full) |=> (FIFOif.overflow == 1));

    assert_underflow: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFOif.empty) |=> (FIFOif.underflow == 1));
    cover_underflow: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFOif.empty) |=> (FIFOif.underflow == 1));

    assert_wr_ack: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |=> (FIFOif.wr_ack == 1));
    cover_wr_ack: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |=> (FIFOif.wr_ack == 1));

    property count_plus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.wr_en && !FIFOif.full && !FIFOif.rd_en) || (FIFOif.wr_en && !FIFOif.full && FIFOif.rd_en && FIFOif.empty) |=> (count == $past(count) + 1);
    endproperty

    property count_minus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.rd_en && !FIFOif.empty && !FIFOif.wr_en) || (FIFOif.rd_en && !FIFOif.empty && FIFOif.wr_en && FIFOif.full) |=> (count == $past(count) - 1);
    endproperty

    property wr_ptr_plus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.wr_en && !FIFOif.full) |=> (wr_ptr == $past(wr_ptr) + 1) || (wr_ptr == 0);
    endproperty

    property rd_ptr_plus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.rd_en && !FIFOif.empty) |=> (rd_ptr == $past(rd_ptr) + 1) || (rd_ptr == 0);
    endproperty
```

```
142
143     property write_pointer_stable;
144         @(posedge FIFOif.clk)
145         disable iff(!FIFOif.rst_n)
146         FIFOif.full && FIFOif.wr_en && !FIFOif.rd_en |=> $stable(wr_ptr);
147     endproperty
148
149     property read_pointer_stable;
150         @(posedge FIFOif.clk)
151         disable iff(!FIFOif.rst_n)
152         FIFOif.empty && FIFOif.rd_en && !FIFOif.wr_en |=> $stable(rd_ptr);
153     endproperty
154
155     //Assertions to the counters
156     assert_count_incr: assert property (count_plus);
157     cover_count_incr: cover property (count_plus);
158
159     assert_count_decr: assert property (count_minus);
160     cover_count_decr: cover property (count_minus);
161
162     assert_wr_ptr_incr: assert property (wr_ptr_plus);
163     cover_wr_ptr_incr: cover property (wr_ptr_plus);
164
165     assert_rd_ptr_incr: assert property (rd_ptr_plus);
166     cover_rd_ptr_incr: cover property (rd_ptr_plus);
167
168     assert_write_pointer_stable: assert property (write_pointer_stable);
169     cover_write_pointer_stable: cover property (write_pointer_stable);
170
171     assert_read_pointer_stable: assert property (read_pointer_stable);
172     cover_read_pointer_stable: cover property (read_pointer_stable);
173
174     always_comb begin
175         if(~FIFOif.rst_n) begin
176             assert_count_rst: assert final (count == 0);
177             cover_count_rst: cover final (count == 0);
178
179             assert_wr_ptr_rst: assert final (wr_ptr == 0);
180             cover_wr_ptr_rst: cover final (wr_ptr == 0);
181
182             assert_rd_ptr_rst: assert final (rd_ptr == 0);
183             cover_rd_ptr_rst: cover final (rd_ptr == 0);
184
185             assert_data_out_rst: assert final (FIFOif.data_out == 0);
186             cover_data_out_rst: cover final (FIFOif.data_out == 0);
187
188             assert_overflow_rst: assert final (FIFOif.overflow == 0);
189             cover_overflow_rst: cover final (FIFOif.overflow == 0);
190
191             assert_underflow_rst: assert final (FIFOif.underflow == 0);
192             cover_underflow_rst: cover final (FIFOif.underflow == 0);
193
194             assert_wr_ack_rst: assert final (FIFOif.wr_ack == 0);
195             cover_wr_ack_rst: cover final (FIFOif.wr_ack == 0);
196         end
197     end
198
199 `endif
200 endmodule
```

QuestaSim Snippets

Wave colour guidelines

1. Cyan for the DUT outputs
2. Gold for the reference outputs
3. Red for the clock
4. Green for the inputs
5. Blue for the internal signals and assertions

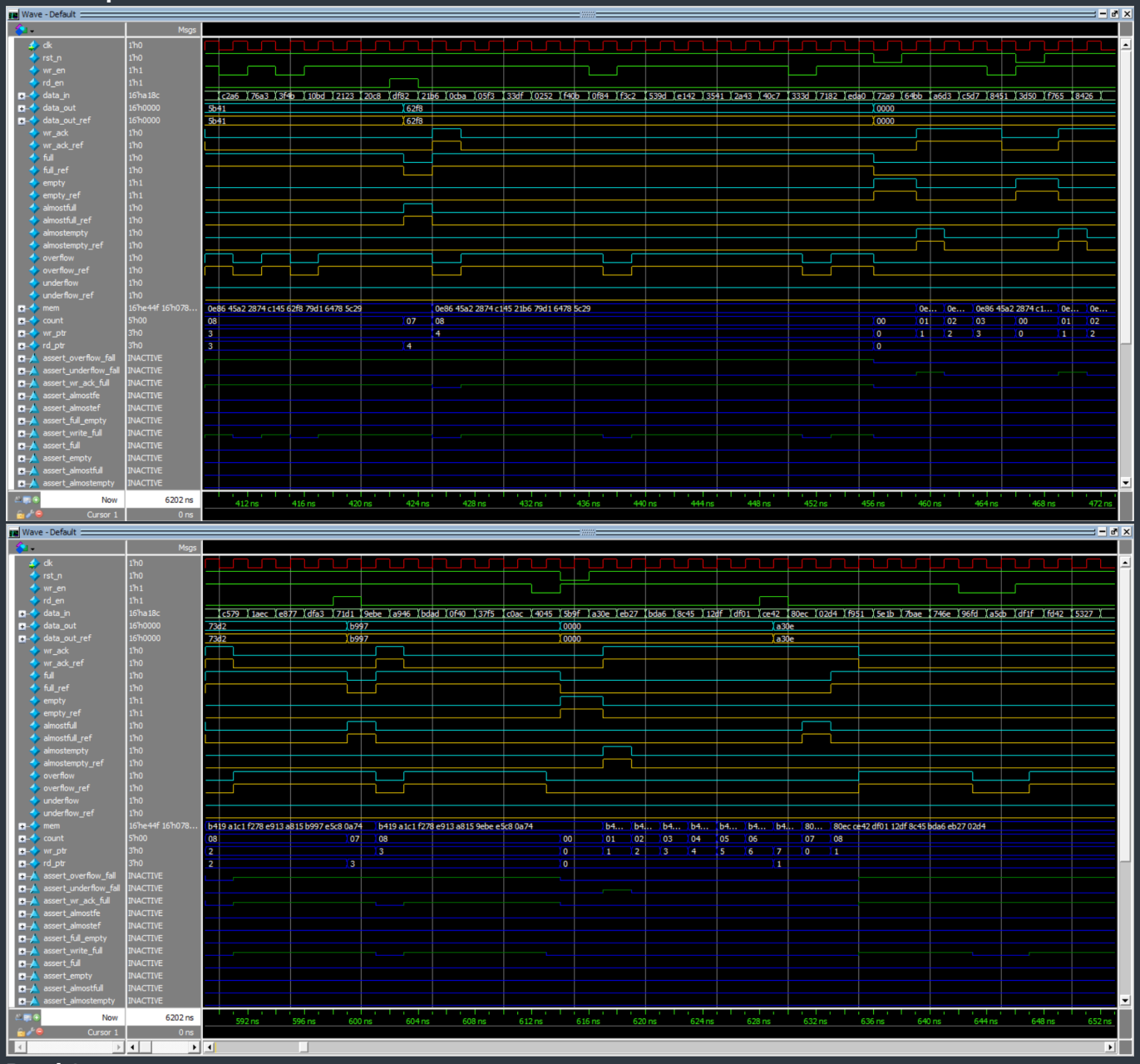
Snippets

Transcript

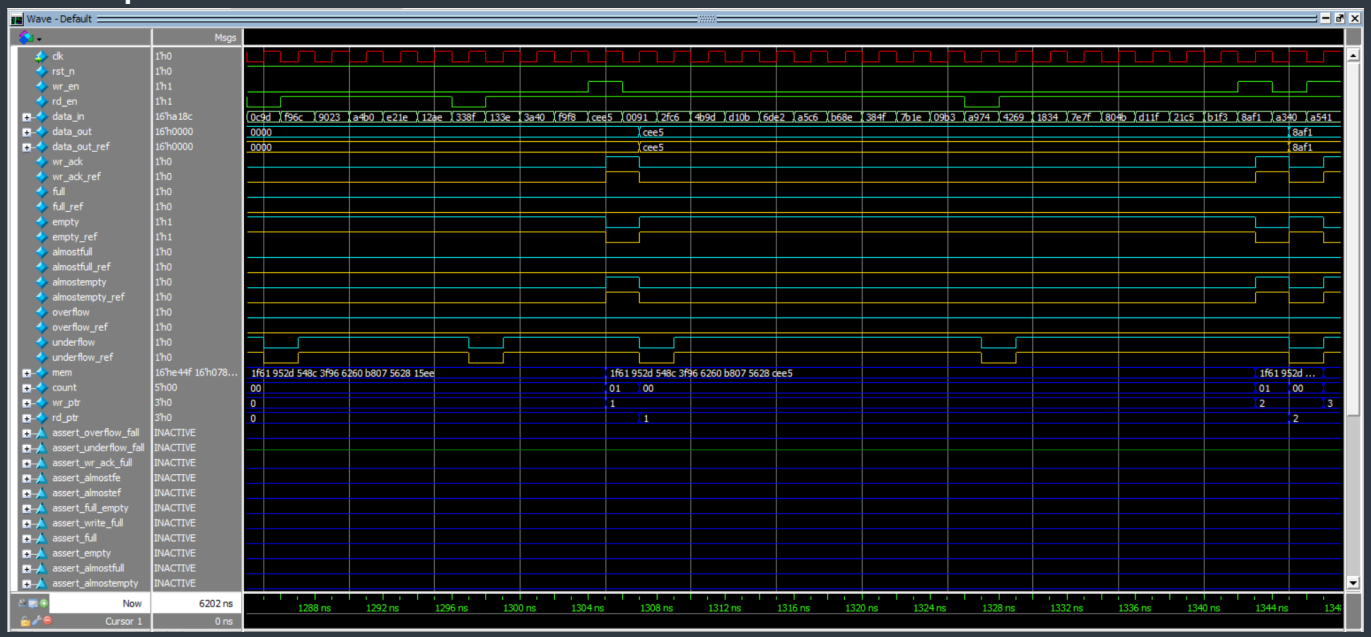
```
# UVM_INFO FIFO_test.sv(47) @ 0: uvm_test_top [run_phase] Reset Asserted
# ****************************************************************************
# * Questa UVM Transaction Recording Turned ON.                              *
# * recording_detail has been set.                                           *
# *  To turn off, set 'recording_detail' to off:                             *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ****************************************************************************
# UVM_INFO FIFO_test.sv(49) @ 2: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO FIFO_test.sv(52) @ 2: uvm_test_top [run_phase] Write Sequence Started
# UVM_INFO FIFO_test.sv(54) @ 1002: uvm_test_top [run_phase] Write Sequence Ended
# UVM_INFO FIFO_test.sv(57) @ 1002: uvm_test_top [run_phase] Read Sequence Started
# UVM_INFO FIFO_test.sv(59) @ 2002: uvm_test_top [run_phase] Read Sequence Ended
# UVM_INFO FIFO_test.sv(62) @ 2002: uvm_test_top [run_phase] Concurrent Sequence Started
# UVM_INFO FIFO_test.sv(64) @ 6002: uvm_test_top [run_phase] Concurrent Sequence Ended
# UVM_INFO FIFO_test.sv(67) @ 6002: uvm_test_top [run_phase] Simultaneous Sequence Started
# UVM_INFO FIFO_test.sv(69) @ 7002: uvm_test_top [run_phase] Simultaneous Sequence Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 7002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(107) @ 7002: uvm_test_top.env.sb [report_phase] Total successful transactions: 28008
# UVM_INFO FIFO_scoreboard.sv(108) @ 7002: uvm_test_top.env.sb [report_phase] Total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   16
# 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]    10
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 7002 ns  Iteration: 61  Instance: /FIFO_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
```

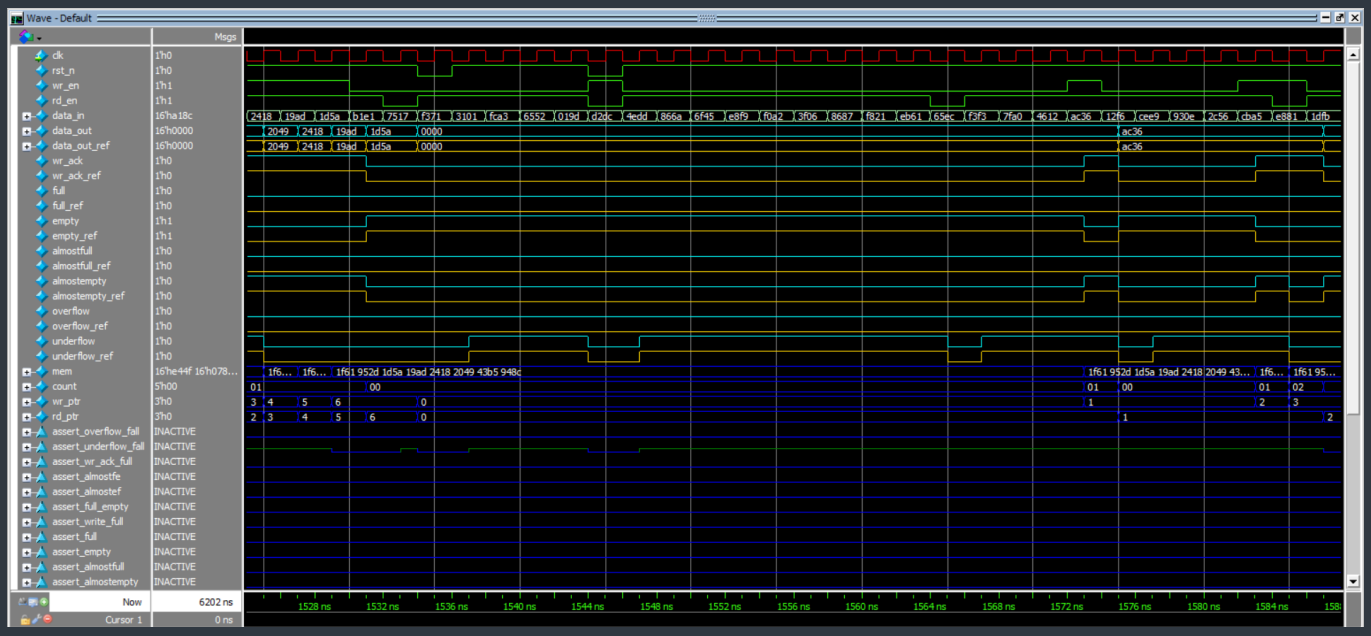
Waveform

Write Sequence

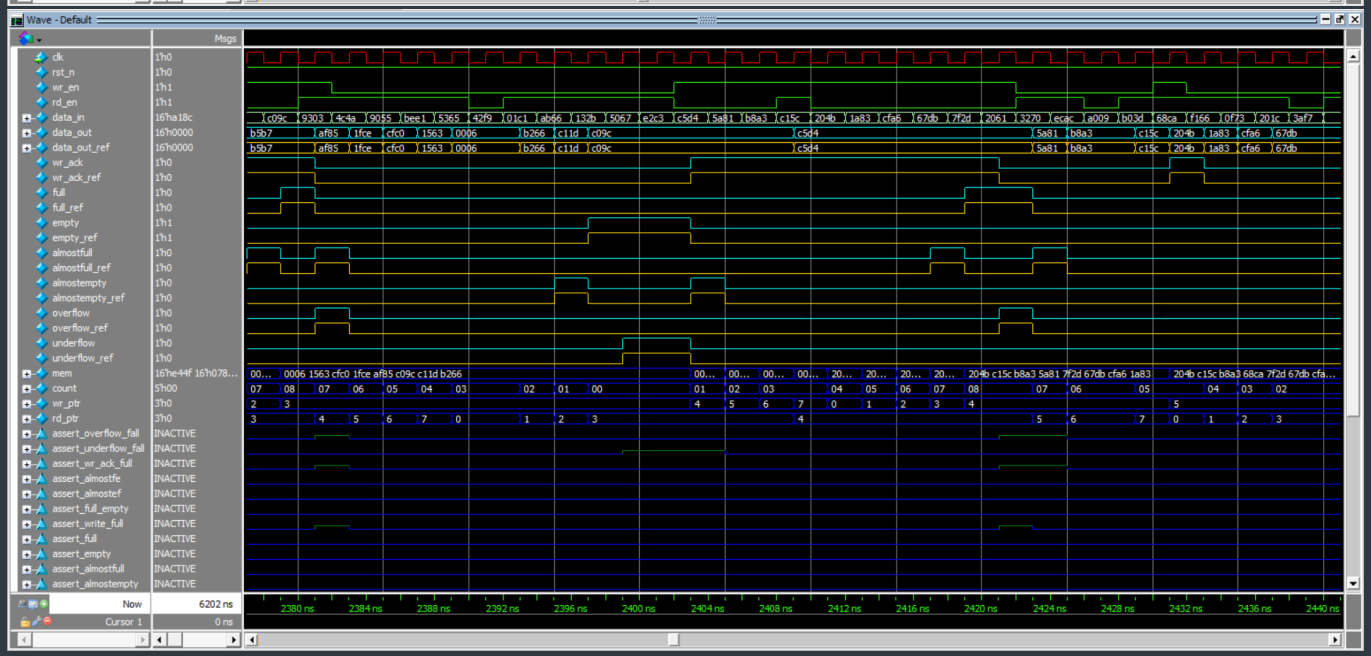


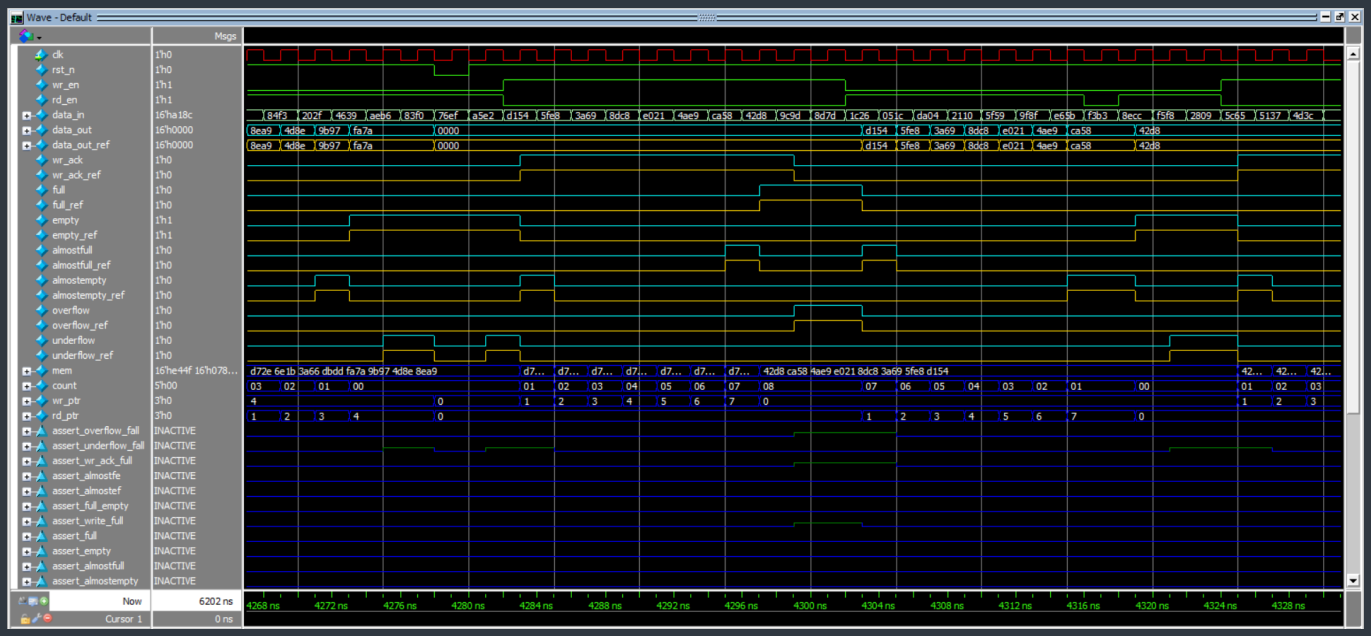
Read Sequence



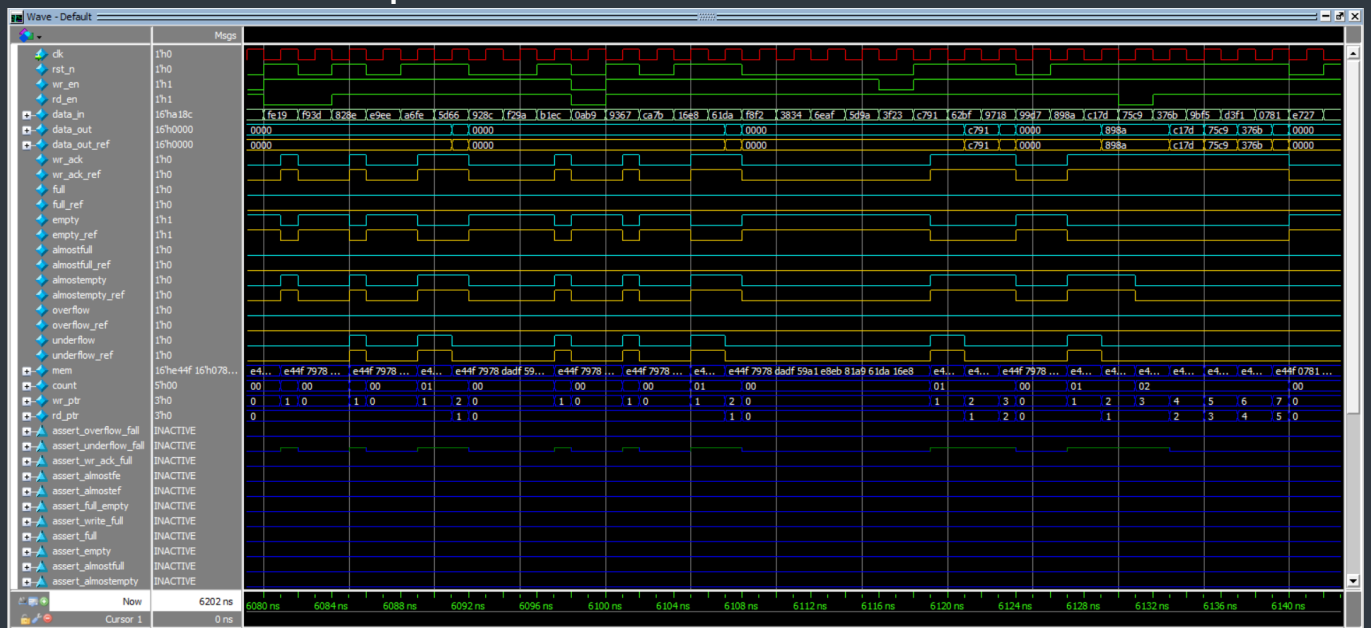


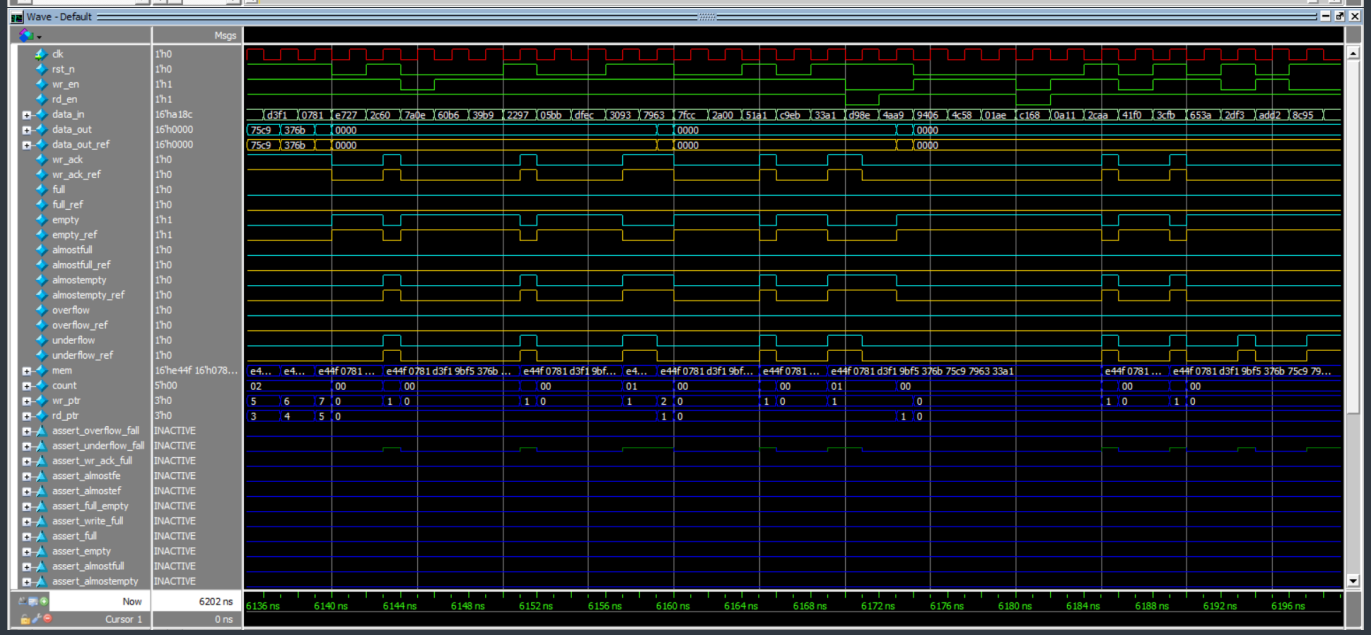
Concurrent Write Read Sequence





Simultaneous Write Read Sequence





Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_p...		100.00%							
TYPE FIFO_cv...		100.00%	100	100.00...		✓	auto(0)		
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CVP FIFO_...		100.00%	100	100.00...		✓			
CROSS FIF...		100.00%	100	100.00...		✓			
CROSS FIF...		100.00%	100	100.00...		✓			
CROSS FIF...		100.00%	100	100.00...		✓			
CROSS FIF...		100.00%	100	100.00...		✓			
CROSS FIF...		100.00%	100	100.00...		✓			
CROSS FIF...		100.00%	100	100.00...		✓			
CROSS FIF...		100.00%	100	100.00...		✓			
INST \/FIF...		100.00%	100	100.00...		✓		0	

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/FIFO_simult_sequence_pkg::FIFO_simult_sequence::body/#ublk#259062743#...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_concurrent_sequence_pkg::FIFO_concurrent_sequence::body/#ublk#19...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_concurrent_sequence_pkg::FIFO_concurrent_sequence::body/#ublk#19...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_read_sequence_pkg::FIFO_read_sequence::body/#ublk#124516487#1...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_write_sequence_pkg::FIFO_write_sequence::body/#ublk#78004439#16...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/DUT/assert_full	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFOif.full)	✓
/FIFO_top/DUT/assert_empty	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFOif.empty)	✓
/FIFO_top/DUT/assert_almostfull	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFOif.almostfull)	✓
/FIFO_top/DUT/assert_almostempty	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFOif.almostempty)	✓
/FIFO_top/DUT/assert_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_wr_ack	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_count_incr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_count_decr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_wr_ptr_incr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_rd_ptr_incr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_write_pointer_stable	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_read_pointer_stable	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/assert_count_rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (count==0)	✓
/FIFO_top/DUT/assert_wr_ptr_rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (wr_ptr==0)	✓
/FIFO_top/DUT/assert_rd_ptr_rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (rd_ptr==0)	✓
/FIFO_top/DUT/assert_data_out_rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFOif.data_out==0)	✓
/FIFO_top/DUT/assert_overflow_rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFOif.overflow)	✓
/FIFO_top/DUT/assert_underflow_rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFOif.underflow)	✓
/FIFO_top/DUT/assert_wr_ack_rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFOif.wr_ack)	✓
/FIFO_top/DUT/FIFO_SVA_inst/assert_overflow_fall	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/FIFO_SVA_inst/assert_underflow_fall	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/FIFO_SVA_inst/assert_wr_ack_full	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/FIFO_SVA_inst/assert_almostfe	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/FIFO_SVA_inst/assert_almostef	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/FIFO_SVA_inst/assert_full_empty	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓
/FIFO_top/DUT/FIFO_SVA_inst/assert_write_full	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disa...	✓

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_top/DUT/cover_full	SVA	✓	Off	102	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_empty	SVA	✓	Off	212	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_almostfull	SVA	✓	Off	187	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_almostempty	SVA	✓	Off	291	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_overflow	SVA	✓	Off	284	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_underflow	SVA	✓	Off	594	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ack	SVA	✓	Off	1135	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_count_incr	SVA	✓	Off	962	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_count_decr	SVA	✓	Off	676	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ptr_incr	SVA	✓	Off	1135	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_rd_ptr_incr	SVA	✓	Off	849	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_write_pointer_stable	SVA	✓	Off	247	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_read_pointer_stable	SVA	✓	Off	517	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_count_rst	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ptr_rst	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_rd_ptr_rst	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_data_out_rst	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_overflow_rst	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_underflow_rst	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ack_rst	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_inst/cover_overflow_...	SVA	✓	Off	32	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_inst/cover_underflow...	SVA	✓	Off	72	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_inst/cover_wr_ack_fu...	SVA	✓	Off	370	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_inst/cover_almostfe	SVA	✓	Off	204	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_inst/cover_almostef	SVA	✓	Off	333	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_inst/cover_full_empty	SVA	✓	Off	2944	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_inst/cover_write_full	SVA	✓	Off	284	1	Unli...	1	100%		✓	0	0	0 ns	0

Coverage Reports

Code Coverage Report

Note: Conditional Coverage didn't reach 100% as the two conditions (!FIFOif.full && FIFOif.wr_en) and (!FIFOif.empty && FIFOif.rd_en) would never happen if else was entered; because in case one of them happened else statement wouldn't be entered. As a result, Total Coverage By Instance dropped to 96.51%.

```
Condition Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----    -------    ------  --------
    Conditions                      24         22         2    91.66%

================================Condition Details================================

Condition Coverage for instance /\FIFO_top#DUT  --

  File FIFO.sv
----------------Focused Condition View-------------------
Line       21 Item    1  (FIFOif.wr_en && (count < FIFOif.FIFO_DEPTH))
Condition totals: 2 of 2 input terms covered = 100.00%

                    Input Term   Covered  Reason for no coverage   Hint
                   -----------  --------  ----------------------  --------------
                  FIFOif.wr_en         Y
  (count < FIFOif.FIFO_DEPTH)          Y

      Rows:       Hits  FEC Target                    Non-masking condition(s)
 ---------  ---------  --------------------          -------------------------
  Row   1:          1  FIFOif.wr_en_0                -
  Row   2:          1  FIFOif.wr_en_1                (count < FIFOif.FIFO_DEPTH)
  Row   3:          1  (count < FIFOif.FIFO_DEPTH)_0  FIFOif.wr_en
  Row   4:          1  (count < FIFOif.FIFO_DEPTH)_1  FIFOif.wr_en

----------------Focused Condition View-------------------
Line       29 Item    1  (FIFOif.full & FIFOif.wr_en)
Condition totals: 1 of 2 input terms covered = 50.00%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
   FIFOif.full         N  '_0' not hit             Hit '_0'
  FIFOif.wr_en         Y

      Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:     ***0***  FIFOif.full 0         FIFOif.wr_en
  Row   2:          1  FIFOif.full_1         FIFOif.wr_en
  Row   3:          1  FIFOif.wr_en_0        FIFOif.full
  Row   4:          1  FIFOif.wr_en_1        FIFOif.full

----------------Focused Condition View-------------------
Line       40 Item    1  (FIFOif.rd_en && (count != 0))
Condition totals: 2 of 2 input terms covered = 100.00%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
  FIFOif.rd_en         Y
  (count != 0)         Y

      Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  FIFOif.rd_en_0        -
  Row   2:          1  FIFOif.rd_en_1        (count != 0)
  Row   3:          1  (count != 0)_0        FIFOif.rd_en
  Row   4:          1  (count != 0)_1        FIFOif.rd_en

----------------Focused Condition View-------------------
Line       46 Item    1  (FIFOif.empty & FIFOif.rd_en)
Condition totals: 1 of 2 input terms covered = 50.00%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
  FIFOif.empty         N  '_0' not hit             Hit '_0'
  FIFOif.rd_en         Y

      Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:     ***0***  FIFOif.empty_0        FIFOif.rd_en
  Row   2:          1  FIFOif.empty_1        FIFOif.rd_en
  Row   3:          1  FIFOif.rd_en_0        FIFOif.empty
  Row   4:          1  FIFOif.rd_en_1        FIFOif.empty
```

```
21      else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
22          mem[wr_ptr] <= FIFOif.data_in;
23          FIFOif.wr_ack <= 1;
24          wr_ptr <= wr_ptr + 1;
25          FIFOif.overflow <= 0;
26      end
27      else begin
28          FIFOif.wr_ack <= 0;
29          if (FIFOif.full & FIFOif.wr_en)
30              FIFOif.overflow <= 1;
31          else
32              FIFOif.overflow <= 0;
33      end
```

```
40      else if (FIFOif.rd_en && count != 0) begin
41          FIFOif.data_out <= mem[rd_ptr];
42          rd_ptr <= rd_ptr + 1;
43          FIFOif.underflow <= 0;
44      end
45      else begin
46          if (FIFOif.empty & FIFOif.rd_en)
47              FIFOif.underflow <= 1;
48          else
49              FIFOif.underflow <= 0;
50      end
```

Note: Toggle coverage didn't reach 100% as count is 4-bits while it resets to zero when reaching 8

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         22        20         2    90.90%

================================Toggle Details================================

Toggle Coverage for instance /\FIFO_top#DUT  --

                                                      Node      1H->0L      0L->1H  "Coverage"
                                                      ----------------------------------------
                                                  count[4]           0           0        0.00
                                                count[3-0]           1           1      100.00
                                               rd_ptr[2-0]           1           1      100.00
                                               wr_ptr[2-0]           1           1      100.00

Total Node Count      =         11
Toggled Node Count    =         10
Untoggled Node Count  =          1

Toggle Coverage       =      90.90% (20 of 22 bins)
```

Functional Coverage Report

```
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        bin empty_to_not                          349         1         -    Covered
    Coverpoint full_to_not_cp                 100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        bin full_to_not                            91         1         -    Covered
    Cross write_read_simlt_cp                 100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt                 4001         1         -    Covered
    Cross full_to_empty_cp                    100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin full_to_empty                      17         1         -    Covered
    Cross write_read_simlt_full_cp            100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt_full            4001         1         -    Covered
    Cross write_read_simlt_empty_cp           100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt_empty           4001         1         -    Covered
    Cross write_nor_read_cp                   100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt                 4001         1         -    Covered
    Cross write_cross_states                  100.00%       100         -    Covered
        covered/total bins:                         7         7         -
        missing/total bins:                         0         7         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_full                       4001         1         -    Covered
            bin write_empty                      4001         1         -    Covered
            bin write_almostfull                 4001         1         -    Covered
            bin write_almostempty                4001         1         -    Covered
            bin write_overflow                   4001         1         -    Covered
            bin write_underflow                  4001         1         -    Covered
            bin write_wr_ack                     4001         1         -    Covered
    Cross read_cross_states                   100.00%       100         -    Covered
        covered/total bins:                         7         7         -
        missing/total bins:                         0         7         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin read_full                        4001         1         -    Covered
            bin read_empty                       4001         1         -    Covered
            bin read_almostfull                  4001         1         -    Covered
            bin read_almostempty                 4001         1         -    Covered
            bin read_overflow                    4001         1         -    Covered
            bin read_underflow                   4001         1         -    Covered
            bin read_wr_ack                      4001         1         -    Covered

TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

Total Coverage By Instance (filtered view): 100.00%
```

Sequential Domain Coverage Report

```
        Auto, Default and User Defined Bins:
            bin write_full                          2944          1          -    Covered
            bin write_empty                         2944          1          -    Covered
            bin write_almostfull                    2944          1          -    Covered
            bin write_almostempty                   2944          1          -    Covered
            bin write_overflow                      2944          1          -    Covered
            bin write_underflow                     2944          1          -    Covered
            bin write_wr_ack                        2944          1          -    Covered
    Cross read_cross_states                       100.00%        100          -    Covered
        covered/total bins:                             7          7          -
        missing/total bins:                             0          7          -
        % Hit:                                    100.00%        100          -
        Auto, Default and User Defined Bins:
            bin read_full                           2944          1          -    Covered
            bin read_empty                          2944          1          -    Covered
            bin read_almostfull                     2944          1          -    Covered
            bin read_almostempty                    2944          1          -    Covered
            bin read_overflow                       2944          1          -    Covered
            bin read_underflow                      2944          1          -    Covered
            bin read_wr_ack                         2944          1          -    Covered

TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/DUT/cover_full                FIFO   Verilog  SVA  FIFO.sv(89)      102 Covered
/FIFO_top/DUT/cover_empty               FIFO   Verilog  SVA  FIFO.sv(94)      212 Covered
/FIFO_top/DUT/cover_almostfull          FIFO   Verilog  SVA  FIFO.sv(99)      187 Covered
/FIFO_top/DUT/cover_almostempty         FIFO   Verilog  SVA  FIFO.sv(104)     291 Covered
/FIFO_top/DUT/cover_overflow            FIFO   Verilog  SVA  FIFO.sv(111)     284 Covered
/FIFO_top/DUT/cover_underflow           FIFO   Verilog  SVA  FIFO.sv(114)     594 Covered
/FIFO_top/DUT/cover_wr_ack              FIFO   Verilog  SVA  FIFO.sv(117)    1135 Covered
/FIFO_top/DUT/cover_count_incr          FIFO   Verilog  SVA  FIFO.sv(155)     962 Covered
/FIFO_top/DUT/cover_count_decr          FIFO   Verilog  SVA  FIFO.sv(158)     676 Covered
/FIFO_top/DUT/cover_wr_ptr_incr         FIFO   Verilog  SVA  FIFO.sv(161)    1135 Covered
/FIFO_top/DUT/cover_rd_ptr_incr         FIFO   Verilog  SVA  FIFO.sv(164)     849 Covered
/FIFO_top/DUT/cover_write_pointer_stable FIFO   Verilog  SVA  FIFO.sv(167)     247 Covered
/FIFO_top/DUT/cover_read_pointer_stable  FIFO   Verilog  SVA  FIFO.sv(170)     517 Covered
/FIFO_top/DUT/cover_count_rst           FIFO   Verilog  SVA  FIFO.sv(175)     123 Covered
/FIFO_top/DUT/cover_wr_ptr_rst          FIFO   Verilog  SVA  FIFO.sv(178)     123 Covered
/FIFO_top/DUT/cover_rd_ptr_rst          FIFO   Verilog  SVA  FIFO.sv(181)     123 Covered
/FIFO_top/DUT/FIFO_SVA_inst/cover_overflow_fall
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(4)    32 Covered
/FIFO_top/DUT/FIFO_SVA_inst/cover_underflow_fall
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(7)    72 Covered
/FIFO_top/DUT/FIFO_SVA_inst/cover_wr_ack_full
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(10)  370 Covered
/FIFO_top/DUT/FIFO_SVA_inst/cover_almostfe
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(13)  204 Covered
/FIFO_top/DUT/FIFO_SVA_inst/cover_almostef
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(16)  333 Covered
/FIFO_top/DUT/FIFO_SVA_inst/cover_full_empty
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(19) 2944 Covered
/FIFO_top/DUT/FIFO_SVA_inst/cover_write_full
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(22)  284 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 23

Total Coverage By Instance (filtered view): 100.00%
```