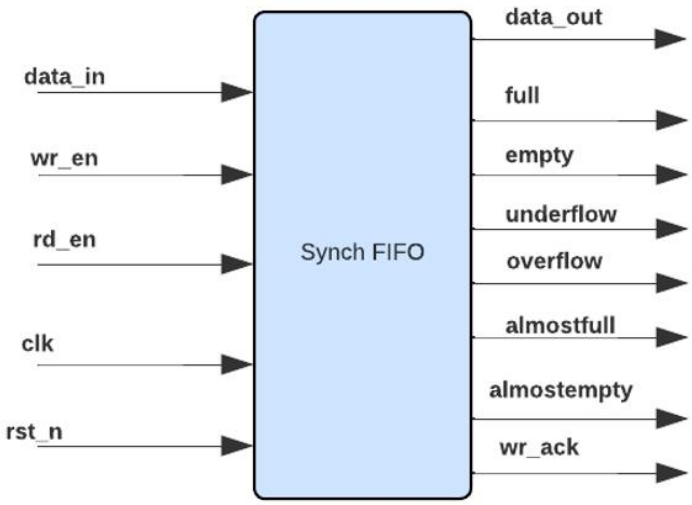
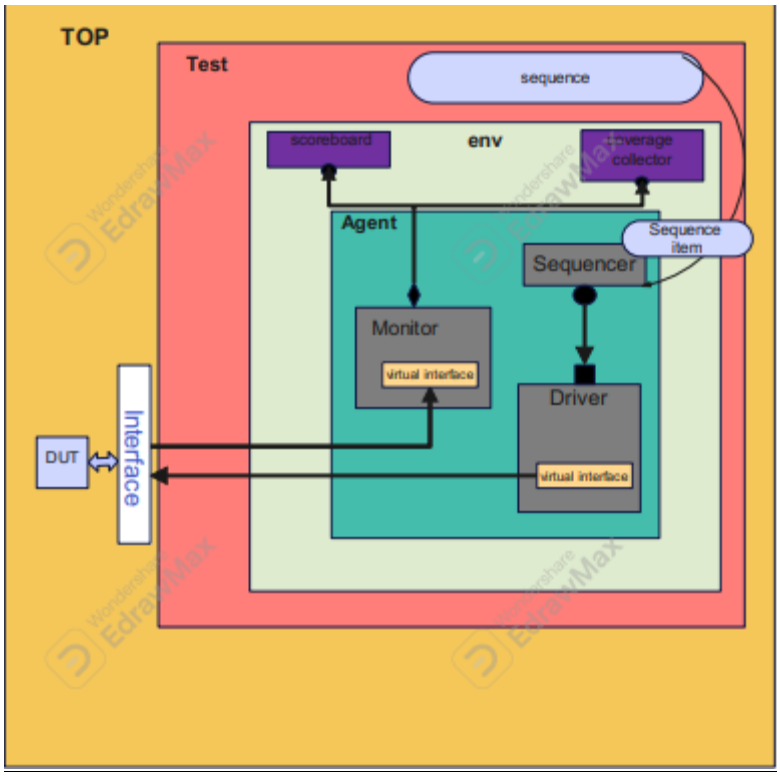
Synchronous FIFO Verification

Block diagram and signal description



Port	Direction	Function
data_in	Input	Write Data: The input data bus used when writing the FIFO.
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
clk		Clock signal
rst_n		Active low asynchronous reset
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 flowing the FIFO is not destructive to the FIFO.
wr_ack		Write Acknowledge: This sequential output signal indicates that a write request (wr_en) has succeeded.

Overview of the project



1. DUT (Device Under Test):

- The FIFO module defines the FIFO functionality.
- It has parameters for FIFO width (FIFO_WIDTH) and depth (FIFO_DEPTH).
- It uses internal registers (mem) to store data.
- It has separate wr_ptr (write pointer) and rd_ptr (read pointer) to track data location.
- Two always_comb blocks handle write and read operations, updating pointers and flags (wr_ack, overflow, underflow, full, empty, almostfull, almostempty).

2. SVA Assertions (SVA.sv):

- This module contains assertions to verify the DUT's behavior.
- It checks the flags (full, empty, almostfull, almostempty) based on the internal counter (count).
- It uses properties to verify write acknowledgement (wr_ack), overflow (overflow), underflow (underflow), counter increment (assert_cnt_inc), and decrement (assert_cnt_dec).
- It uses covergroups to record coverage for these assertions.

3. Top Level (TOP.sv):

- This module instantiates the FIFO DUT and the SVA assertion module.
- It creates a clock signal and binds the DUT interface (f_if) to the SVA assertions.
- It runs a test named "fifo_test" defined in the fifo_test_pkg package.

4. Test Sequence Package (fifo_test_pkg.sv):

- This package defines classes for the test sequence and environment.
- The fifo_test class inherits from uvm_test.
- It creates objects for environment (env_test), configuration (fifo_config_obj_test), sequencer (sqr), driver (drv), monitor (mon), and analysis port (agt_ap).
- The build_phase connects these objects based on configurations obtained from the UVM config database.
- The run_phase resets the DUT, runs different test sequences (rst_seq, wr_only_seq, rd_only_seq, main_seq), and raises/drops objections during the sequence execution.

5. FIFO Agent (fifo_agt.sv):

- This class represents the agent that drives the DUT.
- It contains objects for the sequencer (sqr), driver (drv), monitor (mon), and analysis port (agt_ap).
- It connects these objects based on configurations obtained during the build phase.

6. FIFO Monitor (fifo_mon.sv):

- This class monitors the DUT's interface signals.
- In the run_phase, it samples the interface signals (rst_n, wr_en, rd_en, data_in, data_out, wr_ack, overflow, underflow, full, empty, almostfull, almostempty) at every falling edge of the clock and creates a fifo_seq_item object containing these values.
- It broadcasts the fifo_seq_item object to the coverage collector and scoreboard using the analysis port (mon_ap).

7. Coverage Collector (cvrgclctr.sv):

- This class collects coverage information about the DUT's behavior.
- It defines a covergroup (cg) and coverpoints for various input and output signals.
- It uses cross-coverage to capture interactions between different signals (e.g., wr_enable and wr_ack).
- It samples the received fifo_seq_item and updates the covergroup.

8. Scoreboard (scoreboard_pkg.sv):

- This class acts as a reference model for the DUT.
- It maintains its own internal state (count, data_out_ref, flags) to compare with the DUT's behavior.
- The reference model function emulates the DUT's behavior based on the received fifo_seq_item.

- It compares the DUT's data output (data_out) and flags with its reference model and reports errors if there are any mismatches.

Overall Workflow:

1. The testbench instantiates the DUT and SVA assertions.
2. It runs different test sequences through the sequencer and driver.
3. The driver sends data to the DUT based on the sequence.
4. The monitor samples the DUT's interface signals and creates a fifo_seq_item

1-Verification plan

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, the output data value and the sequential flags should be low ..this shows with the reset seq send at the beginning	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the simulation time.	cover the values of the signals when the reset is asserted	A checker (immediate assertion) in the SVA to make sure the output is correct
FIFO_2	UVM_test will send the write sequence after reseting the DUT where the reset and rd_en signals are off and randomize the data_in for tens of times so the almost_full,full signals can be shown as high and increasing the wr_ptr as well	Randomization under constraints on the wr_en which makes it high through more than 70 % of the time in sequence item	Cover all values of the ouput falgs and cover the cross coverage between 3 signals which are write enable, read enable and each output control signals (outputs except data_out) to make sure that all combinations of write and read enable took place in all state of the	concurrent assertion to check the fuctionlity and the output flags assertions
FIFO_3	UVM_test will send the read sequence after write sequence where the reset and wr_en signals are off so the rd_ptr could increase and and the data_out signals have the same value of the randomized data_in in write only sequence and check the almost_empty ,empty flags	Randomization under constraints on the rd_en which makes it high through more than 30 % of the time .	Cover all values of the ouput falgs and cover the cross coverage between 3 signals which are write enable, read enable and each output control signals (outputs except data_out) to make sure that all combinations of write and read enable took place in all state of the	concurrent assertion to check the fuctionlity and the output flags assertions
FIFO_4	Alternate between read and write operations randomly by looping and randomizing the inputs for 10000 wich guarantee that all conditions of the output signals would happens and compare the values with the reference values and check them	Randomization under constraints on the wr_en which makes it high through more than 70 % of the time and also on rd_en which makes it low for 70% of the time	Cover all values of the ouput falgs and cover the cross coverage between 3 signals which are write enable, read enable and each output control signals (outputs except data_out) to make sure that all combinations of write and read enable took place in all state of the	concurrent assertion to check the fuctionlity and the output flags assertions
FIFO_5	checking for the overflow flag when there are write operations and the full flag is asserted and for the under_flow flag when there are read operations and the empty flag is asserted	Randomization under constraints on the wr_en which makes it high through more than 70 % of the time and also on rd_en which makes it low for 70% of the time	Cover all values of the ouput falgs and cover the cross coverage between 3 signals which are write enable, read enable and each output control signals (outputs except data_out) to make sure that all combinations of write and read enable took place in all state of the	concurrent assertion to check the fuctionlity and the output flags assertions
FIFO_6	checking for the internal signal "counter" ensuring for its proper operation	Randomization under constraints on the wr_en which makes it high through more than 70 % of the time and also on rd_en which makes it low for 70% of the time	cover most of the states of the counter signal to ensure its proper operation	concurrent assertion to check the fuctionlity and the output flags assertions

2-Design with the bugs

```
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg wr_ack, overflow;
output full, empty, almostfull, almostempty, underflow;

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;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    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;
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end

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

assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign underflow = (empty && rd_en)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;

endmodule
```

3-Solved design

(Bugs were in design bugs fixed in FIFO RTL)

1- **In first always block**
   - ❖ Add wr_ack and overflow signals to reset block
2- **In second always block**
   - ❖ Add data_out to reset block
3- **In third always block**
   - ❖ adding the priority conditions in counter logic
4- **in the combinational output flags conditions**
   - ❖ adjusting the conditions of the flags (almost_full).

```
module FIFO(fifo_interface.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; //it has the same bits as the depth so we can equl them

//always block for write process and overflow calculations
always @(posedge f_if.clk or negedge f_if.rst_n) begin
    if (!f_if.rst_n) begin
        wr_ptr <= 0;
        f_if.wr_ack <= 0 ;
        f_if.overflow <= 0 ;
    end
    else if (f_if.wr_en && count < f_if.FIFO_DEPTH) begin //when fifo is full and wr_en and rd_en are high the read process takes place due to this if condition
        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
//always block for read process and  underflow flag
always @(posedge f_if.clk or negedge f_if.rst_n) begin
    if (!f_if.rst_n) begin
        rd_ptr <= 0;
        f_if.data_out <= 0 ;
        f_if.underflow <= 0 ;
    end
    else if (f_if.rd_en && count != 0) begin  //when fifo is empty and wr_en and rd_en are high the write process takes place due to this if condition
        f_if.data_out <= mem[rd_ptr];
```

```
//always block for read process and  underflow flag
always @(posedge f_if.clk or negedge f_if.rst_n) begin
    if (!f_if.rst_n) begin
        rd_ptr <= 0;
        f_if.data_out <= 0 ;
        f_if.underflow <= 0 ;
    end
    else if (f_if.rd_en && count != 0) begin  //when fifo is empty and wr_en and rd_en are high the write process takes place due to this if
        f_if.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else  begin
        if(f_if.empty && f_if.rd_en)
        f_if.underflow <= 1 ;
        else
        f_if.underflow <= 0 ;
    end
end

// always block for evaluating the counter
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.empty)
            count <= count + 1;
        else if (({f_if.wr_en, f_if.rd_en} == 2'b11) && f_if.full)
            count <= count - 1;
```

```
            count <= count - 1;
    end
end

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;
assign f_if.almostempty = (count == 1)? 1 : 0;


endmodule
```

4-Interface

FIFO_IF.sv > fifo_interface

```
interface fifo_interface(clk);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input clk ;

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


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



endinterface
```

5-Assertions

Feature	Assertion
When count equal the FIFO depth the full flag is asserted	if(DUT.count == f_if.FIFO_DEPTH) assert_full : assert (f_if.full==1);
When count equal Zero the empty flag is asserted	if (DUT.count == 0)assert_empty : assert (f_if.empty == 1) ;
When count equal FIFO depth minus one the almost_full flag is asserted	if (DUT.count == f_if.FIFO_DEPTH-1) assert_almostfull : assert (f_if.almostfull ==1) ;
When count equal one the almost_empty flag is asserted	if (DUT.count == 1) assert_almostempty : assert (f_if.almostempty == 1) ;
When disabling rst ,wr_en is high and the count is less than depth wr_ack flag is asserted	(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && DUT.count < f_if.FIFO_DEPTH) |=> (f_if.wr_ack) ;
When disabling rst ,wr_en is high and full is high over_flow flag is asserted	(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.full && f_if.wr_en) |=> (f_if.overflow) ;
When disabling rst ,rd_en is high and empty is high under_flow flag is asserted	(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.empty && f_if.rd_en) |=> (f_if.underflow) ;
When disabling rst ,wr_en is high and full and rd_en are not high => counter will be increamented	(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && !f_if.rd_en && !f_if.full) |=> (DUT.count == $past(DUT.count) + 1) ;
When disabling rst ,rd_en is high , empty and wr_en are not high => counter will be decreamented	(posedge f_if.clk) disable iff(!f_if.rst_n) (!f_if.wr_en && f_if.rd_en && !f_if.empty) |=> (DUT.count == $past(DUT.count) - 1) ;

```
module SVA (fifo_interface.DUT f_if) ;



always_comb begin ;
    if(DUT.count == f_if.FIFO_DEPTH)
    assert_full : assert (f_if.full==1);

    if (DUT.count == 0)
    assert_empty : assert (f_if.empty == 1) ;

    if (DUT.count == f_if.FIFO_DEPTH-1)
    assert_almostfull : assert (f_if.almostfull ==1 ) ;

    if (DUT.count == 1)
    assert_almostempty : assert (f_if.almostempty == 1) ;
end

property  assert_wr_ack;
@(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

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

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

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

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

```

```
property  assert_cnt_dec ;
@(posedge f_if.clk)  disable iff(!f_if.rst_n)  (!f_if.wr_en && f_if.rd_en && !f_if.empty) |=> (DUT.count == $past(DUT.count) - 1 ) ;
endproperty


assert_wr_ack_flag :assert property(assert_wr_ack);
assert_of_flag :assert property(assert_overflow);
assert_un_flag :assert property(assert_underflow);
assert_cnt_increment :assert property(assert_cnt_inc);
assert_cnt_decrement :assert property(assert_cnt_dec);


always_comb begin ;
    if(DUT.count == f_if.FIFO_DEPTH)
    cover_full : cover (f_if.full==1);

    if (DUT.count == 0)
    cover_empty : cover (f_if.empty == 1) ;

    if (DUT.count == f_if.FIFO_DEPTH-1)
    cover_almostfull : cover (f_if.almostfull ==1 ) ;

    if (DUT.count == 1)
    cover_almostempty : cover (f_if.almostempty == 1) ;
end

cover_wr_ack_flag :cover property(assert_wr_ack);
cover_of_flag :cover property(assert_overflow);
cover_un_flag :cover property(assert_underflow);
cover_cnt_increment :cover property(assert_cnt_inc);
cover_cnt_decrement :cover property(assert_cnt_dec);


endmodule
```

6-TOP module

TOP.sv > TOP

```
1   import uvm_pkg::*;
2   import fifo_test_pkg::*;
3
4   `include "uvm_macros.svh"
5   module TOP ();
6
7   bit clk ;
8   initial
9   begin
10      clk =0 ;
11      forever  begin
12      #10 ;
13      clk =~clk ;
14      end
15  end
16
17  fifo_interface  f_if(clk);
18  FIFO  DUT  (f_if);
19
20  bind FIFO SVA sva_asserstion(f_if);
21
22  initial
23  begin
24      uvm_config_db #(virtual fifo_interface )::set(null, "uvm_test_top","FIFO_IF",f_if) ;
25      run_test("fifo_test"); //fifo test class name
26  end
27
28
29
30
31  endmodule
```

7- Configuration object + shared package

sharedPkg.sv > {} sharedpkg

```
1   package sharedpkg ;
2   parameter FIFO_WIDTH = 16;
3   parameter FIFO_DEPTH = 8;
4   int error_count = 0  ;
5   int correct_count = 0;
6
7   endpackage
```

fifo_config_obj.sv > {} fifo_config_obj

```
1   package fifo_config_obj ;
2   import  uvm_pkg::*;
3   `include"uvm_macros.svh"
4
5   class fifo_config_obj extends uvm_object;
6
7   `uvm_object_utils(fifo_config_obj);
8
9   virtual fifo_interface fifo_config_vif;
10
11  function new(string name ="fifo_config_obj");
12  super.new(name);
13  endfunction
14
15  endclass
16
17
18  endpackage
```

8- sequence item

```
1   package fifo_seqitem;
2   import uvm_pkg::*;
3   import sharedpkg::*;
4   `include "uvm_macros.svh"
5
6   class fifo_seq_item extends uvm_sequence_item ;
7   `uvm_object_utils(fifo_seq_item);
8
9   function new(string name ="fifo_seq_item");
10      super.new(name);
11  endfunction
12
13  rand bit [FIFO_WIDTH-1:0] data_in;
14  rand bit  rst_n, wr_en, rd_en;
15  logic [FIFO_WIDTH-1:0]  data_out ;
16  logic wr_ack, overflow;
17  logic full, empty, almostfull, almostempty, underflow;
18
```

```
////////**********************
function string convert2string();
return $sformatf("%s reset = %0b, wr_en =%0b, rd_en =%0b,data_in = %0b",super.convert2string,rst_n,wr_en,rd_en,data_in);

endfunction

function string convert2string_stimulus();

return $sformatf(" reset = %0b, wr_en =%0b, rd_en =%0b,data_in = %0b",rst_n,wr_en,rd_en,data_in);

endfunction
////////**********************

//constrain block
int RD_EN_ON_DIST = 30  ;
int WR_EN_ON_DIST = 70 ;


constraint rst_cnstrs {
  rst_n dist {1:= 98 , 0 :=2 };  //u should make it enabled more than 1 to make it toggle more than one
}

constraint wr_enble {
  wr_en dist {1:= WR_EN_ON_DIST , 0 := 100-WR_EN_ON_DIST};
}

constraint rd_enable {
  rd_en dist {1:= RD_EN_ON_DIST , 0 := 100-RD_EN_ON_DIST};
}
```

9-sequences

1)reset sequence

rst_seq_f.sv > {} rst_seq_f > fifo_reset_seq

```
1   package rst_seq_f;
2   import uvm_pkg::*;
3   import fifo_seqitem::*;
4   `include "uvm_macros.svh"
5
6   class fifo_reset_seq extends uvm_sequence #(fifo_seq_item);
7   `uvm_object_utils(fifo_reset_seq);
8   fifo_seq_item seq_item ;
9
10  function  new (string name = "fifo_reset_seq");
11  super.new(name);
12  endfunction
13
14  task body ;
15  seq_item = fifo_seq_item::type_id::create("seq_item");
16  start_item(seq_item);
17  seq_item.rst_n = 0 ;
18  seq_item.data_out = 0 ;
19  seq_item.wr_ack = 0 ;
20  seq_item.overflow = 0 ;
21  seq_item.underflow = 0 ;
22  //shold i reset the counter ??...how ??
23  finish_item(seq_item);
24
25  endtask
26
27  endclass
28
29  endpackage
```

2)read only sequence

```
package rd_only_seq_f ;
import uvm_pkg::*;
import fifo_seqitem::*;
`include "uvm_macros.svh"
class fifo_rd_only_seq extends uvm_sequence #(fifo_seq_item);
`uvm_object_utils(fifo_rd_only_seq);

fifo_seq_item seq_item ;

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

task body ;
repeat (50) begin
seq_item = fifo_seq_item::type_id::create("seq_item");
start_item(seq_item);
seq_item.rst_n = 1 ;
seq_item.wr_en = 0 ;
seq_item.rd_en = 1 ;
finish_item(seq_item);
end
endtask

endclass

endpackage
```

3)write only sequence

```
wr_only-seq_f.sv > { } wr_only_seq_f > fifo_wr_only_seq
package wr_only_seq_f ;
import uvm_pkg::*;
import fifo_seqitem::*;
`include "uvm_macros.svh"
class fifo_wr_only_seq extends uvm_sequence #(fifo_seq_item);
`uvm_object_utils(fifo_wr_only_seq);

fifo_seq_item seq_item ;

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

task body ;
repeat (80) begin
seq_item = fifo_seq_item::type_id::create("seq_item");
start_item(seq_item);
seq_item.rst_n = 1 ;
seq_item.wr_en = 1 ;
seq_item.rd_en = 0 ;
seq_item.rand_mode(0);
seq_item.data_in.rand_mode(1);
assert (seq_item.randomize());

finish_item(seq_item);
end


endtask

endclass

endpackage

```

4)main sequence

```
package main_seq;
import uvm_pkg::*;
import fifo_seqitem::*;
`include "uvm_macros.svh"
class fifo_main_seq extends uvm_sequence #(fifo_seq_item);
`uvm_object_utils(fifo_main_seq);

fifo_seq_item seq_item ;

function new(string name ="fifo_main_seq");
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
```

10- Test

```
package fifo_test_pkg ;
import  uvm_pkg::*;
import  fifo_env::*;
import  fifo_config_obj::*;
import  main_seq::*;
import  rd_only_seq_f::*;
import  wr_only_seq_f::*;
import  rst_seq_f::*;

`include"uvm_macros.svh"

class fifo_test extends uvm_test ;

`uvm_component_utils(fifo_test)
//object declration
fifo_env env_test;
fifo_config_obj fifo_config_obj_test ;
virtual fifo_interface fifo_test_vif ;
fifo_reset_seq  rst_seq;
fifo_wr_only_seq wr_only_seq;
fifo_rd_only_seq rd_only_seq;
fifo_main_seq    main_seq ;


//constructs the class
function new (string name ="fifo_test" , uvm_component parent = null);
super.new(name,parent);
endfunction

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

env_test =fifo_env::type_id::create("env_test",this) ;
fifo_config_obj_test = fifo_config_obj::type_id::create("fifo_config_obj_test");
rst_seq = fifo_reset_seq::type_id::create("rst_seq");
wr_only_seq = fifo_wr_only_seq::type_id::create("wr_only_seq");
rd_only_seq = fifo_rd_only_seq::type_id::create("rd_only_seq");
main_seq = fifo_main_seq::type_id::create("main_seq");

if (!uvm_config_db #(virtual fifo_interface) :: get(this ,"","FIFO_IF",fifo_config_obj_test.fifo_config_vif))
`uvm_fatal("build_phase","TEST - unable to get the virtual interface of the  fifo from the uvm_config_db");

uvm_config_db #(fifo_config_obj)::set (this ,"*","CFG",fifo_config_obj_test);
endfunction
```

```
//remember test is the intellegent boy who knows which seq opens to which sqr
task run_phase (uvm_phase phase) ;
super.run_phase(phase);
phase.raise_objection(this);

`uvm_info("run_phase","reset asserted",UVM_LOW)
rst_seq.start(env_test.agt.sqr) ;
`uvm_info("run_phase","reset deasserted",UVM_LOW)

`uvm_info("run_phase","write only seq is asserted",UVM_LOW)
wr_only_seq.start(env_test.agt.sqr) ;
`uvm_info("run_phase","write only seq is deasserted",UVM_LOW)

`uvm_info("run_phase","read only seq is asserted",UVM_LOW)
rd_only_seq.start(env_test.agt.sqr) ;
`uvm_info("run_phase","read only seq is deasserted",UVM_LOW)

`uvm_info("run_phase"," randomized stimulus generation started",UVM_LOW)
main_seq.start(env_test.agt.sqr) ;
`uvm_info("run_phase","stimulus generation finished",UVM_LOW)
phase.drop_objection(this);

endtask

endclass
endpackage
```

11-Env

fifo_env.sv > {} fifo_env

```
1  package fifo_env ;
2  import  uvm_pkg::*;
3  import  fifo_agt::*;
4  import  cvrgclctr::*;
5  import  scoreboard_pkg::*;
6
7  `include"uvm_macros.svh"
8
9  class fifo_env extends uvm_env ;
10
11 `uvm_component_utils(fifo_env)
12
13 fifo_agent agt;
14 FIFO_scoreboard sb ;
15 fifo_cvrg cov;
16
17
18 function new (string name ="fifo_env" , uvm_component parent = null);
19 super.new(name,parent);
20 endfunction
21
22
23 function void build_phase (uvm_phase phase);
24 super.build_phase(phase);
25 agt = fifo_agent::type_id::create("agt",this);
26 sb = FIFO_scoreboard::type_id::create("sb",this);
27 cov = fifo_cvrg::type_id::create("cov",this);
28
29 endfunction
30
31 function void connect_phase (uvm_phase phase);
32 super.connect_phase(phase);
33 agt.agt_ap.connect(sb.sb_export);
34 agt.agt_ap.connect(cov.cov_export);
35 endfunction
36
37 endclass
38 endpackage
```

12-scoreboard

fifo_scoreboard.sv > {} scoreboard_pkg > FIFO_scoreboard

```
package scoreboard_pkg ;
import uvm_pkg::*;
import sharedpkg::*;
import fifo_seqitem::*;
`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 ;

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


function void connect_phase(uvm_phase phase);
   super.connect_phase(phase);
   sb_export.connect(sb_fifo.analysis_export);   //this is the internal connection between the fifo and the sb
endfunction

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

bit [6:0] scrbrd_out_flags ;
bit [6:0] dut_out_flags ;
bit [FIFO_WIDTH-1:0] data_q[$] ;


function comb_flags_calc ;
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;
endfunction
```

```

fork
 begin

if(!seq_item_chk.rst_n) begin
wr_ack_ref =0 ;
overflow_ref =0;
data_q.delete();
full_ref = 0 ;
almostfull_ref = 0 ;
end
else if (seq_item_chk.wr_en && count<FIFO_DEPTH )
begin
    data_q.push_back(seq_item_chk.data_in);
    wr_ack_ref =1;
end
else begin
    wr_ack_ref = 0 ;
    if (full_ref && seq_item_chk.wr_en)
        overflow_ref = 1;
    else
        overflow_ref = 0;
end
end

begin
    if(!seq_item_chk.rst_n) begin
    data_out_ref = 0;
    underflow_ref =0;
    empty_ref = 0 ;
    almostempty_ref = 0 ;
    end
    else if (seq_item_chk.rd_en && count != 0)
    data_out_ref = data_q.pop_front();
    else begin
    if (empty_ref && seq_item_chk.rd_en)
    underflow_ref = 1 ;
    else
    underflow_ref = 0 ;
    end
end
```

```
join
if(!seq_item_chk.rst_n)
    count = 0 ;
else begin
if  ( ({seq_item_chk.wr_en, seq_item_chk.rd_en} == 2'b10) && !full_ref)
    count = count + 1;
else if ( ({seq_item_chk.wr_en, seq_item_chk.rd_en} == 2'b01) && !empty_ref)
    count = count - 1;
else if (({seq_item_chk.wr_en, seq_item_chk.rd_en} == 2'b11) && empty_ref)
    count = count + 1;
else if (({seq_item_chk.wr_en, seq_item_chk.rd_en} == 2'b11) && full_ref)
          count = count - 1;
end
comb_flags_calc();
endfunction


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

        reference_model(seq_item_sb);

        scrbrd_out_flags ={wr_ack_ref,overflow_ref,full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref} ;
        dut_out_flags ={seq_item_sb.wr_ack,seq_item_sb.overflow,seq_item_sb.full,seq_item_sb.empty,seq_item_sb.almostfull,seq_item_sb.almostempty,seq_item_sb.underflow} ;

       if (seq_item_sb.data_out !== data_out_ref) begin
          `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s While the reference output:%0b", seq_item_sb.convert2string(),data_out_ref));
          error_count++;
        end

       if (dut_out_flags !== scrbrd_out_flags) begin
          `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:%s While the reference output:%0p", seq_item_sb.convert2string(),scrbrd_out_flags));
          error_count++;
       end

        if (seq_item_sb.data_out == data_out_ref && dut_out_flags == scrbrd_out_flags) begin
          `uvm_info("run_phase", $sformatf("Correct fifo out: %s  ", seq_item_sb.convert2string()), UVM_HIGH);
          correct_count++;
        end

      end
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",error_count), UVM_MEDIUM);
  endfunction
  /*
  function check_data ( FIFO_transaction seq_item_sb) ;
  reference_model(seq_item_sb);
  scrbrd_out_flags ={wr_ack_ref,overflow_ref,full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref} ;
  dut_out_flags ={seq_item_sb.wr_ack,seq_item_sb.overflow,seq_item_sb.full,seq_item_sb.empty,seq_item_sb.almostfull,seq_item_sb.almostempty,seq_item_sb.underflow} ;

  if (seq_item_sb.data_out !== data_out_ref) begin
  $display ("time %0t the output of the design and golden model are not equal",$time);
  error_cntr++ ;
  end
  if (dut_out_flags !== scrbrd_out_flags) begin
  $display ("time %0t the output flags of the design and golden model are not equal",$time);
  error_cntr++ ;
  end
  if (seq_item_sb.data_out == data_out_ref && dut_out_flags == scrbrd_out_flags ) begin
    $display ("time %0t Succedded comarison with data = %0p",$time,data_q);
  correct_cntr++ ;
  end
  endfunction
  */
  endclass
  endpackage
```

13-Coverage collector

```
package  cvrgcltr ;
import uvm_pkg::*;
import fifo_seqitem::*;
`include "uvm_macros.svh"

class fifo_cvrg extends uvm_component ;

`uvm_component_utils(fifo_cvrg) ;

uvm_analysis_export #(fifo_seq_item) cov_export ;
uvm_tlm_analysis_fifo #(fifo_seq_item)  cov_fifo ;

fifo_seq_item seq_item_cov;

covergroup cg ;
//////coverpoints for input sigs//////
wr_enable_cvp : coverpoint seq_item_cov.wr_en{
    bins wr_en0 ={0};
    bins wr_en1 ={1};
    option.weight =0 ;}
rd_enable_cvp : coverpoint seq_item_cov.wr_en{
    bins rd_en0 ={0};
    bins rd_en1 ={1};
    option.weight =0 ;}

//////coverpoints for output sigs//////
wr_ack_cvp :coverpoint  seq_item_cov.wr_ack{
    bins wr_ak0 ={0};
    bins wr_ak1 ={1};
    option.weight =0 ;}

overf_cvp :coverpoint  seq_item_cov.overflow{
    bins overflow0 ={0};
    bins overflow1 ={1};
    option.weight =0 ;}

fl_cvp :coverpoint  seq_item_cov.full{
    bins full0 ={0};
    bins full1 ={1};
    option.weight =0 ;}

mt_cvp :coverpoint  seq_item_cov.empty{
    bins empty0 ={0};
    bins empty1 ={1};
    option.weight =0 ;}

al_f_cvp :coverpoint  seq_item_cov.almostfull{
    bins almostfull0 ={0};
```

```
al_f_cvp :coverpoint  seq_item_cov.almostfull{
    bins almostfull0 ={0};
    bins almostfull1 ={1};
    option.weight =0 ;}

al_e_cvp :coverpoint  seq_item_cov.almostempty{
    bins almostempty0 ={0};
    bins almostempty1 ={1};
    option.weight =0 ;}

underf_cvp :coverpoint  seq_item_cov.underflow{
    bins underflow0 ={0};
    bins underflow1 ={1};
    option.weight =0 ;}

cross wr_enable_cvp,rd_enable_cvp,wr_ack_cvp {
    bins wren_ack  = binsof(wr_enable_cvp.wr_en1) && binsof (wr_ack_cvp.wr_ak1) ;
    bins rden_ack  = binsof(wr_enable_cvp.wr_en1) && binsof (wr_ack_cvp.wr_ak1) && binsof (rd_enable_cvp.rd_en1) ;  //must add wr_en because wr_ack wont be 1 unless wr_en is 1
    option.cross_auto_bin_max = 0 ;
}

cross wr_enable_cvp,rd_enable_cvp,overf_cvp {
    bins wren_of  = binsof (wr_enable_cvp.wr_en1) && binsof (overf_cvp.overflow1) ;
    bins rden_of =  binsof (rd_enable_cvp.rd_en1) && binsof (overf_cvp.overflow1) ;   //should not happen ???
    option.cross_auto_bin_max = 0 ;
}

cross wr_enable_cvp,rd_enable_cvp,underf_cvp {
    bins wren_uf  = binsof (wr_enable_cvp.wr_en1) && binsof (underf_cvp.underflow1) ;
    bins rden_uf =  binsof (rd_enable_cvp.rd_en1) && binsof (underf_cvp.underflow1) ;
    option.cross_auto_bin_max = 0 ;
}


cross wr_enable_cvp,rd_enable_cvp,fl_cvp {
    bins wren_f  = binsof (wr_enable_cvp.wr_en1) && binsof (fl_cvp.full1) ;
    bins rden_f =  binsof (rd_enable_cvp.rd_en1) && binsof (fl_cvp.full1) ;
    option.cross_auto_bin_max = 0 ;
}

cross wr_enable_cvp,rd_enable_cvp,mt_cvp {
    bins wren_f  = binsof (wr_enable_cvp.wr_en1) && binsof (mt_cvp.empty1) ;
    bins rden_f =  binsof (rd_enable_cvp.rd_en1) && binsof (mt_cvp.empty1) ;
    option.cross_auto_bin_max = 0 ;
}
```

```
    option.cross_auto_bin_max = 0 ;
}

cross wr_enable_cvp,rd_enable_cvp,al_f_cvp {
    bins wren_f  = binsof (wr_enable_cvp.wr_en1) && binsof (al_f_cvp.almostfull1) ;
    bins rden_f =  binsof (rd_enable_cvp.rd_en1) && binsof (al_f_cvp.almostfull1) ;
    option.cross_auto_bin_max = 0 ;
}

cross wr_enable_cvp,rd_enable_cvp,al_e_cvp {
    bins wren_f  = binsof (wr_enable_cvp.wr_en1) && binsof (al_e_cvp.almostempty1) ;
    bins rden_f =  binsof (rd_enable_cvp.rd_en1) && binsof (al_e_cvp.almostempty1) ;
    option.cross_auto_bin_max = 0 ;
}

endgroup

function new(string name = "fifo_cvrg", uvm_component parent = null);
   super.new(name, parent);
   cg =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


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);
      cg.sample();
   end
endtask


endclass

endpackage
```

14-Agent

```
package fifo_agt ;
import  uvm_pkg::*;
import  fifo_driver::*;
import  fifo_sqr::*;
import  fifo_config_obj::*;
import  fifo_mon::*;
import  fifo_seqitem::*;

`include"uvm_macros.svh"


class fifo_agent extends uvm_agent;

`uvm_component_utils(fifo_agent)

fifo_sequencer sqr;
fifo_driver drv;
fifo_config_obj agt_cnfg;
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_obj)::get(this ,"","CFG",agt_cnfg))
`uvm_fatal("build_phase","AGENT - unable to get the virtual interface  set by the test  of fifo from the uvm_config_db");

sqr = fifo_sequencer::type_id::create("sqr",this);
drv = fifo_driver::type_id::create("drv",this);
mon = fifo_monitor::type_id::create("mon",this);
agt_ap = new ("agt_ap" ,this) ;

endfunction
```

```
28
29  function void build_phase (uvm_phase phase);
30  super.build_phase(phase);
31
32  if (!uvm_config_db #(fifo_config_obj)::get(this ,"","CFG",agt_cnfg))
33  `uvm_fatal("build_phase","AGENT - unable to get the virtual interface  set by the test  of fifo from the uvm_config_db");
34
35  sqr = fifo_sequencer::type_id::create("sqr",this);
36  drv = fifo_driver::type_id::create("drv",this);
37  mon = fifo_monitor::type_id::create("mon",this);
38  agt_ap = new ("agt_ap" ,this) ;
39
40  endfunction
41
42  function void connect_phase (uvm_phase phase) ;
43  super.connect_phase (phase) ;
44  drv.fifo_driver_vif = agt_cnfg.fifo_config_vif ;
45  mon.fifo_mon_vif = agt_cnfg.fifo_config_vif ;
46  drv.seq_item_port.connect(sqr.seq_item_export);
47  mon.mon_ap.connect(agt_ap) ;
48
49  endfunction
50
51
52  endclass
53
54
55  endpackage
```

15-Driver

fifo_driver.sv > { } fifo_driver > fifo_driver

```
1   package fifo_driver ;
2   import uvm_pkg::*;
3   import fifo_seqitem::*;
4   `include "uvm_macros.svh"
5
6   class fifo_driver extends uvm_driver #(fifo_seq_item) ;
7
8   `uvm_component_utils(fifo_driver)
9   //creating the virtual interface and config_obj handle
10
11  virtual fifo_interface fifo_driver_vif ;
12  fifo_seq_item  stim_seq_item;
13
14  function new (string name ="fifo_driver" , uvm_component parent = null);
15  super.new(name,parent);
16  endfunction
17
18
19  task run_phase (uvm_phase phase) ;
20  super.run_phase(phase);
21  forever
22  begin
23  stim_seq_item =fifo_seq_item::type_id::create("stim_seq_item");
24  seq_item_port.get_next_item(stim_seq_item); //request the seq_item from sequencer=====> sequence
25
26  fifo_driver_vif.rst_n = stim_seq_item.rst_n;
27  fifo_driver_vif.wr_en = stim_seq_item.wr_en;
28  fifo_driver_vif.rd_en = stim_seq_item.rd_en;
29  fifo_driver_vif.data_in = stim_seq_item.data_in;
30
31  @(negedge fifo_driver_vif.clk );
32  seq_item_port.item_done() ;
33  `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
34  end
35
36  endtask
37
38  endclass
39
40  endpackage
```

16-Monitor

```
1   package fifo_mon ;
2   import  uvm_pkg::*;
3   import  fifo_seqitem::*;
4
5   `include "uvm_macros.svh"
6
7   class fifo_monitor extends uvm_monitor ;
8
9   `uvm_component_utils(fifo_monitor)
10
11  virtual fifo_interface fifo_mon_vif ;
12  fifo_seq_item  rsp_seq_item;
13  uvm_analysis_port #(fifo_seq_item) mon_ap ;
14
15  function new (string name ="fifo_driver" , uvm_component parent = null);
16  super.new(name,parent);
17  endfunction
18
19
20  function  void build_phase (uvm_phase phase) ;
21  super.build_phase(phase) ;
22  mon_ap = new ("mon_ap",this);
23  endfunction
24
```

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


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_mon_vif.clk );  // here the delay shows that each negedge the monitor sends the data to the scoreboard .... not all monitors have the same delay so we use sb_fifo p3 min35

//inputs
rsp_seq_item.rst_n = fifo_mon_vif.rst_n ;
rsp_seq_item.wr_en = fifo_mon_vif.wr_en ;
rsp_seq_item.rd_en = fifo_mon_vif.rd_en ;
rsp_seq_item.data_in = fifo_mon_vif.data_in ;

//outputs
rsp_seq_item.data_out = fifo_mon_vif.data_out ;
rsp_seq_item.wr_ack = fifo_mon_vif.wr_ack ;
rsp_seq_item.overflow = fifo_mon_vif.overflow ;
rsp_seq_item.underflow = fifo_mon_vif.underflow ;
rsp_seq_item.full = fifo_mon_vif.full ;
rsp_seq_item.empty = fifo_mon_vif.empty ;
rsp_seq_item.almostfull = fifo_mon_vif.almostfull ;
rsp_seq_item.almostempty = fifo_mon_vif.almostempty ;

mon_ap.write (rsp_seq_item); //broadcasting the recieved seq from the interface to the coverage collector and scoreboards

`uvm_info("run_phase",rsp_seq_item.convert2string_stimulus(),UVM_HIGH)
end

endtask

endclass

endpackage
```

17-sequencer

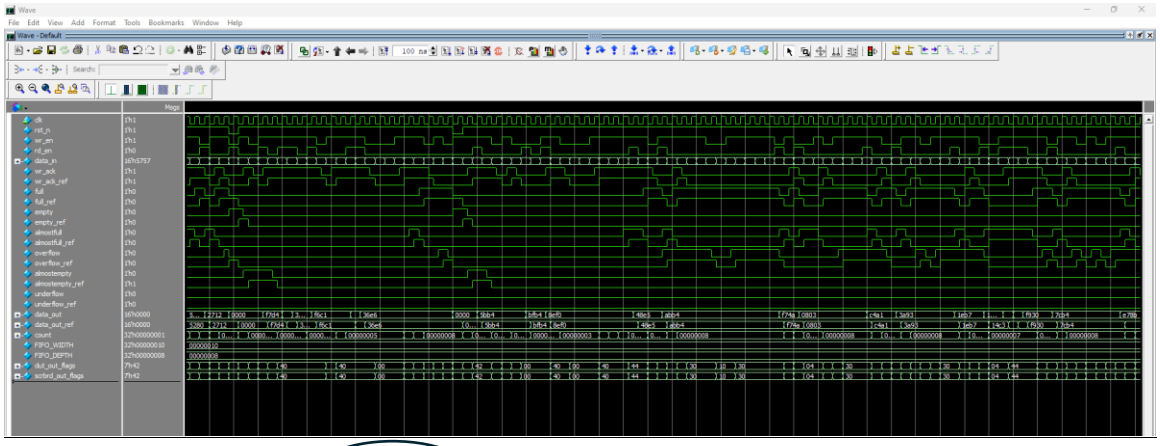
sequencer_f.sv > ...

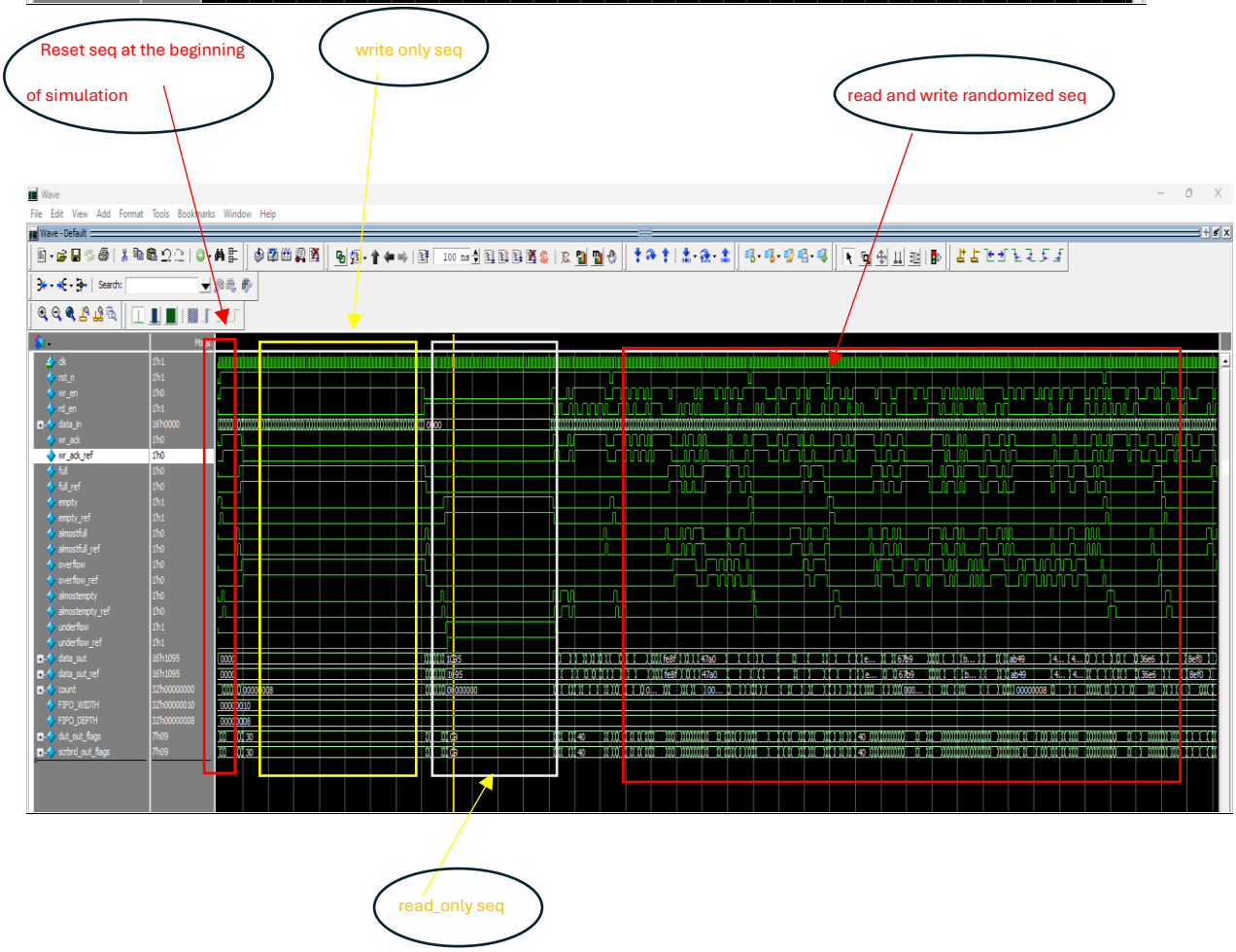
```
package fifo_sqr ;

import uvm_pkg::*;
import fifo_seqitem::*;
`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
```

Run.do file

```
vlib work
vlog -f sourcefiles.txt +cover
vsim -voptargs=+acc work.TOP -cover -classdebug -uvmcontrol=all
add wave /TOP/f_if/*
coverage save TOP.ucdb -onexit
run -all
```

Waveform and coverage reports





```
VSIM 80> run -all
# UVM_INFO fifo_test.sv(54) @ 20: uvm_test_top [run_phase] reset deasserted
# UVM_INFO fifo_test.sv(56) @ 20: uvm_test_top [run_phase] write only seq is asserted
# UVM_INFO fifo_test.sv(58) @ 1620: uvm_test_top [run_phase] write only seq is deasserted
# UVM_INFO fifo_test.sv(60) @ 1620: uvm_test_top [run_phase] read only seq is asserted
# UVM_INFO fifo_test.sv(62) @ 2620: uvm_test_top [run_phase] read only seq is deasserted
# UVM_INFO fifo_test.sv(64) @ 2620: uvm_test_top [run_phase]  randomized stimulus generation started
# UVM_INFO fifo_test.sv(66) @ 202620: uvm_test_top [run_phase] stimulus generation finished
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 202620: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO fifo_scoreboard.sv(139) @ 202620: uvm_test_top.env_test.sb [report_phase] Total successful transactions: 10131
# UVM_INFO fifo_scoreboard.sv(140) @ 202620: uvm_test_top.env_test.sb [report_phase] Total failed transactions: 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    : C:/Mentor_Graphics_QuestaSim_2021.1x64-20240819T153030Z-001/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 202620 ns  Iteration: 61  Instance: /TOP
# Break in Task uvm_pkg/uvm_root::run_test at C:/Mentor_Graphics_QuestaSim_2021.1x64-20240819T153030Z-001/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($
/wr_only_seq_f::fifo_wr_only_seq::body/#ublk#3248232...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (ra
/main_seq::fifo_main_seq::body/#ublk#134508273#16/i...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (ra
/TOP/DUT/sva_asserstion/assert_full	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.f
/TOP/DUT/sva_asserstion/assert_empty	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.f
/TOP/DUT/sva_asserstion/assert_almostfull	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.f
/TOP/DUT/sva_asserstion/assert_almostempty	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.f
/TOP/DUT/sva_asserstion/assert_wr_ack_flag	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@
/TOP/DUT/sva_asserstion/assert_of_flag	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@
/TOP/DUT/sva_asserstion/assert_un_flag	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@
/TOP/DUT/sva_asserstion/assert_cnt_increment	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@
/TOP/DUT/sva_asserstion/assert_cnt_decrement	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	C
/TOP/DUT/sva_asserstion/cover_full	SVA	✓	Off	1299	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_empty	SVA	✓	Off	209	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_almostfull	SVA	✓	Off	1600	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_almostempty	SVA	✓	Off	252	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_wr_ack_flag	SVA	✓	Off	3855	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_of_flag	SVA	✓	Off	2961	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_un_flag	SVA	✓	Off	136	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_cnt_increment	SVA	✓	Off	2698	1	Unli...	1	100%		✓	0	0	0 ns	
/TOP/DUT/sva_asserstion/cover_cnt_decrement	SVA	✓	Off	840	1	Unli...	1	100%		✓	0	0	0 ns	

Covergroups

Name	Coverage	Goal	% of Goal	Status	Merge_instances	Get_inst_coverage	Comment
/cvrgclctr/fifo_cvrg...	100.00%						
TYPE cg	100.00%	100	100.00...		auto(1)		
CVP cg::wr...	100.00%	100	100.00...				
CVP cg::rd...	100.00%	100	100.00...				
CVP cg::wr...	100.00%	100	100.00...				
CVP cg::ov...	100.00%	100	100.00...				
CVP cg::fl_...	100.00%	100	100.00...				
CVP cg::mt...	100.00%	100	100.00...				
CVP cg::al...	100.00%	100	100.00...				
CVP cg::al...	100.00%	100	100.00...				
CVP cg::un...	100.00%	100	100.00...				
CROSS cg:...	100.00%	100	100.00...				
CROSS cg:...	100.00%	100	100.00...				
CROSS cg:...	100.00%	100	100.00...				
CROSS cg:...	100.00%	100	100.00...				
CROSS cg:...	100.00%	100	100.00...				
CROSS cg:...	100.00%	100	100.00...				
bin wre...	1737	1	100.00...				
bin rden...	1737	1	100.00...				
CROSS cg:...	100.00%	100	100.00...				

Analysis | Assertions | Cover Directives | Covergroups | uvm_root.svh | TOP.sv

```
================================================================================
=== Instance: /TOP/DUT
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        25        25         0   100.00%

================================Branch Details================================
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      28        28         0   100.00%

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

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

```
    75              1                       4691     assign f_if.almostempty = (count == 1

Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         20        20         0   100.00%

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

Toggle Coverage for instance /TOP/DUT --
```

THANK YOU