Synchronous FIFO
UVM Verification

By: Yousef Gamal

Contents

1. Overview

The Synchronous FIFO design in this project is a First-In-First-Out (FIFO) memory buffer. It allows data to be written into a queue and read from it in the same sequential order, ensuring smooth communication between processes operating on the same clock domain.

Key Features:

• FIFO_WIDTH: Defines the width of data input and output buses, as well as the memory word width (default: 16 bits).

• FIFO_DEPTH: Determines the depth of the FIFO, representing the number of data entries that can be stored (default: 8 entries).

FIFO Signals:

Signal	Direction	Description
data_in	Input	Write Data: The input data bus used when writing to the FIFO.
wr_en	Input	Write Enable: Enables data writing when the FIFO is not full.
rd_en	Input	Read Enable: Enables data reading when the FIFO is not empty.
clk	Input	Clock: The clock signal for synchronizing operations.
rst_n	Input	Reset: Active-low asynchronous reset signal.
data_out	Output	Read Data: The data output bus when reading from the FIFO.
full	Output	Full Flag: Indicates the FIFO is full and cannot accept more data.
almostfull	Output	Almost Full: Indicates one more write can be performed before full.
empty	Output	Empty Flag: Indicates the FIFO is empty.
almostempty	Output	Almost Empty: Indicates one more read can be performed before empty.
overflow	Output	Overflow: Indicates a rejected write operation due to full FIFO.
underflow	Output	Underflow: Indicates a rejected read operation due to empty FIFO.
wr_ack	Output	Write Acknowledge: Indicates a successful write operation.

2. UVM Structure

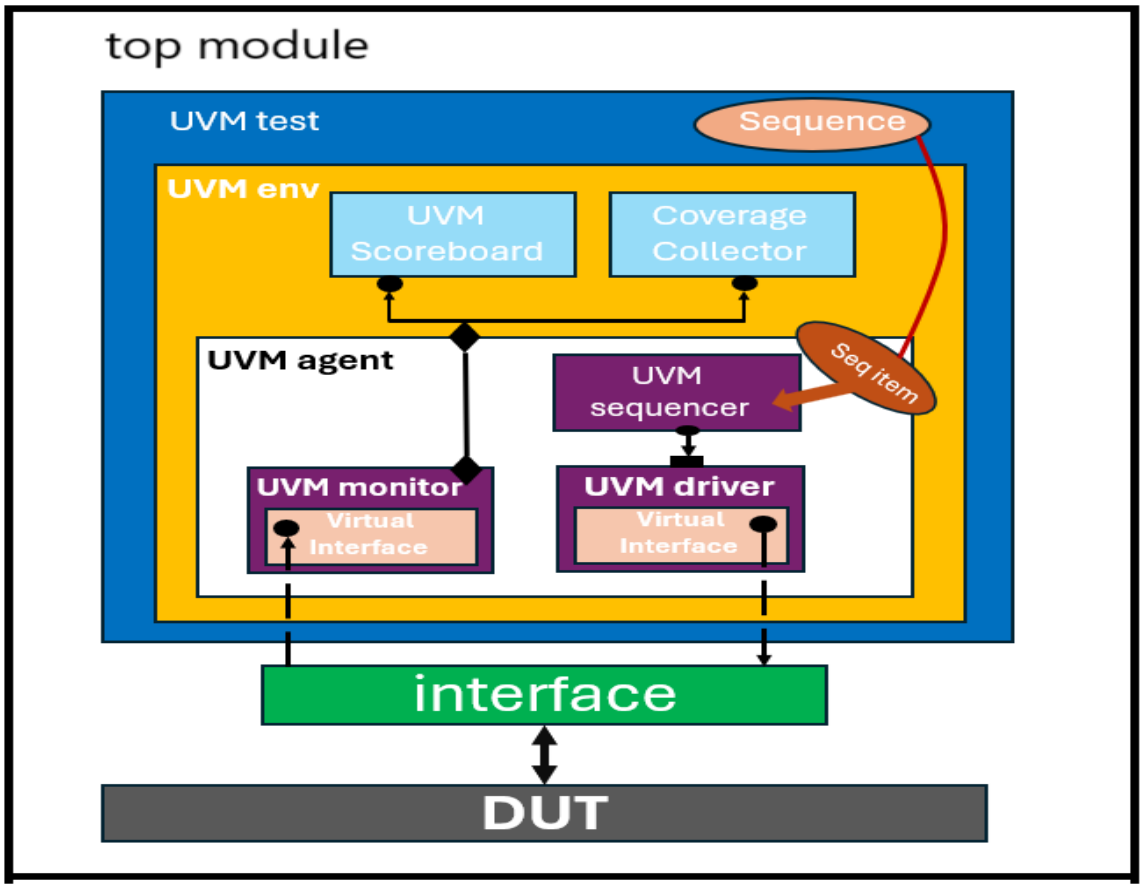


Figure 1: UVM structure

3. Flow illustration

Illustrated of the flow shown in Figure 1:

1. **Top module:**
   - The execution starts here, where the UVM test is invoked. This module represents the root of the hierarchy
   - This module is responsible for generating the clock, instantiating the DUT, importing the test packages, binding the assertions, setting the virtual interface in the UVM configuration database, and finally, running the test

2. **UVM Test:**
   - Building the environment and sequences
   - Retrieves the virtual interface from the database and sets the configuration object in the database
   - Start sequences on the sequencer

3. **UVM Sequence:**
   - The sequence is considered the core stimulus of the verification environment
   - Generates several sequence items

4. **UVM Sequence item:**
   - Contains the data to communicate with DUT
   - Random stimuli are generated based on the constraints that are defined

5. **UVM Environment:**
   - Builds and connect the UVM agent and analysis components which are the scoreboard and the coverage collector

6. **UVM Agent:**
   - Builds the monitor, sequencer, and driver
   - Connect the driver and the sequencer
   - Retrieve the configuration object from the database and assign its virtual interface to the ones within the driver and monitor

7. **UVM Sequencer:**
   - It is a FIFO to store sequence items (transactions) that are sent from the sequence and deliver them to the driver

8. **UVM Driver:**
   - Pulls the sequence items from the sequencer
   - Assign the input signals from the sequence items to the virtual interface that directly interacts with the DUT

9. **UVM Monitor:**
   - Gather all the signals of the DUT from the interface and translate these signals into a sequence item.
   - Broadcast this sequence item to the analysis components to do their tasks

10. **UVM analysis components:**
    - UVM Scoreboard: receives a sequence item from the monitor and compares the outputs with the reference model to check the correctness of the functionality

    - Functional coverage collector: receives a sequence item from the monitor and samples the data for functional coverage

4. Verification plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_2	When RD_EN is high and the FIFO is not empty, the FIFO will deliver the output to the data_out port	parameterized Randomization with default values for RD_EN to be High 30% of simulation time	Cross Coverage for all possibilities for WR_EN and Empty Flag and RD_EN to make sure that we cover all cases	Self-checking in monitor module
FIFO_3	When reset is asserted the output Flag full = 0 , empty = 1 , almostempty = 0 and almostfull = 0	Directed at the start of the testbench and randomized during the simulation to be most of the time off	--	Immediate assertion to verify these output flags
FIFO_4	When The count equal to Zero the output Flag full = 0 empty = 1 , almostempty = 0 and almostfull = 0	--	--	Immediate assertion to verify these output flags
FIFO_5	When The count equal to FIFO Depth the output Flag full = 1 empty = 0 , almostempty = 0 and almostfull = 0	--	--	Immediate assertion to verify these output flags
FIFO_6	When The count equal to FIFO Depth minus one the output Flag full = 0 , empty = 0 , almostempty = 0 and almostfull = 1	--	Cross Coverage for all possibilities for WR_EN and almostfull Flag and RD_EN to make sure that we cover all cases	Immediate assertion to verify these output flags
FIFO_7	When The count equal to one the output Flag full = 0 , empty =0 , almostempty = 1 and almostfull = 0	--	Cross Coverage for all possibilities for WR_EN and almostempty Flag and RD_EN to make sure that we cover all cases	Immediate assertion to verify these output flags
FIFO_8	When RD_EN is high, and Count is not zero, and RD_ptr is less than the maximum value, the RD_ptr should increment	--	--	Concurrent assertion to check the Read pointer
FIFO_9	When WR_EN is high, and Count is less than the FIFO depth, and WR_ptr is less than the maximum value, the WR_ptr should increment	--	--	Concurrent assertion to check the Write pointer
FIFO_10	When WR_EN is high, and FIFO is Full the Overflow Flag should be High	--	Cross Coverage for all possibilities for WR_EN and Overflow Flag and RD_EN to make sure that we cover all cases	Concurrent assertion to check the overflow flag
FIFO_11	When RD_EN is high, and FIFO is Empty the Underflow Flag should be High	--	Cross Coverage for all possibilities for WR_EN and underflow Flag and RD_EN to make sure that we cover all cases	Concurrent assertion to check the underflow flag
FIFO_12	When WR_EN is high, and FIFO is Not full the WR_ack Flag should be High	--	Cross Coverage for all possibilities for WR_EN and underflow Flag and RD_EN to make sure that we cover all cases	Concurrent assertion to check the WR_ack flag
FIFO_13	When WR_EN is high, and FIFO is full the WR_ack Flag should be LOW	--	Cross Coverage for all possibilities for WR_EN and WR_ack Flag and RD_EN to make sure that we cover all cases	Concurrent assertion to check the WR_ack flag

5. Bugs

Bug	Why
overflow <= 0;	It is missed so it is added to cleared with reset
wr_ack <= 0;	It is missed so it is added to cleared with reset
underflow <= 0;	It is missed so it is added to cleared with reset
overflow <= 0;	Added to be cleared if we already wrote in FIFO
underflow <= 0;	Added to be cleared if we already read from FIFO
almostfull = (count ==FIFO_DEPTH-1)? 1 : 0;	Adjusted to be FIFO_DEPTH-1 instead of FIFO_DEPTH-2

`else`
`if (empty && rd_en)`
`underflow <= 1;`
`else`
`underflow <= 0;`
	Added to be sequential output not combinational

6. Assertions used

FIFO Assertions Table

Feature	Assertions
When the reset is asserted, the flags will be Full = 0, Empty = 1, AlmostEmpty = 0, AlmostFull = 0	assert final (!full && empty && !almostempty && !almostfull)
When the count equals zero, the flags will be Full = 0, Empty = 1, AlmostEmpty = 0, AlmostFull = 0	assert final (!full && empty && !almostempty && !almostfull)
When the count equals FIFO depth, the flags will be Full = 1, Empty = 0, AlmostEmpty = 0, AlmostFull = 0	assert (full && !empty && !almostempty && !almostfull)
When the count equals (FIFO depth - 1), the flags will be Full = 0, Empty = 0, AlmostEmpty = 0, AlmostFull = 1	assert (!full && !empty && !almostempty && almostfull)
When the count equals 1, the flags will be Full = 0, Empty = 0, AlmostEmpty = 1, AlmostFull = 0	assert (!full && !empty && almostempty && !almostfull)
When write is enabled and FIFO is not full, acknowledgment will be sent	@(posedge clk) (wr_en && ! full) |=> (wr_ack) ;
When write is enabled and FIFO is full, acknowledgment will be low	@(posedge clk) (wr_en && full) |=> (!wr_ack) ;
When write is enabled and FIFO is full, overflow will occur	@(posedge clk) (wr_en && full) |=> (overflow) ;
When read is enabled and FIFO is empty, underflow will occur	@(posedge clk) (rd_en && empty) |=> (underflow) ;
When read is enabled and the FIFO is not empty, the read pointer will increment by 1, wrapping around when it reaches the FIFO depth.	@(posedge clk) (rd_en && (count != 0) && rd_ptr < 7) |=> ((rd_ptr == ($past(rd_ptr) + 1)) % FIFO_DEPTH);
When write is enabled and the FIFO is not full, the write pointer will increment by 1, wrapping around when it reaches the FIFO depth.	@(posedge clk) (wr_en && (count < FIFO_DEPTH) && wr_ptr < 7) |=> ((wr_ptr == ($past(wr_ptr) + 1)) % FIFO_DEPTH);

7. Code

7.1. Updated Design RTL

```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: FIFO Design
//
////////////////////////////////////////////////////////////////////////////////
module FIFO #(parameter FIFO_WIDTH = 16 , FIFO_DEPTH = 8 )
(
   input clk,
   input rst_n,
   input wr_en,
   input rd_en,
   input [FIFO_WIDTH-1:0] data_in,
   output full,
   output empty,
   output almostfull,
   output almostempty,
   output reg wr_ack,
   output reg overflow,
   output reg underflow,
   output reg [FIFO_WIDTH-1:0] data_out
);

/*------------------------------internal signals--------------------------------*/
localparam max_fifo_addr = $clog2(FIFO_DEPTH);
reg [FIFO_WIDTH-1 : 0] mem [FIFO_DEPTH-1 : 0];
reg [max_fifo_addr-1 : 0] wr_ptr, rd_ptr;
reg [max_fifo_addr : 0] count;


always @(posedge clk or negedge rst_n)
begin
    if (!rst_n)
        begin
            wr_ptr <= 0;
            overflow <= 0; /* added to cleared with reset */
            wr_ack <= 0;   /* added to cleared with reset */
        end
    else if (wr_en && count < FIFO_DEPTH)
        begin
            mem[wr_ptr] <= data_in;
            wr_ack <= 1;
            wr_ptr <= wr_ptr + 1;
            overflow <= 0; /* added to be cleared if we already write in FIFO */
        end
    else
        begin
```

```
            wr_ack <= 0;
            if (full & wr_en)
                overflow <= 1;
            else
                overflow <= 0;
        end
end

always @(posedge clk or negedge rst_n)
begin
    if (!rst_n)
        begin
            rd_ptr <= 0;
            underflow <= 0; /* added to cleared with reset */
        end
    else if (rd_en && (count != 0) )
        begin
            data_out <= mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
            underflow <= 0; /* added to be cleared if we already read from FIFO */
        end
    else  /* added to be sequential output not combinational */
        if (empty && rd_en)
            underflow <= 1;
        else
            underflow <= 0;
end
// always block specialized for counter signal
always @(posedge clk or negedge rst_n)
begin
    if (!rst_n)
        count <= 0;
    else if (wr_en && rd_en)
        begin
            if (empty)
                count <= count + 1;  /* Prioritize write if FIFO is empty */

            else if (full)
                count <= count - 1;  /* Prioritize read if FIFO is full */
        end
    else if( wr_en && !full )
                count <= count + 1;

    else if ( rd_en && !empty )
                count <= count - 1;

    end

assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; /* adjusted to be FIFO_DEPTH-1 */
assign almostempty = (count == 1)? 1 : 0;

endmodule
```

7.2. SVA module

```
module SVA #(parameter FIFO_WIDTH = 16 , FIFO_DEPTH = 8 )
(
   input logic clk,
   input logic rst_n,
   input logic wr_en,
   input logic rd_en,
   input logic wr_ack,
   input logic full,
   input logic empty,
   input logic overflow,
   input logic underflow,
   input logic almostfull,
   input logic almostempty,
   input logic [2:0]wr_ptr,
   input logic [2:0]rd_ptr,
   input logic [FIFO_WIDTH-1:0] data_in,
   input logic [FIFO_WIDTH-1:0] data_out,
   input logic [$clog2(FIFO_DEPTH) : 0] count

);


always_comb
begin
if( !rst_n )
begin
    assert_reset_falgs:  assert final ( !full && empty && !almostempty && !almostfull  ) ;
    cover_reset_falgs :  cover  ( !full && empty && !almostempty && !almostfull  ) ;
end
else
begin
        if( count == 0 )
        begin
            assert_empty: assert (!full && empty && !almostempty && !almostfull );
            cover_empty : cover  (!full && empty && !almostempty && !almostfull );
        end

        else if( count == FIFO_DEPTH )
        begin
            assert_full: assert (full && !empty && !almostempty && !almostfull ) ;
            cover_full : cover  (full && !empty && !almostempty && !almostfull);
        end
        else if( count == (FIFO_DEPTH - 1'b1) )
        begin
            assert_almostfull: assert (!full && !empty && !almostempty && almostfull) ;
            cover_almostfull : cover  (!full && !empty && !almostempty && almostfull) ;
        end



 else if( count == 1'b1 )
        begin
```

```
            assert_almostempty: assert (!full && !empty && almostempty && !almostfull) ;
            cover_almostempty : cover  (!full && !empty && almostempty && !almostfull) ;
        end
end
end

property RD_ptr;
    @(posedge clk) disable iff(!rst_n) (rd_en && (count != 0) && rd_ptr < 7 ) |=> (
(rd_ptr == ($past(rd_ptr) + 1)) % FIFO_DEPTH );
endproperty

property WR_ptr;
    @(posedge clk) disable iff(!rst_n) (wr_en && (count < FIFO_DEPTH) && wr_ptr < 7) |=>
((wr_ptr == ($past(wr_ptr) + 1)) % FIFO_DEPTH);
endproperty

property prop_overflow ;
    @(posedge clk) disable iff (!rst_n) (wr_en && full) |=> (overflow) ;
endproperty

property prop_underflow ;
    @(posedge clk) disable iff (!rst_n) (rd_en && empty) |=> (underflow) ;
endproperty

property wr_ack_1 ;
    @(posedge clk) disable iff (!rst_n) (wr_en && ! full) |=> (wr_ack) ;
endproperty

property wr_ack_0 ;
    @(posedge clk) disable iff (!rst_n) (wr_en && full) |=> (!wr_ack) ;
endproperty

RD_PTR_assert : assert property (RD_ptr);
RD_PTR_cover  : cover  property (RD_ptr);

WR_PTR_assert : assert property (WR_ptr);
WR_PTR_cover  : cover  property (WR_ptr);


assert_overflow : assert property (prop_overflow) ;
cover_overflow  : cover  property (prop_overflow) ;

assert_underflow : assert property (prop_underflow) ;
cover_underflow  : cover  property (prop_underflow) ;

assert_wr_ack_1 : assert property (wr_ack_1) ;
cover_wr_ack_1  : cover  property (wr_ack_1) ;

assert_wr_ack_0 : assert property (wr_ack_0) ;
cover_wr_ack_0  : cover  property (wr_ack_0) ;

endmodule
```

7.3. Interface

```
interface FIFO_if (clk) ;

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

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

endinterface
```

7.4. Configuration object

```
package config_obj_pkg;

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

class FIFO_config_obj extends uvm_object;
    `uvm_object_utils(FIFO_config_obj)

    virtual FIFO_if FIFOif;

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

7.5. Top module

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

module top ();
    bit clk; // Clock signal
    // Initial block to generate the clock signal
    initial
    begin
        clk = 0; // Initialize clock to 0
        forever #1 clk = ~clk; // Toggle clock every 1 time unit
```

```
    end
    // Instantiate the FIFO interface
    FIFO_if FIFOif (clk);

    // Instantiate the FIFO design under test (DUT)
    FIFO DUT (
                .clk         (  clk              ),
                .rst_n       (FIFOif.rst_n       ),
                .wr_en       (FIFOif.wr_en       ),
                .rd_en       (FIFOif.rd_en       ),
                .full        (FIFOif.full        ),
                .empty       (FIFOif.empty       ),
                .wr_ack      (FIFOif.wr_ack      ),
                .data_in     (FIFOif.data_in     ),
                .data_out    (FIFOif.data_out    ),
                .overflow    (FIFOif.overflow    ),
                .underflow   (FIFOif.underflow   ),
                .almostfull  (FIFOif.almostfull  ),
                .almostempty (FIFOif.almostempty )
             );

    bind FIFO SVA SVA_inst  (
                                .clk         (  clk               ),
                                .rst_n       (FIFOif.rst_n        ),
                                .wr_en       (FIFOif.wr_en        ),
                                .rd_en       (FIFOif.rd_en        ),
                                .full        (FIFOif.full         ),
                                .empty       (FIFOif.empty        ),
                                .wr_ack      (FIFOif.wr_ack       ),
                                .data_in     (FIFOif.data_in      ),
                                .data_out    (FIFOif.data_out     ),
                                .overflow    (FIFOif.overflow     ),
                                .underflow   (FIFOif.underflow    ),
                                .almostfull  (FIFOif.almostfull   ),
                                .almostempty (FIFOif.almostempty  ),
                                .count       (DUT.count           ),
                                .wr_ptr      (DUT.wr_ptr          ),
                                .rd_ptr      (DUT.rd_ptr          )
                            );
    initial
    begin
        uvm_config_db #(virtual FIFO_if)::set(null , "uvm_test_top" , "FIFOif" , FIFOif);
        run_test("FIFO_test");
    end

endmodule
```

7.6. Test Package

```
package test_pkg ;

    import uvm_pkg::* ;
    import env_pkg::* ;
    import sequence_pkg::* ;
    import config_obj_pkg::* ;

    `include "uvm_macros.svh"

class FIFO_test extends uvm_test ;
`uvm_component_utils(FIFO_test)

FIFO_env  env ;
FIFO_config_obj cnf_obj   ;
FIFO_reset_seq  rst_seq   ;
FIFO_write_seq  wr_seq    ;
FIFO_read_seq   rd_seq    ;
FIFO_rd_wr_seq  rd_wr_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);
    cnf_obj   = FIFO_config_obj::type_id::create("cnf_obj");
    env       = FIFO_env::type_id::create("env", this);
    rst_seq   = FIFO_reset_seq::type_id::create("rst_seq" , this);
    wr_seq    = FIFO_write_seq::type_id::create("wr_seq", this);
    rd_seq    = FIFO_read_seq::type_id::create("rd_seq" , this);
    rd_wr_seq = FIFO_rd_wr_seq::type_id::create("rd_wr_seq" , this);

    if(!uvm_config_db #(virtual FIFO_if)::get(this , "" , "FIFOif" , cnf_obj.FIFOif ) )
    `uvm_fatal("build_phase" , "Test - couldn't get the interface")

    uvm_config_db#(FIFO_config_obj)::set(this , "*" , "CFG" , cnf_obj);

endfunction

task run_phase( uvm_phase phase );
    super.run_phase(phase);
    phase.raise_objection(this);

    `uvm_info("run-phase" , "reset sequence asserted"   , UVM_LOW);
    rst_seq.start(env.agt.sequencer);
    `uvm_info("run-phase" , "reset sequence deasserted" , UVM_LOW);

    `uvm_info("run-phase" , "write sequence asserted"   , UVM_LOW);
     wr_seq.start(env.agt.sequencer);
    `uvm_info("run-phase" , "write sequence deasserted" , UVM_LOW);

    `uvm_info("run-phase" , "read sequence asserted"   , UVM_LOW);
```

```
     rd_seq.start(env.agt.sequencer);
    `uvm_info("run-phase" , "read sequence deasserted" , UVM_LOW);

    `uvm_info("run-phase" , "rd_wr sequence asserted"   , UVM_LOW);
     rd_wr_seq.start(env.agt.sequencer);
    `uvm_info("run-phase" , "rd_wr sequence deasserted" , UVM_LOW);

    phase.drop_objection(this) ;
endtask
endclass
endpackage
```

7.7. Sequence Package

```
package sequence_pkg ;

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

/*----------------------------reset sequence-----------------------------*/
class FIFO_reset_seq extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_reset_seq)

FIFO_seq_item seq_item;

function new(string name = "FIFO_reset_seq");
    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 ;
        finish_item(seq_item);
endtask
endclass


/*----------------------------Write only sequence----------------------------*/
class FIFO_write_seq extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_write_seq)

FIFO_seq_item seq_item;

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

task body( );
    repeat(500)
    begin
    seq_item = FIFO_seq_item::type_id::create("seq_item");
```

```
    seq_item.WR_EN_cons.constraint_mode(0); /* disable the constraint of write Enable */
    seq_item.RD_EN_cons.constraint_mode(0); /* disable the constraint of read  Enable */
    seq_item.read_only.constraint_mode(0);  /* disable the constraint of read only    */

    seq_item.rst_cons.constraint_mode(1);   /* Enable the constraint of reset         */
    seq_item.write_only.constraint_mode(1); /* Enable the constraint of write only    */

        start_item(seq_item);
        assert(seq_item.randomize());
        finish_item(seq_item);
    end
endtask
endclass

/*----------------------------read only sequence-----------------------------*/
class FIFO_read_seq extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_read_seq)

FIFO_seq_item seq_item;

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

task body();

    repeat(500)
    begin
    seq_item = FIFO_seq_item::type_id::create("seq_item");
    seq_item.WR_EN_cons.constraint_mode(0); /* disable the constraint of write Enable */
    seq_item.RD_EN_cons.constraint_mode(0); /* disable the constraint of read  Enable */
    seq_item.write_only.constraint_mode(0); /* disable the constraint of write only   */

    seq_item.rst_cons.constraint_mode(1);   /* Enable the constraint of reset         */
    seq_item.read_only.constraint_mode(1);  /* Enable the constraint of read only     */
        start_item(seq_item);
        assert(seq_item.randomize());
        finish_item(seq_item);
    end
endtask
endclass

/*----------------------------read write sequence----------------------------*/
class FIFO_rd_wr_seq extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_rd_wr_seq)

FIFO_seq_item seq_item;

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

task body();
```

```
    repeat(100000)
    begin
    seq_item = FIFO_seq_item::type_id::create("seq_item");
    seq_item.write_only.constraint_mode(0); /* disable the constraint of write only   */
    seq_item.read_only.constraint_mode(0);  /* disable the constraint of read only    */

    seq_item.rand_value( );                /* set the default values for randomization */
    seq_item.rst_cons.constraint_mode(1);  /* Enable the constraint of reset           */
    seq_item.WR_EN_cons.constraint_mode(1);/* Enable the constraint of write Enable    */
    seq_item.RD_EN_cons.constraint_mode(1);/* Enable the constraint of read  Enable    */
        start_item(seq_item);
        assert(seq_item.randomize());
        finish_item(seq_item);
    end
endtask
endclass
endpackage
```

7.8. Sequence item Package

```
package seq_item_pkg;

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

class FIFO_seq_item extends uvm_sequence_item ;
    `uvm_object_utils(FIFO_seq_item)

localparam FIFO_DEPTH = 8 ;
localparam FIFO_WIDTH = 16 ;
rand bit  rst_n, wr_en, rd_en ;
rand bit [FIFO_WIDTH-1:0] data_in;

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

int RD_EN_ON_DIST = 0 ;
int WR_EN_ON_DIST = 0 ;

        function void rand_value ( int RD_EN_ON_DIST = 30 , int WR_EN_ON_DIST = 70 ) ;
            this.WR_EN_ON_DIST = WR_EN_ON_DIST ;
            this.RD_EN_ON_DIST = RD_EN_ON_DIST ;
        endfunction

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





function string convert2string( );
```

```
return $sformatf("%s , rst_n: %0b , wr_en: %0b , rd_en: %0b , data_in: %0d , data_out: %0d
, full: %0b , almostfull: %0b , empty: %0b , almostempty: %0b , overflow: %0b , underflow:
%0b , wr_ack: %0b",
super.convert2string() , rst_n , wr_en , rd_en , data_in , data_out , full , almostfull ,
empty , almostempty , overflow , underflow , wr_ack );
endfunction

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

        /*-------------------- Constraints -------------------------*/
        constraint rst_cons   { rst_n dist {0:/5 , 1:/95 } ;}
        constraint WR_EN_cons { wr_en dist {0:/(100-WR_EN_ON_DIST) , 1:/WR_EN_ON_DIST } ;}
        constraint RD_EN_cons { rd_en dist {0:/(100-RD_EN_ON_DIST) , 1:/RD_EN_ON_DIST } ;}
        constraint write_only { rd_en == 0 ; wr_en == 1 ;}
        constraint read_only  { rd_en == 1 ; wr_en == 0 ;}

endclass
endpackage
```

7.9. Environment Package

```
package env_pkg ;

    import uvm_pkg::* ;
    import cvg_pkg::* ;
    import agent_pkg::* ;
    import scoreboard_pkg::* ;
    `include "uvm_macros.svh"

class FIFO_env extends uvm_env ;
`uvm_component_utils(FIFO_env)

FIFO_agent         agt ;
FIFO_scoreboard    sb  ;
FIFO_cvg_collector cvg ;


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);
    cvg = FIFO_cvg_collector::type_id::create("cvg", this);
endfunction



function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
```

```
    agt.agent_aport.connect(sb.sb_export);
    agt.agent_aport.connect(cvg.cvg_export);
endfunction

endclass
endpackage
```

7.10. Agent Package

```
package agent_pkg ;

    import uvm_pkg::*;
    import driver_pkg::*;
    import monitor_pkg::*;
    import sequencer_pkg::*;
    import config_obj_pkg::*;
    import seq_item_pkg::*;
    `include "uvm_macros.svh"
class FIFO_agent extends uvm_agent ;
`uvm_component_utils(FIFO_agent)

FIFO_driver driver ;
FIFO_monitor monitor ;
FIFO_Sequencer sequencer ;
FIFO_config_obj config_obj ;
uvm_analysis_port #(FIFO_seq_item) agent_aport ;
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_obj)::get(this , "" , "CFG" , config_obj)  )
        `uvm_fatal("build_phase" , "Agent - couldn't get the configuration object")

        driver = FIFO_driver::type_id::create("driver" , this);
        monitor = FIFO_monitor::type_id::create("monitor" , this);
        sequencer = FIFO_Sequencer::type_id::create("sequencer" , this);
        agent_aport = new("agent_aport" , this);
endfunction

function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        driver.FIFOif  = config_obj.FIFOif;
        monitor.FIFOif = config_obj.FIFOif;
        driver.seq_item_port.connect(sequencer.seq_item_export);
        monitor.monitor_aport.connect(agent_aport);
endfunction

endclass
endpackage
```

7.11. Driver Package

```
package driver_pkg ;

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

class FIFO_driver extends uvm_driver #(FIFO_seq_item) ;
`uvm_component_utils(FIFO_driver)

virtual FIFO_if FIFOif ;
FIFO_seq_item seq_item ;

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

    function void build_phase (uvm_phase phase);
            super.build_phase(phase);
    endfunction

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

7.12. Monitor Package

```
package monitor_pkg ;

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

class FIFO_monitor extends uvm_monitor  ;
`uvm_component_utils(FIFO_monitor)

virtual FIFO_if FIFOif ;
FIFO_seq_item seq_item ;
uvm_analysis_port #(FIFO_seq_item) monitor_aport ;

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);
        monitor_aport = new("monitor_aport" , this);
endfunction

task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever
        begin
            seq_item = FIFO_seq_item::type_id::create("seq_item" , this);
            @(negedge FIFOif.clk);
            seq_item.rst_n        = FIFOif.rst_n        ;
            seq_item.wr_en        = FIFOif.wr_en        ;
            seq_item.rd_en        = FIFOif.rd_en        ;
            seq_item.full         = FIFOif.full         ;
            seq_item.empty        = FIFOif.empty        ;
            seq_item.wr_ack       = FIFOif.wr_ack       ;
            seq_item.data_in      = FIFOif.data_in      ;
            seq_item.data_out     = FIFOif.data_out     ;
            seq_item.overflow     = FIFOif.overflow     ;
            seq_item.underflow    = FIFOif.underflow    ;
            seq_item.almostfull   = FIFOif.almostfull   ;
            seq_item.almostempty  = FIFOif.almostempty  ;
            monitor_aport.write(seq_item); /*broadcast the seq_item to analysis component */
        end
endtask

endclass
endpackage
```

7.13. Sequencer Package

```
package sequencer_pkg;

import uvm_pkg::*;
import seq_item_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
```

7.14. Scoreboard Package

```
package scoreboard_pkg ;
    import uvm_pkg::*;
    import seq_item_pkg::*;
    `include "uvm_macros.svh"

class FIFO_scoreboard extends uvm_scoreboard ;
`uvm_component_utils(FIFO_scoreboard)

FIFO_seq_item seq_item_sb ;
uvm_analysis_export #(FIFO_seq_item) sb_export ;
uvm_tlm_analysis_fifo #(FIFO_seq_item) sb_fifo;

/*---------------signals of reference model -----------------------------*/
localparam FIFO_DEPTH = 8;
localparam FIFO_WIDTH = 16;

logic [FIFO_WIDTH-1 : 0] data_out_ref ;
bit [$clog2(FIFO_DEPTH) : 0] counter ;
int My_ref_Queue[$] ;

    int error_count = 0;
    int correct_count = 0;


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_fifo   = new("sb_fifo"  , this);
    sb_export = new("sb_export", this);
endfunction

function void connect_phase(uvm_phase phase);
```

```
    super.connect_phase(phase);
    sb_export.connect(sb_fifo.analysis_export);
endfunction


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

        if(data_out_ref != seq_item_sb.data_out)
            begin
                error_count++;
                `uvm_error("run_phase" , $sformatf("transaction received : %s , output
expected = %0d " ,
                                                seq_item_sb.convert2string() , data_out_ref  )
)
                                                $stop ;
            end
        else
            begin
                correct_count++ ;
            end
    end
endtask


task golden_model(  FIFO_seq_item seq_item_sb );
fork
    begin // first thread -> write operation
        if (!seq_item_sb.rst_n)
            begin
                My_ref_Queue.delete() ;
                counter = 0 ;
            end
        else
             begin
                if ( seq_item_sb.wr_en && (counter < FIFO_DEPTH) )
                    begin
                        My_ref_Queue.push_front(seq_item_sb.data_in) ;
                    end
             end
    end

    begin // second thread -> read operation
        if (seq_item_sb.rst_n)
        begin
            if ( seq_item_sb.rd_en && (counter != 0) )
            begin
                data_out_ref = My_ref_Queue.pop_back() ;
            end
        end
```

```
    end

    begin //  third thread -> Counter updating
        if (!seq_item_sb.rst_n)
            counter = 0;
        else
            begin
                casex ({seq_item_sb.wr_en, seq_item_sb.rd_en, (counter == FIFO_DEPTH),
(counter == 0) })
                    4'b11_01: // Both write and read enabled, FIFO empty
                        counter = counter + 1; // Prioritize write if FIFO is empty

                    4'b11_10: // Both write and read enabled, FIFO full
                        counter = counter - 1; // Prioritize read if FIFO is full

                    4'b10_0x: // Write enabled, not full
                        counter = counter + 1;

                    4'b01_x0: // Read enabled, not empty
                        counter = counter - 1;
                 endcase
            end
    end
join
endtask

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

endclass
endpackage
```

7.15. Coverage Collector Package

```
package cvg_pkg ;

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

class FIFO_cvg_collector extends uvm_component ;
`uvm_component_utils(FIFO_cvg_collector)

FIFO_seq_item seq_item ;
uvm_analysis_export   #(FIFO_seq_item) cvg_export ;
uvm_tlm_analysis_fifo #(FIFO_seq_item) cvg_fifo   ;

covergroup  cvr_gp  ;
```

```
cross_coverage_almostfull  : cross  seq_item.wr_en, seq_item.rd_en, seq_item.almostfull  ;
cross_coverage_almostempty : cross  seq_item.wr_en, seq_item.rd_en, seq_item.almostempty ;

 /* whenever the RD_EN = 1 the full = 0 so we will ignore the bins of RD_EN = 1 and full = 1 */
cross_coverage_full        : cross  seq_item.wr_en, seq_item.rd_en, seq_item.full        ;

  /* whenever the WR_EN = 1 the empty = 0 so we will ignore the bins of WR_EN = 1 and
                                                                                  empty = 1 */
cross_coverage_empty       : cross  seq_item.wr_en, seq_item.rd_en, seq_item.empty       ;

  /* whenever the WR_EN = 0 the WR_ACK = 0 so we will ignore the bins of wr_en = 0 and
                                                                                wr_ack = 1 */
cross_coverage_wr_ack      : cross  seq_item.wr_en, seq_item.rd_en, seq_item.wr_ack      ;

  /* whenever the WR_EN = 0 the overflow = 0 as we not intend to write so ignore the bins
       of wr_en = 0 and overflow = 1 */
cross_coverage_overflow    : cross  seq_item.wr_en, seq_item.rd_en,
seq_item.overflow    ;

  /* whenever the RD_EN = 0 the underflow = 0 as we not intend to read so ignore the bins
       of rd_en = 0 and underflow = 1 */
cross_coverage_underflow   : cross  seq_item.wr_en, seq_item.rd_en, seq_item.underflow   ;

endgroup

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

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

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    cvg_export.connect(cvg_fifo.analysis_export);
endfunction

task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever
    begin
        cvg_fifo.get(seq_item);
        cvr_gp.sample();
    end
endtask

endclass
endpackage
```

8. Do file

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
coverage save top.ucdb -onexit -du FIFO
run 0
## Add the signals on wave
add wave -position insertpoint  \
sim:/top/FIFOif/clk \
sim:/top/FIFOif/rst_n \
sim:/top/FIFOif/wr_en \
sim:/top/FIFOif/rd_en \
sim:/top/FIFOif/wr_ack \
sim:/top/FIFOif/full \
sim:/top/FIFOif/empty \
sim:/top/FIFOif/overflow \
sim:/top/FIFOif/underflow \
sim:/top/FIFOif/almostfull \
sim:/top/FIFOif/almostempty \
sim:/top/FIFOif/data_in \
sim:/top/FIFOif/data_out \
sim:@FIFO_scoreboard@1.data_out_ref \
sim:/top/DUT/count \
sim:@FIFO_scoreboard@1.counter
## Add assertions to wave
add wave /top/DUT/SVA_inst/assert_reset_falgs
add wave /top/DUT/SVA_inst/assert_empty
add wave /top/DUT/SVA_inst/assert_full
add wave /top/DUT/SVA_inst/assert_almostfull
add wave /top/DUT/SVA_inst/assert_almostempty
add wave /top/DUT/SVA_inst/RD_PTR_assert
add wave /top/DUT/SVA_inst/WR_PTR_assert
add wave /top/DUT/SVA_inst/assert_overflow
add wave /top/DUT/SVA_inst/assert_underflow
add wave /top/DUT/SVA_inst/assert_wr_ack_1
add wave /top/DUT/SVA_inst/assert_wr_ack_0
## Excluding the illegal bins
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_wr_ack/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_wr_ack/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_overflow/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_overflow/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_underflow/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_underflow/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_full/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_full/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_empty/}
coverage exclude -cvgpath {/cvg_pkg/FIFO_cvg_collector/cvr_gp/cross_coverage_empty/}
run -all
#vcover report top.ucdb -details -annotate -all -output coverage_report.txt
```

9. Code coverage

```
Assertion Coverage:
    Assertions                      11        11         0   100.00%

============================================================================
Directive Coverage:
    Directives                      11        11         0   100.00%

============================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       1         1         0   100.00%

============================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        106       106         0   100.00%

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

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

10. Functional coverage

Covergroups

File Bookmarks Window Help

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
TYPE cvr_gp		100.00%	100	100.00...		✓	auto(1)		
CVP cvr_gp::{#seq_item.wr_en__0#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.rd_en__1#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.underflow__2#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.wr_en__3#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.rd_en__4#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.overflow__5#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.wr_en__6#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.rd_en__7#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.wr_ack__8#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.wr_en__9#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.rd_en__10#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.empty__11#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.wr_en__12#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.rd_en__13#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.full__14#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.wr_en__15#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.rd_en__16#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.almostempty__17#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.wr_en__18#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.rd_en__19#}		100.00%	100	100.00...		✓			
CVP cvr_gp::{#seq_item.almostfull__20#}		100.00%	100	100.00...		✓			
CROSS cvr_gp::cross_coverage_almostfull		100.00%	100	100.00...		✓			
CROSS cvr_gp::cross_coverage_almostempty		100.00%	100	100.00...		✓			
CROSS cvr_gp::cross_coverage_full		100.00%	100	100.00...		✓			
CROSS cvr_gp::cross_coverage_empty		100.00%	100	100.00...		✓			
CROSS cvr_gp::cross_coverage_wr_ack		100.00%	100	100.00...		✓			
CROSS cvr_gp::cross_coverage_overflow		100.00%	100	100.00...		✓			
CROSS cvr_gp::cross_coverage_underflow		100.00%	100	100.00...		✓			

11. Assertions coverage

Assertions

File Edit View Add Bookmarks Window Help

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/im...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/imm...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/sequence_pkg::FIFO_write_seq::body/#ublk#50851543#40/im...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/sequence_pkg::FIFO_read_seq::body/#ublk#50851543#69/imm...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/sequence_pkg::FIFO_rd_wr_seq::body/#ublk#50851543#97/im...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/top/DUT/SVA_inst/assert_reset_falgs	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~full&empty&~almostempty...	✓
/top/DUT/SVA_inst/assert_empty	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~full&empty&~almostempty...	✓
/top/DUT/SVA_inst/assert_full	Immediate	SVA	on	0	1	-	-	-	-		off	assert (full&~empty&~almostempty...	✓
/top/DUT/SVA_inst/assert_almostfull	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((full~|empty)&~almostempt...	✓
/top/DUT/SVA_inst/assert_almostempty	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((full~|empty)&almostempty...	✓
/top/DUT/SVA_inst/RD_PTR_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA_inst/WR_PTR_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA_inst/assert_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA_inst/assert_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA_inst/assert_wr_ack_1	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA_inst/assert_wr_ack_0	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓

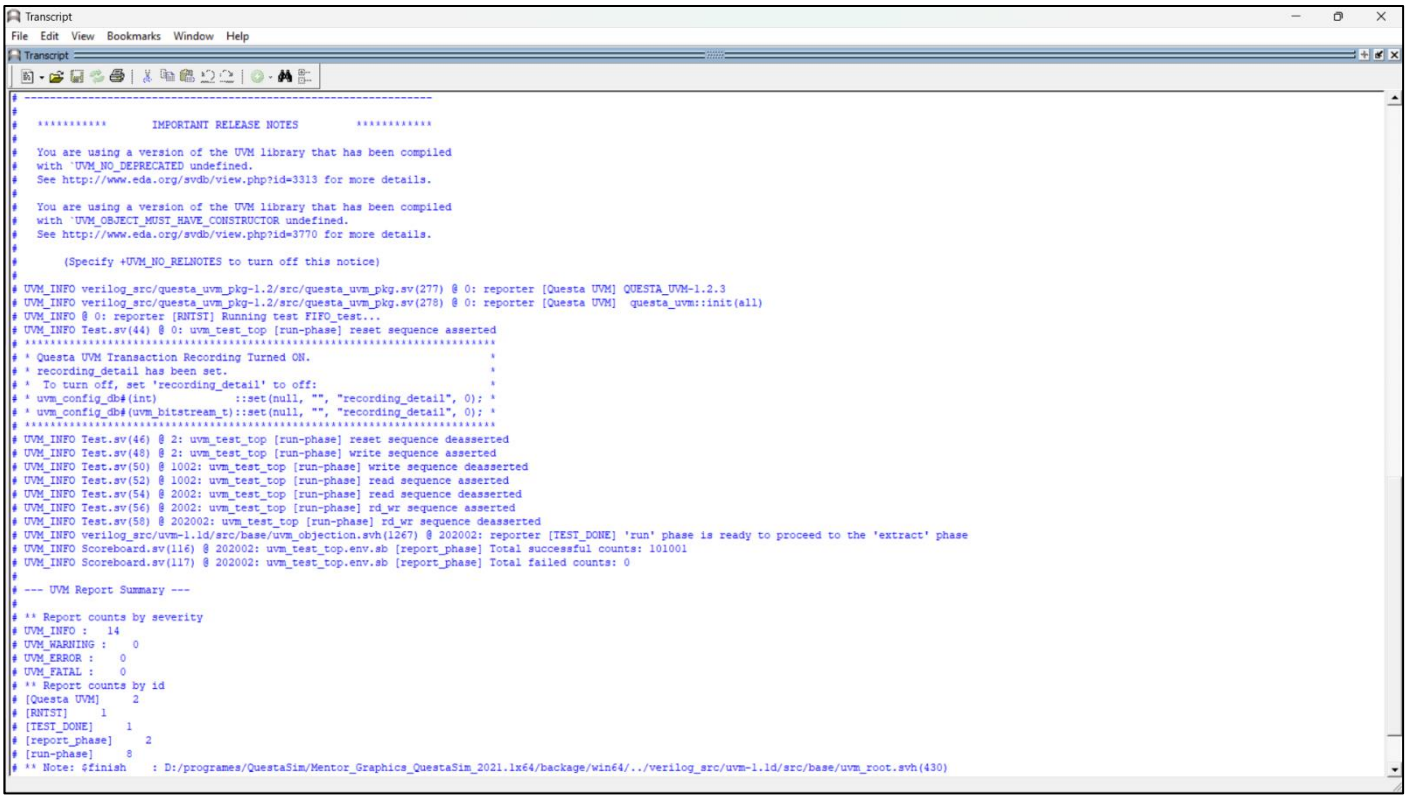
Cover Directives

File Edit View Add Bookmarks Window Help

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT/SVA_inst/WR_PTR_cover	SVA	✓	Off	43840	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/RD_PTR_cover	SVA	✓	Off	23289	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_wr_ack_1	SVA	✓	Off	47997	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_wr_ack_0	SVA	✓	Off	15862	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_underflow	SVA	✓	Off	2670	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_reset_falgs	SVA	✓	Off	4794	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_overflow	SVA	✓	Off	15862	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_full	SVA	✓	Off	8235	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_empty	SVA	✓	Off	5709	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_almostfull	SVA	✓	Off	10667	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT/SVA_inst/cover_almostempty	SVA	✓	Off	6325	1	Unli...	1	100%		✓	0	0	0 ns	0

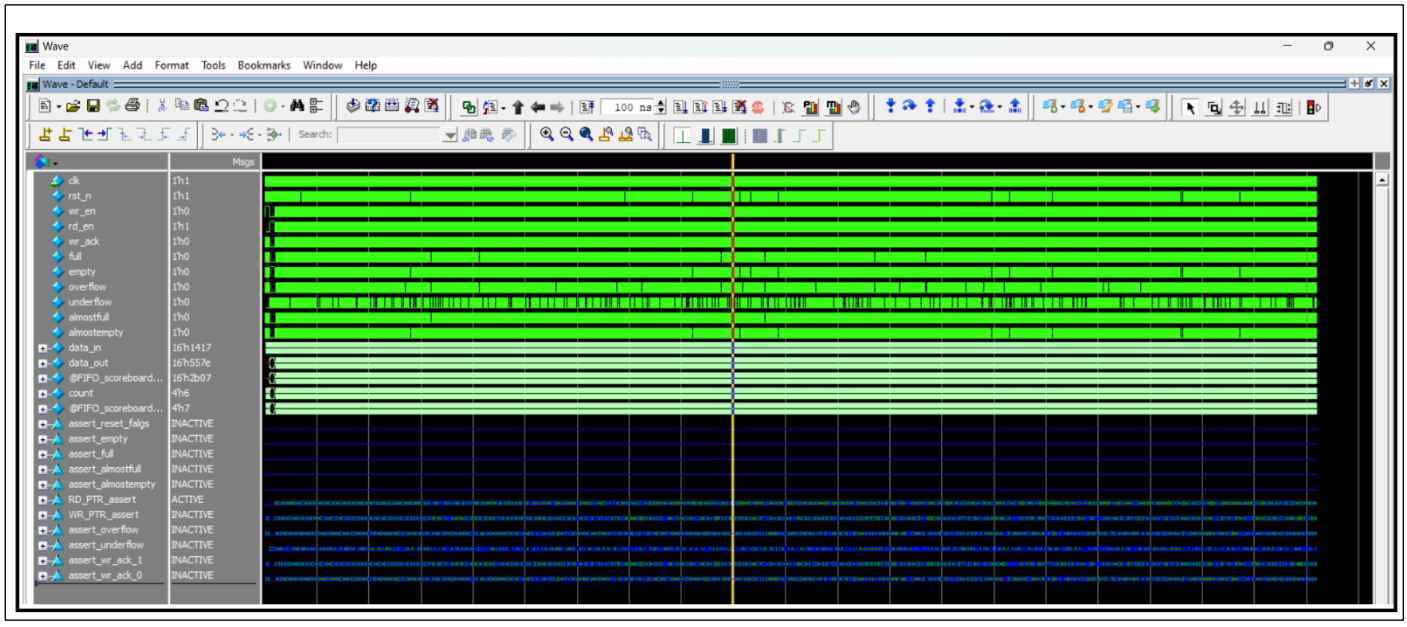
12. Questa Sim Snippets

12.1. Transcript

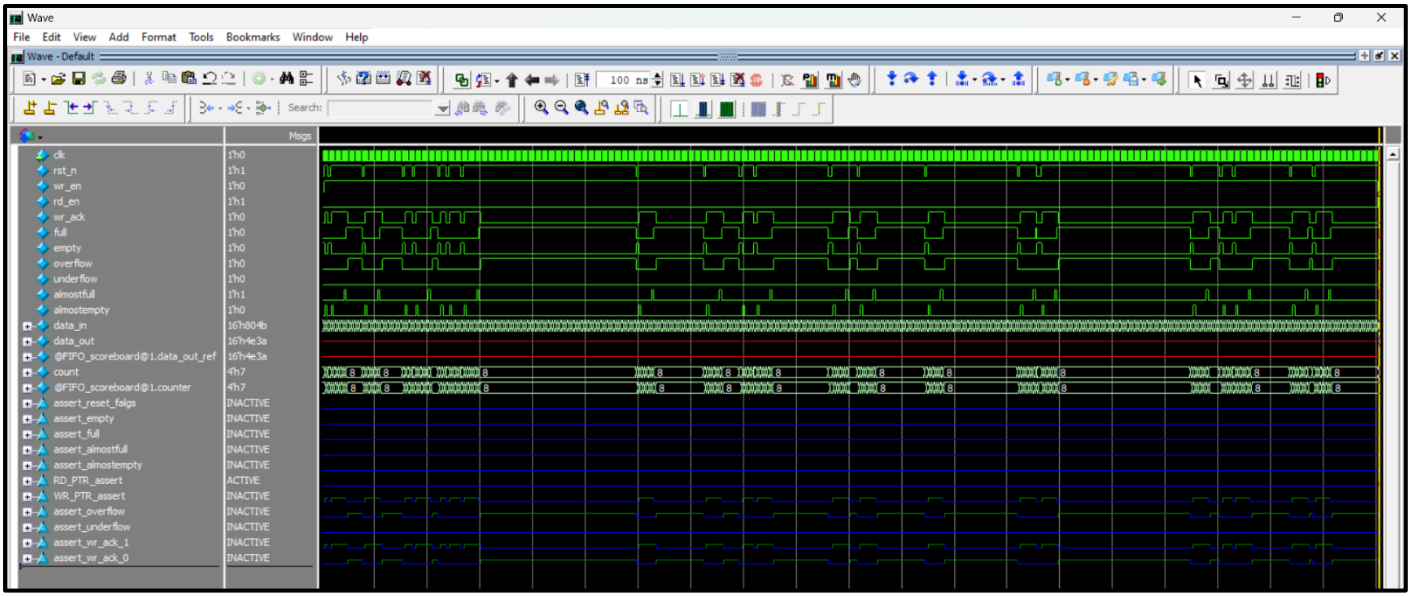


```
# ----------------------------------------------------------------
#
#    ************        IMPORTANT RELEASE NOTES         ************
#
#    You are using a version of the UVM library that has been compiled
#    with `UVM_NO_DEPRECATED undefined.
#    See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#    You are using a version of the UVM library that has been compiled
#    with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#    See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#        (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# 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 Test.sv(44) @ 0: uvm_test_top [run-phase] reset sequence 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 Test.sv(46) @ 2: uvm_test_top [run-phase] reset sequence deasserted
# UVM_INFO Test.sv(48) @ 2: uvm_test_top [run-phase] write sequence asserted
# UVM_INFO Test.sv(50) @ 1002: uvm_test_top [run-phase] write sequence deasserted
# UVM_INFO Test.sv(52) @ 1002: uvm_test_top [run-phase] read sequence asserted
# UVM_INFO Test.sv(54) @ 2002: uvm_test_top [run-phase] read sequence deasserted
# UVM_INFO Test.sv(56) @ 2002: uvm_test_top [run-phase] rd_wr sequence asserted
# UVM_INFO Test.sv(58) @ 202002: uvm_test_top [run-phase] rd_wr sequence deasserted
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 202002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO Scoreboard.sv(116) @ 202002: uvm_test_top.env.sb [report_phase] Total successful counts: 101001
# UVM_INFO Scoreboard.sv(117) @ 202002: uvm_test_top.env.sb [report_phase] Total failed counts: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   14
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run-phase]     8
# ** Note: $finish    : D:/programes/QuestaSim/Mentor_Graphics_QuestaSim_2021.1x64/backage/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
```

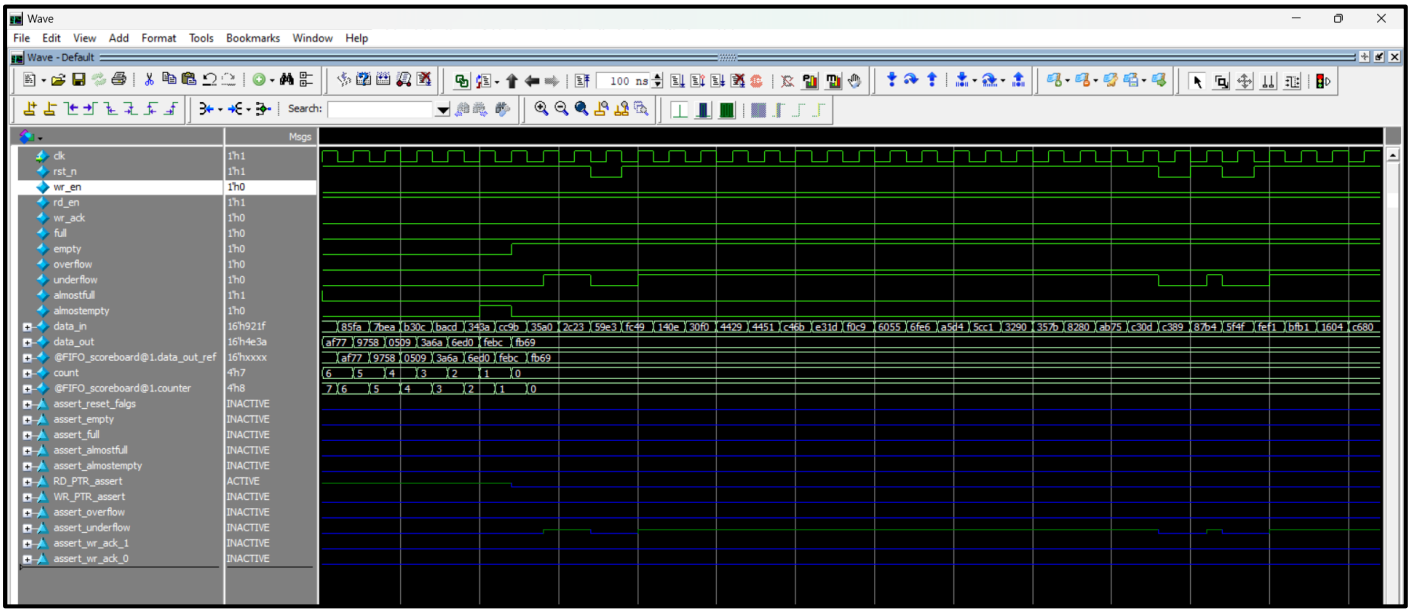
12.2. Full Waveform



12.3. Write Only Waveform



12.4. Read only Waveform



12.5. Read write Waveform