Verification plan :

	A	B	C	D	E
1	label		Stimulus Generation	Functional Coverage	Functionality Check
2	FIFO_1	When the reset is asserted, the output of fifo value should be low and making all outputs signals equals zero and intialize the counter signals and the test finish flag	Directed at the start of the simulation	-	golden model in refrence_model task in the scoreborad package to check the result to check the dataout but all other output signals are being checked by assertions in the design file
3	FIFO_2	randomizing some inputs with some constraints :1. Assert reset less often 2. Constraint the write enable to be high with distribution of the value WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DIST 3. Constraint the read enable the same as write enable but using RD_EN_ON_DIST	Directed during simulation	The coverage needed is cross coverage between 3 signals which are write enable, read enable and each output control signals (outputs except data_out) to make sure that all combinations of write and read enable took place in all state of the FIFO.	golden model in refrence_model task in the scoreborad package to check the result to check the dataout but all other output signals are being checked by assertions in the design file
4	FIFO_3	When the reset is asserted, the output of fifo value should be low and making all outputs signals equals zero and intialize the counter signals	Directed during simulation	_	golden model in refrence_model task in the scoreborad package to check the result to check the dataout but all other output signals are being checked by assertions in the design file

Interface code :

```
interface FIFO_interface (clk);

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);//put the local param before the signals
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 data_in , rst_n, wr_en, rd_en , clk,
output data_out ,wr_ack, overflow , full, empty, almostfull, almostempty, underflow
);

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_out ,wr_ack, overflow , full, empty, almostfull, almostempty,
underflow ,
data_in , rst_n, wr_en, rd_en);


endinterface //FIFO_interface (clk)
```

design and detected bugs and correction of them:

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

reg [FIFO_if.FIFO_WIDTH-1:0] mem [FIFO_if.FIFO_DEPTH-1:0];
reg [FIFO_if.max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [FIFO_if.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;
  end
```

```
  else if (FIFO_if.rd_en && count != 0) begin
    FIFO_if.data_out <= mem[rd_ptr];
    rd_ptr <= rd_ptr + 1;
  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;
  end
end

always @(*) begin //make it in always block pure combinational
 FIFO_if.full = (count == FIFO_if.FIFO_DEPTH)? 1 : 0;
 FIFO_if.empty = (count == 0)? 1 : 0;
 FIFO_if.underflow = (FIFO_if.empty && FIFO_if.rd_en)? 1 : 0;
 FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-2)? 1 : 0;
 FIFO_if.almostempty = (count == 1)? 1 : 0;
 FIFO_if.overflow = (FIFO_if.full && FIFO_if.wr_en)? 1 : 0; //adding overflow
end



//assertions

assert property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH) |-> FIFO_if.full
);
assert property (@(posedge FIFO_if.clk) (count == 0) |-> FIFO_if.empty );
assert property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH-2) |->
FIFO_if.almostfull );
assert property (@(posedge FIFO_if.clk) (count == 1) |-> FIFO_if.almostempty );
assert property (@(posedge FIFO_if.clk) (FIFO_if.empty && FIFO_if.rd_en) |->
FIFO_if.underflow);
assert property (@(posedge FIFO_if.clk) (FIFO_if.full && FIFO_if.wr_en) |->
FIFO_if.overflow);

cover property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH) |->
FIFO_if.full );
cover property (@(posedge FIFO_if.clk) (count == 0) |-> FIFO_if.empty );
cover property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH-2) |->
FIFO_if.almostfull );
```

```
cover property (@(posedge FIFO_if.clk) (count == 1) |-> FIFO_if.almostempty );
cover property (@(posedge FIFO_if.clk) (FIFO_if.empty && FIFO_if.rd_en) |->
FIFO_if.underflow);
cover property (@(posedge FIFO_if.clk) (FIFO_if.full && FIFO_if.wr_en) |->
FIFO_if.overflow);


endmodule
```

Top module code :

```
module top ();

bit clk ;

//clock generation
initial begin
 clk = 0 ;
 forever begin
 #1 clk =~ clk;
 end
end

FIFO_interface FIFO_if (clk) ;
FIFO DUT (FIFO_if.DUT) ;
FIFO_tb TEST (FIFO_if.TEST) ;

endmodule
```

Packages codes :

1.transactions:

```
package FIFO_transaction_pkg ;

import shared_pkg::*;

class FIFO_transaction;

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);//put the local param before the signals
bit clk_poc ;
randc logic [FIFO_WIDTH-1:0] data_in_poc;
randc logic rst_n_poc;
randc logic wr_en_poc, rd_en_poc;
```

```
logic  [FIFO_WIDTH-1:0] data_out_poc;
logic  wr_ack_poc, overflow_poc;
logic full_poc, empty_poc, almostfull_poc, almostempty_poc, underflow_poc;

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; // Initialize read enable distance
    this.WR_EN_ON_DIST = WR_EN_ON_DIST ; // Initialize write enable distance
  endfunction //new()

 constraint reset_signal { rst_n_poc dist { 0:= 5 , 1:= 95 } ; } //reset contstraint

 constraint write_enable_signal { wr_en_poc dist { 0:= 100-WR_EN_ON_DIST  , 1:=
WR_EN_ON_DIST  } ; } //reset contstraint

 constraint read_enable_signal { rd_en_poc dist { 0:= 100-RD_EN_ON_DIST  , 1:=
RD_EN_ON_DIST  } ; } //reset contstraint

endclass //

endpackage
```

2.coverage:

```
package FIFO_coverage_pkg;

import shared_pkg::*;
import FIFO_transaction_pkg::*;

class FIFO_coverage;

FIFO_transaction F_cvg_txn = new (30,70) ;


covergroup FIFO_cover;

write : coverpoint F_cvg_txn.wr_en_poc ;
read : coverpoint F_cvg_txn.rd_en_poc ;
full : coverpoint F_cvg_txn.full_poc ;
almostfull : coverpoint F_cvg_txn.almostfull_poc ;
empty : coverpoint F_cvg_txn.empty_poc  ;
almostempty : coverpoint F_cvg_txn.almostempty_poc ;
overflow : coverpoint F_cvg_txn.overflow_poc ;
underflow : coverpoint F_cvg_txn.underflow_poc ;
wr_ack : coverpoint F_cvg_txn.wr_ack_poc ;
```

```
  a0: cross write , full ;
  a1: cross write , almostfull ;
  a2: cross write , wr_ack ;
  a4: cross write , overflow ;
  b0: cross read , empty ;
  b1: cross read , almostempty ;
  b2: cross read , underflow ;

endgroup

// Constructor
  function new();
    FIFO_cover = new(); // Initialize the covergroup instance
  endfunction


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

endclass

endpackage
```

3.scoreboard:

```
package FIFO_scoreboard_pkg;

import shared_pkg::*;
import FIFO_transaction_pkg::*;

class FIFO_scoreboard;

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);//put the local param before the signals
logic [FIFO_WIDTH-1:0] data_out_ref ;

  function void check_data ( FIFO_transaction output_received);
    output_received = new (30,70);
    refrence_model(output_received);
      if (output_received.data_out_poc !== data_out_ref) begin
      $display("Error: Data Out Mismatch! Expected: %0d, Got: %0d", data_out_ref,
output_received.data_out_poc );
      error_counter++;
    end else begin
```

```
      correct_counter++;
    end

  endfunction

  function void refrence_model (FIFO_transaction output_received_checking);
    // Static variables to simulate FIFO memory and pointers
  static bit [FIFO_WIDTH-1:0] fifo_memory[FIFO_DEPTH-1:0]; // FIFO storage
  static int write_pointer = 0; // Pointer for writing data
  static int read_pointer = 0;  // Pointer for reading data
  output_received_checking = new (30,70);

  // Initialize data_out_ref based on FIFO state
  if (output_received_checking.wr_en_poc  && !((write_pointer + 1) %
output_received_checking.FIFO_DEPTH == read_pointer)) begin
    // If write enable is high and FIFO is not full, write data
    fifo_memory[write_pointer] = output_received_checking.data_in_poc ; // Write data
into memory
    write_pointer = (write_pointer + 1) % output_received_checking.FIFO_DEPTH; // Move
write pointer forward
  end

  if (output_received_checking.rd_en_poc  && (read_pointer != write_pointer)) begin
    // If read enable is high and FIFO is not empty, read data
    data_out_ref = fifo_memory[read_pointer]; // Set expected output data from read
pointer
    read_pointer = (read_pointer + 1) % output_received_checking.FIFO_DEPTH ; // Move
read pointer forward
  end else if (!output_received_checking.rd_en_poc ) begin
    // If no read operation, maintain current expected output
    data_out_ref = fifo_memory[read_pointer]; // Keep the same expected output if no
read occurs
  end

  endfunction
endclass //FIFO_scoreboard

endpackage
```

Testbench code :

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

module FIFO_tb ( FIFO_interface.TEST FIFO_in );
```

```
  FIFO_transaction transactions;
  FIFO_coverage coverage ;
  FIFO_scoreboard scoreboard;

initial begin

  transactions = new (30,70);
  scoreboard = new ;
  coverage = new ;


  //asserting rst (FIFO_1)
  error_counter = 0 ;
  correct_counter = 0 ;
  test_finished = 0 ;

  transactions.rst_n_poc  = 0 ;
  @(negedge FIFO_if.clk);
  transactions.rst_n_poc = 1 ;
  @(negedge FIFO_if.clk);

  //randomization (FIFO_2)
  for (int i = 0 ; i<1000 ; i++ ) begin
     @(negedge FIFO_if.clk);
       assert(transactions.randomize()) else begin
         $fatal("Randomization failed");
       end
     transactions.clk_poc = FIFO_if.clk;

     FIFO_if.data_in = transactions.data_in_poc ;
     FIFO_if.wr_en = transactions.wr_en_poc ;
     FIFO_if.rd_en = transactions.rd_en_poc ;
     FIFO_if.rst_n = transactions.rst_n_poc ;

     transactions.full_poc =  FIFO_if.full;
     transactions.empty_poc =  FIFO_if.empty;
     transactions.almostfull_poc =  FIFO_if.almostfull;
     transactions.almostempty_poc =  FIFO_if.almostempty;
     transactions.underflow_poc =  FIFO_if.underflow;
     transactions.overflow_poc =  FIFO_if.overflow;
     transactions.wr_ack_poc =  FIFO_if.wr_ack;
     transactions.data_out_poc =  FIFO_if.data_out;

     scoreboard.check_data(transactions);

     coverage.sample_data(transactions);
```

```
    @(negedge FIFO_if.clk);
  end

    test_finished = 1;
    @(negedge FIFO_if.clk);
  if (test_finished) begin
    $display("number of errors = %0d and number of correct operations = %0d",
error_counter , correct_counter);
    $stop ;
  end

  //asserting_rst (FIFO_3)
  transactions.rst_n_poc = 0 ;
  @(negedge FIFO_if.clk);
  transactions.rst_n_poc = 1 ;

end

endmodule
```

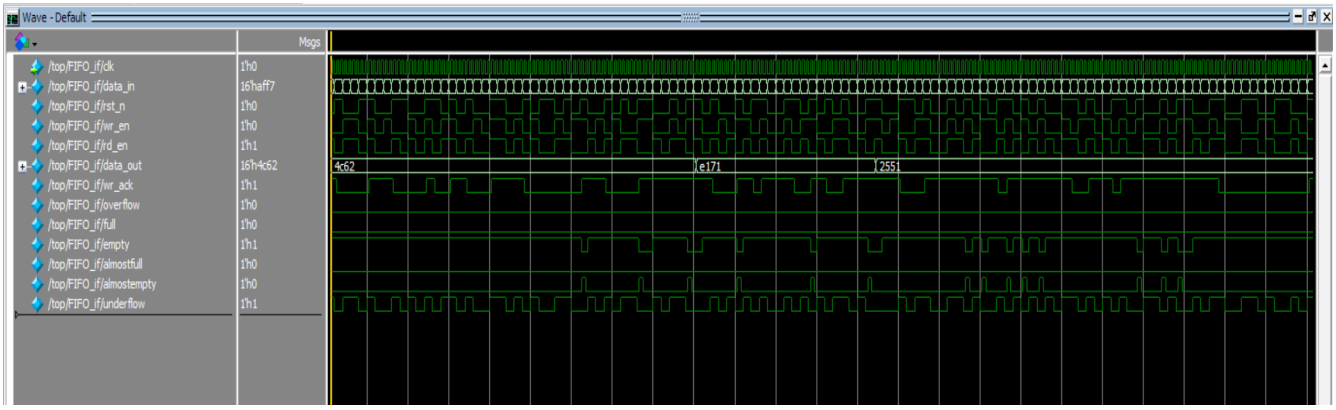
src files :

```
shared_pkg.sv
FIFO_transaction_pkg.sv
FIFO_coverage_pkg.sv
FIFO_scoreboard_pkg.sv
FIFO_interface.sv
FIFO.sv
FIFO_tb.sv
FIFO_top.sv
```

DO file :

```
vlib work
vlog -f src_files.list
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all
add wave /top/FIFO_if/*
coverage save top.ucdb -onexit
run -all
```

Simulation :



functional coverage :

```
TOTAL COVERGROUP COVERAGE: 75.0%  COVERGROUP TYPES: 1
```

Assertions coverage :

```
DIRECTIVE COVERAGE:
-------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Count Status
                                        Unit   UnitType
-------------------------------------------------------------------------------------------
/top/DUT/cover__5                       FIFO   Verilog  SVA  FIFO.sv(79)         0 ZERO
/top/DUT/cover__4                       FIFO   Verilog  SVA  FIFO.sv(78)       966 Covered
/top/DUT/cover__3                       FIFO   Verilog  SVA  FIFO.sv(77)       126 Covered
/top/DUT/cover__2                       FIFO   Verilog  SVA  FIFO.sv(76)         0 ZERO
/top/DUT/cover__1                       FIFO   Verilog  SVA  FIFO.sv(75)      1812 Covered
/top/DUT/cover__0                       FIFO   Verilog  SVA  FIFO.sv(74)         0 ZERO

TOTAL DIRECTIVE COVERAGE: 50.0%  COVERS: 6

ASSERTION RESULTS:
-----------------------------------------------------
Name                 File(Line)           Failure Pass
                                          Count   Count
-----------------------------------------------------
/top/DUT/assert__5   FIFO.sv(72)                0     0
/top/DUT/assert__4   FIFO.sv(71)                0     1
/top/DUT/assert__3   FIFO.sv(70)                0     1
/top/DUT/assert__2   FIFO.sv(69)                0     0
/top/DUT/assert__1   FIFO.sv(68)                0     1
/top/DUT/assert__0   FIFO.sv(67)                0     0
/top/TEST/#anonblk#182146786#30#4#/#ublk#182146786#30/immed__32
                     FIFO_tb.sv(32)             0     1
```