Synchronous FIFO Verification

Project Done By :

Yousef Sherif Ali Hassan Elgendy

Introduction to FIFO

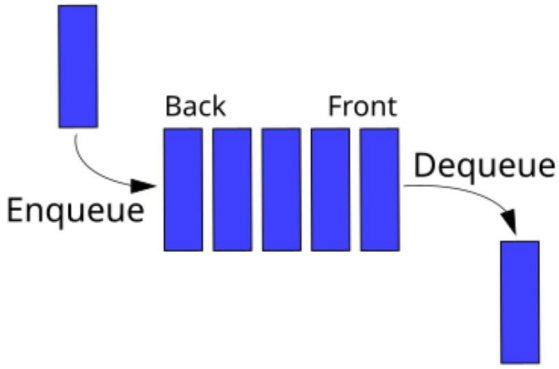
FIFO (First In, First Out) is a type of memory structure widely used in digital systems. Its behavior is similar to a queue: the first data element that enters is also the first one to leave. This makes it very useful for handling temporary data storage and smooth data transfer between hardware blocks that may not operate at the same rate.

Main Characteristics

- **Ordered Storage:** Keeps information in the same sequence it was received and delivers it in that order.
- **Write/Read Separation:** Maintains a dedicated pointer for inserting data and another one for retrieving it.
- **Cross-Clock Support:** Acts as a buffer when data must move between two circuits that run on different clock signals.
- **Error Prevention:** Detects and signals conditions like *full buffer* (cannot accept more data) or *empty buffer* (nothing to read).

Where It Is Used

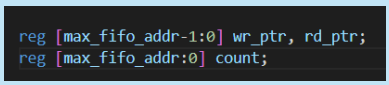
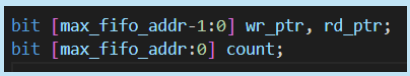
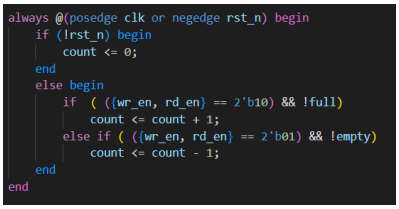
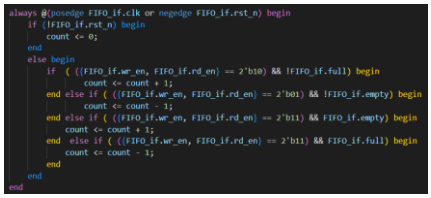
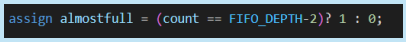
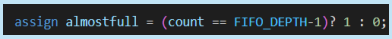
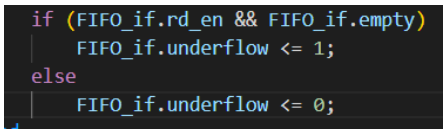
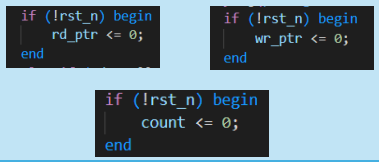
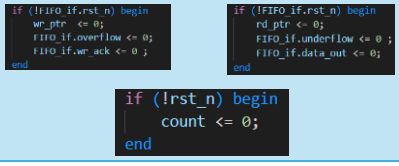
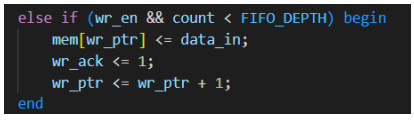
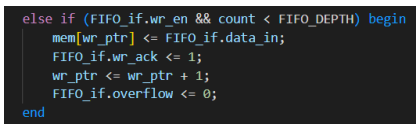
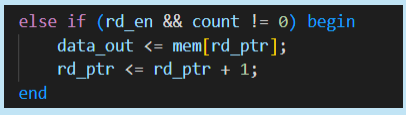
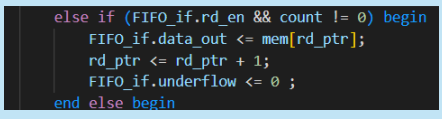
- **System Communication:** Balances the speed difference between processors and connected devices by storing data in transit.
- **Pipeline Operations:** Ensures continuous data availability across pipeline stages, improving efficiency.
- **Clock Domain Transfer:** Provides a safe channel for exchanging information between asynchronous clock domains.
- **Media Applications:** Commonly found in audio and video systems to avoid interruptions and ensure steady playback.



Verification Plan

Label	Design requirement Description	Stimulus generation	Functional coverage	Functionality check
FIFO_1	When the reset is asserted, the outputs , pointers and counters should be zero	Directed at the start of the simulation , then randomized with constraint that drive the reset to be off most of the simulation time.	-----	**Immediate assertion** in the DUT + Checker in the testbench (scoreboard) in reference_model
FIFO_2	when write enable is asserted and read enable is disasserted then write operation is done on FIFO	Directed at simulation	-----	Checker in the testbench (scoreboard) in reference_model
FIFO_3	when read enable is asserted and write enable is disasserted then read operation is done on FIFO	Directed at simulation	-----	Checker in the testbench (scoreboard) in reference_model
FIFO_4	When write and read operations are done simultansly but when empty only write is done and when full only read is done	Directed at simulation	-----	Checker in the testbench (scoreboard) in reference_model
FIFO_5	Randomzing all the inputs to see how the DUT outputs	Randomized with constraints on write, read and reset	-----	Checker in the testbench (scoreboard
Write ACK	When a write enable signal (wr_en) is active and the FIFO is not full, wr_ack should be asserted to confirm the write operation.	Write enable is Randomized with constraint to be 70% of the time active	Cross coverage between wr_ack signal and read and write enables combinations	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model
Overflow	If a write is attempted when the FIFO is full, overflow should be asserted	Write enable is Randomized with constraint to be 70% of the time active	Cross coverage between overflow signal and read and write enables combinations	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model
Underflow	If a read is attempted when the FIFO is empty, underflow should be asserted	Write enable is Randomized with constraint to be 30% of the time active	Cross coverage between underflow signal and read and write enables combinations	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model
Empty	When the internal count is zero, the empty flag should be asserted	Read enable is randomized with constraint to be 30% of the time active	Cross coverage between empty signal and read and write enables combinations	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model
Full	In When the internal count equals the FIFO depth, the full flag should be asserted	Write enable is Randomized with constraint to be 70% of the time active	Cross coverage between full signal and read and write enables combinations	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model
Almost Full	When the count reaches FIFO depth - 1, almost_full should be asserted	Write enable is Randomized with constraint to be 30% of the time active	Cross coverage between almostfull signal and read and write enables combinations	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model
Almost Empty	When the count equals 1, the almostempty signal should be asserted	Read enable is randomized with constraint to be 30% of the time active	Cross coverage between almostempty signal and read and write enables combinations	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model
Pointer Wraparound	After writing or reading FIFO_DEPTH entries (0 to 7), the write or read pointer should eventually wrap around back to 0. Same applies for the counter (0 to 8) on reset	-----	-----	**concurrent assertion**
Pointer Threshold	Internal pointers cannot exceed the FIFO_DEPTH entries in any given time. Same applies for the counter	-----	-----	**concurrent assertion** Checker in the testbench (scoreboard) in reference_model

Bugs Report

Explaination	Bug	After Fixing
Should be bit or you can initialize the reg with 0 cause reg is 4 state and the default of it was x.		
When write and read are asserted, the dut wasn't handling either to decrement the count or increment it depending on its case at that time which causes errors		
should be -1 not -2 in case of almost_full		
Underflow should be sequential not combinational		
Missing signals during the reset operation 1. overflow 2. underflow 3. wr_ack		
Overflow must be zero in successful write operations especially when read and write enables are asserted together here causes bug in flag		
underflow must be zero in successful read operations especially when read and write enables are asserted together here causes bug in flag		

Design After modification :

```
import shared_pkg::*;

module FIFO(FIFO_interface.DUT FIFO_if);

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

bit [max_fifo_addr-1:0] wr_ptr, rd_ptr;
bit [max_fifo_addr:0] count;

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

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

```
            FIFO_if.underflow <= 1;
        else
            FIFO_if.underflow <= 0;
    end
end

always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    if (!FIFO_if.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full) begin
                count <= count + 1;
        end else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty) begin
                count <= count - 1;
        end else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty) begin
            count <= count + 1;
        end  else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full) begin
            count <= count - 1;
        end
    end
end

assign FIFO_if.full =   (count == FIFO_DEPTH)? 1 : 0;
assign FIFO_if.empty =  (count == 0)? 1 : 0;
assign FIFO_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
assign FIFO_if.almostempty =  (count == 1)? 1 : 0;

`ifdef SIM
    always_comb begin
        if(!FIFO_if.rst_n)
        rst_assert: assert final(FIFO_if.data_out === 0 && wr_ptr ===0 && rd_ptr ===0 && count
=== 0)
            else $error("Assertion reset failed!");
    end

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

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

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

    property empty_prop ;
        @(posedge FIFO_if.clk)disable iff (!FIFO_if.rst_n) (count === 0) |-> FIFO_if.empty ;
    endproperty

    property full_prop ;
        @(posedge FIFO_if.clk)disable iff (!FIFO_if.rst_n) (count === FIFO_DEPTH) |->
FIFO_if.full ;
    endproperty

    property almostfull_prop ;
        @(posedge FIFO_if.clk)disable iff (!FIFO_if.rst_n) (count === FIFO_DEPTH-1) |->
FIFO_if.almostfull ;
    endproperty

    property almostempty_prop ;
        @(posedge FIFO_if.clk)disable iff (!FIFO_if.rst_n) (count === 1) |->
FIFO_if.almostempty ;
    endproperty

    property rd_wrap_around_prop ;
        @(posedge FIFO_if.clk)disable iff (!FIFO_if.rst_n) (rd_ptr === FIFO_DEPTH-1 &&
FIFO_if.rd_en && !FIFO_if.empty) |=> (rd_ptr === 0) ;
    endproperty

    property wr_wrap_around_prop ;
        @(posedge FIFO_if.clk)disable iff (!FIFO_if.rst_n) (wr_ptr === FIFO_DEPTH-1 &&
FIFO_if.wr_en && !FIFO_if.full) |=> (wr_ptr === 0) ;
    endproperty

    property pointer_threshold_prop ;
        @(posedge FIFO_if.clk) (wr_ptr < FIFO_DEPTH && rd_ptr < FIFO_DEPTH && count <=
FIFO_DEPTH ) ;
    endproperty


    wr_ack_assertion : assert property (wr_ack_prop)
        else $error("Assertion wr_ack failed!");
    overflow_assertion : assert property (overflow_prop)
        else $error("Assertion overflow failed!");
```

```
    undeflow_assertion : assert property (underflow_prop)
        else $error("Assertion undeflow failed!");
    empty_assertion : assert property (empty_prop)
        else $error("Assertion empty failed!");
    full_assertion : assert property (full_prop)
        else $error("Assertion full failed!");
    almostfull_assertion : assert property (almostfull_prop)
        else $error("Assertion almostfull failed!");
    almostempty_assertion : assert property (almostempty_prop)
        else $error("Assertion almostempty failed!");
    rd_wrap_around_assertion : assert property (rd_wrap_around_prop)
        else $error("Assertion rd_wrap failed!");
    wr_wrap_around_assertion : assert property (wr_wrap_around_prop)
        else $error("Assertion wr_wrap failed!");
    pointer_threshold_assertion : assert property (pointer_threshold_prop)
        else $error("Assertion pointer_threshold failed!");

    wr_ack_cover            : cover property (wr_ack_prop)            ;
    overflow_cover          : cover property (overflow_prop)          ;
    undeflow_cover          : cover property (underflow_prop)         ;
    empty_cover             : cover property (empty_prop)             ;
    full_cover              : cover property
(full_prop)             ;
    almostfull_cover        : cover property
(almostfull_prop)       ;
    almostempty_cover       : cover property (almostempty_prop)       ;
    rd_wrap_around_cover    : cover property (rd_wrap_around_prop)    ;
    wr_wrap_around_cover    : cover property (wr_wrap_around_prop)    ;
    pointer_threshold_cover : cover property (pointer_threshold_prop) ;
`endif


Endmodule
```

TOP module

```
module FIFO_top ();
    bit clk ;
    always #1 clk =! clk ;

    FIFO_interface FIFO_if (clk);
    FIFO my_FIFO (FIFO_if);
    FIFO_tb my_tb (FIFO_if);
    FIFO_monitor my_monitor (FIFO_if);
endmodule
```

Interface Module

```
import shared_pkg::*;

interface FIFO_interface(clk);

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

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

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

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

endinterface
```

Shared Pkg :

```
package shared_pkg;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    localparam max_fifo_addr = $clog2(FIFO_DEPTH);
    event done ;
    bit test_finished ;
    int error_count , correct_count;

endpackage
```

ScoreBoard :

```
package FIFO_scoreboard_pkg ;
import FIFO_transaction_pkg::*;
import shared_pkg::*;

class FIFO_scoreboard;
    logic [FIFO_WIDTH-1:0] data_out_ref;
    logic wr_ack_ref, overflow_ref ;
    logic full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref ;
    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;
task check_data(input FIFO_transaction FIFO_tr);
    reference_model(FIFO_tr);


    if (FIFO_tr.data_out !== data_out_ref) begin
        $display("Error in dataout");
        $stop();
        error_count ++ ;
    end else begin
        correct_count++ ;
    end

    if (FIFO_tr.wr_ack !== wr_ack_ref) begin
        $display("Error in wr_ack");
        $stop();
        error_count ++ ;
    end else begin
        correct_count++ ;
    end


    if (FIFO_tr.overflow !==overflow_ref) begin
        $display("Error in overflow");
        $stop();
        error_count ++ ;
    end else begin
        correct_count++ ;
    end

    if (FIFO_tr.full !== full_ref) begin
        $display("Error in full");
        $stop();
```

```
            error_count ++ ;
        end else begin
            correct_count++ ;
        end

        if (FIFO_tr.empty !== empty_ref) begin
            $display("Error in empty");
            $stop();
            error_count ++ ;
        end else begin
            correct_count++ ;
        end

        if (FIFO_tr.almostfull !== almostfull_ref) begin
            $display("Error in almostfull");
            $stop();
            error_count ++ ;
        end else begin
            correct_count++ ;
        end

        if (FIFO_tr.almostempty !== almostempty_ref) begin
            $display("Error in almostempty");
            $stop();
            error_count ++ ;
        end else begin
            correct_count++ ;
        end
        if (FIFO_tr.underflow !== underflow_ref) begin
            $display("Error in underflow");
            $stop();
            error_count ++ ;
        end else begin
            correct_count++ ;
        end
    endtask
    task reference_model(input FIFO_transaction FIFO_tr) ;
            if (!FIFO_tr.rst_n) begin
                wr_ptr = 0;
                count  = 0;
                rd_ptr = 0;
                {data_out_ref,wr_ack_ref, overflow_ref ,underflow_ref} = 0;
            end
            else  begin
                // writing
```

```
            if (FIFO_tr.wr_en && count < FIFO_DEPTH) begin
                mem[wr_ptr] = FIFO_tr.data_in;
                wr_ack_ref  = 1 ;
                wr_ptr = wr_ptr + 1;
                overflow_ref = 0;
            end else begin
                wr_ack_ref = 0;
                if (full_ref & FIFO_tr.wr_en)
                    overflow_ref = 1;
                else
                    overflow_ref = 0;
            end
            // reading
            if (FIFO_tr.rd_en && count != 0) begin
            data_out_ref = mem[rd_ptr];
            rd_ptr = rd_ptr + 1;
            underflow_ref = 0;
            end else begin
                if (empty_ref && FIFO_tr.rd_en)
                    underflow_ref = 1;
                else
                    underflow_ref = 0;
            end

            // counting
            if( ({FIFO_tr.wr_en, FIFO_tr.rd_en} == 2'b10) && !full_ref) begin
                if (count < FIFO_DEPTH)
                    count = count + 1;
                end
            else if ( ({FIFO_tr.wr_en, FIFO_tr.rd_en} == 2'b01) && !empty_ref)   begin
                if (count > 0)
                    count = count - 1;
            end else if ( ({FIFO_tr.wr_en, FIFO_tr.rd_en} == 2'b11) && empty_ref) begin
                count = count + 1;
            end  else if ( ({FIFO_tr.wr_en, FIFO_tr.rd_en} == 2'b11) && full_ref) begin
                count = count - 1;
            end
        end
    full_ref = (count == FIFO_DEPTH)? 1 : 0;
     empty_ref = (count == 0)? 1 : 0;
     almostfull_ref = (count == FIFO_DEPTH-1)? 1 : 0;
     almostempty_ref = (count == 1)? 1 : 0;
endtask
endclass
endpackage
```

Transaction pkg

```
package FIFO_transaction_pkg;
    import shared_pkg::*;

    class FIFO_transaction;

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


        function new (input integer arg_1 = 30 , input integer arg_2 = 70);
            RD_EN_ON_DIST = arg_1;
            WR_EN_ON_DIST = arg_2;
        endfunction

        constraint rst_con {
            rst_n dist {0:=2 , 1:=98} ;
        }

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

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

    endclass
endpackage
```

Coverage

```
package FIFO_coverage_pkg;
import FIFO_transaction_pkg::*;
    class FIFO_coverage;
        FIFO_transaction F_cvg_txn ;
        covergroup COV_gp ;
       Write_en    : coverpoint F_cvg_txn.wr_en       iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       Read_en     : coverpoint F_cvg_txn.rd_en       iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       write_ack   : coverpoint F_cvg_txn.wr_ack      iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       overflow    : coverpoint F_cvg_txn.overflow    iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       full        : coverpoint F_cvg_txn.full        iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       empty       : coverpoint F_cvg_txn.empty       iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       almostfull  : coverpoint F_cvg_txn.almostfull  iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       almostempty : coverpoint F_cvg_txn.almostempty iff (F_cvg_txn.rst_n) {option.weight =0 ; }
       underflow   : coverpoint F_cvg_txn.underflow   iff (F_cvg_txn.rst_n) {option.weight =0 ; }

      cross_wr_rd_wrack       : cross Write_en , Read_en , write_ack   {
          illegal_bins WR0_Ack1 = (binsof(Write_en) intersect {0} && binsof(write_ack) intersect {1});
      }
      cross_wr_rd_overflow    : cross Write_en , Read_en , overflow    {
          illegal_bins WR0_overflow1 = (binsof(Write_en) intersect {0} && binsof(overflow) intersect {1});
      }
      cross_wr_rd_underflow   : cross Write_en , Read_en , underflow   {
          illegal_bins WR0_overflow1 = (binsof(Read_en) intersect {0} && binsof(underflow) intersect {1});
      }
      cross_wr_rd_full        : cross Write_en , Read_en , full        {
          illegal_bins WR0_overflow1 = (binsof(Read_en) intersect {1} && binsof(full) intersect {1});
      }
      cross_wr_rd_empty       : cross Write_en , Read_en , empty       {
          illegal_bins WR0_overflow1 = (binsof(Write_en) intersect {1} && binsof(empty) intersect {1});
      }
        cross_wr_rd_almostfull  : cross Write_en , Read_en , almostfull  ;
        cross_wr_rd_almostempty : cross Write_en , Read_en , almostempty ;
        endgroup
        function new ();
               COV_gp = new() ;
        endfunction
        function void sample_data(input FIFO_transaction F_txn);
            F_cvg_txn = F_txn ;
            COV_gp.sample();
        endfunction
    endclass
endpackage
```

Monitor

```
import shared_pkg::*;
import FIFO_transaction_pkg::*;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;
module FIFO_monitor (FIFO_interface.MONITOR FIFO_if);
    FIFO_transaction trans  = new ;
    FIFO_scoreboard  score  = new ;
    FIFO_coverage  coverage = new ;

    initial begin
        forever begin
            wait (done.triggered);
             @(negedge FIFO_if.clk)                 ;

            trans.data_in     = FIFO_if.data_in     ;
            trans.rst_n       = FIFO_if.rst_n       ;
            trans.wr_en       = FIFO_if.wr_en       ;
            trans.rd_en       = FIFO_if.rd_en       ;
            trans.data_out    = FIFO_if.data_out    ;
            trans.wr_ack      = FIFO_if.wr_ack      ;
            trans.overflow    = FIFO_if.overflow    ;
            trans.full        = FIFO_if.full        ;
            trans.empty       = FIFO_if.empty       ;
            trans.almostfull  = FIFO_if.almostfull  ;
            trans.almostempty = FIFO_if.almostempty ;
            trans.underflow   = FIFO_if.underflow   ;
            fork
                begin
                    coverage.sample_data(trans);
                end
                begin
                    score.check_data(trans);
                end
            join
            if(test_finished) begin
                $display("Finished Successfully");
                $display("Error count = %0d , correct count = %0d" , error_count , correct_count);
                $stop;
            end
        end
    end
endmodule
```

Testbench :

```
import shared_pkg::*;
import FIFO_transaction_pkg::*;
import FIFO_scoreboard_pkg::*;
module FIFO_tb (FIFO_interface.TEST FIFO_if);

FIFO_transaction myclass = new ;
FIFO_scoreboard myscore = new ;

initial begin
    assert_reset();
        // writing only
     for (int i=0; i<10; i++) begin
            assert(myclass.randomize());
            FIFO_if.data_in = myclass.data_in ;
            FIFO_if.rst_n   = 1   ;
            FIFO_if.wr_en   = 1   ;
            FIFO_if.rd_en   = 0   ;
             -> done ;
            @(negedge FIFO_if.clk);
    end

    // both reading and writing to check reading only done in 1st cycle
    for (int i=0; i<10; i++) begin
            assert(myclass.randomize());
            FIFO_if.data_in = myclass.data_in ;
            FIFO_if.rst_n   = 1   ;
            FIFO_if.wr_en   = 1   ;
            FIFO_if.rd_en   = 1   ;
             -> done ;
            @(negedge FIFO_if.clk);
    end

       // writing only
     for (int i=0; i<10; i++) begin
            assert(myclass.randomize());
            FIFO_if.data_in = myclass.data_in ;
            FIFO_if.rst_n   = 1   ;
            FIFO_if.wr_en   = 1   ;
            FIFO_if.rd_en   = 0   ;
             -> done ;
            @(negedge FIFO_if.clk);
    end
```

```
//reading only
for (int i=0; i<10; i++) begin
        assert(myclass.randomize());
        FIFO_if.data_in = myclass.data_in ;
        FIFO_if.rst_n   = 1   ;
        FIFO_if.wr_en   = 0   ;
        FIFO_if.rd_en   = 1   ;
         -> done ;
        @(negedge FIFO_if.clk);
end

// both reading and writing
for (int i=0; i<10; i++) begin
        assert(myclass.randomize());
        FIFO_if.data_in = myclass.data_in ;
        FIFO_if.rst_n   = 1   ;
        FIFO_if.wr_en   = 1   ;
        FIFO_if.rd_en   = 1   ;
         -> done ;
        @(negedge FIFO_if.clk);
end
// no reading nor writing

 for (int i=0; i<10; i++) begin
        assert(myclass.randomize());
        FIFO_if.data_in = myclass.data_in ;
        FIFO_if.rst_n   = 1   ;
        FIFO_if.wr_en   = 0   ;
        FIFO_if.rd_en   = 0   ;
         -> done ;
        @(negedge FIFO_if.clk);
end

// randomize everything

for (int i=0; i<10000; i++) begin
        assert(myclass.randomize());
        FIFO_if.data_in = myclass.data_in ;
        FIFO_if.rst_n   = myclass.rst_n   ;
        FIFO_if.wr_en   = myclass.wr_en   ;
        FIFO_if.rd_en   = myclass.rd_en   ;
         -> done ;
        @(negedge FIFO_if.clk);
end
// end
```

```
    -> done ;
    test_finished = 1 ;

end


task assert_reset();
    FIFO_if.rst_n = 0 ;
    -> done ;
    @(negedge FIFO_if.clk);
    FIFO_if.rst_n = 1 ;
endtask
endmodule
```

Do File :

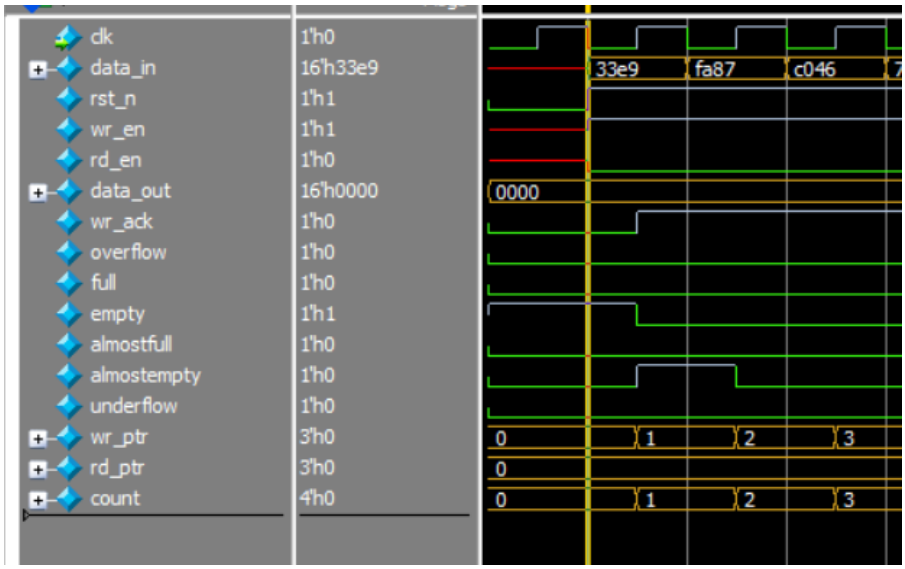
```
vlib work
vlog -f src_files.list +define+SIM   +cover -covercells
vsim -voptargs=+acc work.FIFO_top  -cover
add wave /FIFO_top/FIFO_if/* /FIFO_top/my_FIFO/*
coverage save FIFO.ucdb -onexit
run -all

coverage exclude -src FIFO.sv -line 24 -code c
coverage exclude -src FIFO.sv -line 42 -code c

quit -sim
vcover report FIFO.ucdb -details -annotate -all -output FIFO_coverage.txt

```

Source files :

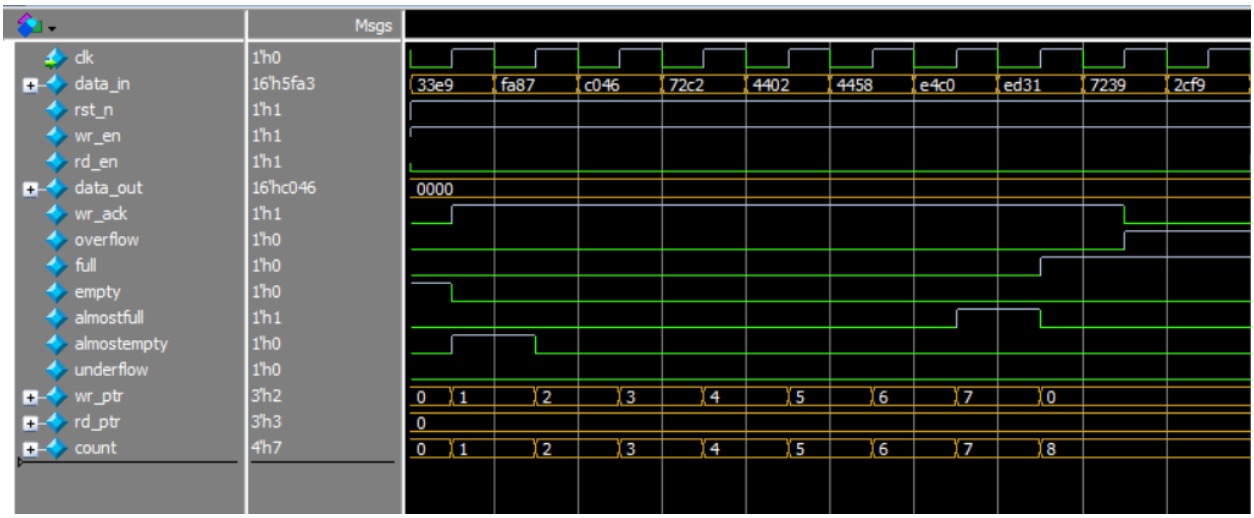
```
shared_pkg.sv
FIFO.sv
FIFO_scoreboard.sv
FIFO_interface.sv
FIFO_transaction.sv
FIFO_coverage.sv
FIFO_monitor.sv
FIFO_tb.sv
FIFO_top.sv
```

WaveForm Snippets

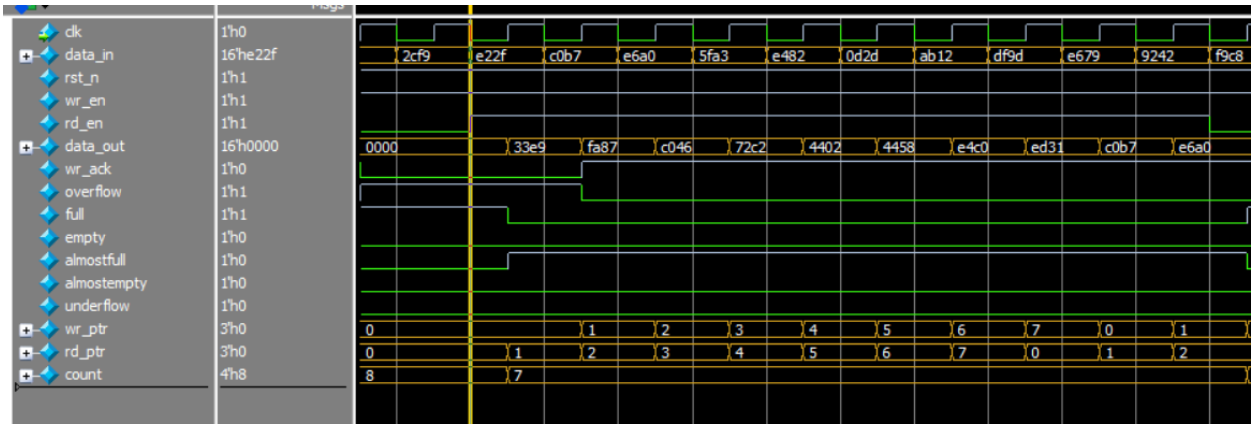
1. At reset



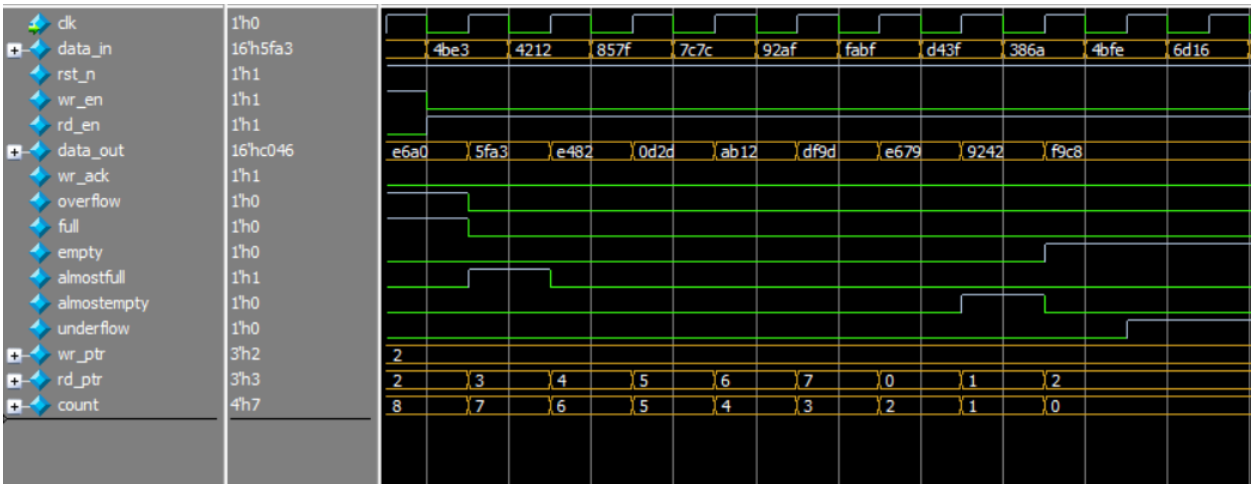
2. At writing only



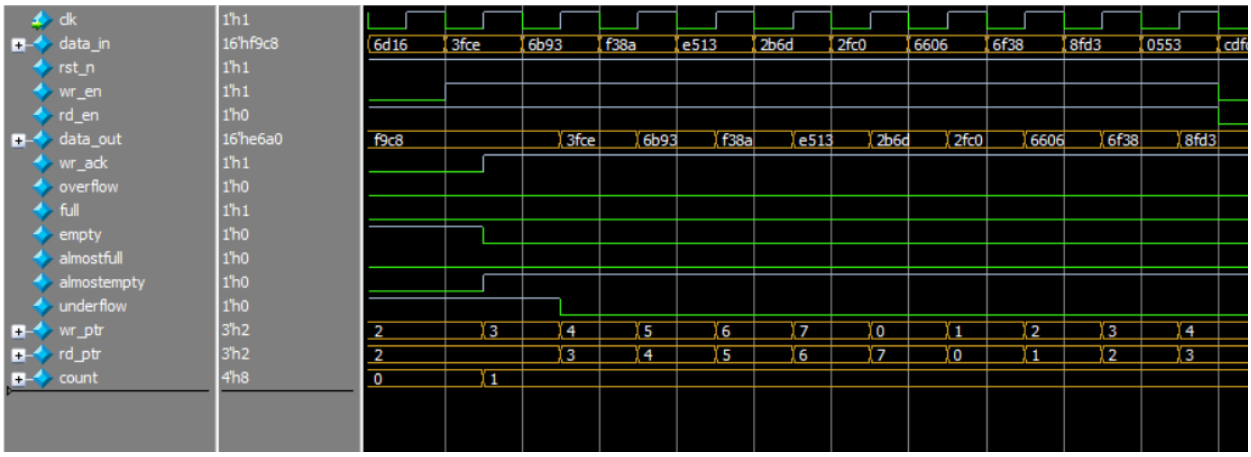
3. At both writing and reading after full



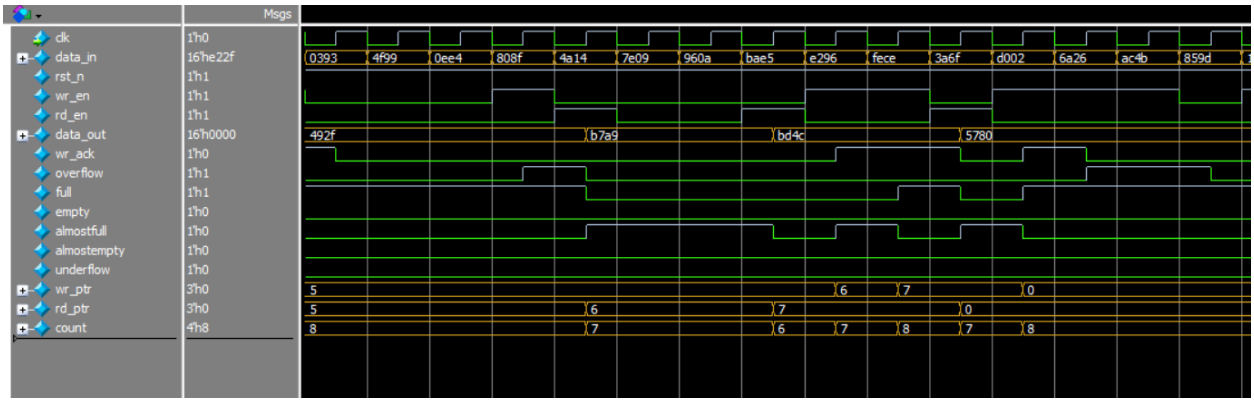
4. At reading only :



5. At both writing and reading after empty



6. At Randomization :



Coverage Report :

Code coverage

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        27        27         0   100.00%
```

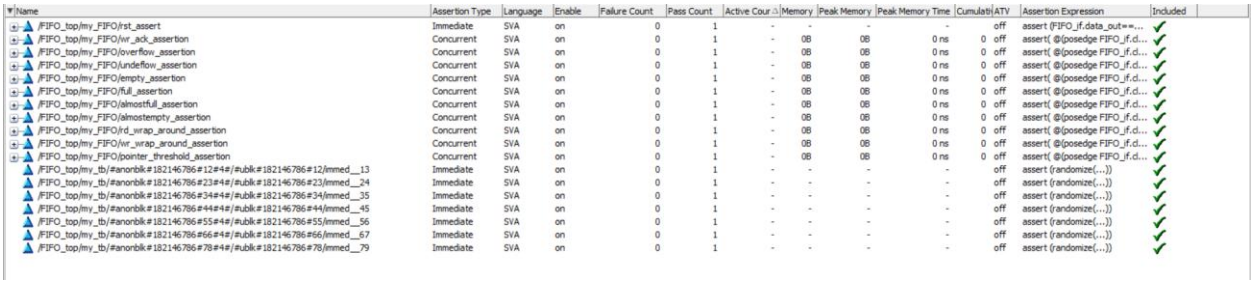
```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                      20        20         0   100.00%
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      31        31         0   100.00%
```

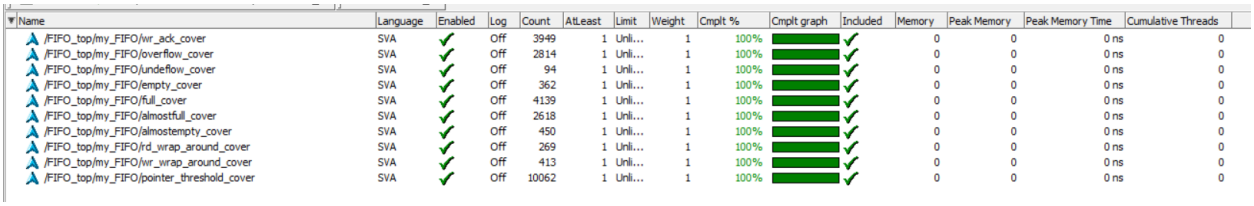
```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         20        20         0   100.00%
```

Assertion Coverage

From Questasim



Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Cour	Memory	Peak Memory	Peak Memory Time	Cumulati	ATV	Assertion Expression	Included
/FIFO_top/my_FIFO/rst_assert	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.data_out==...	✓
/FIFO_top/my_FIFO/wr_ack_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/overflow_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/undeflow_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/empty_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/full_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/almostfull_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/almostempty_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/rd_wrap_around_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/wr_wrap_around_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_FIFO/pointer_threshold_assertion	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.d...	✓
/FIFO_top/my_tb/#anonblk#182146786#12#4#/#ublk#182146786#12/immed__13	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/my_tb/#anonblk#182146786#23#4#/#ublk#182146786#23/immed__24	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/my_tb/#anonblk#182146786#34#4#/#ublk#182146786#34/immed__35	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/my_tb/#anonblk#182146786#44#4#/#ublk#182146786#44/immed__45	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/my_tb/#anonblk#182146786#55#4#/#ublk#182146786#55/immed__56	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/my_tb/#anonblk#182146786#66#4#/#ublk#182146786#66/immed__67	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/my_tb/#anonblk#182146786#78#4#/#ublk#182146786#78/immed__79	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓



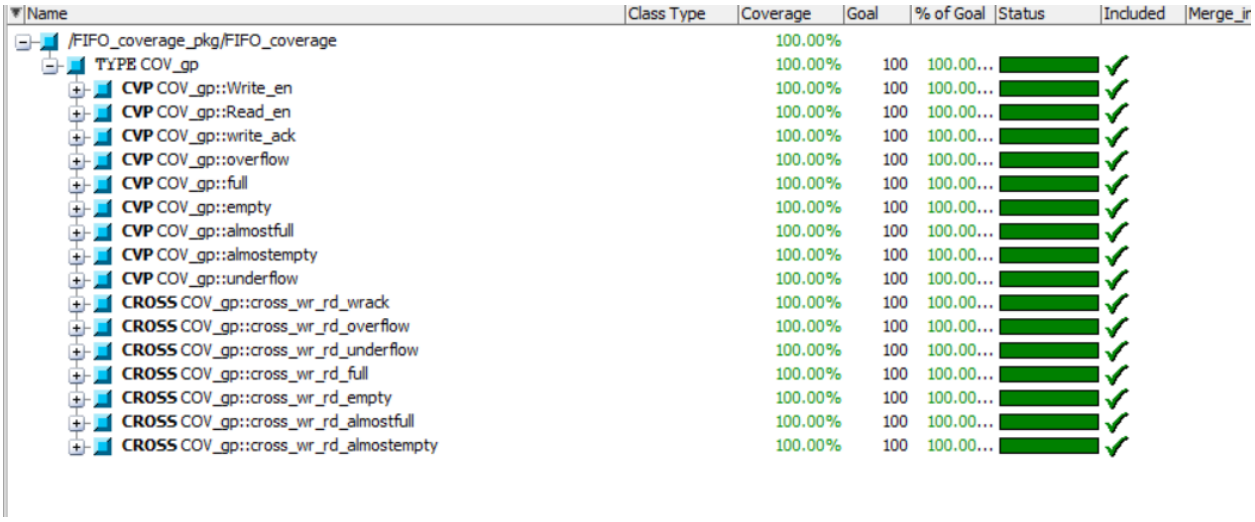
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_top/my_FIFO/wr_ack_cover	SVA	✓	Off	3949	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/overflow_cover	SVA	✓	Off	2814	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/undeflow_cover	SVA	✓	Off	94	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/empty_cover	SVA	✓	Off	362	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/full_cover	SVA	✓	Off	4139	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/almostfull_cover	SVA	✓	Off	2618	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/almostempty_cover	SVA	✓	Off	450	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/rd_wrap_around_cover	SVA	✓	Off	269	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/wr_wrap_around_cover	SVA	✓	Off	413	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/my_FIFO/pointer_threshold_cover	SVA	✓	Off	10062	1	Unli...	1	100%		✓	0	0	0 ns	0

From Text file

```
Assertion Coverage:
    Assertions                      11        11         0   100.00%
--------------------------------------------------------------------------------
Name                 File(Line)                    Failure      Pass
                                                   Count        Count
--------------------------------------------------------------------------------
/FIFO_top/my_FIFO/rst_assert
                     FIFO.sv(78)                         0          1
/FIFO_top/my_FIFO/wr_ack_assertion
                     FIFO.sv(123)                        0          1
/FIFO_top/my_FIFO/overflow_assertion
                     FIFO.sv(125)                        0          1
/FIFO_top/my_FIFO/undeflow_assertion
                     FIFO.sv(127)                        0          1
/FIFO_top/my_FIFO/empty_assertion
                     FIFO.sv(129)                        0          1
/FIFO_top/my_FIFO/full_assertion
                     FIFO.sv(131)                        0          1
/FIFO_top/my_FIFO/almostfull_assertion
                     FIFO.sv(133)                        0          1
/FIFO_top/my_FIFO/almostempty_assertion
                     FIFO.sv(135)                        0          1
/FIFO_top/my_FIFO/rd_wrap_around_assertion
                     FIFO.sv(137)                        0          1
/FIFO_top/my_FIFO/wr_wrap_around_assertion
                     FIFO.sv(139)                        0          1
/FIFO_top/my_FIFO/pointer_threshold_assertion
                     FIFO.sv(141)                        0          1
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        27        27         0   100.00%
```

```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/my_FIFO/wr_ack_cover          FIFO   Verilog  SVA  FIFO.sv(144)    3949 Covered
/FIFO_top/my_FIFO/overflow_cover        FIFO   Verilog  SVA  FIFO.sv(145)    2814 Covered
/FIFO_top/my_FIFO/undeflow_cover        FIFO   Verilog  SVA  FIFO.sv(146)      94 Covered
/FIFO_top/my_FIFO/empty_cover           FIFO   Verilog  SVA  FIFO.sv(147)     362 Covered
/FIFO_top/my_FIFO/full_cover            FIFO   Verilog  SVA  FIFO.sv(148)    4139 Covered
/FIFO_top/my_FIFO/almostfull_cover      FIFO   Verilog  SVA  FIFO.sv(149)    2618 Covered
/FIFO_top/my_FIFO/almostempty_cover     FIFO   Verilog  SVA  FIFO.sv(150)     450 Covered
/FIFO_top/my_FIFO/rd_wrap_around_cover  FIFO   Verilog  SVA  FIFO.sv(151)     269 Covered
/FIFO_top/my_FIFO/wr_wrap_around_cover  FIFO   Verilog  SVA  FIFO.sv(152)     413 Covered
/FIFO_top/my_FIFO/pointer_threshold_cover
                                        FIFO   Verilog  SVA  FIFO.sv(153)   10062 Covered
```

Functional Coverage :




```
================================================================================
=== Instance: /FIFO_coverage_pkg
=== Design Unit: work.FIFO_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                 1        na        na   100.00%
        Coverpoints/Crosses    16        na        na        na
            Covergroup Bins    64        64         0   100.00%
```