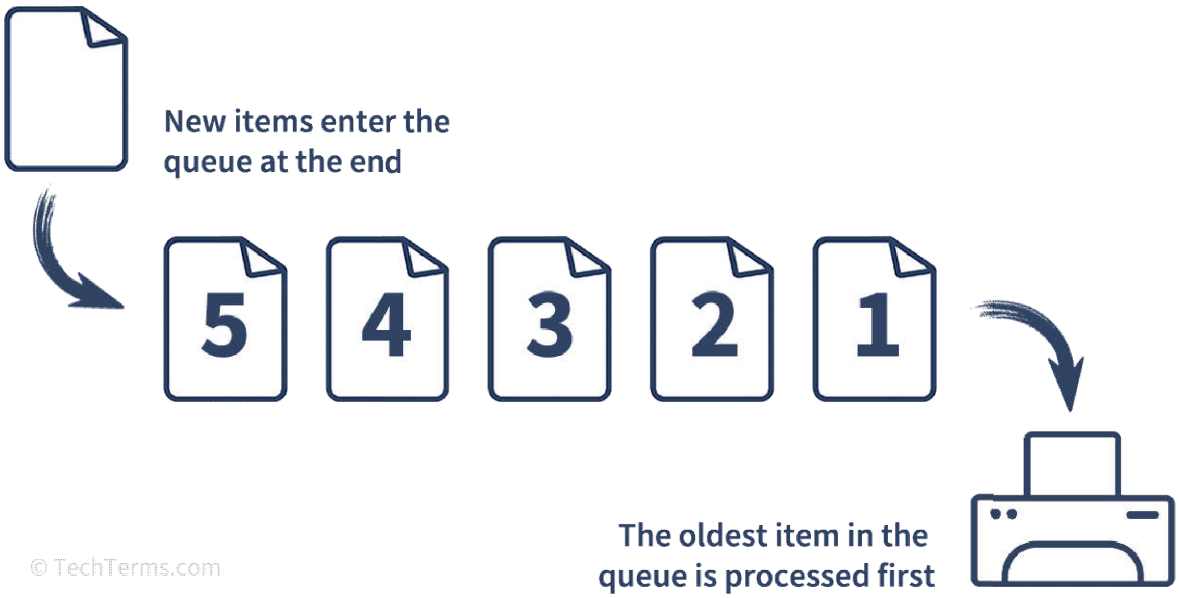
UVM FIFO Project



Mahmoud S. Zahran

Table of contents

I. Introduction to FIFO

A First-In, First-Out (FIFO) buffer is a fundamental data structure used in both hardware and software systems to manage data flow. As the name suggests, it operates on a queueing principle where the first piece of data to enter the buffer is also the first to leave, making it ideal for sequential data handling. FIFOs are widely utilized in scenarios that require orderly data transmission and reception, especially where data needs to be processed in the exact order it arrives.

In digital systems, FIFO buffers are essential for temporary data storage and are often implemented in hardware as part of memory structures or as standalone modules. In software, FIFOs can be used in algorithms or operating system-level tasks, such as inter-process communication.

Key Applications of FIFO:

1. Data Communication: FIFOs are crucial in buffering data between systems with different clock domains, such as between a processor and a peripheral device, ensuring that data is not lost when speeds differ.

2. Networking: In network routers and switches, FIFOs are used to queue packets before forwarding them, managing traffic flow and avoiding data collisions or loss.

3. Audio/Video Streaming: FIFO buffers smooth out the streaming of audio and video signals by temporarily holding the data before processing, preventing interruptions due to uneven data transmission.

4. Digital Signal Processing (DSP): In signal processing systems, FIFOs manage continuous data flow between different stages of a pipeline to avoid data overwrites or underflows.

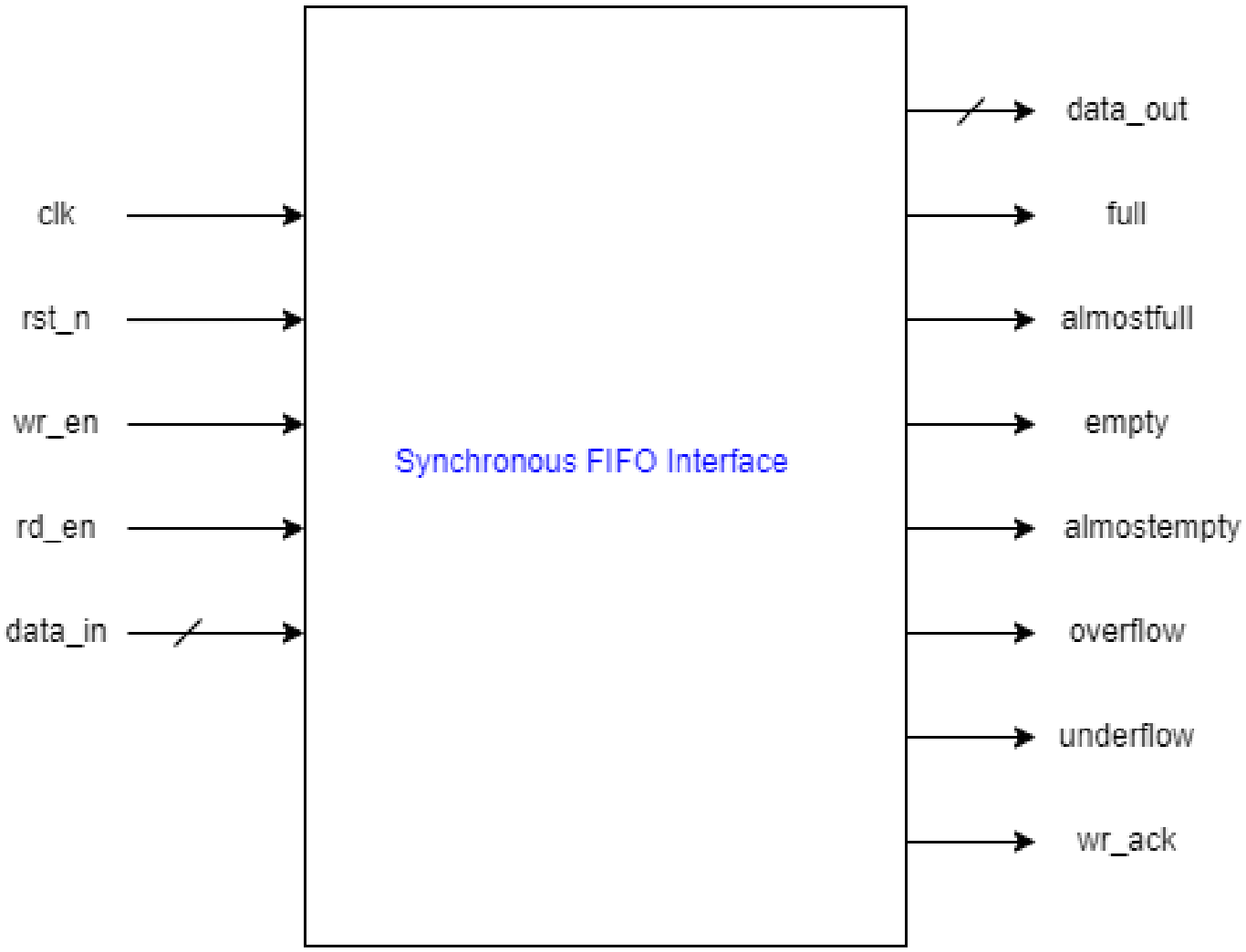
5. Microprocessor Systems: FIFOs are often used in CPU peripherals (such as UARTs) to queue incoming and outgoing data, enabling efficient data handling and processing.

6. Control Systems: In embedded systems, FIFOs help manage sensor data and ensure that control loops receive and process input in the correct order.

II. FIFO Interface

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

Ports table


clk
rst_n
wr_en
rd_en
data_in
Synchronous FIFO Interface
data_out
full
almostfull
empty
almostempty
overflow
underflow
wr_ack

FIFO IF

III. Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, the internal pointers, counter and overflow, underflow will reset	Directed at the start of the sim, then randomized with constraint that drives reset to be off most of the sim time	-	immediate assertion to check the reset functionality
FIFO_2	When wr_en and rd_en are high and the fifo is empty, the priorty is to write to fifo	Randomization under constraint on wr_en to be high 70% of the time and rd_en to be high 30% 0f the time	coverpoints wr_en and rd_e n and cross cover with empty and full signals	concurent assertion to check count is increased
FIFO_3	When wr_en and rd_en are high and the fifo is full, the priorty is to read from fifo	Randomization under constraint on wr_en to be high 70% of the time and rd_en to be high 30% 0f the time	coverpoints wr_en and rd_en and cross cover with empty and full signals	concurent assertion to check count is decreased
FIFO_4	When wr_en is high and rd_en is low and the fifo is not full, write operation is done	Randomization under constraint on wr_en to be high 70% of the time	cross cover between wr_en and full signal	concurent assertion to check count is increased, self check using reference model
FIFO_5	When wr_en is low and rd_en is high and the fifo is not empty, read operation is done	Randomization under constraint on wr_en to be high 30% of the time	cross cover between rd_en and empty signal	concurent assertion to check count is decreased, self check using reference model
FIFO_6	When count is equal to depth - 1		cross cover between almostfull and wr_en and rd_en	immediate assertion to check almostfull is high, self check using reference model
FIFO_7	When count is equal to 1		cross cover between almostempty and wr_en and rd_en	immediate assertion to check almostempty is high, self check using reference model
FIFO_8	When wr_en is high and fifo is not full. wr_ack is high		cross cover between wr_ack and wr_en and rd_en	concurent assertion to check wr_ack is high, self check using reference model
FIFO_9	When wr_en is high and fifo is full, overflow is high		cross cover between overflow and wr_en and rd_en	concurent assertion to check overflow is high, self check using reference model
FIFO_10	When rd_en is high and fifo is empty, underflow is high		cross cover between underflow and wr_en and rd_en	concurent assertion to check underflow is high, self check using reference model
FIFO_11	When count = depth, fifo is full		cross cover between full and wr_en and rd_en	immediate assertion to check full is high, self check using reference model
FIFO_12	When count = 0, fifo is empty		cross cover between empty and wr_en and rd_en	immediate assertion to check empty is high, self check using reference model
FIFO_13	data_in	Randomized during the simulation		

IV. RTL Bugs Report

1.

Original design

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

Modified design

```
if (!f_if.rst_n) begin
  wr_ptr <= 0;
  f_if.overflow <= 0; // reset overflow signal
  f_if.wr_ack <= 0; // reset wr_ack signal
end
```

2.

Original design

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

Modified design

```
if (!f_if.rst_n) begin
  rd_ptr <= 0;
  f_if.underflow <= 0; // reset underflow signal
end
```

3.

Original design

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

Modified design

```
else begin
  if (f_if.empty && f_if.rd_en)  // make underflow signal sequential
    f_if.underflow <= 1;
  else
    f_if.underflow <= 0;
end
```

4.

Original design

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

Modified design

```
assign f_if.almostfull = (count == f_if.FIFO_DEPTH-1) ? 1 : 0; // modify the almostfull signal to match design specs
```

5.

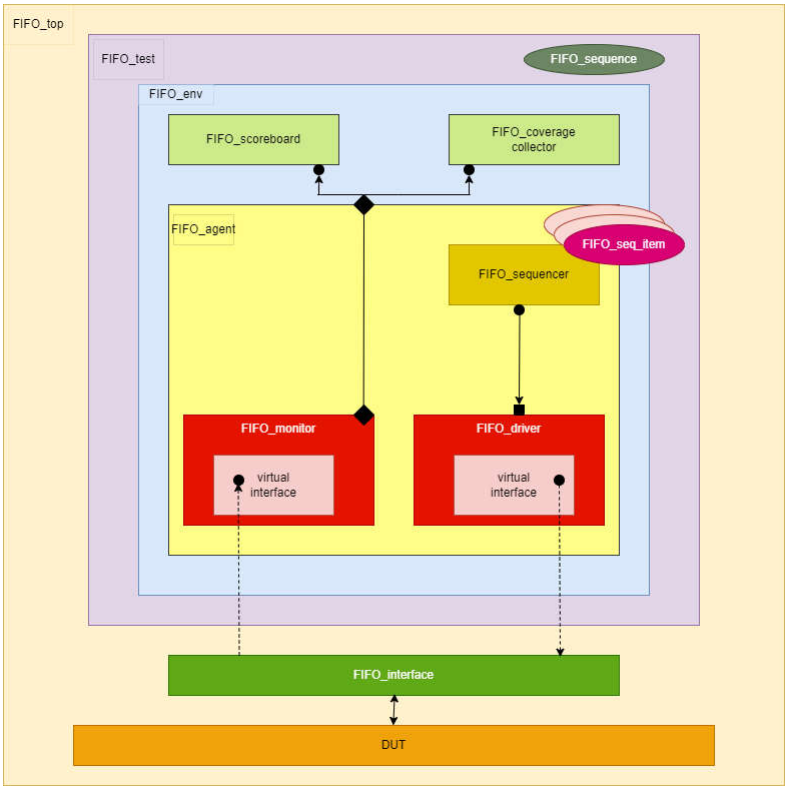
Original design

```
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
    end
```

Modified design

```
else begin
  if ({f_if.wr_en, f_if.rd_en} == 2'b10 && !f_if.full)
    count <= count + 1;
  else if ({f_if.wr_en, f_if.rd_en} == 2'b01 && !f_if.empty)
    count <= count - 1;
  else if ({f_if.wr_en, f_if.rd_en} == 2'b11 && f_if.full) // add unhandled case
    count <= count - 1;
  else if ({f_if.wr_en, f_if.rd_en} == 2'b11 && f_if.empty) // add unhandled case
    count <= count + 1;
end
```

V. UVM Structure and description



- UVM flow of FIFO

1. FIFO_top: root of the hierarchy, instantiate the design, interface and bind assertions, generate the clock, set the interface into the database and finally run the test
2. Config. Database: a shared database between all components
3. Config. Object: an object that holds configuration settings and parameters for components.
4. FIFO_test: here we build the environment, config, and different sequences like reset sequence and other scenarios, get the virtual interface from the database and put into container and set it into database again, in run phase we raise objection and run all the different sequences then drop objection
5. FIFO_sequence: core stimulus of any verification plan, made up of several sequence items, parametrized class by type, when sequence starts it tells the sequencer it has data to produce then the sequencer waits for the driver to pull data by telling get_next_item(); then sequencer take the data from the sequence to driver when finish_item means sequence is ready to be sent. In FIFO project I have created many sequences
   - Reset sequence
   - Write and read randomly
   - Write only no read
   - Read only no write
   - Write to the FIFO until it is full, then start reading until the FIFO is empty
   - Full sequence: write to the FIFO and continue writing after it is fills up to check overflow
   - Empty sequence: read from the FIFO and continue reading after it is empty to check underflow
6. FIFO_seq_item: data field to communicate with the design, randomized data is generated, constraints is added to stimulus
7. FIFO_env: build agent, scoreboard, and coverage collector and connect analysis port of agent to scoreboard an coverage collector exports
8. FIFO_agent: build sequencer, driver, and monitor, get the config. Object from database, create agent analysis port connect driver virtual interface and monitor interface to cofig. Object virtual interface, connect driver port to sequencer export and mon analysis port to agent analysis port
9. FIFO_sequencer: generate transactions as class objects and sends it to the driver (acting as fifo)

10. FIFO_driver: pulls the data from the sequence by get_next_item(); as mentioned above, drive the sequence item in the run phase task using the virtual interface
11. FIFO_monitor: capture signal information from the design ports and translate it into seq items then broadcasts those sequence items analysis components like scoreboard and coverage collector
12. FIFO_scoreboard: receives seq items from the monitor, check them using reference model and compare them with design outputs, build analysis port and export and connect them, finally report phase to report correct and wrong transactions
13. FIFO_coverage: receives seq items from the monitor, sample the data fields for functional coverage, buil analysis port and export and connect them

VI. Design and UVM codes

1. FIFO_top

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

module FIFO_top();
    bit clk;
    bit rst_n;

   // clock generation
    initial begin
        clk = 0;
        forever
            #1 clk = ~clk;
    end

    //instantiates DUT, interface and bind assertions
    FIFO_if f_if(clk);
    FIFO DUT (f_if);
    bind FIFO FIFO_SVA FIFO_SVA_INSTA(f_if);

    initial begin //passes the interface using config. database
        uvm_config_db#(virtual FIFO_if)::set(null, "uvm_test_top", "FIFO_IF", f_if);

        run_test("FIFO_test");  //run the test
    end
endmodule
```

2. FIFO_interface

```
interface FIFO_if (clk);
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;
bit [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;

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


endinterface //FIFO_if
```

3. Modified design

```
module FIFO(FIFO_if.DUT f_if);
  localparam max_fifo_addr = $clog2(f_if.FIFO_DEPTH);

  reg [f_if.FIFO_WIDTH-1:0] mem [f_if.FIFO_DEPTH-1:0];
  reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
  reg [max_fifo_addr:0] count;

  //write operation
  always @(posedge f_if.clk or negedge f_if.rst_n) begin
    if (!f_if.rst_n) begin
      wr_ptr <= 0;
      f_if.overflow <= 0; // reset overflow signal
      f_if.wr_ack <= 0; // reset wr_ack signal
    end
    else if (f_if.wr_en && count < f_if.FIFO_DEPTH) begin
      mem[wr_ptr] <= f_if.data_in;
      f_if.wr_ack <= 1;
      wr_ptr <= wr_ptr + 1;
    end
    else begin
      f_if.wr_ack <= 0;
      if (f_if.full && f_if.wr_en)
        f_if.overflow <= 1;
      else
        f_if.overflow <= 0;
    end
  end

  //read operation
  always @(posedge f_if.clk or negedge f_if.rst_n) begin
    if (!f_if.rst_n) begin
      rd_ptr <= 0;
      f_if.underflow <= 0; // reset underflow signal
    end
    else if (f_if.rd_en && count != 0) begin
      f_if.data_out <= mem[rd_ptr];
      rd_ptr <= rd_ptr + 1;
    end
    else begin
      if (f_if.empty && f_if.rd_en)  // make underflow signal sequential
        f_if.underflow <= 1;
      else
        f_if.underflow <= 0;
    end
  end
  // count operations
  always @(posedge f_if.clk or negedge f_if.rst_n) begin
    if (!f_if.rst_n) begin
      count <= 0;
    end
    else begin
      if ({f_if.wr_en, f_if.rd_en} == 2'b10 && !f_if.full)
        count <= count + 1;
      else if ({f_if.wr_en, f_if.rd_en} == 2'b01 && !f_if.empty)
        count <= count - 1;
      else if ({f_if.wr_en, f_if.rd_en} == 2'b11 && f_if.full) // add unhandled case
        count <= count - 1;
      else if ({f_if.wr_en, f_if.rd_en} == 2'b11 && f_if.empty) // add unhandled case
        count <= count + 1;
    end
  end
  // flags operations
  assign f_if.full = (count == f_if.FIFO_DEPTH) ? 1 : 0;
  assign f_if.empty = (count == 0) ? 1 : 0;
  assign f_if.almostfull = (count == f_if.FIFO_DEPTH-1) ? 1 : 0; // modify the almostfull signal to match design specs
  assign f_if.almostempty = (count == 1) ? 1 : 0;

endmodule
```

4. FIFO_config

```
package FIFO_config_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_config extends uvm_object;
`uvm_object_utils(FIFO_config)

virtual FIFO_if FIFO_vif;

function new(string name = "FIFO_config");
    super.new(name);
endfunction
endclass
endpackage
```

5. FIFO_test

```
package FIFO_test_pkg;
import FIFO_env_pkg::*;
import FIFO_config_pkg::*;
import FIFO_write_read_sequence_pkg::*;
import FIFO_read_only_sequence_pkg::*;
import FIFO_write_only_sequence_pkg::*;
import FIFO_write_then_read_sequence_pkg::*;
import FIFO_full_sequence_pkg::*;
import FIFO_empty_sequence_pkg::*;
import FIFO_reset_sequence_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_test extends uvm_test;
`uvm_component_utils(FIFO_test)

FIFO_env env;
FIFO_config FIFO_cfg;
FIFO_write_read_sequence write_read_seq;
FIFO_write_then_read_sequence write_then_read_seq;
FIFO_write_only_sequence write_seq;
FIFO_read_only_sequence read_seq;
FIFO_reset_sequence reset_seq;
FIFO_full_sequence full_seq;
FIFO_empty_sequence empty_seq;

function new(string name = "FIFO_test", uvm_component parent = null);
    super.new(name, parent);
endfunction
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");
write_read_seq = FIFO_write_read_sequence::type_id::create("write_read_seq");
write_seq = FIFO_write_only_sequence::type_id::create("write_seq");
read_seq = FIFO_read_only_sequence::type_id::create("read_seq");
write_then_read_seq = FIFO_write_then_read_sequence::type_id::create("write_then_read_seq");
reset_seq = FIFO_reset_sequence::type_id::create("reset_seq");
full_seq = FIFO_full_sequence::type_id::create("full_seq");
empty_seq = FIFO_empty_sequence::type_id::create("empty_seq");
```

```
if(!uvm_config_db #(virtual FIFO_if)::get(this,"", "FIFO_IF", FIFO_cfg.FIFO_vif))
`uvm_fatal("build_phase", "Test - Unable to get the virtual interface of thr FIFO from the uvm_config_db");

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

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

    `uvm_info("run_phase", "write_read Sequence Generation Started", UVM_LOW)
    write_read_seq.start(env.agt.sqr);
    `uvm_info("run_phase", "write_read Sequence Generation Ended", UVM_LOW)

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

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

    `uvm_info("run_phase", "Reset Asserted", UVM_LOW)
    reset_seq.start(env.agt.sqr);
    `uvm_info("run_phase", "Reset Desserted", UVM_LOW)

    `uvm_info("run_phase", "Write then Read Sequence Generation Started", UVM_LOW)
    write_then_read_seq.start(env.agt.sqr);
    `uvm_info("run_phase", "Write then Read Sequence Generation Ended", UVM_LOW)
```

```
    `uvm_info("run_phase", "Full Sequence Generation Started", UVM_LOW)
    full_seq.start(env.agt.sqr);
    `uvm_info("run_phase", "Full Sequence Generation Ended", UVM_LOW)

    `uvm_info("run_phase", "Empty Sequence Generation Started", UVM_LOW)
    empty_seq.start(env.agt.sqr);
    `uvm_info("run_phase", "Empty Sequence Generation Ended", UVM_LOW)
    phase.drop_objection(this);
endtask: run_phase
endclass: FIFO_test
endpackage
```

6. FIFO_sequences
   - Reset sequence

```
package FIFO_reset_sequence_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_reset_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_reset_sequence);
FIFO_seq_item  seq_item;

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

task body;
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
seq_item.rst_n = 0;
seq_item.wr_en = 0;
seq_item.rd_en = 0;
seq_item.data_in = 0;
finish_item(seq_item);
endtask
endclass
endpackage
```

   - Write and read sequence

```
package FIFO_write_read_sequence_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_write_read_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_write_read_sequence);
FIFO_seq_item seq_item;

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

task body;
repeat (10000) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize());
finish_item(seq_item);
end
endtask
endclass
endpackage
```

- Write only sequence

```
package FIFO_write_only_sequence_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_write_only_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_write_only_sequence);
FIFO_seq_item seq_item;

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

task body;
repeat (10000) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 1;    // Write enabled
                rd_en == 0;    // Read disabled
            });
finish_item(seq_item);
end
endtask
endclass
endpackage
```

- Read only sequence

```
package FIFO_read_only_sequence_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_read_only_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_read_only_sequence);
FIFO_seq_item seq_item;

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

task body;
repeat (10000) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 0;    // Write disabled
                rd_en == 1;    // Read enabled
            });
finish_item(seq_item);
end
endtask
endclass
endpackage
```

- Write then read sequence

```
package FIFO_write_then_read_sequence_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_write_then_read_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_write_then_read_sequence);
FIFO_seq_item seq_item;

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

task body;
repeat (8) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 1;    // Write enabled
                rd_en == 0;    // Read disabled
                rst_n == 1;
            });
finish_item(seq_item);
end
repeat (8) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 0;    // Write disabled
                rd_en == 1;    // Read enabled
                rst_n == 1;
            });
finish_item(seq_item);
end
endtask
endclass
endpackage
```

- Full sequence

```
package FIFO_full_sequence_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_full_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_full_sequence);
FIFO_seq_item seq_item;

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

task body;
seq_item = FIFO_seq_item::type_id::create("seq_item");
repeat (seq_item.FIFO_DEPTH) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 1;    // Write enabled
                rd_en == 0;    // Read disabled
                rst_n == 1;
            });
finish_item(seq_item);
end

repeat (5) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 1;    // Write enabled
                rd_en == 0;    // Read disabled
                rst_n == 1;
            });
finish_item(seq_item);
end
endtask
endclass
endpackage
```

- Empty sequence

```
package FIFO_empty_sequence_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_empty_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_empty_sequence);
FIFO_seq_item seq_item;

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

task body;
seq_item = FIFO_seq_item::type_id::create("seq_item");
repeat (seq_item.FIFO_DEPTH) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 0;    // Write disabled
                rd_en == 1;    // Read enabled
                rst_n == 1;
            });
finish_item(seq_item);
end

repeat (5) begin
seq_item = FIFO_seq_item::type_id::create("seq_item");
start_item(seq_item);
assert(seq_item.randomize()  with {
                wr_en == 0;    // Write disabled
                rd_en == 1;    // Read enabled
                rst_n == 1;
            });
finish_item(seq_item);
end
endtask
endclass
endpackage
```

7. FIFO_seq_item

```
package FIFO_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_seq_item extends uvm_sequence_item;
`uvm_object_utils(FIFO_seq_item);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
rand logic rst_n,wr_en, rd_en;
rand logic [FIFO_WIDTH-1:0] data_in;
logic [FIFO_WIDTH-1:0] data_out;
logic full, empty, almostfull, almostempty, underflow, wr_ack, overflow;

int RD_EN_ON_DIST = 30;
int WR_EN_ON_DIST = 70;


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

function string convert2string();
    return $sformatf("%s rst_n = 0b%b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b, data_out = 0b%0b, full = 0b%0b, empty = %0s, almostfull = 0b%0b, almostempty = 0b%0b, ove
     overflow, underflow, wr_ack);
endfunction

function string convert2string_stimulus();
    return $sformatf("rst_n = 0b%b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b",  rst_n, wr_en, rd_en, data_in);
endfunction

constraint rst_dist{rst_n dist {0:=1, 1:=99};} //rst
constraint write_enable_dist { wr_en dist {0 := 100 - WR_EN_ON_DIST, 1 := WR_EN_ON_DIST}; } //write
constraint read_enable_dist { rd_en dist {0 := 100 - RD_EN_ON_DIST, 1 := RD_EN_ON_DIST}; } //read

endclass

endpackage
```

8. FIFO_env

```
package FIFO_env_pkg;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;
import FIFO_agent_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_env extends uvm_env;
`uvm_component_utils(FIFO_env)

FIFO_agent agt;
FIFO_scoreboard sb;
FIFO_coverage cov;

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

function void build_phase(uvm_phase phase);
super.build_phase(phase);
agt = FIFO_agent::type_id::create("agt", this);
sb = FIFO_scoreboard::type_id::create("sb", this);
cov = FIFO_coverage::type_id::create("cov", this);
endfunction: build_phase

function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
agt.agt_ap.connect(sb.sb_export);
agt.agt_ap.connect(cov.cov_export);
endfunction: connect_phase
endclass

endpackage
```

9. FIFO_agent

```
package FIFO_agent_pkg;
import FIFO_config_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_monitor_pkg::*;
import FIFO_seq_item_pkg::*;
import FIFO_sequencer_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_agent extends uvm_agent;
`uvm_component_utils(FIFO_agent)

FIFO_driver driver;
FIFO_sequencer sqr;
FIFO_config FIFO_cfg;
FIFO_monitor mon;
uvm_analysis_port #(FIFO_seq_item) agt_ap;
function new(string name = "FIFO_agent", uvm_component parent = null);
    super.new(name, parent);
endfunction
function void build_phase(uvm_phase phase);
super.build_phase(phase);
if(!uvm_config_db #(FIFO_config)::get(this,"", "CFG", FIFO_cfg)) begin
`uvm_fatal("build_phase", "Driver - Unable to get configuration object");
end
driver = FIFO_driver::type_id::create("driver", this);
sqr = FIFO_sequencer::type_id::create("sqr", this);
mon = FIFO_monitor::type_id::create("mon", 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;
mon.FIFO_vif = FIFO_cfg.FIFO_vif;
driver.seq_item_port.connect(sqr.seq_item_export);
mon.mon_ap.connect(agt_ap);
endfunction: connect_phase
endclass
endpackage
```

10. FIFO_sequencer

```
package FIFO_sequencer_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_sequencer extends uvm_sequencer #(FIFO_seq_item);
`uvm_component_utils(FIFO_sequencer);

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

endclass
endpackage
```

11. FIFO_driver

```
package FIFO_driver_pkg;
import FIFO_config_pkg::*;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_driver extends uvm_driver #(FIFO_seq_item);
`uvm_component_utils(FIFO_driver)

virtual FIFO_if FIFO_vif;
FIFO_seq_item stim_seq_item;

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
    stim_seq_item = FIFO_seq_item::type_id::create("stim_seq_item");
    seq_item_port.get_next_item(stim_seq_item);
    FIFO_vif.rst_n = stim_seq_item.rst_n;
    FIFO_vif.wr_en = stim_seq_item.wr_en;
    FIFO_vif.rd_en = stim_seq_item.rd_en;
    FIFO_vif.data_in = stim_seq_item.data_in;
    @(negedge FIFO_vif.clk);
    seq_item_port.item_done();
    `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
end
endtask
endclass
endpackage
```

12. FIFO_monitor

```
package FIFO_monitor_pkg;
import FIFO_seq_item_pkg::*;
import uvm_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
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.rst_n = FIFO_vif.rst_n;
    rsp_seq_item.rd_en = FIFO_vif.rd_en;
    rsp_seq_item.wr_en = FIFO_vif.wr_en;
    rsp_seq_item.data_in = FIFO_vif.data_in;
    rsp_seq_item.data_out = FIFO_vif.data_out;
    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.overflow = FIFO_vif.overflow;
    rsp_seq_item.underflow = FIFO_vif.underflow;
    rsp_seq_item.wr_ack = FIFO_vif.wr_ack;
    mon_ap.write(rsp_seq_item);
    `uvm_info("run_phase", rsp_seq_item.convert2string_stimulus(), UVM_HIGH)
end
endtask
endclass
endpackage
```

13. FIFO_scoreboard

```
package FIFO_scoreboard_pkg;
import FIFO_seq_item_pkg::*;
import uvm_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 seq_item_sb;
  int erorr_count = 0;
  int correct_count = 0;

  parameter FIFO_WIDTH = 16;
  parameter FIFO_DEPTH = 8;
  localparam max_fifo_addr = $clog2(FIFO_DEPTH);

  bit [FIFO_WIDTH-1:0] data_out_ref;
  bit wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
  bit [max_fifo_addr:0] count;
  bit [6:0] ref_flags, dut_flags;

  bit [FIFO_WIDTH-1:0] mem_queue[$];


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

  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(seq_item_sb);
      ref_model(seq_item_sb);
      ref_flags = {full_ref, empty_ref, almostfull_ref, almostempty_ref, overflow_ref, underflow_ref, wr_ack_ref};
      dut_flags = {seq_item_sb.full, seq_item_sb.empty, seq_item_sb.almostfull, seq_item_sb.almostempty, seq_item_sb.overflow, seq_item_sb.underflow, seq_item_sb.wr_ack};
      if (seq_item_sb.data_out != data_out_ref && ref_flags != dut_flags) begin
        `uvm_error("run_phase", $sformatf("comparison failed, Transaction received by the DUT: 0b%0b and DUT FLAGS: %b While the reference out: 0b%0b and REF FLAGS: %b", seq_i
        erorr_count++;
      end else begin
        `uvm_info("run_phase", $sformatf("correct FIFO out: %s", seq_item_sb.convert2string()), UVM_HIGH);
        correct_count++;
      end
    end
  endtask

task ref_model(FIFO_seq_item seq_item_chk);
    fork
      begin // write
        if (!seq_item_chk.rst_n) begin
          wr_ack_ref = 0;
          overflow_ref = 0;
          full_ref = 0;
          almostfull_ref = 0;
          mem_queue.delete();
        end else if (seq_item_chk.wr_en && count < FIFO_DEPTH) begin
          wr_ack_ref = 1;
          mem_queue.push_back(seq_item_chk.data_in);
        end else begin
          wr_ack_ref = 0;
          overflow_ref = (full_ref && seq_item_chk.wr_en) ? 1 : 0;
        end
      end
```

```
      begin //read
        if (!seq_item_chk.rst_n) begin
          empty_ref = 1;
          underflow_ref = 0;
          almostempty_ref = 0;
        end else if (seq_item_chk.rd_en && count != 0) begin
          data_out_ref = mem_queue.pop_front();
        end else begin
          underflow_ref = (empty_ref && seq_item_chk.rd_en) ? 1 : 0;
        end
      end
    join
    if (!seq_item_chk.rst_n) begin // count
      count = 0;
    end else if (seq_item_chk.wr_en && !seq_item_chk.rd_en && !full_ref) begin
      count = count + 1;
    end else if (!seq_item_chk.wr_en && seq_item_chk.rd_en && !empty_ref) begin
      count = count - 1;
    end else if (seq_item_chk.wr_en && seq_item_chk.rd_en && full_ref) begin
      count = count - 1;
    end else if (seq_item_chk.wr_en && seq_item_chk.rd_en && empty_ref) begin
      count = count + 1;
    end
    //flags
    full_ref = (count == FIFO_DEPTH) ? 1 : 0;
    empty_ref = (count == 0) ? 1 : 0;
    almostfull_ref = (count == FIFO_DEPTH - 1) ? 1 : 0;
    almostempty_ref = (count == 1) ? 1 : 0;
endtask
  function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info("report_phase", $sformatf("Total Successful Transactions : %0d", correct_count), UVM_MEDIUM);
    `uvm_info("report_phase", $sformatf("Total FAILED Transactions : %0d", erorr_count), UVM_MEDIUM);
  endfunction: report_phase
endclass
endpackage
```

14. FIFO_coverage

```
package FIFO_coverage_pkg;
import FIFO_seq_item_pkg::*;
import uvm_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 seq_item_cov;

    covergroup FIFO_cvgr;
    // Coverpoints
    wr_en_cp:       coverpoint seq_item_cov.wr_en;
    rd_en_cp:       coverpoint seq_item_cov.rd_en;
    full_cp:        coverpoint seq_item_cov.full;
    empty_cp:       coverpoint seq_item_cov.empty;
    almostfull_cp:  coverpoint seq_item_cov.almostfull;
    almostempty_cp: coverpoint seq_item_cov.almostempty;
    overflow_cp:    coverpoint seq_item_cov.overflow;
    underflow_cp:   coverpoint seq_item_cov.underflow;
    wr_ack_cp:      coverpoint seq_item_cov.wr_ack;
    // Cross coverage
    wr_full:        cross wr_en_cp, full_cp;
    wr_empty:       cross wr_en_cp, empty_cp;
    wr_almostfull:  cross wr_en_cp, almostfull_cp;
    wr_almostempty: cross wr_en_cp, almostempty_cp;
    wr_overflow:    cross wr_en_cp, overflow_cp{

        ignore_bins wr_en0_overflow1 = !binsof(wr_en_cp) intersect{1} && binsof(overflow_cp) intersect{1};
    }
    wr_underflow:   cross wr_en_cp, underflow_cp;
    wr_wr_ack:      cross wr_en_cp, wr_ack_cp{

        ignore_bins wr_en0_wr_ack1 = !binsof(wr_en_cp) intersect{1} && binsof(wr_ack_cp) intersect{1};
    }

    rd_full:        cross rd_en_cp, full_cp{

        ignore_bins rd_en1_full1 = binsof(rd_en_cp) intersect{1} && binsof(full_cp) intersect{1};
```

```
    }
    rd_empty:       cross rd_en_cp, empty_cp;
    rd_almostfull:  cross rd_en_cp, almostfull_cp;
    rd_almostempty: cross rd_en_cp, almostempty_cp;
    rd_overflow:    cross rd_en_cp, overflow_cp;
    rd_underflow:   cross rd_en_cp, underflow_cp{

        ignore_bins rd_en0_underflow1 = !binsof(rd_en_cp) intersect{1} && binsof(underflow_cp) intersect{1};
    }
    rd_wr_ack:      cross rd_en_cp, wr_ack_cp;
endgroup

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

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
cov_fifo.get(seq_item_cov);
FIFO_cvgr.sample();
end
endtask
endclass
endpackage
```

15. Assertions

```
module FIFO_SVA(FIFO_if.DUT f_if);

// reset asseration
always_comb begin
if(!f_if.rst_n) begin
reset_assert: assert final(!DUT.wr_ptr && !DUT.rd_ptr && !DUT.count);
reset_cover: cover final(!DUT.wr_ptr && !DUT.rd_ptr && !DUT.count);
end
end
//immediate asserations
always_comb begin
//full
if(f_if.rst_n && (DUT.count== f_if.FIFO_DEPTH)) begin
full_flag_assert: assert final(f_if.full && !f_if.empty && !f_if.almostempty && !f_if.almostfull);
full_flag_cover: cover final(f_if.full && !f_if.empty && !f_if.almostempty && !f_if.almostfull);
end
//almostfull
if(f_if.rst_n && (DUT.count== f_if.FIFO_DEPTH-1)) begin
almostfull_flag_assert: assert final(f_if.almostfull && !f_if.empty && !f_if.almostempty && !f_if.full);
almostfull_flag_cover: cover final(f_if.almostfull && !f_if.empty && !f_if.almostempty && !f_if.full);
end
//empty
if(f_if.rst_n && (DUT.count== 0)) begin
empty_flag_assert: assert final(f_if.empty && !f_if.almostempty && !f_if.full && !f_if.almostfull);
empty_flag_acover: cover final(f_if.empty && !f_if.almostempty && !f_if.full && !f_if.almostfull);
end
//almostempty
if(f_if.rst_n && (DUT.count== 1)) begin
almostempty_flag_assert: assert final(f_if.almostempty && !f_if.empty && !f_if.full && !f_if.almostfull);
almostempty_flag_cover: cover final(f_if.almostempty && !f_if.empty && !f_if.full && !f_if.almostfull);
end
end
```

```
//concurrent assertions
//overflow
property overflow_flag;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.full && f_if.wr_en) |=> f_if.overflow;
endproperty
overflow_flag_assert: assert property(overflow_flag);
overflow_flag_cover: cover property(overflow_flag);
//underflow
property underflow_flag;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.empty && f_if.rd_en) |=> f_if.underflow;
endproperty
underflow_flag_assert: assert property(underflow_flag);
underflow_flag_cover: cover property(underflow_flag);
//wr_ack_high
property wr_ack_high;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && DUT.count< f_if.FIFO_DEPTH) |=> f_if.wr_ack;
endproperty
wr_ack_high_flag_assert: assert property(wr_ack_high);
wr_ack_high_flag_cover: cover property(wr_ack_high);
//wr_ack_high
property wr_ack_low;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && f_if.full) |=> !f_if.wr_ack;
endproperty
wr_ack_low_flag_assert: assert property(wr_ack_low);
wr_ack_low_flag_cover: cover property(wr_ack_low);
//write
property write_op;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && !f_if.rd_en && !f_if.full) |=> $past(DUT.count+ 1'b1);
endproperty
write_op_assert: assert property(write_op);
write_op_cover: cover property(write_op);
//read
property read_op;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (!f_if.wr_en && f_if.rd_en && !f_if.empty) |=> (DUT.count+ 1'b1);
endproperty
read_op_assert: assert property(read_op);
read_op_cover: cover property(read_op);
```

```
//write priority
property write_pri;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && f_if.rd_en && f_if.full) |=> (DUT.count+ 1'b1);
endproperty
write_pri_assert: assert property(write_pri);
write_pri_cover: cover property(write_pri);
//read priority
property read_pri;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && f_if.rd_en && f_if.empty) |=> $past(DUT.count+ 1'b1);
endproperty
read_pri_assert: assert property(read_pri);
read_pri_cover: cover property(read_pri);
// read pointer
property read_ptr;
@(posedge f_if.clk) disable iff (!f_if.rst_n) (f_if.rd_en && (DUT.count != 0)) |=> (DUT.rd_ptr == ($past(DUT.rd_ptr) + 1) % f_if.FIFO_DEPTH);
endproperty
read_ptr_assert: assert property(read_ptr);
read_ptr_cover: cover property(read_ptr);
// write pointer
property write_ptr;
@(posedge f_if.clk) disable iff (!f_if.rst_n) (f_if.wr_en && (DUT.count < f_if.FIFO_DEPTH)) |=> (DUT.wr_ptr == ($past(DUT.wr_ptr) + 1) % f_if.FIFO_DEPTH);
endproperty
write_ptr_assert: assert property(write_ptr);
write_ptr_cover: cover property(write_ptr);

endmodule
```

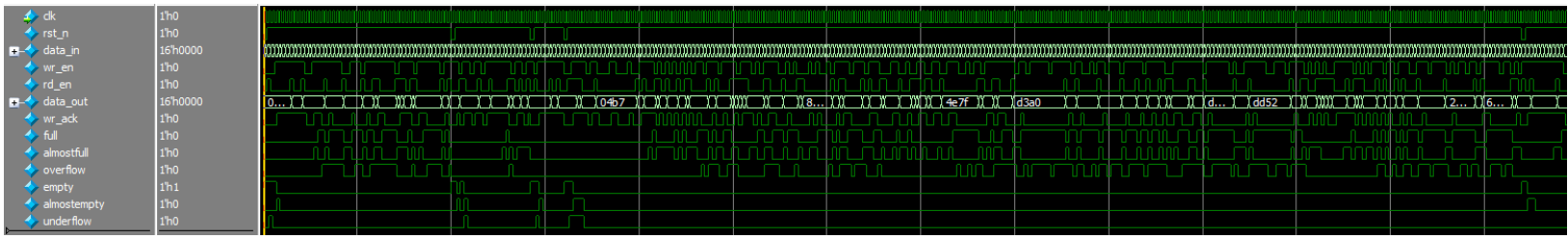
VII. Questa snippets

1. Report of the results

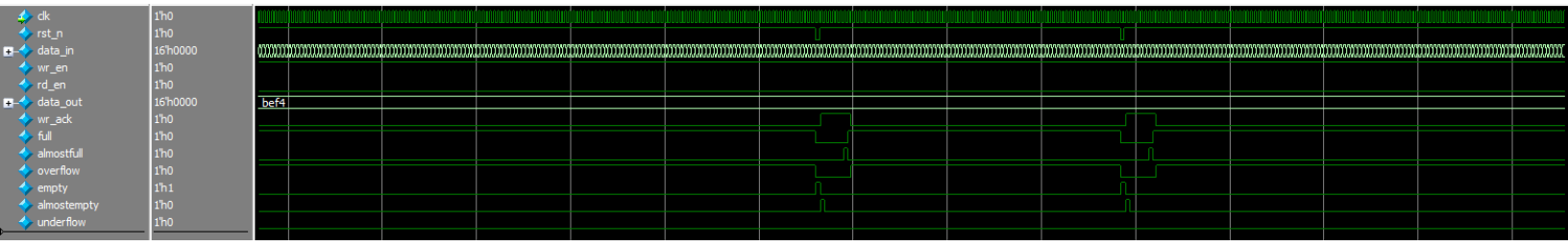
```
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(all)
# UVM_INFO @ 0: reporter [RNTST] Running test FIFO_test...
# UVM_INFO FIFO_test.sv(52) @ 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(54) @ 2: uvm_test_top [run_phase] Reset Desserted
# UVM_INFO FIFO_test.sv(56) @ 2: uvm_test_top [run_phase] write_read Sequence Generation Started
# UVM_INFO FIFO_test.sv(58) @ 20002: uvm_test_top [run_phase] write_read Sequence Generation Ended
# UVM_INFO FIFO_test.sv(60) @ 20002: uvm_test_top [run_phase] Write Sequence Generation Started
# UVM_INFO FIFO_test.sv(62) @ 40002: uvm_test_top [run_phase] Write Sequence Generation Ended
# UVM_INFO FIFO_test.sv(64) @ 40002: uvm_test_top [run_phase] Read Sequence Generation Started
# UVM_INFO FIFO_test.sv(66) @ 60002: uvm_test_top [run_phase] Read Sequence Generation Ended
# UVM_INFO FIFO_test.sv(68) @ 60002: uvm_test_top [run_phase] Reset Asserted
# UVM_INFO FIFO_test.sv(70) @ 60004: uvm_test_top [run_phase] Reset Desserted
# UVM_INFO FIFO_test.sv(72) @ 60004: uvm_test_top [run_phase] Write then Read Sequence Generation Started
# UVM_INFO FIFO_test.sv(74) @ 60036: uvm_test_top [run_phase] Write then Read Sequence Generation Ended
# UVM_INFO FIFO_test.sv(76) @ 60036: uvm_test_top [run_phase] Full Sequence Generation Started
# UVM_INFO FIFO_test.sv(78) @ 60062: uvm_test_top [run_phase] Full Sequence Generation Ended
# UVM_INFO FIFO_test.sv(80) @ 60062: uvm_test_top [run_phase] Empty Sequence Generation Started
# UVM_INFO FIFO_test.sv(82) @ 60088: uvm_test_top [run_phase] Empty Sequence Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 60088: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(113) @ 60088: uvm_test_top.env.sb [report_phase] Total Successful Transactions : 30044
# UVM_INFO FIFO_scoreboard.sv(114) @ 60088: uvm_test_top.env.sb [report_phase] Total FAILED Transactions : 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   22
# 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]    16
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 60088 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
```

2. Each UVM sequence waveform

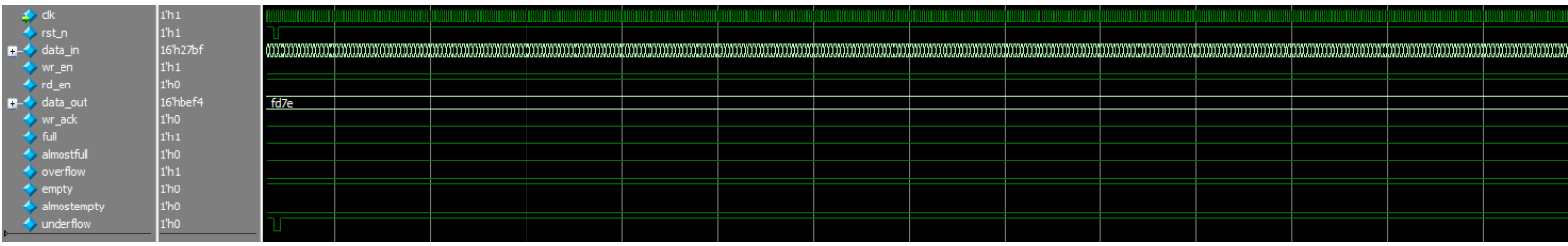
- Write and read sequence



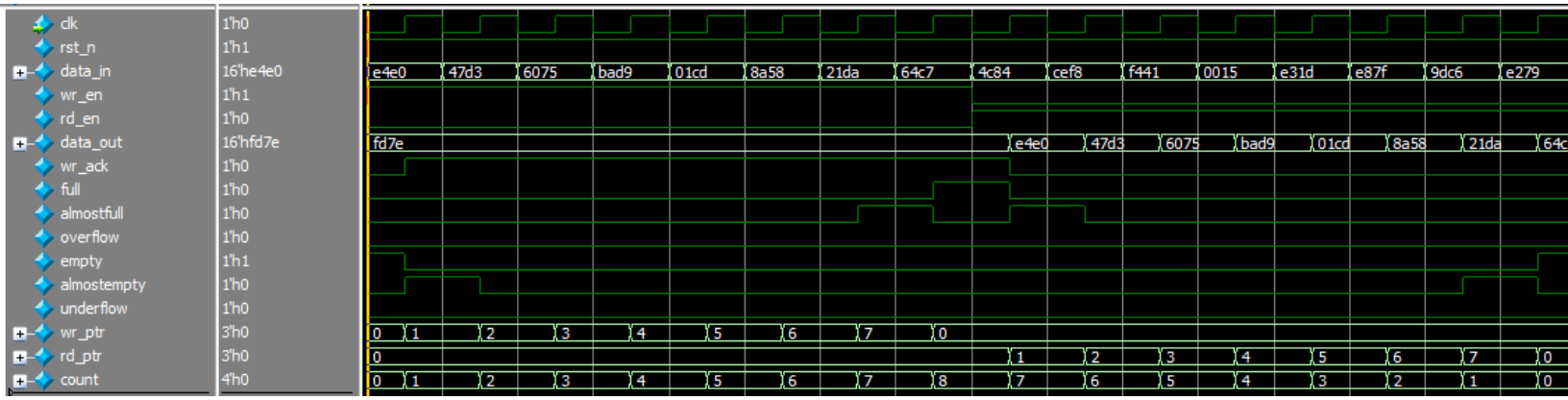
- Write only sequence



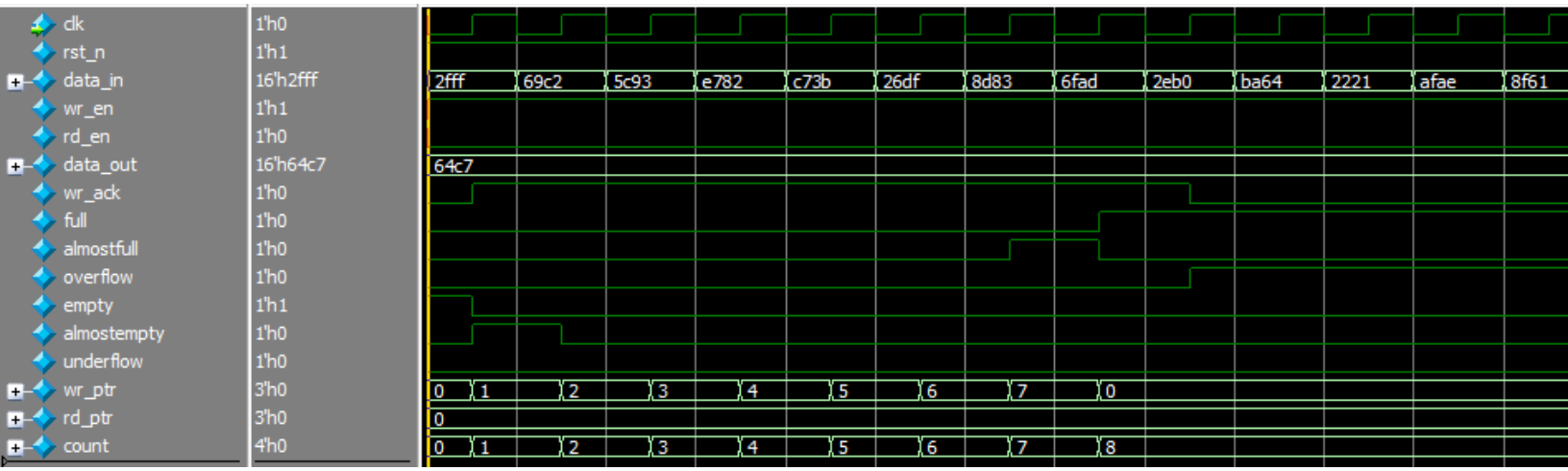
- Read only sequence



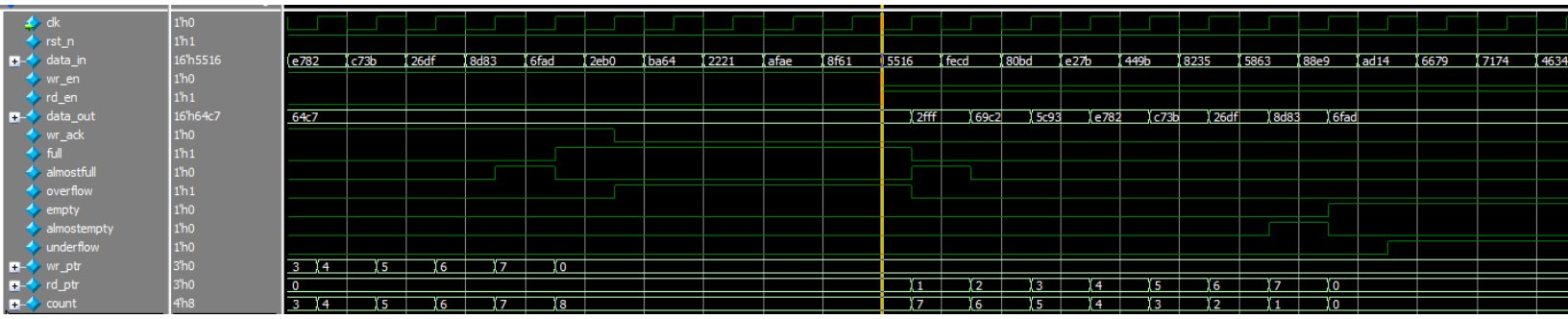
- Write then read sequence



- Full sequence



- Empty sequence



3. Coverage snippets

- Assertions passed

/FIFO_empty_sequence_pkg::FIFO_empty_sequence::body/#ublk#67718807#...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_empty_sequence_pkg::FIFO_empty_sequence::body/#ublk#67718807#...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_full_sequence_pkg::FIFO_full_sequence::body/#ublk#123788807#15/im...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_full_sequence_pkg::FIFO_full_sequence::body/#ublk#123788807#26/im...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_write_then_read_sequence_pkg::FIFO_write_then_read_sequence::bod...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_write_then_read_sequence_pkg::FIFO_write_then_read_sequence::bod...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_write_only_sequence_pkg::FIFO_write_only_sequence::body/#ublk#392...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_read_only_sequence_pkg::FIFO_read_only_sequence::body/#ublk#1805...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_write_read_sequence_pkg::FIFO_write_read_sequence::body/#ublk#33...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/reset_assert	Immediate	SVA	on	0	1	-	-	-	-		off	assert (!DUT.wr_ptr&!DUT.rd_ptr&...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/full_flag_assert	Immediate	SVA	on	0	1	-	-	-	-		off	assert (f_if.full&~f_if.empty&~f_if....	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/almostfull_flag_assert	Immediate	SVA	on	0	1	-	-	-	-		off	assert (f_if.almostfull&~f_if.empty...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/empty_flag_assert	Immediate	SVA	on	0	1	-	-	-	-		off	assert (f_if.empty&~f_if.almostem...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/almostempty_flag_assert	Immediate	SVA	on	0	1	-	-	-	-		off	assert (f_if.almostempty&~f_if.em...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/overflow_flag_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/underflow_flag_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_high_flag_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_low_flag_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/write_op_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/read_op_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/write_pri_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/read_pri_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/read_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓
/FIFO_top/DUT/FIFO_SVA_INSTA/write_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge f_if.clk) disable...	✓

- 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/FIFO_SVA_INSTA/reset_cover	SVA	✓	Off	301	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/full_flag_cover	SVA	✓	Off	1568	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/almostfull_flag_cov...	SVA	✓	Off	1858	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/empty_flag_acover	SVA	✓	Off	320	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/almostempty_flag_...	SVA	✓	Off	244	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/overflow_flag_cov...	SVA	✓	Off	12391	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/underflow_flag_co...	SVA	✓	Off	9842	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_high_flag_...	SVA	✓	Off	4277	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_low_flag_c...	SVA	✓	Off	12391	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/write_op_cover	SVA	✓	Off	3266	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/read_op_cover	SVA	✓	Off	904	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/write_pri_cover	SVA	✓	Off	963	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/read_pri_cover	SVA	✓	Off	32	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/read_ptr_cover	SVA	✓	Off	2846	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/FIFO_SVA_INSTA/write_ptr_cover	SVA	✓	Off	4277	1	Unli...	1	100%		✓	0	0	0 ns	0

- Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_pkg/FIFO_coverage		100.00%							
TYPE FIFO_cvgr		100.00%	100	100.00...			auto(1)		
CVP FIFO_cvgr::wr_en_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::rd_en_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::full_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::empty_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::almostfull_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::almostempty_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::overflow_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::underflow_cp		100.00%	100	100.00...					
CVP FIFO_cvgr::wr_ack_cp		100.00%	100	100.00...					
CROSS FIFO_cvgr::wr_full		100.00%	100	100.00...					
CROSS FIFO_cvgr::wr_empty		100.00%	100	100.00...					
CROSS FIFO_cvgr::wr_almostfull		100.00%	100	100.00...					
CROSS FIFO_cvgr::wr_almostempty		100.00%	100	100.00...					
CROSS FIFO_cvgr::wr_overflow		100.00%	100	100.00...					
CROSS FIFO_cvgr::wr_underflow		100.00%	100	100.00...					
CROSS FIFO_cvgr::wr_wr_ack		100.00%	100	100.00...					
CROSS FIFO_cvgr::rd_full		100.00%	100	100.00...					
CROSS FIFO_cvgr::rd_empty		100.00%	100	100.00...					
CROSS FIFO_cvgr::rd_almostfull		100.00%	100	100.00...					
CROSS FIFO_cvgr::rd_almostempty		100.00%	100	100.00...					
CROSS FIFO_cvgr::rd_overflow		100.00%	100	100.00...					
CROSS FIFO_cvgr::rd_underflow		100.00%	100	100.00...					
CROSS FIFO_cvgr::rd_wr_ack		100.00%	100	100.00...					

- If signals toggle

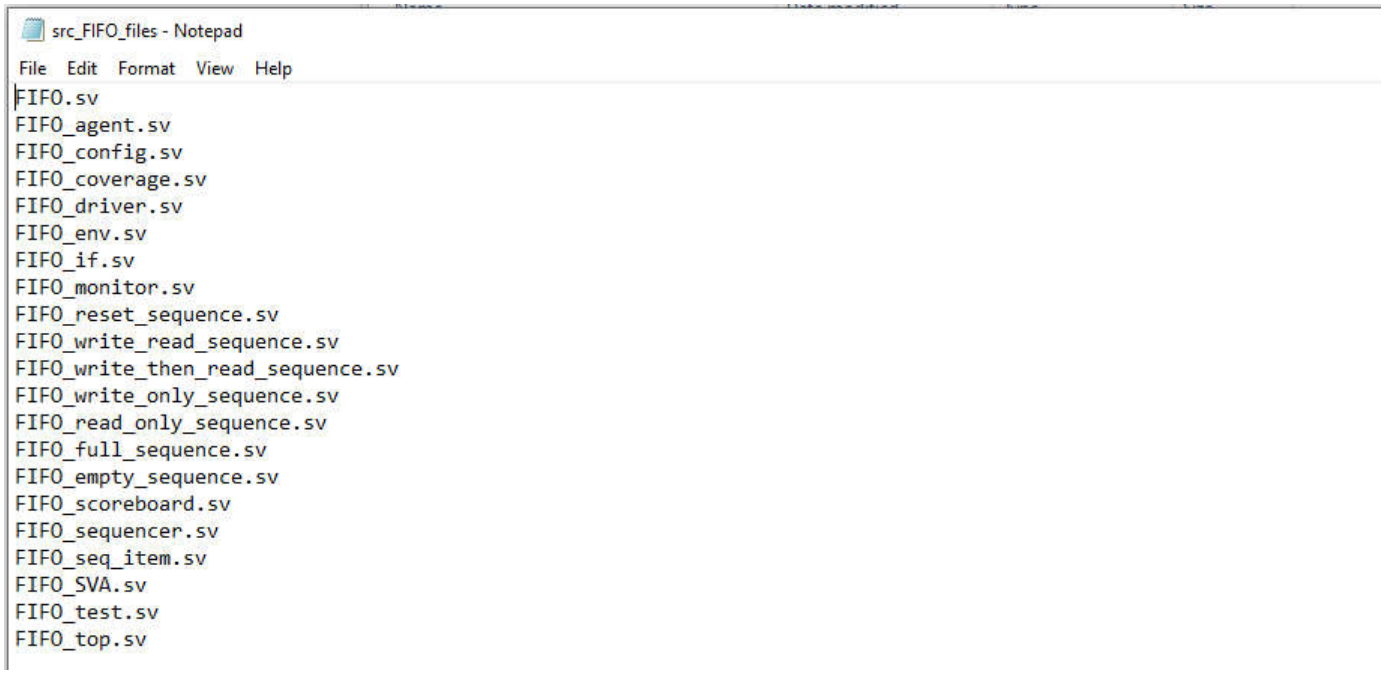
- Branch coverage

- Statement coverage

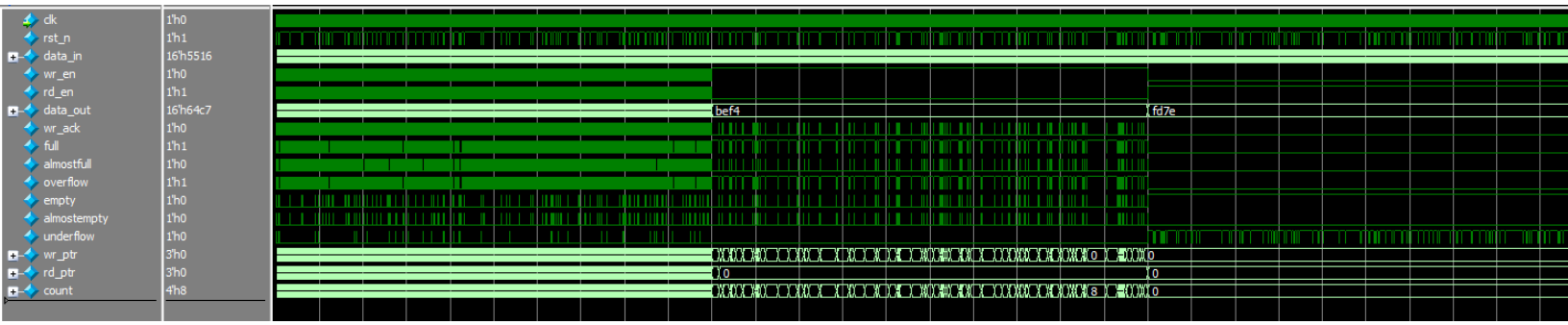
VIII. Do File and source files

```
vlib work
vlog -f src_FIFO_files.txt -mfcu +define+SIM +cover
vsim -voptargs=+acc work.FIFO_top -classdebug -uvmcontrol=all -cover
add wave /FIFO_top/f_if/*
coverage save FIFO_top.ucdb -onexit
run -all

quit -sim
vcover report FIFO_top.ucdb -details -annotate -all -output FIFO_coverage_rpt.txt
```



IX. Full wave of FIFO



X. FIFO Coverage report

1. Code coverage

- Branch coverage

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

================================Branch Details================================

Branch Coverage for instance /FIFO_top/DUT
```

- Statement coverage

```
  Statement Coverage:
      Enabled Coverage              Bins      Hits    Misses  Coverage
      ----------------              ----      ----    ------  --------
      Statements                      27        27         0   100.00%

  ================================Statement Details================================

  Statement Coverage for instance /FIFO_top/DUT --
```

- Toggle coverage

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

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

Toggle Coverage for instance /FIFO_top/f_if --
```

2. Assertions coverage

➢ Assertions passed

```
================================================================================
=== Instance: /FIFO_top/DUT/FIFO_SVA_INSTA
=== Design Unit: work.FIFO_SVA
================================================================================

Assertion Coverage:
    Assertions                      15        15         0   100.00%
--------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------
/FIFO_top/DUT/FIFO_SVA_INSTA/reset_assert
                     FIFO_SVA.sv(6)                     0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/full_flag_assert
                     FIFO_SVA.sv(14)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/almostfull_flag_assert
                     FIFO_SVA.sv(19)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/empty_flag_assert
                     FIFO_SVA.sv(24)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/almostempty_flag_assert
                     FIFO_SVA.sv(29)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/overflow_flag_assert
                     FIFO_SVA.sv(39)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/underflow_flag_assert
                     FIFO_SVA.sv(45)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_high_flag_assert
                     FIFO_SVA.sv(51)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_low_flag_assert
                     FIFO_SVA.sv(57)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/write_op_assert
                     FIFO_SVA.sv(63)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/read_op_assert
                     FIFO_SVA.sv(69)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/write_pri_assert
                     FIFO_SVA.sv(75)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/read_pri_assert
                     FIFO_SVA.sv(81)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/read_ptr_assert
                     FIFO_SVA.sv(87)                    0          1
/FIFO_top/DUT/FIFO_SVA_INSTA/write_ptr_assert
                     FIFO_SVA.sv(93)                    0          1
Branch Coverage:
```

➢ Cover directives

```
Directive Coverage:
    Directives                      15        15         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/DUT/FIFO_SVA_INSTA/reset_cover FIFO_SVA Verilog  SVA  FIFO_SVA.sv(7)   301 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/full_flag_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(15) 1568 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/almostfull_flag_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(20) 1858 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/empty_flag_acover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(25)  320 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/almostempty_flag_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(30)  244 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/overflow_flag_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(40) 12391 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/underflow_flag_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(46) 9842 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_high_flag_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(52) 4277 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/wr_ack_low_flag_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(58) 12391 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/write_op_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(64) 3266 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/read_op_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(70)  904 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/write_pri_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(76)  963 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/read_pri_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(82)   32 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/read_ptr_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(88) 2846 Covered
/FIFO_top/DUT/FIFO_SVA_INSTA/write_ptr_cover
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(94) 4277 Covered
Statement Coverage:
```

3. Functional coverage

```
Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses        23        na        na        na
            Covergroup Bins        70        70         0   100.00%
----------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status
----------------------------------------------------------------------------------------------------------------
 TYPE /FIFO_coverage_pkg/FIFO_coverage/FIFO_cvgr    100.00%        100          -    Covered
    covered/total bins:                                  70         70          -
    missing/total bins:                                   0         70          -
    % Hit:                                           100.00%        100          -
    Coverpoint wr_en_cp                              100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    13042          1          -    Covered
        bin auto[1]                                    17002          1          -    Covered
    Coverpoint rd_en_cp                              100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    17078          1          -    Covered
        bin auto[1]                                    12966          1          -    Covered
    Coverpoint full_cp                               100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    15911          1          -    Covered
        bin auto[1]                                    14133          1          -    Covered
    Coverpoint empty_cp                              100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    19780          1          -    Covered
        bin auto[1]                                    10264          1          -    Covered
    Coverpoint almostfull_cp                         100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    26983          1          -    Covered
        bin auto[1]                                     3061          1          -    Covered
    Coverpoint almostempty_cp                        100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    29712          1          -    Covered
        bin auto[1]                                      332          1          -    Covered
    Coverpoint overflow_cp                           100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    16683          1          -    Covered
        bin auto[1]                                    13361          1          -    Covered
    Coverpoint underflow_cp                          100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    20089          1          -    Covered
        bin auto[1]                                     9955          1          -    Covered
    Coverpoint wr_ack_cp                             100.00%        100          -    Covered
        covered/total bins:                                2          2          -
        missing/total bins:                                0          2          -
        % Hit:                                       100.00%        100          -
        bin auto[0]                                    25720          1          -    Covered
        bin auto[1]                                     4324          1          -    Covered
    Cross wr_full                                    100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
        Auto, Default and User Defined Bins:
            bin <auto[1],auto[1]>                      13104          1          -    Covered
            bin <auto[0],auto[1]>                       1029          1          -    Covered
            bin <auto[1],auto[0]>                       3898          1          -    Covered
            bin <auto[0],auto[0]>                      12013          1          -    Covered
    Cross wr_empty                                   100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
        Auto, Default and User Defined Bins:
            bin <auto[1],auto[1]>                        170          1          -    Covered
            bin <auto[0],auto[1]>                      10094          1          -    Covered
            bin <auto[1],auto[0]>                      16832          1          -    Covered
            bin <auto[0],auto[0]>                       2948          1          -    Covered
    Cross wr_almostfull                              100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
        Auto, Default and User Defined Bins:
            bin <auto[1],auto[1]>                       1993          1          -    Covered
            bin <auto[0],auto[1]>                       1068          1          -    Covered
            bin <auto[1],auto[0]>                      15009          1          -    Covered
            bin <auto[0],auto[0]>                      11974          1          -    Covered
```

Cross wr_almostempty	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	262	1	-	Covered
bin <auto[0],auto[1]>	70	1	-	Covered
bin <auto[1],auto[0]>	16740	1	-	Covered
bin <auto[0],auto[0]>	12972	1	-	Covered
Cross wr_overflow	100.00%	100	-	Covered
covered/total bins:	3	3	-	
missing/total bins:	0	3	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	13361	1	-	Covered
bin <auto[1],auto[0]>	3641	1	-	Covered
bin <auto[0],auto[0]>	13042	1	-	Covered
Illegal and Ignore Bins:				
ignore_bin wr_en0_overflow1	0		-	ZERO
Cross wr_underflow	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	39	1	-	Covered
bin <auto[0],auto[1]>	9916	1	-	Covered
bin <auto[1],auto[0]>	16963	1	-	Covered
bin <auto[0],auto[0]>	3126	1	-	Covered
Cross wr_wr_ack	100.00%	100	-	Covered
covered/total bins:	3	3	-	
missing/total bins:	0	3	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	4324	1	-	Covered
bin <auto[1],auto[0]>	12678	1	-	Covered
bin <auto[0],auto[0]>	13042	1	-	Covered
Illegal and Ignore Bins:				
ignore_bin wr_en0_wr_ack1	0		-	ZERO
Cross rd_full	100.00%	100	-	Covered
covered/total bins:	3	3	-	
missing/total bins:	0	3	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[0]>	12966	1	-	Covered
bin <auto[0],auto[1]>	14133	1	-	Covered
bin <auto[0],auto[0]>	2945	1	-	Covered
Illegal and Ignore Bins:				
ignore_bin rd_en1_full1	0		-	ZERO

Cross rd_empty	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	10071	1	-	Covered
bin <auto[0],auto[1]>	193	1	-	Covered
bin <auto[1],auto[0]>	2895	1	-	Covered
bin <auto[0],auto[0]>	16885	1	-	Covered
Cross rd_almostfull	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	2029	1	-	Covered
bin <auto[0],auto[1]>	1032	1	-	Covered
bin <auto[1],auto[0]>	10937	1	-	Covered
bin <auto[0],auto[0]>	16046	1	-	Covered
Cross rd_almostempty	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	104	1	-	Covered
bin <auto[0],auto[1]>	228	1	-	Covered
bin <auto[1],auto[0]>	12862	1	-	Covered
bin <auto[0],auto[0]>	16850	1	-	Covered
Cross rd_overflow	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	1230	1	-	Covered
bin <auto[0],auto[1]>	12131	1	-	Covered
bin <auto[1],auto[0]>	11736	1	-	Covered
bin <auto[0],auto[0]>	4947	1	-	Covered
Cross rd_underflow	100.00%	100	-	Covered
covered/total bins:	3	3	-	
missing/total bins:	0	3	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	9955	1	-	Covered
bin <auto[1],auto[0]>	3011	1	-	Covered
bin <auto[0],auto[0]>	17078	1	-	Covered
Illegal and Ignore Bins:				
ignore_bin rd_en0_underflow1	0		-	ZERO
Cross rd_wr_ack	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1]>	1029	1	-	Covered
bin <auto[0],auto[1]>	3295	1	-	Covered
bin <auto[1],auto[0]>	11937	1	-	Covered
bin <auto[0],auto[0]>	13783	1	-	Covered

Statement Coverage:

XI. Assertions table

Feature	Assertion
Whenever the reset asserted, count and pointers reset to zero	if(!f_if.rst_n) begin reset_assert: assert final(!DUT.wr_ptr CC !DUT.rd_ptr CC !DUT.count);
Whenever count is equal to fifo_depth, FIFO is full	if(f_if.rst_n CC (DUT.count== f_if.FIFO_DEPTH)) begin full_flag_assert: assert final(f_if.full CC !f_if.enpty CC !f_if.alnostenpty CC !f_if.alnostfull);
Whenever count is equal to fifo_depth – 1, FIFO is alnost full	if(f_if.rst_n CC (DUT.count== f_if.FIFO_DEPTH-1)) begin alnostfull_flag_assert: assert final(f_if.alnostfull CC !f_if.enpty CC !f_if.alnostenpty CC !f_if.full);
Whenever count is equal to zero, FIFO is enpty	if(f_if.rst_n CC (DUT.count== 0)) begin enpty_flag_assert: assert final(f_if.enpty CC !f_if.alnostenpty CC !f_if.full CC !f_if.alnostfull);
Whenever count is equal to 1, FIFO is alnost enpty	if(f_if.rst_n CC (DUT.count== 1)) begin alnostenpty_flag_assert: assert final(f_if.alnostenpty CC !f_if.enpty CC !f_if.full CC !f_if.alnostfull);
Whenever FIFO is full and wr_en is high, overflow occurs	@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.full CC f_if.wr_en) |=» f_if.overflow;
Whenever FIFO is enpty and rd_en is high, underflow occurs	@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.enpty CC f_if.rd_en) |=» f_if.underflow;
Whenever FIFO is not full and wr_en is high, wr_ack is high	@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en CC DUT.count" f_if.FIFO_DEPTH) |=» f_if.wr_ack;

Whenever FIFO is full and wr_en is high, wr_ack is zero	@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en CC f_if.full) |=» !f_if.wr_ack;
Whenever wr_en is high, rd_en is low and FIFO is not full, write operation occurs	@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en CC !f_if.rd_en CC !f_if.full) |=» $past(DUT.count+ 1'b1);
Whenever wr_en is low, rd_en is high and FIFO is not enpty, read operation occurs	@(posedge f_if.clk) disable iff(!f_if.rst_n) (!f_if.wr_en CC f_if.rd_en CC !f_if.enpty) |=» (DUT.count+ 1'b1);
Whenever wr_en is high, rd_en is high and FIFO is not enpty, read operation takes place	@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en CC f_if.rd_en CC f_if.full) |=» (DUT.count+ 1'b1);
Whenever wr_en is high, rd_en is high and FIFO is enpty, write operation takes place	@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en CC f_if.rd_en CC f_if.enpty) |=» $past(DUT.count+ 1'b1);
Whenever rd_en is high and FIFO is not enpty, Read pointer increases	@(posedge f_if.clk) disable iff (!f_if.rst_n) (f_if.rd_en CC (DUT.count != 0)) |=» (DUT.rd_ptr == ($past(DUT.rd_ptr) + 1) % f_if.FIFO_DEPTH);
Whenever wr_en is high and FIFO is not full, Write pointer increases	@(posedge f_if.clk) disable iff (!f_if.rst_n) (f_if.wr_en CC (DUT.count ” f_if.FIFO_DEPTH)) |=» (DUT.wr_ptr == ($past(DUT.wr_ptr) + 1) % f_if.FIFO_DEPTH);