Synchronous FIFO Verification Project

By: Yousef Gamal

Under the Supervision of Eng. Kareem Waseem

Contents

1. Design RTL including the assertions

```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: FIFO Design
//
////////////////////////////////////////////////////////////////////////////////
module FIFO( FIFO_if.DUT FIFO_if_obj );

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

/*------------------------------------------------------------------------------*/
logic rst_n, wr_en, rd_en , wr_ack ;
logic full, empty, almostfull, almostempty, underflow , overflow ;

assign full = FIFO_if_obj.full ;
assign empty = FIFO_if_obj.empty ;
assign almostempty = FIFO_if_obj.almostempty ;
assign almostfull = FIFO_if_obj.almostfull ;
assign overflow = FIFO_if_obj.overflow ;
assign underflow = FIFO_if_obj.underflow ;
assign rst_n = FIFO_if_obj.rst_n ;
assign wr_en = FIFO_if_obj.wr_en ;
assign rd_en = FIFO_if_obj.rd_en ;
assign wr_ack = FIFO_if_obj.wr_ack ;

/*-------------------------------------------assertions-----------------------------------
--------*/
`ifdef SIM
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_if_obj.FIFO_DEPTH )
        begin
            assert_full: assert (full && !empty && !almostempty && !almostfull ) ;
            cover_full : cover  (full && !empty && !almostempty && !almostfull)
;
        end

        else if( count == (FIFO_if_obj.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 FIFO_if_obj.clk) disable iff(!rst_n) (rd_en && (count != 0) && rd_ptr < 7 )
                  |=> ( (rd_ptr == ($past(rd_ptr) + 1)) % FIFO_if_obj.FIFO_DEPTH );
endproperty

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

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

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

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

property wr_ack_0 ;
    @(posedge FIFO_if_obj.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);
```

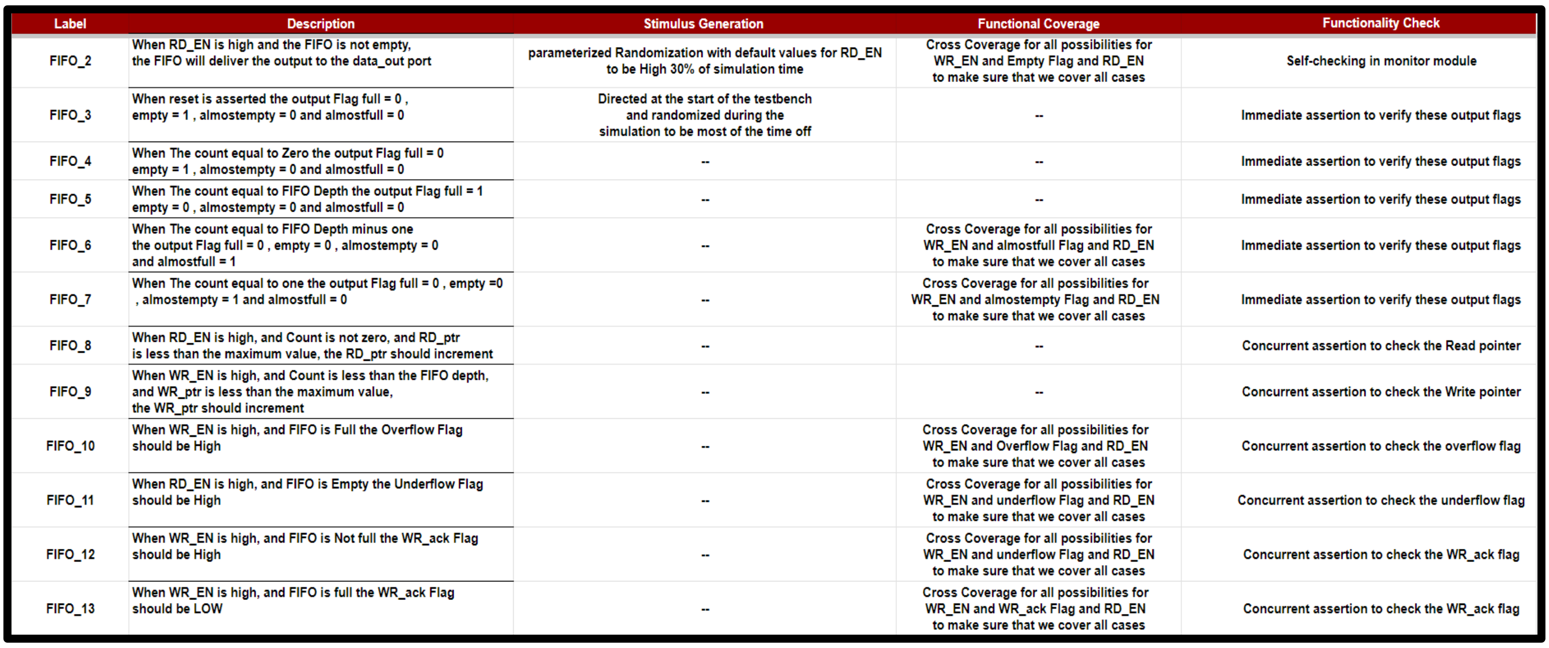
```
assert_overflow : assert property (prop_overflow) ;
cover_overflow  : cover  property (prop_overflow) ;

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) ;

`endif

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

always @(posedge FIFO_if_obj.clk or negedge FIFO_if_obj.rst_n)
begin
    if (!FIFO_if_obj.rst_n)
        begin
            rd_ptr <= 0;
            FIFO_if_obj.underflow <= 0; /* added to cleared with reset */
        end
    else if (FIFO_if_obj.rd_en && (count != 0) )
        begin
          FIFO_if_obj.data_out <= mem[rd_ptr];
          rd_ptr <= rd_ptr + 1;
          FIFO_if_obj.underflow <= 0;/*added to be cleared if we already read from FIFO */
        end
    else  /* added to be sequential output not combinational */
```

```
        if (FIFO_if_obj.empty && FIFO_if_obj.rd_en)
            FIFO_if_obj.underflow <= 1;

       else
            FIFO_if_obj.underflow <= 0;
end

// always block specialized for counter signal
always @(posedge FIFO_if_obj.clk or negedge FIFO_if_obj.rst_n)
begin
    if (!FIFO_if_obj.rst_n)
        count <= 0;
    else if (FIFO_if_obj.wr_en && FIFO_if_obj.rd_en)
        begin
            if (FIFO_if_obj.empty)
                count <= count + 1;  /* Prioritize write if FIFO is empty */

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

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

    end

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

endmodule
```

2. Detected Bugs

- Clear the **overflow** flag upon reset.
- Clear the **underflow** flag upon reset.
- Clear the **wr_ack** flag upon reset.
- Ensure the **overflow** flag remains LOW when the **wr_ack** flag is HIGH.
- Set the **underflow** flag to LOW when the FIFO delivers data to the **data_out** port.
- Sequentially update the **overflow** flag based on the **rd_en** signal and the **empty** flag.
- Set the **almostfull** flag to HIGH when the count reaches one less than the FIFO depth.

3. 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

4. Verification codes

4.1. Interface package

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

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

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

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

4.2. Shared package

```
package shared_pkg_0;

bit test_finished;
int error_count;
int correct_count;

endpackage
```

4.3. Transaction package

```
package Transaction_pkg ;
class FIFO_transaction  #(parameter FIFO_WIDTH = 16) ;

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

constraint rst_N_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 } ;}

function new ( 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

endclass

endpackage
```

4.4. Coverage package

```
package coverage_pkg ;

import Transaction_pkg::* ;

class FIFO_coverage ;
FIFO_transaction F_cvg_txn   ;

covergroup code_cov ;

cross_coverage_full : cross  F_cvg_txn.wr_en, F_cvg_txn.rd_en, F_cvg_txn.full;

cross_coverage_empty : cross  F_cvg_txn.wr_en, F_cvg_txn.rd_en, F_cvg_txn.empty ;

cross_coverage_wr_ack : cross F_cvg_txn.wr_en, F_cvg_txn.rd_en, F_cvg_txn.wr_ack ;

cross_coverage_overflow : cross  F_cvg_txn.wr_en, F_cvg_txn.rd_en, F_cvg_txn.overflow   ;
```

```
cross_coverage_underflow : cross  F_cvg_txn.wr_en, F_cvg_txn.rd_en, F_cvg_txn.underflow ;

cross_coverage_almostfull : cross  F_cvg_txn.wr_en, F_cvg_txn.rd_en, F_cvg_txn.almostfull
;

cross_coverage_almostempty : cross  F_cvg_txn.wr_en, F_cvg_txn.rd_en,
F_cvg_txn.almostempty ;

endgroup


function void sample_data(input FIFO_transaction F_txn) ;
    F_cvg_txn = F_txn; /* because the sampling is done for F_cvg_txn */
    code_cov.sample();
endfunction

function new();
F_cvg_txn = new ; /* create an object from transaction class */
code_cov  = new ; /* create an object from covergroup */
endfunction
endclass
endpackage
```

4.5. Testbench

```
import Transaction_pkg::* ;
import shared_pkg_0::* ;

module FIFO_TB( FIFO_if.TEST FIFO_if_obj) ;

FIFO_transaction tr ;
    initial
    begin
        tr = new ;

        $display("##########################################################");
        $display("------------------------start simulation------------------");

        FIFO_if_obj.rst_n = 0 ;
        #18 ;
        FIFO_if_obj.rst_n = 1 ;


        // Write phase
        tr.constraint_mode(0);
        tr.rst_n = 1 ; tr.wr_en = 1; tr.rd_en = 0;
        tr.rand_mode(0) ;
        tr.data_in.rand_mode(1);

        repeat (1000)
        begin
            assertions_TB_wr : assert(tr.randomize);
            connect_if_to_class();
            #20;
        end

        // Read phase
        tr.constraint_mode(0);
        tr.rst_n = 1 ; tr.wr_en = 0; tr.rd_en = 1;
        tr.rand_mode(0) ;

        repeat (1000)
        begin
            assertions_TB_rd : assert(tr.randomize);
            connect_if_to_class();
            #20;
        end

        tr.constraint_mode(1);
        tr.rand_mode(1);

        repeat (10000)
        begin   // Mixed
           assertions_TB_Mix: assert( tr.randomize ) ;
            connect_if_to_class();
            #20 ;
        end


        test_finished = 1 ;

    end

function void connect_if_to_class();
    FIFO_if_obj.data_in  = tr.data_in;
    FIFO_if_obj.rst_n    = tr.rst_n;
    FIFO_if_obj.wr_en    = tr.wr_en;
    FIFO_if_obj.rd_en    = tr.rd_en;
endfunction
endmodule
```

4.6. Scoreboard package

```
package Scoreboard_pkg ;

import shared_pkg_0::* ;
import Transaction_pkg::* ;

class FIFO_Scoreboard #(parameter FIFO_WIDTH = 16 , FIFO_DEPTH = 8 ) ;

logic [FIFO_WIDTH-1 : 0] data_out_ref ;

bit [$clog2(FIFO_DEPTH) : 0] counter ;
int My_ref_Queue[$] ;
FIFO_transaction F_tr = new();

function void check_data(input FIFO_transaction F_tr);

    reference_model(F_tr);
```

```
        if (F_tr.data_out != data_out_ref)
        begin
            $display("At time = %0t , the output of the DUT (data_out = %0d) , doesn't
Match with the Golden model output (data_out_ref =
%0d)",$time,F_tr.data_out,data_out_ref);
            error_count++;
            $stop;
        end
    else
    begin
        correct_count++;
    end

endfunction

function void reference_model(input FIFO_transaction F_txn);
fork

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

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

    end

    begin //  third thread -> Counter updating
        if (!F_txn.rst_n)
            counter <= 0;
        else
            begin
                casex ({F_txn.wr_en, F_txn.rd_en, F_txn.full, F_txn.empty})
                    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
endfunction

function new() ;
    error_count = 0 ;
    correct_count = 0 ;
endfunction

endclass
endpackage
```

4.7. Monitor module

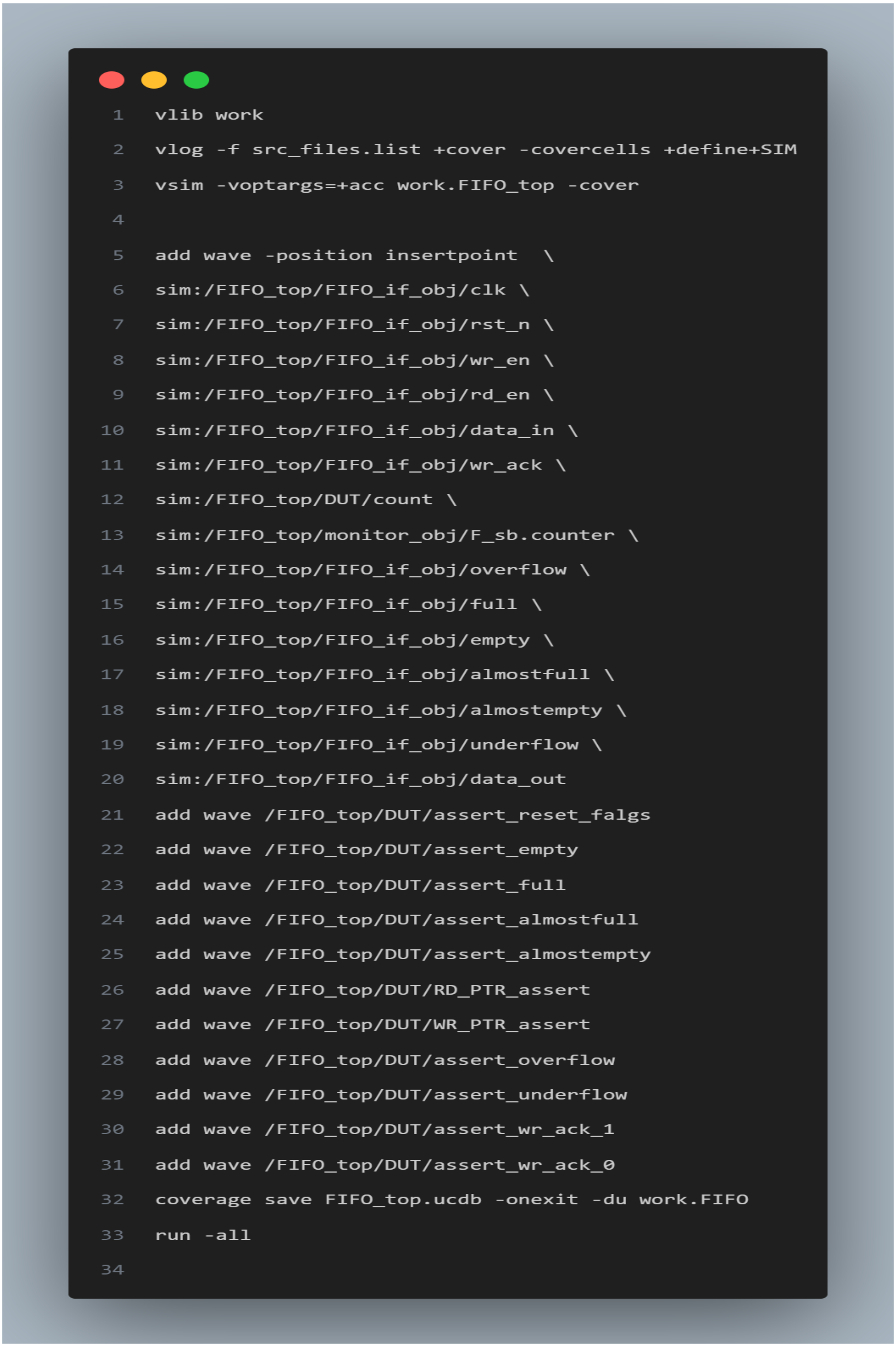
```
import coverage_pkg::* ;      // import coverge package
import Scoreboard_pkg::* ;    // import scoreboard package
import Transaction_pkg::* ;   // import trnasaction package
import shared_pkg_0::* ;      // import shared package

module monitor (FIFO_if.monitor FIFO_if_obj) ;
FIFO_transaction F_tr  ;
FIFO_Scoreboard  F_sb  ;
FIFO_coverage    F_cov ;
initial
    begin
    F_tr  =  new() ;  // create an object of class FIFO_transaction
    F_sb  =  new() ;  // create an object of class FIFO_Scoreboard
    F_cov =  new() ;  // create an object of class FIFO_coverage

        forever
        begin
                @(negedge FIFO_if_obj.clk) ;
                connect_class_to_if() ; /* pass the randomized Data to class to get
sampled and compared to golden model */
                fork
                    begin
                            F_cov.sample_data(F_tr)  ; /* Sample the data */
                    end

                    begin
                            F_sb.check_data( F_tr ) ;   /* Compare with Golden Model
*/
```

```
                    end
                join

                if( test_finished )
                begin
                    $display("correct count = 0d%0d" , correct_count);
                    $display("Error   count = 0d%0d" , error_count);
                    $display("Test finished, stopping simulation");
                    $display("------------------------END   simulation-----------------
");

                    $display("##########################################################")
;


                    $stop;
                end

        end
    end









function void connect_class_to_if( );
    F_tr.data_in     = FIFO_if_obj.data_in;
    F_tr.rst_n       = FIFO_if_obj.rst_n;
    F_tr.wr_en       = FIFO_if_obj.wr_en;
    F_tr.rd_en       = FIFO_if_obj.rd_en;
    F_tr.data_out    = FIFO_if_obj.data_out;
    F_tr.wr_ack      = FIFO_if_obj.wr_ack;
    F_tr.overflow    = FIFO_if_obj.overflow;
    F_tr.underflow   = FIFO_if_obj.underflow;
    F_tr.full        = FIFO_if_obj.full;
    F_tr.almostfull  = FIFO_if_obj.almostfull;
    F_tr.empty       = FIFO_if_obj.empty;
    F_tr.almostempty = FIFO_if_obj.almostempty;
endfunction

endmodule
```

4.8. Top Module

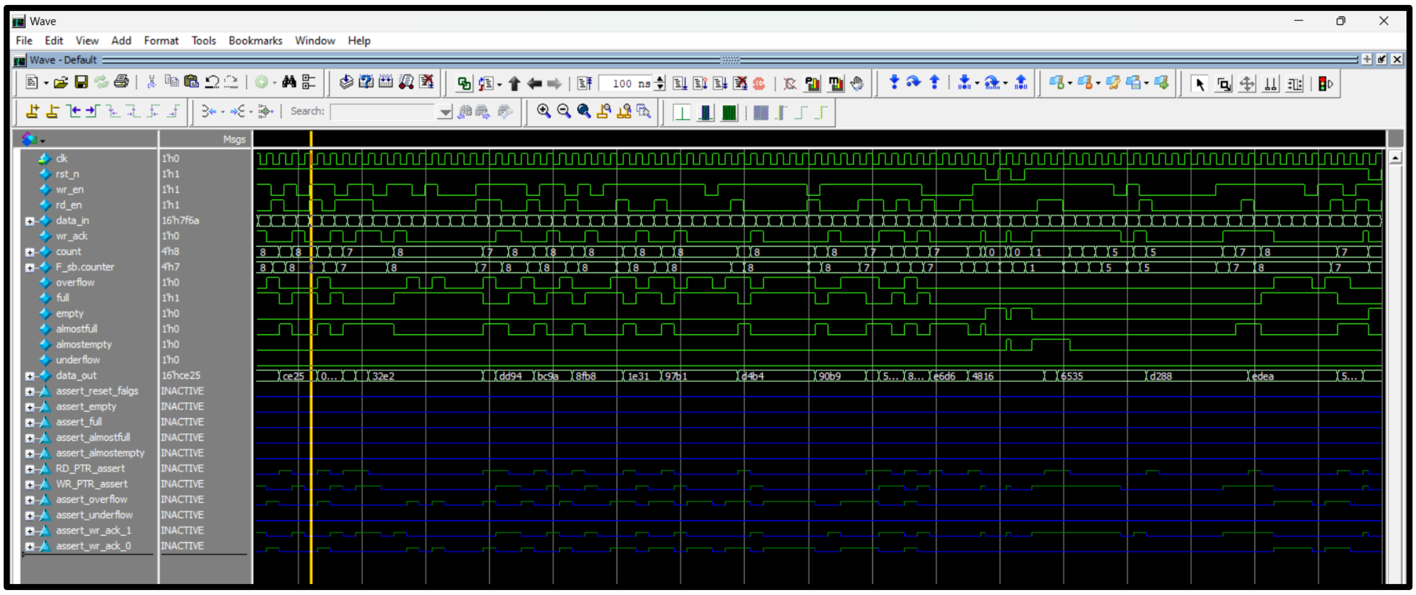
```
module FIFO_top ;
    bit clk ;  // Clock signal

    // Initial block to generate the clock signal
    initial
    begin
        clk = 0 ;  // Initialize clock to 0
        forever
        begin
            #10 clk = ~clk ;  // Toggle clock every 10 time units
        end
    end

    // Instantiate the FIFO interface
    FIFO_if FIFO_if_obj(clk) ;

    // Instantiate the FIFO design under test (DUT)
    FIFO DUT (FIFO_if_obj) ;

    // Instantiate the FIFO testbench
    FIFO_TB TB (FIFO_if_obj) ;

    // Instantiate a monitor for observing signals
    monitor monitor_obj (FIFO_if_obj) ;
endmodule
```

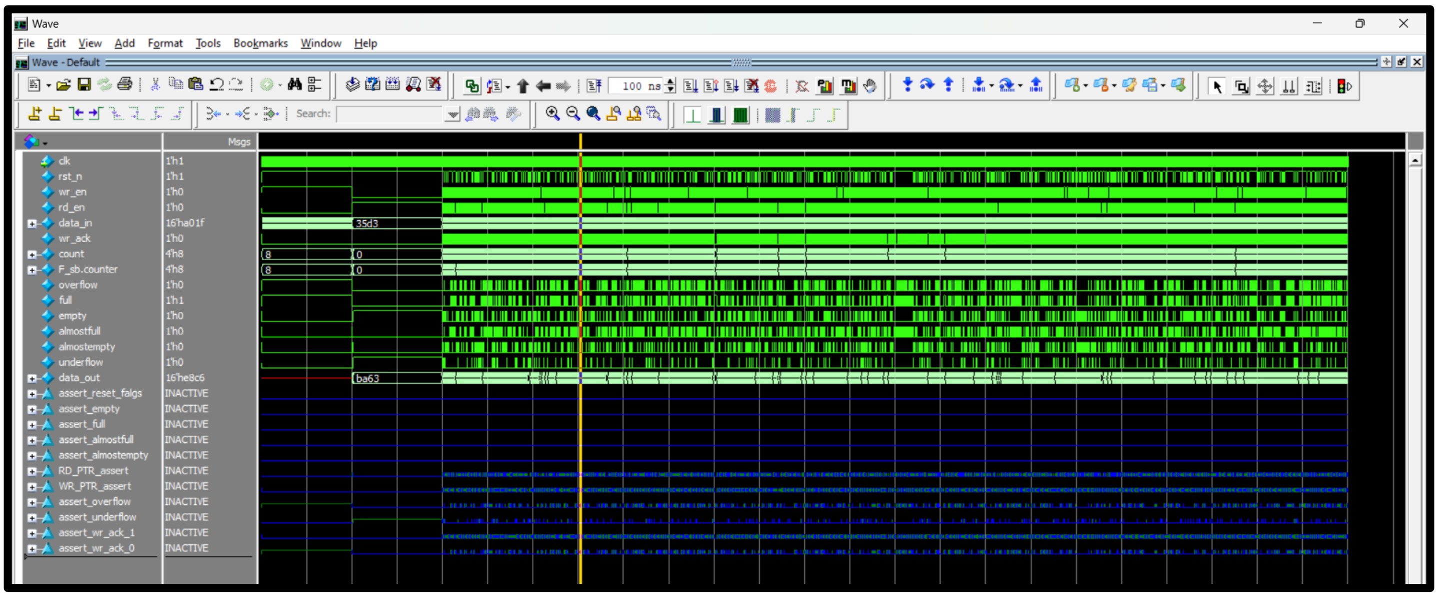
5. Do file



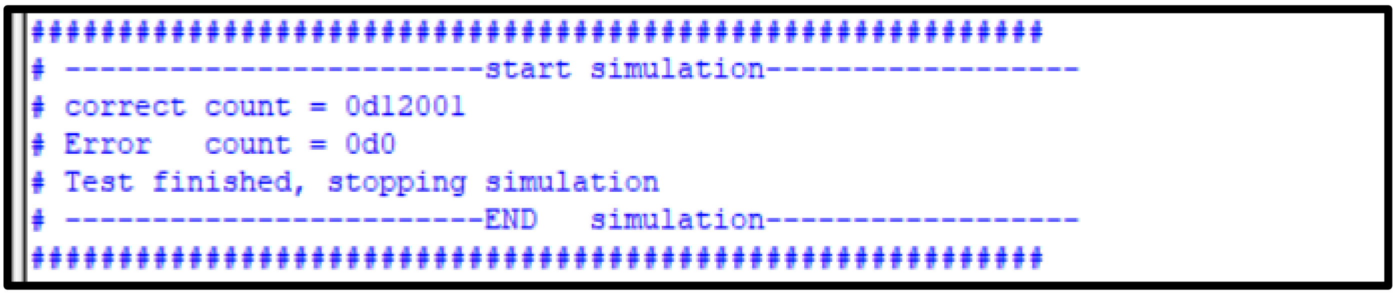
```
vlib work
vlog -f src_files.list +cover -covercells +define+SIM
vsim -voptargs=+acc work.FIFO_top -cover

add wave -position insertpoint  \
sim:/FIFO_top/FIFO_if_obj/clk \
sim:/FIFO_top/FIFO_if_obj/rst_n \
sim:/FIFO_top/FIFO_if_obj/wr_en \
sim:/FIFO_top/FIFO_if_obj/rd_en \
sim:/FIFO_top/FIFO_if_obj/data_in \
sim:/FIFO_top/FIFO_if_obj/wr_ack \
sim:/FIFO_top/DUT/count \
sim:/FIFO_top/monitor_obj/F_sb.counter \
sim:/FIFO_top/FIFO_if_obj/overflow \
sim:/FIFO_top/FIFO_if_obj/full \
sim:/FIFO_top/FIFO_if_obj/empty \
sim:/FIFO_top/FIFO_if_obj/almostfull \
sim:/FIFO_top/FIFO_if_obj/almostempty \
sim:/FIFO_top/FIFO_if_obj/underflow \
sim:/FIFO_top/FIFO_if_obj/data_out
add wave /FIFO_top/DUT/assert_reset_falgs
add wave /FIFO_top/DUT/assert_empty
add wave /FIFO_top/DUT/assert_full
add wave /FIFO_top/DUT/assert_almostfull
add wave /FIFO_top/DUT/assert_almostempty
add wave /FIFO_top/DUT/RD_PTR_assert
add wave /FIFO_top/DUT/WR_PTR_assert
add wave /FIFO_top/DUT/assert_overflow
add wave /FIFO_top/DUT/assert_underflow
add wave /FIFO_top/DUT/assert_wr_ack_1
add wave /FIFO_top/DUT/assert_wr_ack_0
coverage save FIFO_top.ucdb -onexit -du work.FIFO
run -all

```

6. Waveform



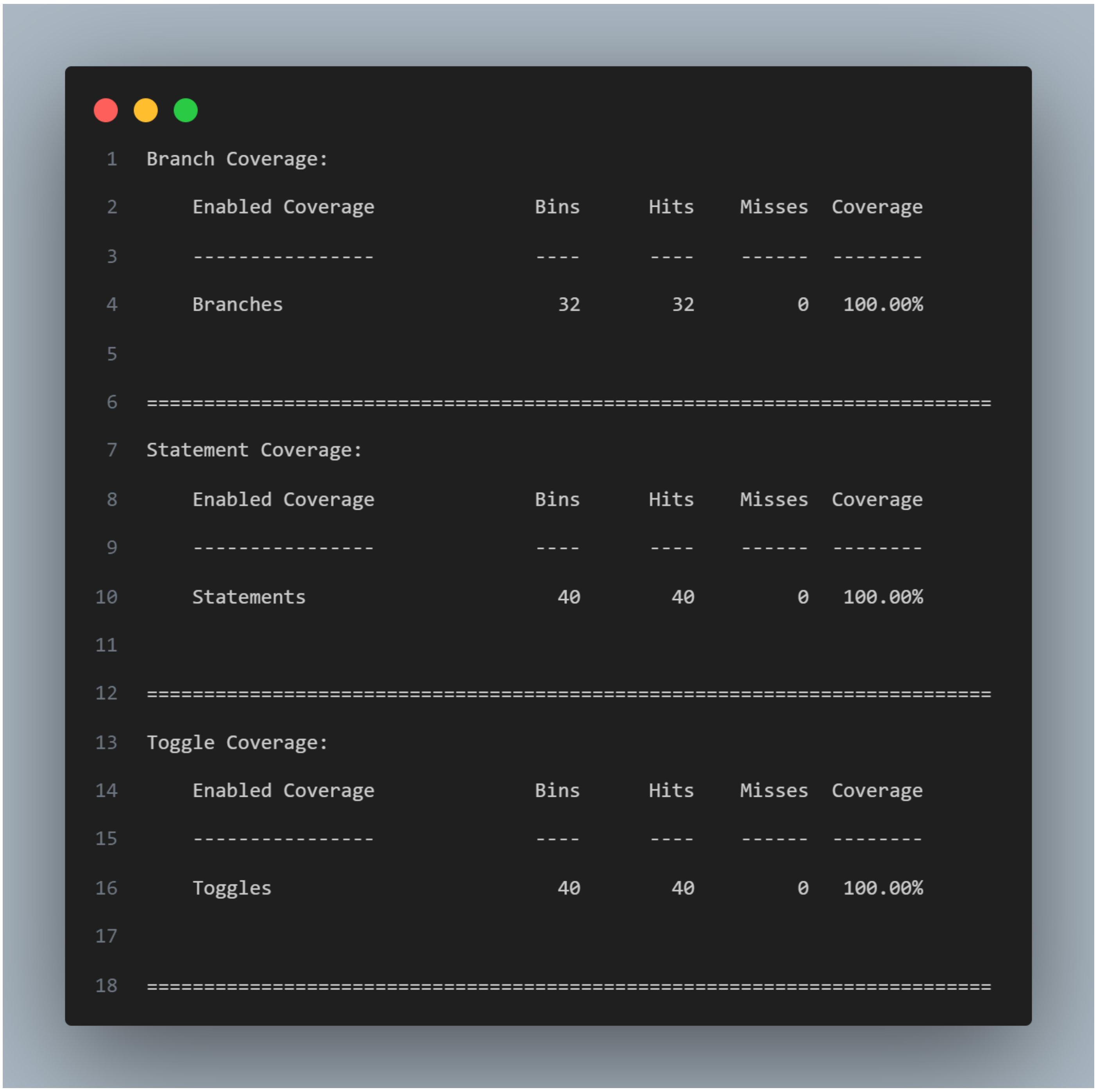


7. Transcript



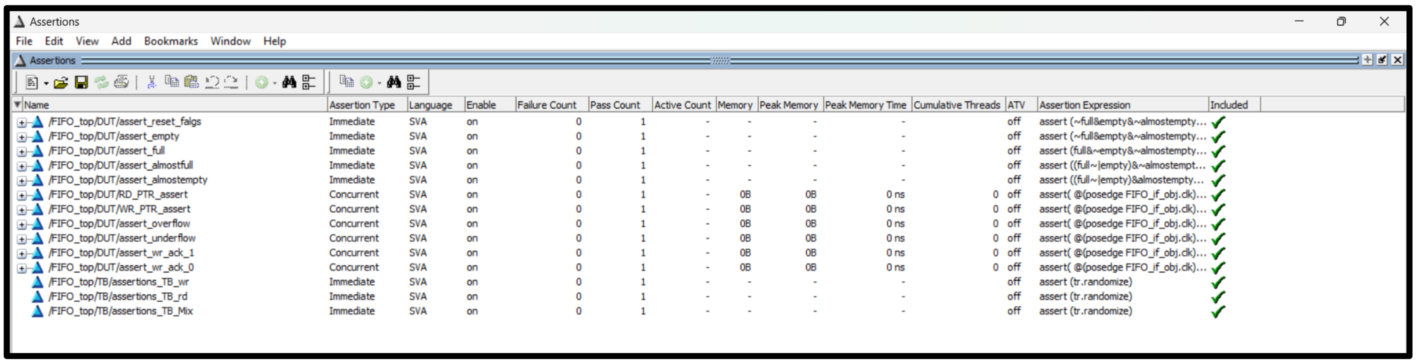
```
###########################################################
# ------------------------start simulation------------------
# correct count = 0d12001
# Error   count = 0d0
# Test finished, stopping simulation
# ------------------------END   simulation------------------
###########################################################
```

8. Code coverage

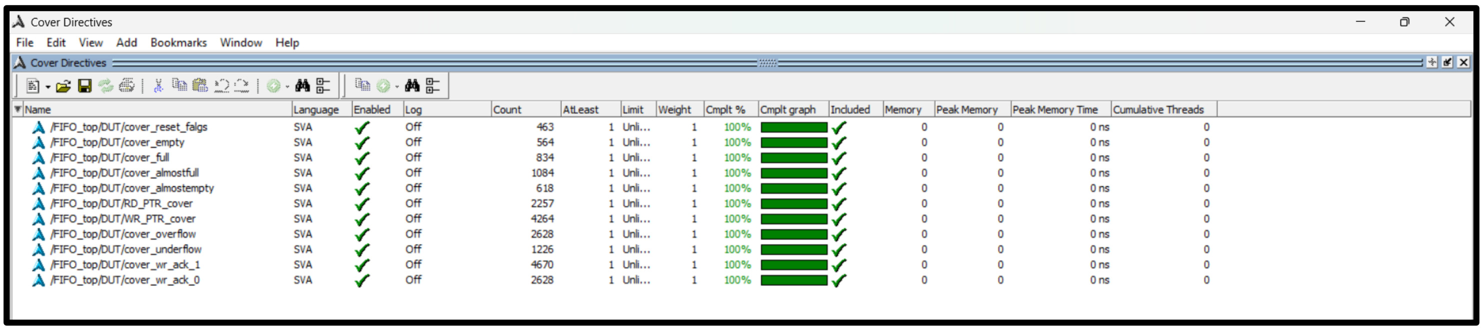


```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        32        32         0   100.00%

=========================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      40        40         0   100.00%

=========================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         40        40         0   100.00%

=========================================================================
```

9. Assertion coverage

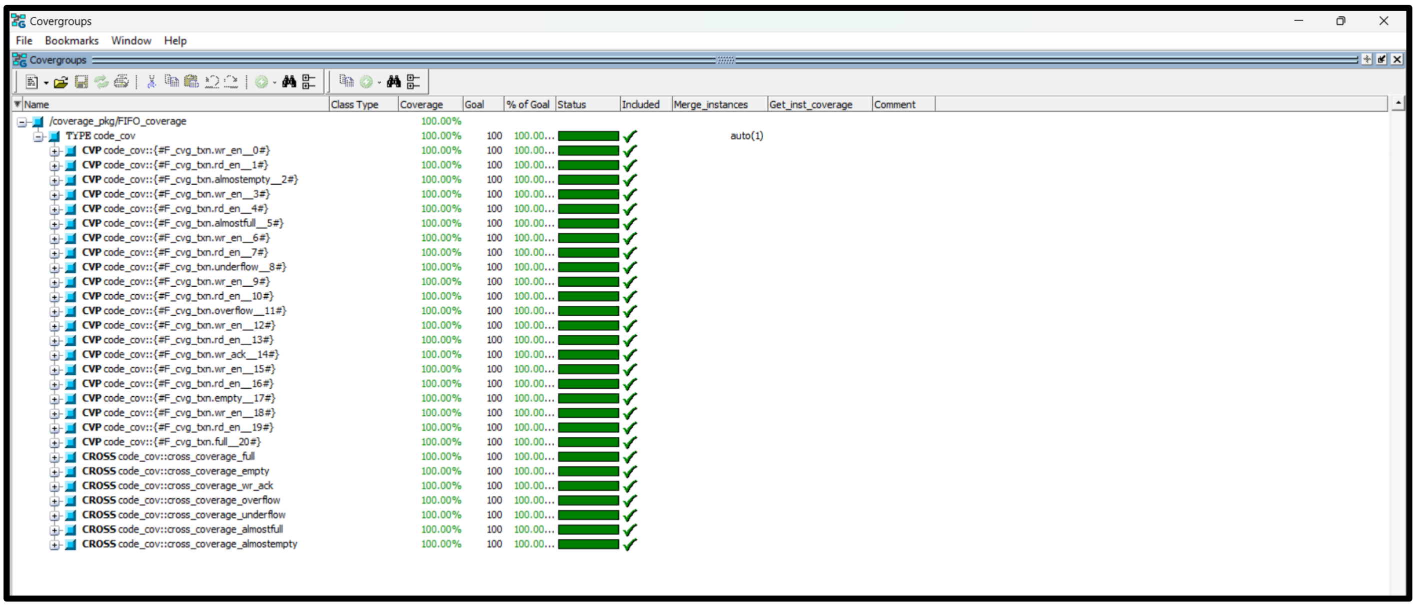


Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/FIFO_top/DUT/assert_reset_falgs	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~full&empty&~almostempty...	✓
/FIFO_top/DUT/assert_empty	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~full&empty&~almostempty...	✓
/FIFO_top/DUT/assert_full	Immediate	SVA	on	0	1	-	-	-	-		off	assert (full&~empty&~almostempty...	✓
/FIFO_top/DUT/assert_almostfull	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((full~|empty)&~almostempt...	✓
/FIFO_top/DUT/assert_almostempty	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((full~|empty)&almostempty...	✓
/FIFO_top/DUT/RD_PTR_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if_obj.clk)...	✓
/FIFO_top/DUT/WR_PTR_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if_obj.clk)...	✓
/FIFO_top/DUT/assert_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if_obj.clk)...	✓
/FIFO_top/DUT/assert_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if_obj.clk)...	✓
/FIFO_top/DUT/assert_wr_ack_1	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if_obj.clk)...	✓
/FIFO_top/DUT/assert_wr_ack_0	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if_obj.clk)...	✓
/FIFO_top/TB/assertions_TB_wr	Immediate	SVA	on	0	1	-	-	-	-		off	assert (tr.randomize)	✓
/FIFO_top/TB/assertions_TB_rd	Immediate	SVA	on	0	1	-	-	-	-		off	assert (tr.randomize)	✓
/FIFO_top/TB/assertions_TB_Mix	Immediate	SVA	on	0	1	-	-	-	-		off	assert (tr.randomize)	✓



Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_top/DUT/cover_reset_falgs	SVA	✓	Off	463	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_empty	SVA	✓	Off	564	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_full	SVA	✓	Off	834	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_almostfull	SVA	✓	Off	1084	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_almostempty	SVA	✓	Off	618	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/RD_PTR_cover	SVA	✓	Off	2257	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/WR_PTR_cover	SVA	✓	Off	4264	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_overflow	SVA	✓	Off	2628	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_underflow	SVA	✓	Off	1226	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ack_1	SVA	✓	Off	4670	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ack_0	SVA	✓	Off	2628	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0

10. Functional coverage



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/coverage_pkg/FIFO_coverage		100.00%							
TYPE code_cov		100.00%	100	100.00...	🟩	✓		auto(1)	
CVP code_cov::{#F_cvg_txn.wr_en__0#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.rd_en__1#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.almostempty__2#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.wr_en__3#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.rd_en__4#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.almostfull__5#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.wr_en__6#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.rd_en__7#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.underflow__8#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.wr_en__9#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.rd_en__10#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.overflow__11#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.wr_en__12#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.rd_en__13#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.wr_ack__14#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.wr_en__15#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.rd_en__16#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.empty__17#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.wr_en__18#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.rd_en__19#}		100.00%	100	100.00...	🟩	✓			
CVP code_cov::{#F_cvg_txn.full__20#}		100.00%	100	100.00...	🟩	✓			
CROSS code_cov::cross_coverage_full		100.00%	100	100.00...	🟩	✓			
CROSS code_cov::cross_coverage_empty		100.00%	100	100.00...	🟩	✓			
CROSS code_cov::cross_coverage_wr_ack		100.00%	100	100.00...	🟩	✓			
CROSS code_cov::cross_coverage_overflow		100.00%	100	100.00...	🟩	✓			
CROSS code_cov::cross_coverage_underflow		100.00%	100	100.00...	🟩	✓			
CROSS code_cov::cross_coverage_almostfull		100.00%	100	100.00...	🟩	✓			
CROSS code_cov::cross_coverage_almostempty		100.00%	100	100.00...	🟩	✓			