Project

Class-Based Verification for a Synchronous FIFO USING SYSTEMVERILOG

By: Mohamed Sayed Mohamed Soliman

Contents

1. Introduction to the Synchronous FIFO

A Synchronous FIFO (First-In-First-Out) : is a fundamental hardware component used in digital systems for data buffering and transferring between different subsystems. It ensures that data is read in the same order as it was written, and both write and read operations are synchronized to a common clock. This makes it particularly useful for applications where data flow needs to be managed in a controlled and sequential manner, often between two devices or subsystems operating at different speeds but synchronized to the same clock.

Key Features of a Synchronous FIFO:

1. **Single Clock Domain**: Both read and write operations are synchronized to the same clock signal, eliminating the need for complex clock domain crossing techniques.
2. **Data Storage**: The FIFO stores data in an internal memory (typically implemented as a register array or RAM). The depth (number of entries) and width (bits per entry) of the FIFO can be parameterized to fit the specific requirements of the application.
3. **Control Flags**:
    - **Full**: Indicates the FIFO is full and cannot accept more write operations until space is freed by reading.
    - **Empty**: Indicates the FIFO is empty and no data is available for reading.
    - **Almost Full/Almost Empty**: Optional flags that provide an early warning before the FIFO becomes completely full or empty.
4. **Read/Write Pointers**: The FIFO internally maintains pointers for tracking the current positions for reading and writing, ensuring data is managed in a circular buffer fashion.

Applications of Synchronous FIFO:

- **Data Rate Matching**: Used between subsystems with different data rates but synchronized clocks (e.g., between a CPU and a memory controller).
- **Buffering**: Temporary storage for bursty data streams or packet data, smoothing out fluctuations in data rates.
- **Pipeline Processing**: Helps in decoupling stages of a digital pipeline, allowing one stage to write data while the next stage reads it at a different rate.

Operational Overview:

- **Write Operation**: Data is written to the FIFO at the location pointed to by the write pointer. After a successful write, the write pointer increments, and the **full** flag is updated if necessary.
- **Read Operation**: Data is read from the FIFO at the location pointed to by the read pointer. The read pointer increments after a read operation, and the **empty** flag is updated accordingly.

The FIFO must manage these operations without data loss or overwriting, maintaining data integrity across various conditions.

Design Considerations:

- **Depth**: The size of the FIFO (number of entries) is chosen based on the maximum burst length or data buffering requirements of the system.
- **Width**: The data width is parameterized, depending on the size of the data being transferred (e.g., 8-bit, 16-bit, or 32-bit).
- **Latency**: The delay between a write or read request and when the data becomes available is minimal in a well-designed FIFO.

Synchronous FIFOs are widely used in digital designs, including networking hardware, processors, and communication systems, where reliable and ordered data transfer is critical.

2. Design RTL code include the assertion

```
////////////////////////////////////////////////////////////////////////////////
module FIFO(FIFO_inter.DUT  FIFO_if);

localparam max_fifo_addr = $clog2(FIFO_if.FIFO_DEPTH);

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

reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [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;
    end
    else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
        mem[wr_ptr] <= FIFO_if.data_in;
        FIFO_if.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    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.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;
    end
    else begin
        if (FIFO_if.empty && FIFO_if.rd_en)
        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)
            count <= count + 1;
        else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty)
            count <= count - 1;
        else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty)
            count <= count + 1;
        else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full)
            count <= count - 1;
    end
end
```

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

        property prop1 ;
            @(posedge FIFO_if.clk )
            disable iff (!FIFO_if.rst_n)
            (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) |=>  (FIFO_if.wr_ack ) ;
        endproperty
        prop1_assertion:  assert property (prop1) ;
        prop1_cover: cover property (prop1) ;

        property prop2 ;
            @(posedge FIFO_if.clk )
            disable iff (!FIFO_if.rst_n)
            ( FIFO_if.full && FIFO_if.wr_en ) |=>  (FIFO_if.overflow ) ;
        endproperty
        prop2_assertion:  assert property (prop2) ;
        prop2_cover: cover property (prop2) ;
```

```
    always_comb begin
        if (count == 0) begin
            prop3_assertion : assert (FIFO_if.empty && !FIFO_if.full && !FIFO_if.almostempty && !FIFO_if.almostfull) ;
            prop3_cover     : cover  (FIFO_if.empty && !FIFO_if.full && !FIFO_if.almostempty && !FIFO_if.almostfull)     ;
        end
        if (count == 1) begin
            prop4_assertion : assert (!FIFO_if.empty && !FIFO_if.full && FIFO_if.almostempty && !FIFO_if.almostfull) ;
            prop4_cover     : cover  (!FIFO_if.empty && !FIFO_if.full && FIFO_if.almostempty && !FIFO_if.almostfull)     ;
        end
        if (count == FIFO_if.FIFO_DEPTH-1) begin
            prop5_assertion : assert (!FIFO_if.empty && !FIFO_if.full && !FIFO_if.almostempty && FIFO_if.almostfull);
            prop5_cover     : cover  (!FIFO_if.empty && !FIFO_if.full && !FIFO_if.almostempty && FIFO_if.almostfull)    ;
        end
        if (count == FIFO_if.FIFO_DEPTH) begin
            prop6_assertion : assert (!FIFO_if.empty && FIFO_if.full && !FIFO_if.almostempty && !FIFO_if.almostfull) ;
            prop6_cover     : cover  (!FIFO_if.empty && FIFO_if.full && !FIFO_if.almostempty && !FIFO_if.almostfull)     ;
        end
    end
```

```
    property prop7 ;
        @(posedge FIFO_if.clk )
        disable iff (!FIFO_if.rst_n)
        (FIFO_if.empty && FIFO_if.rd_en) |=>  ( FIFO_if.underflow ) ;
    endproperty
    prop7_assertion:  assert property (prop7) ;
    prop7_cover: cover property (prop7) ;


    property prop8 ;
        @(posedge FIFO_if.clk )
        disable iff (!FIFO_if.rst_n)
        (FIFO_if.wr_en && !FIFO_if.rd_en && !FIFO_if.full ) |=>  ( count === ($past(count) + 1 ) ) ;
    endproperty
    prop8_assertion:  assert property (prop8) ;
    prop8_cover: cover property (prop8) ;

    property prop9 ;
        @(posedge FIFO_if.clk )
        disable iff (!FIFO_if.rst_n)
        ( !FIFO_if.wr_en && FIFO_if.rd_en && !FIFO_if.empty ) |=>  ( count === ($past(count) - 1 ) ) ;
    endproperty
    prop9_assertion:  assert property (prop9) ;
    prop9_cover: cover property (prop9) ;
```

```
    property prop10 ;
        @(posedge FIFO_if.clk )
        disable iff (!FIFO_if.rst_n)
        ( FIFO_if.wr_en && FIFO_if.rd_en && FIFO_if.empty ) |=>  ( count === ($past(count) + 1 ) ) ;
    endproperty
    prop10_assertion:  assert property (prop10) ;
    prop10_cover: cover property (prop10) ;


    property prop11 ;
        @(posedge FIFO_if.clk )
        disable iff (!FIFO_if.rst_n)
        ( FIFO_if.wr_en && FIFO_if.rd_en && FIFO_if.full ) |=>  ( count === ($past(count) - 1 ) ) ;
    endproperty
    prop11_assertion:  assert property (prop11) ;
    prop11_cover: cover property (prop11) ;

endmodule
```

3. Verification plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, the output value should be low & All the location of FIFO is cleard	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the simulation time.	-	A check_data function in the FIFO_scoreboard to make sure the output is correct
FIFO_2	When the write enable is asserted , the read enable is deasserted and not full so we will write data_in inside the FIFO	Randomization under constraints on the data_in & write enable	include cover point for the wr_en signal & all flag signal	A check_data function in the FIFO_scoreboard to make sure the output is correct
FIFO_3	When the write enable is deasserted , the read enable is asserted and not empty so we will read from the FIFO & get the value to be stored in data_out signal	Randomization under constraints for read enable signal	include cover point for the rd_en signal & all flag signal	A check_data function in the FIFO_scoreboard to make sure the output is correct
FIFO_4	When the write enable is asserted && the read enable is asserted at the same time we then concentrate on the output flag => if full = 1 then the priority will be for read else if empty = 1 then the priority will be for write	-	-	A check_data function in the FIFO_scoreboard to make sure the output is correct

4. Verification code

4.1. Top design

```
module TOP_FIFO ();

    bit clk ;
    initial begin
        clk = 0;
        forever
            #10 clk = ~clk;
      end

      FIFO_inter  FIFO_if (clk) ;
      FIFO     DUT (FIFO_if);
      FIFO_monitor    mon (FIFO_if);
      FIFO_tb  test (FIFO_if);

endmodule
```

4.2. Interface

```
interface FIFO_inter(clk) ;

    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

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

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

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


endinterface
```

4.3. FIFO_transaction package

```
package pack_FIFO_transaction;

    class FIFO_transaction ;

      parameter FIFO_WIDTH = 16;
      parameter FIFO_DEPTH = 8;
      bit clk;
      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( integer RD_EN_ON_DIST = 30  ,  integer WR_EN_ON_DIST = 70  ) ;
        this.RD_EN_ON_DIST = RD_EN_ON_DIST ;
        this.WR_EN_ON_DIST = WR_EN_ON_DIST ;
      endfunction

      constraint write {wr_en dist {1:/(WR_EN_ON_DIST) , 0:/(100-WR_EN_ON_DIST)  } ; }
      constraint read  {rd_en dist {1:/(RD_EN_ON_DIST) , 0:/(100-RD_EN_ON_DIST)  } ; }
      constraint rst {rst_n dist {1:/99 , 0:/1  } ; }

    endclass

endpackage
```

4.4. FIFO_coverage package

```
package pack_FIFO_coverage;

  import pack_FIFO_transaction::* ;
    class FIFO_coverage ;

      FIFO_transaction F_cvg_txn = new() ;


      covergroup cvr_gp ;
        wr_en_cp : coverpoint F_cvg_txn.wr_en ;
        rd_en_cp : coverpoint F_cvg_txn.rd_en ;
        wr_ack_cp : coverpoint F_cvg_txn.wr_ack ;
        overflow_cp : coverpoint F_cvg_txn.overflow ;
        full_cp : coverpoint F_cvg_txn.full ;
        empty_cp : coverpoint F_cvg_txn.empty ;
        almostfull_cp : coverpoint F_cvg_txn.almostfull ;
        almostempty_cp : coverpoint F_cvg_txn.almostempty ;
        underflow_cp : coverpoint F_cvg_txn.underflow ;


        wr_full_cp : cross  wr_en_cp  , full_cp ;
        wr_wr_ack_cp : cross  wr_en_cp , wr_ack_cp ;
        wr_overflow_cp : cross  wr_en_cp , overflow_cp ;
        wr_empty_cp : cross  wr_en_cp  , empty_cp ;
        wr_almostfull_cp : cross  wr_en_cp  , almostfull_cp ;
        wr_almostempty_cp : cross  wr_en_cp  , almostempty_cp ;
        wr_underflow_cp : cross  wr_en_cp  , underflow_cp ;

      rd_full_cp : cross  rd_en_cp  , full_cp
      {
        ignore_bins rd_full = binsof(rd_en_cp) intersect {1} && binsof(full_cp) intersect {1} ;
      }
      rd_wr_ack_cp : cross  rd_en_cp ,  wr_ack_cp ;
      rd_overflow_cp : cross  rd_en_cp , overflow_cp ;
      rd_empty_cp : cross  rd_en_cp  , empty_cp ;
      rd_almostfull_cp : cross  rd_en_cp  , almostfull_cp ;
      rd_almostempty_cp : cross  rd_en_cp  , almostempty_cp ;
      rd_underflow_cp : cross  rd_en_cp  , underflow_cp ;


      endgroup
    function new() ;
      cvr_gp  = new;
    endfunction

      function void  sample_data(input FIFO_transaction F_txn ) ;
        F_cvg_txn = F_txn ;
       cvr_gp.sample();
      endfunction

    endclass

endpackage
```

4.5. FIFO_scoreboard package

```
package pack_FIFO_scoreboard;

  import pack_FIFO_transaction::* ;
  import shared_pkg::* ;

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

      bit [FIFO_WIDTH-1:0] mem_queue[$] ;
      reg [max_fifo_addr:0] count;

      function void  reference_model(input FIFO_transaction obj_tran ) ;
        fork
            begin
                if (!obj_tran.rst_n) begin
                    mem_queue.delete();
                end
                else if (obj_tran.wr_en && count < obj_tran.FIFO_DEPTH) begin
                    mem_queue.push_back(obj_tran.data_in) ;
                    wr_ack_ref = 1;
                end
                else begin
                    wr_ack_ref = 0;
                    if (full_ref & obj_tran.wr_en)
                    overflow_ref = 1;
                    else
                    overflow_ref = 0;
                end
            end
            begin
                if (!obj_tran.rst_n) begin
                    data_out_ref = 0 ;
                end
                else if (obj_tran.rd_en && count != 0) begin
                    data_out_ref = mem_queue.pop_front() ;
                end
                else begin
                    if (empty_ref && obj_tran.rd_en)
                    underflow_ref = 1;
                    else
                    underflow_ref = 0 ;
                end
            end
        join

        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;
```

```
56          if (!obj_tran.rst_n) begin
57              count = 0;
58          end
59          else begin
60              if  ( ({obj_tran.wr_en, obj_tran.rd_en} == 2'b10) && !full_ref)
61                  count = count + 1;
62              else if ( ({obj_tran.wr_en, obj_tran.rd_en} == 2'b01) && !empty_ref)
63                  count = count - 1;
64              else if ( ({obj_tran.wr_en, obj_tran.rd_en} == 2'b11) && empty_ref)
65                  count = count + 1;
66              else if ( ({obj_tran.wr_en, obj_tran.rd_en} == 2'b11) && full_ref)
67                  count = count - 1;
68          end
69      endfunction
70      function void check_data(input FIFO_transaction obj_tran_check ) ;
71       logic [6:0] flag_test , flag_dut ;
72        reference_model(obj_tran_check);
73        flag_test ={ wr_ack_ref , overflow_ref , full_ref , empty_ref , almostfull_ref , almostempty_ref , underflow_ref } ;
74        flag_dut ={ obj_tran_check.wr_ack , obj_tran_check.overflow , obj_tran_check.full , obj_tran_check.empty , obj_tran_check.almostfull , obj_tran_check.almostempty , obj_tran_check.underflow } ;
75
76        if( (obj_tran_check.data_out !== data_out_ref) && (flag_dut !== flag_test) )
77        begin
78            $display ("At time = %0t ::error ==> obj_tran_check.data_out  = %0d  ,  data_out_ref  = %0d  , flag_dut  = %0b , flag_test  = %0b  , obj_tran_check.rst_n  = %0b  ,  obj_tran_check.wr_en  = %0b
79            error_count++ ;
80        end
81    else
82        begin
83            correct_count++ ;
84        end
85      endfunction
86    endclass
87  endpackage
```

4.6. Testbench

```
1   import pack_FIFO_transaction::* ;
2   import pack_FIFO_coverage::* ;
3   import shared_pkg::* ;
4
5   module FIFO_tb(FIFO_inter.TEST  FIFO_if);
6
7     FIFO_transaction  transaction = new() ;
8     FIFO_coverage     coverage    = new() ;
9     initial
10      begin
11          forever
12          #10 transaction.clk = FIFO_if.clk;
13      end
14    initial begin
15      rest ();
16
17      test_finished = 0 ;
18      FIFO_if.data_in = 1;
19      FIFO_if.rst_n = 1;
20      FIFO_if.wr_en =1 ;
21      FIFO_if.rd_en = 1 ;
```

```
22          @(negedge FIFO_if.clk) ;
23          repeat(100000)
24          begin
25            transaction.randomize();
26            FIFO_if.data_in = transaction.data_in ;
27            FIFO_if.rst_n = transaction.rst_n ;
28            FIFO_if.wr_en = transaction.wr_en ;
29            FIFO_if.rd_en = transaction.rd_en ;
30            @(negedge FIFO_if.clk) ;
31          end
32          test_finished = 1 ;
33        end
34        task rest ();
35          FIFO_if.rst_n = 0 ;
36          #10
37          FIFO_if.rst_n = 1 ;
38
39      endtask
40
41      endmodule
```

4.7. Monitor

```
1   import pack_FIFO_transaction::* ;
2   import pack_FIFO_scoreboard::* ;
3   import pack_FIFO_coverage::* ;
4   import shared_pkg::* ;
5
6   module FIFO_monitor(FIFO_inter.MONITOR  FIFO_if);
7
8     FIFO_transaction  transaction = new()  ;
9     FIFO_scoreboard  scoreboard = new();
10    FIFO_coverage  coverage = new();
11
12      initial begin
13        forever
14        begin
15            @(negedge FIFO_if.clk );
16            transaction.clk = FIFO_if.clk ;
17            transaction.data_in = FIFO_if.data_in ;
18            transaction.rst_n = FIFO_if.rst_n ;
19            transaction.wr_en = FIFO_if.wr_en ;
20            transaction.rd_en = FIFO_if.rd_en ;
21            transaction.data_out = FIFO_if.data_out ;
22            transaction.wr_ack = FIFO_if.wr_ack ;
23            transaction.overflow = FIFO_if.overflow ;
24            transaction.full = FIFO_if.full ;
25            transaction.empty = FIFO_if.empty ;
26            transaction.almostfull = FIFO_if.almostfull ;
27            transaction.almostempty = FIFO_if.almostempty ;
28            transaction.underflow = FIFO_if.underflow ;
29
30
```

```
            fork

              begin
                coverage.sample_data(transaction);
              end

              begin  // 2nd thread
                @(posedge FIFO_if.clk);
                #10;
                scoreboard.check_data(transaction);
              end

            join

            if(test_finished == 1) begin
              $display("no.of error_count    :%0d " , error_count   ) ;
              $display("no.of correct_count :%0d " , correct_count) ;
              $stop;
            end
        end
        end


endmodule
```

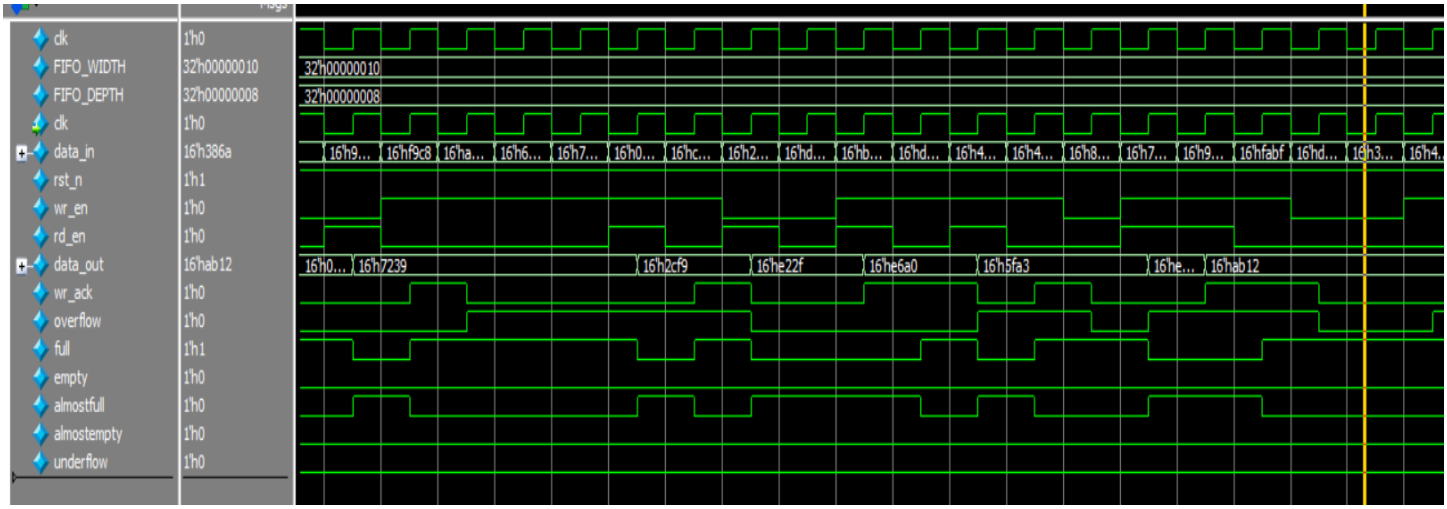
4.8. Shared package

```
package shared_pkg;
  bit test_finished;

  int error_count, correct_count;

endpackage

```

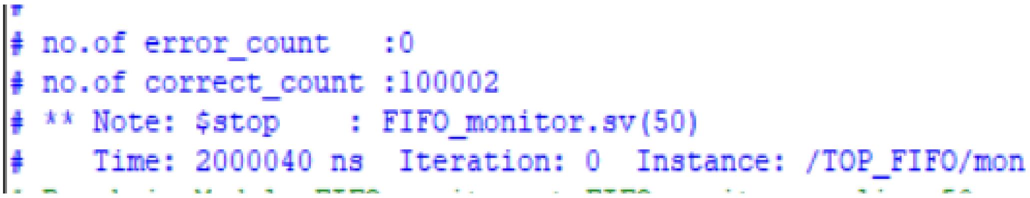
5. Do file

```
vlib work
vlog FIFO_inter.sv TOP_FIFO.sv FIFO.sv FIFO_tb.sv FIFO_monitor.sv pack_FIFO_transaction.sv pack_FIFO_coverage.sv pack_FIFO_scoreboard.sv shared_pkg.sv +cover -covercells
vsim -voptargs=+acc work.TOP_FIFO -cover
add wave *
coverage save TOP_FIFO.ucdb -onexit
add wave -position insertpoint  \
sim:/TOP_FIFO/FIFO_if/FIFO_WIDTH \
sim:/TOP_FIFO/FIFO_if/FIFO_DEPTH \
sim:/TOP_FIFO/FIFO_if/clk \
sim:/TOP_FIFO/FIFO_if/data_in \
sim:/TOP_FIFO/FIFO_if/rst_n \
sim:/TOP_FIFO/FIFO_if/wr_en \
sim:/TOP_FIFO/FIFO_if/rd_en \
sim:/TOP_FIFO/FIFO_if/data_out \
sim:/TOP_FIFO/FIFO_if/wr_ack \
sim:/TOP_FIFO/FIFO_if/overflow \
sim:/TOP_FIFO/FIFO_if/full \
sim:/TOP_FIFO/FIFO_if/empty \
sim:/TOP_FIFO/FIFO_if/almostfull \
sim:/TOP_FIFO/FIFO_if/almostempty \
sim:/TOP_FIFO/FIFO_if/underflow

run -all

quit -sim
vcover report TOP_FIFO.ucdb -details -all -output coverage.txt
```

6. Waveform

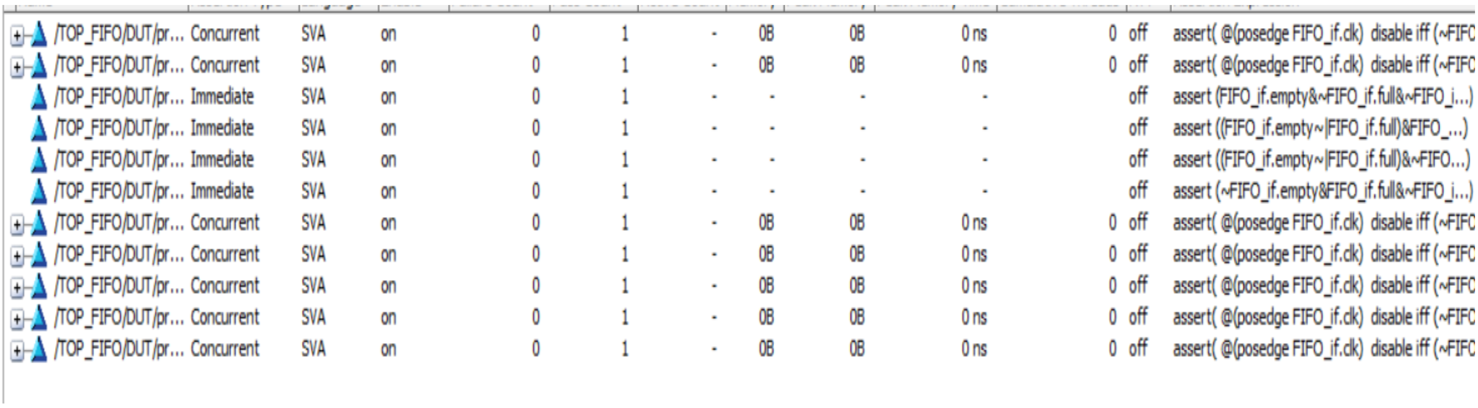


7. Transcript



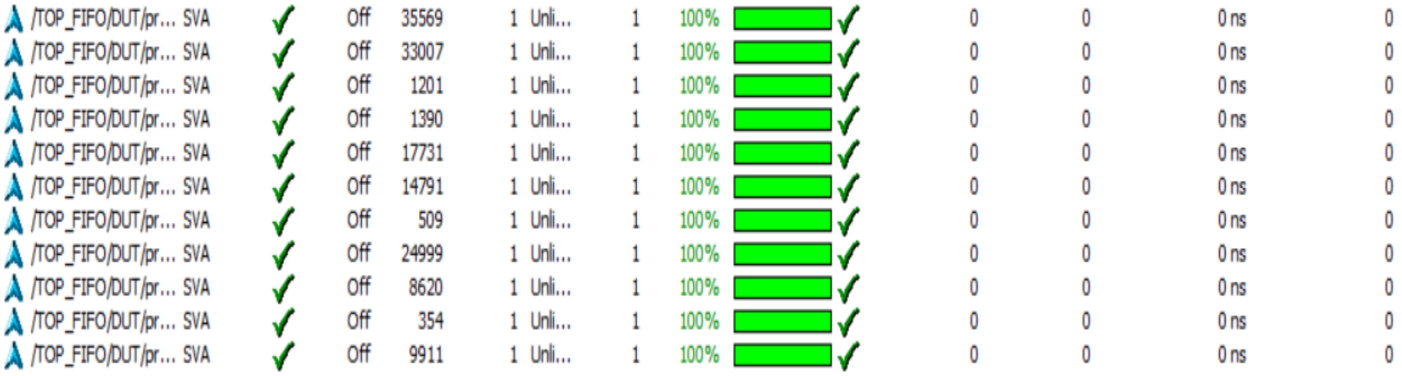
```
# no.of error_count   :0
# no.of correct_count :100002
# ** Note: $stop    : FIFO_monitor.sv(50)
#    Time: 2000040 ns  Iteration: 0  Instance: /TOP_FIFO/mon
```

8. Assertion



/TOP_FIFO/DUT/pr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) disable iff (~FIFO
/TOP_FIFO/DUT/pr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) disable iff (~FIFO
/TOP_FIFO/DUT/pr...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.empty&~FIFO_if.full&~FIFO_i...)
/TOP_FIFO/DUT/pr...	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((FIFO_if.empty~|FIFO_if.full)&FIFO_...)
/TOP_FIFO/DUT/pr...	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((FIFO_if.empty~|FIFO_if.full)&~FIFO...)
/TOP_FIFO/DUT/pr...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFO_if.empty&FIFO_if.full&~FIFO_i...)
/TOP_FIFO/DUT/pr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) disable iff (~FIFO
/TOP_FIFO/DUT/pr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) disable iff (~FIFO
/TOP_FIFO/DUT/pr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) disable iff (~FIFO
/TOP_FIFO/DUT/pr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) disable iff (~FIFO
/TOP_FIFO/DUT/pr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) disable iff (~FIFO

9. Assertion coverage



/TOP_FIFO/DUT/pr...	SVA	✓	Off	35569	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	33007	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	1201	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	1390	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	17731	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	14791	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	509	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	24999	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	8620	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	354	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0
/TOP_FIFO/DUT/pr...	SVA	✓	Off	9911	1	Unli...	1	100%	🟩 ✓	0	0	0 ns	0

10. Function coverage

/pack_FIFO_cover...			
TYPE cvr_gp	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CVP cvr_g...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%
CROSS cvr...	100.0%	100	100.0%

11. Code Coverage

```
3   ==============================================================================
4   === File: FIFO.sv
5   ==============================================================================
6   Statement Coverage:
7       Enabled Coverage            Active      Hits    Misses % Covered
8       ----------------            ------      ----    ------ ---------
9       Stmts                           25        25         0     100.0
10
11  ================================Statement Details================================
12
13  Statement Coverage for file FIFO.sv --
14
15      1                                       ////////////////////////////////////////////////////////////////////////////////
16      2                                       // Author: Kareem Waseem
17      3                                       // Course: Digital Verification using SV & UVM
18      4                                       //
19      5                                       // Description: FIFO Design
20      6                                       //
21      7                                       ////////////////////////////////////////////////////////////////////////////////
22      8                                       module FIFO(FIFO_inter.DUT  FIFO_if);
23      9
24      10                                      localparam max_fifo_addr = $clog2(FIFO_if.FIFO_DEPTH);
25      11
26      12                                      reg [FIFO_if.FIFO_WIDTH-1:0] mem [FIFO_if.FIFO_DEPTH-1:0];
27      13
28      14                                      reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
29      15                                      reg [max_fifo_addr:0] count;
30      16
31      17              1                101004 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
32      18                                         if (!FIFO_if.rst_n) begin
33      19              1                  2011            wr_ptr <= 0;
34      20                                         end
35      21                                         else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
```

```
Branch Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Branches                        23        23         0     100.0

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

Branch Coverage for file FIFO.sv --

----------------------------------IF Branch----------------------------------
    18                                     101004     Count coming in to IF
    18              1                       2011        if (!FIFO_if.rst_n) begin
    21              1                      35936        else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
    26              1                      63057        else begin
Branch totals: 3 hits of 3 branches = 100.0%

----------------------------------IF Branch----------------------------------
    28                                     63057     Count coming in to IF
    28              1                      33332            if (FIFO_if.full && FIFO_if.wr_en)
    30              1                      29725            else
Branch totals: 2 hits of 2 branches = 100.0%

----------------------------------IF Branch----------------------------------
    36                                    101004     Count coming in to IF
    36              1                       2011        if (!FIFO_if.rst_n) begin
    40              1                      29029        else if (FIFO_if.rd_en && count != 0) begin
    45              1                        516            if (FIFO_if.empty && FIFO_if.rd_en)
    47              1                      69448            else
Branch totals: 4 hits of 4 branches = 100.0%

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

Toggle Coverage for File FIFO.sv --

        Line                                Node      1H->0L      0L->1H  "Coverage"
--------------------------------------------------------------------------------------
          14                            wr_ptr[2]           1           1      100.00
          14                            wr_ptr[1]           1           1      100.00
          14                            wr_ptr[0]           1           1      100.00
          14                            rd_ptr[2]           1           1      100.00
          14                            rd_ptr[1]           1           1      100.00
          14                            rd_ptr[0]           1           1      100.00
          15                             count[3]           1           1      100.00
          15                             count[2]           1           1      100.00
          15                             count[1]           1           1      100.00
          15                             count[0]           1           1      100.00

Total Node Count     =          10
Toggled Node Count   =          10
Untoggled Node Count =           0

Toggle Coverage      =      100.0% (20 of 20 bins)

--------------------------------------------------------------------------------
```