Momen Mostafa Mohamed Elzaghawy

FIFO project

Verification plan:

label	design requirement description	stimulus generation	functional coverage	functionality check
FIFO_1	the rst_n is asserted to check the reset functionality	directed and randomization		assertion to check rst_n functionality
FIFO_2	the rst_n is desserted and wr_en is asserted to check write functionality	randomization	cross coverage	assertion to check write functionality
FIFO_3	the rst_n is desserted and rd_en is asserted to check read functionality	randomization	cross coverage	assertion to check read functionality

Design (with bugs):

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

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

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

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

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

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

```
52  always @(posedge clk or negedge rst_n) begin
53      if (!rst_n) begin
54          count <= 0;
55      end
56      else begin
57          if  ( ({wr_en, rd_en} == 2'b10) && !full)
58              count <= count + 1;
59          else if ( ({wr_en, rd_en} == 2'b01) && !empty)
60              count <= count - 1;
61      end
62  end
63
64  assign full = (count == FIFO_DEPTH)? 1 : 0;
65  assign empty = (count == 0)? 1 : 0;
66  assign underflow = (empty && rd_en)? 1 : 0;
67  assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
68  assign almostempty = (count == 1)? 1 : 0;
69
70  endmodule
```

Bugs:

1- At rst_n is asserted overflow and underflow and wr_ack is deasserted
2- In line 35 full && wr_en
3- The logic when wr_en and rd_en is asserted is missing
4- underflow logic is not right
5- in line 67 almost full logic is not right (FIFO_DEPTH-1)
6- overflow must be 0 in successive write and underflow must be 0 in successive read

Design (with no bugs):

```
module FIFO(FIFO_if.DUT fifoif);


localparam max_fifo_addr = $clog2(fifoif.FIFO_DEPTH);

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

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

always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
        wr_ptr <= 0;
        fifoif.overflow <=0;  // bug1
        fifoif.wr_ack <=0;
    end
    else if (fifoif.wr_en && count < fifoif.FIFO_DEPTH) begin
        mem[wr_ptr] <= fifoif.data_in;
        fifoif.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
       fifoif.overflow <=0;
    end
    else begin
        fifoif.wr_ack <= 0;
        if (fifoif.full && fifoif.wr_en)  // bug2
            fifoif.overflow <= 1;
        else
            fifoif.overflow <= 0;
    end
end

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

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

always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({fifoif.wr_en, fifoif.rd_en} == 2'b10) && !fifoif.full)
            count <= count + 1;
        else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b01) && !fifoif.empty)
            count <= count - 1;
          else if ({fifoif.wr_en , fifoif.rd_en} == 2'b11) begin   //bug3
            if (fifoif.full)
            count<=count-1;
            if (fifoif.empty)
            count<=count+1;
          end
    end
end

assign fifoif.full = (count == fifoif.FIFO_DEPTH)? 1 : 0;
assign fifoif.empty = (count == 0)? 1 : 0;
assign fifoif.almostfull = (count == fifoif.FIFO_DEPTH-1)? 1 : 0; //bug4
assign fifoif.almostempty = (count == 1)? 1 : 0;


endmodule
```

Assertions:

```
property no_1;
@(posedge fifoif.clk) !fifoif.rst_n |=> $past(!wr_ptr) && $past(!rd_ptr) && count == 0;
endproperty
property no_2;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && !fifoif.full |=> fifoif.wr_ack;
endproperty
property no_3;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && fifoif.full |=> fifoif.overflow;
endproperty
property no_4;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.rd_en && fifoif.empty |=> fifoif.underflow;
endproperty
property no_5;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) count==0 |=> $past(fifoif.empty);
endproperty
property no_6;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) count==fifoif.FIFO_DEPTH |=> $past(fifoif.full);
endproperty
property no_7;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) count==fifoif.FIFO_DEPTH-1 |=> $past(fifoif.almostfull);
endproperty
property no_8;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) count==1 |=> $past(fifoif.almostempty);
endproperty
property no_9;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && !fifoif.full && wr_ptr==fifoif.FIFO_DEPTH-1 |=> wr_ptr==0 ;
endproperty
property no_10;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n)  fifoif.rd_en && !fifoif.empty && rd_ptr==fifoif.FIFO_DEPTH-1 |=>  rd_ptr==0 ;
endproperty
property no_11;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) wr_ptr<=fifoif.FIFO_DEPTH-1 && rd_ptr<=fifoif.FIFO_DEPTH-1 && count<=fifoif.FIFO_DEPTH ;
endproperty
```

```
reset_assert:assert property (no_1);
reset_cover:cover property (no_1);
wr_ack_assert:assert property (no_2);
wr_ack_cover:cover property (no_2);
overflow_assert:assert property (no_3);
overflow_cover:cover property (no_3);
underflow_assert:assert property (no_4);
underflow_cover:cover property (no_4);
empty_assert:assert property (no_5);
empty_cover:cover property (no_5);
full_assert:assert property (no_6);
full_cover:cover property (no_6);
almostfull_assert:assert property (no_7);
almostfull_cover:cover property (no_7);
almostempty_assert:assert property (no_8);
almostempty_cover:cover property (no_8);
wraparound_wr_assert:assert property (no_9);
wraparound_wr_cover:cover property (no_9);
wraparound_rd_assert:assert property (no_10);
wraparound_rd_cover:cover property (no_10);
threshold_assert:assert property (no_11);
threshold_cover:cover property (no_11);
endmodule
```

Interface:

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

Internal package:

```
package internal_pkg;
bit test_finished ;
integer error_count=0,correct_count=0;

endpackage
```

Transaction package:

```
package FIFO_transaction_pkg;
 class FIFO_transaction ;
 parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

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 RD_EN_ON_DIST=30,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 rst_n_const {rst_n dist {0:=2,1:=98};}
constraint wr_en_const {
    wr_en dist{1:=WR_EN_ON_DIST,0:=(100-WR_EN_ON_DIST)};
}
constraint rd_en_const {
    rd_en dist{1:=RD_EN_ON_DIST,0:=(100-RD_EN_ON_DIST)};
}
 endclass
endpackage
```

Coverage package:

```
package FIFO_coverage_pkg;
import FIFO_transaction_pkg::*;

class FIFO_coverage;
    FIFO_transaction F_cvg_txn ;
    covergroup cg ;
   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;
   underflow_cp:coverpoint F_cvg_txn.underflow;
   full_cp:coverpoint F_cvg_txn.full;
   almostempty_cp:coverpoint F_cvg_txn.almostempty;
   almostfull_cp:coverpoint F_cvg_txn.almostfull;
   empty_cp:coverpoint F_cvg_txn.empty;

   cross_1: cross wr_en_cp,rd_en_cp,wr_ack_cp;
   cross_2: cross wr_en_cp,rd_en_cp,overflow_cp;
   cross_3: cross wr_en_cp,rd_en_cp,underflow_cp;
   cross_4: cross wr_en_cp,rd_en_cp,full_cp;
   cross_5: cross wr_en_cp,rd_en_cp,almostempty_cp;
   cross_6: cross wr_en_cp,rd_en_cp,almostfull_cp;
   cross_7: cross wr_en_cp,rd_en_cp,empty_cp;
    endgroup
    function new();
        cg=new();
    endfunction
    function void sample_data(input FIFO_transaction F_txn);
      F_cvg_txn = F_txn ;
      cg.sample();
    endfunction
endclass
endpackage
```

Scoreboard package:

```
package FIFO_scoreboard_pkg;
import FIFO_transaction_pkg::*;
import internal_pkg::*;
class FIFO_scoreboard;
 FIFO_transaction tra = new();
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

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

logic [FIFO_WIDTH-1:0] fifo_queue [$];
int count_fifo =0;

task check_data (input FIFO_transaction t_obj) ;
   reference_model(t_obj) ;
    if ( ( data_out_ref===t_obj.data_out) )
  correct_count++;
  else begin
    error_count++;
    $display("%t,Error!,data_out=%0h,data_out_ref=%0h",$time,t_obj.data_out,data_out_ref);
  end
endtask
function void reference_model(input FIFO_transaction t_obj);
if(!t_obj.rst_n) begin
    fifo_queue <= {};
    count_fifo <= 0 ;
end
else begin
    if (t_obj.wr_en && count_fifo < FIFO_DEPTH) begin
        fifo_queue.push_back(t_obj.data_in);
        count_fifo <= fifo_queue.size();
    end
    if (t_obj.rd_en && count_fifo!=0) begin
        data_out_ref <= fifo_queue.pop_front();
        count_fifo <= fifo_queue.size();
    end
end

endfunction
endclass
endpackage
```

Monitor:

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

module FIFO_MONITOR (FIFO_if.MON fifoif);
    FIFO_transaction tra = new();
    FIFO_scoreboard sco = new();
    FIFO_coverage cov= new() ;

    initial begin

        forever begin
            @(negedge fifoif.clk);
            assert(tra.randomize());
            tra.rst_n= fifoif.rst_n;
            tra.data_in = fifoif.data_in;
            tra.wr_en = fifoif.wr_en;
            tra.rd_en = fifoif.rd_en;
            tra.full = fifoif.full ;
            tra.empty = fifoif.empty ;
            tra.wr_ack  = fifoif.wr_ack ;
            tra.almostempty  = fifoif.almostempty ;
            tra.almostfull  = fifoif.almostfull ;
            tra.data_out  = fifoif.data_out ;
            tra.overflow  = fifoif.overflow ;
            tra.underflow  = fifoif.underflow ;

           fork
               begin
                   cov.sample_data(tra);
               end
               begin
                   sco.check_data(tra);
               end
            join
             if (test_finished) begin
            $display("correct_count= %0d,error_count=%0d",correct_count,error_count);
            $stop;
        end
    end
    end

endmodule
```

Testbench:

```
import FIFO_transaction_pkg::*;
import FIFO_scoreboard_pkg::*;
import internal_pkg::*;
module FIFO_tb (FIFO_if.TEST fifoif);
    FIFO_transaction tra = new();

    initial begin
      fifoif.rst_n=0;
      fifoif.rd_en=0;
      fifoif.wr_en=0;
      fifoif.data_in=0;
      @(negedge fifoif.clk);
      fifoif.rst_n=1;
      @(negedge fifoif.clk);
      @(negedge fifoif.clk);
     repeat (1000)begin
        assert(tra.randomize());
        fifoif.rst_n=tra.rst_n;
        fifoif.data_in=tra.data_in;
        fifoif.wr_en=tra.wr_en;
        fifoif.rd_en=tra.rd_en;
        @(negedge fifoif.clk);
        @(negedge fifoif.clk);
      end


      test_finished = 1;


    end
endmodule
```

Top:

```
module FIFO_top ();

    bit clk;
    initial begin
        clk=0;
        forever #1 clk=~clk;
    end
    FIFO_if fifoif (clk);
    FIFO DUT (fifoif);
    FIFO_tb TEST (fifoif);
    FIFO_MONITOR MON (fifoif);

endmodule
```

Src_files_FIFO.list:

```
FIFO_if.sv
internal_pkg.sv
FIFO_transaction.sv
FIFO_scoreboard.sv
FIFO_coverage.sv
FIFO.sv
FIFO_tb.sv
FIFO_MONITOR.sv
FIFO_top.sv
```

Do file:

```
vlib work
vlog -f src_files_FIFO.list   +cover -covercells
vsim -voptargs=+acc work.FIFO_top -cover
add wave /FIFO_top/fifoif/*
coverage save FIFO_tb.ucdb -onexit -du FIFO
run 0
```

Code coverage:

```
Coverage Report by DU with details

================================================================================
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        28        28         0   100.00%

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

Branch Coverage for Design Unit work.FIFO

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File FIFO.sv
------------------------------------IF Branch------------------------------------
    19                                      2024     Count coming in to IF
    19             1                          62        if (!fifoif.rst_n) begin
    24             1                         743        else if (fifoif.wr_en && count < fifoif.FIFO_DEPTH) begin
    30             1                        1219        else begin
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
    32                                      1219     Count coming in to IF
    32             1                         659                if (fifoif.full && fifoif.wr_en)  // bug2
    34             1                         560                else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    40                                      2024     Count coming in to IF
    40             1                          62        if (!fifoif.rst_n) begin
    44             1                         590        else if (fifoif.rd_en && count != 0) begin
    48             1                        1372         else begin
Branch totals: 3 hits of 3 branches = 100.00%
```

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

================================Condition Details================================

Condition Coverage for Design Unit work.FIFO --

  File FIFO.sv
----------------Focused Condition View-------------------
Line       24 Item    1  (fifoif.wr_en && (count < fifoif.FIFO_DEPTH))
Condition totals: 2 of 2 input terms covered = 100.00%

----------------Focused Condition View-------------------
Line       32 Item    1  (fifoif.full && fifoif.wr_en)
Condition totals: 2 of 2 input terms covered = 100.00%

----------------Focused Condition View-------------------
Line       44 Item    1  (fifoif.rd_en && (count != 0))
Condition totals: 2 of 2 input terms covered = 100.00%

----------------Focused Condition View-------------------
Line       49 Item    1  (fifoif.empty && fifoif.rd_en)
Condition totals: 2 of 2 input terms covered = 100.00%

----------------Focused Condition View-------------------
Line       61 Item    1  ((~fifoif.rd_en && fifoif.wr_en) && ~fifoif.full)
Condition totals: 3 of 3 input terms covered = 100.00%
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      27        27         0   100.00%

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

Statement Coverage for Design Unit work.FIFO --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File FIFO.sv
    8                                                module FIFO(FIFO_if.DUT fifoif);
    9
    10
    11                                               localparam max_fifo_addr = $clog2(fifoif.FIFO_DEPTH);
    12
    13                                               reg [fifoif.FIFO_WIDTH-1:0] mem [fifoif.FIFO_DEPTH-1:0];
    14
    15                                               reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
    16                                               reg [max_fifo_addr:0] count;
    17
    18             1                          2024  always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    19                                                  if (!fifoif.rst_n) begin
    20             1                            62                wr_ptr <= 0;
    21             1                            62             fifoif.overflow <=0;  // bug1
    22             1                            62             fifoif.wr_ack <=0;
    23                                                  end
    24                                                  else if (fifoif.wr_en && count < fifoif.FIFO_DEPTH) begin
    25             1                           743                mem[wr_ptr] <= fifoif.data_in;
    26             1                           743                fifoif.wr_ack <= 1;
    27             1                           743                wr_ptr <= wr_ptr + 1;
    28
```

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

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

Toggle Coverage for Design Unit work.FIFO

                                            Node      1H->0L      0L->1H  "Coverage"
                                            ---------------------------------------


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

Toggle Coverage      =     100.00% (20 of 20 bins)

================================================================================
=== Design Unit: work.FIFO_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%

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

Toggle Coverage for Design Unit work.FIFO_if
```

Functional coverage:

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_p...		100.00%							
TYPE cg		100.00%	100	100.00...		✓	auto(1)		
CVP cg::wr...		100.00%	100	100.00...		✓			
CVP cg::rd...		100.00%	100	100.00...		✓			
CVP cg::wr...		100.00%	100	100.00...		✓			
CVP cg::ov...		100.00%	100	100.00...		✓			
CVP cg::un...		100.00%	100	100.00...		✓			
CVP cg::ful...		100.00%	100	100.00...		✓			
CVP cg::al...		100.00%	100	100.00...		✓			
CVP cg::al...		100.00%	100	100.00...		✓			
CVP cg::e...		100.00%	100	100.00...		✓			
CROSS cg:...		100.00%	100	100.00...		✓			
CROSS cg:...		100.00%	100	100.00...		✓			
CROSS cg:...		100.00%	100	100.00...		✓			
CROSS cg:...		100.00%	100	100.00...		✓			
CROSS cg:...		100.00%	100	100.00...		✓			
CROSS cg:...		100.00%	100	100.00...		✓			
CROSS cg:...		100.00%	100	100.00...		✓			

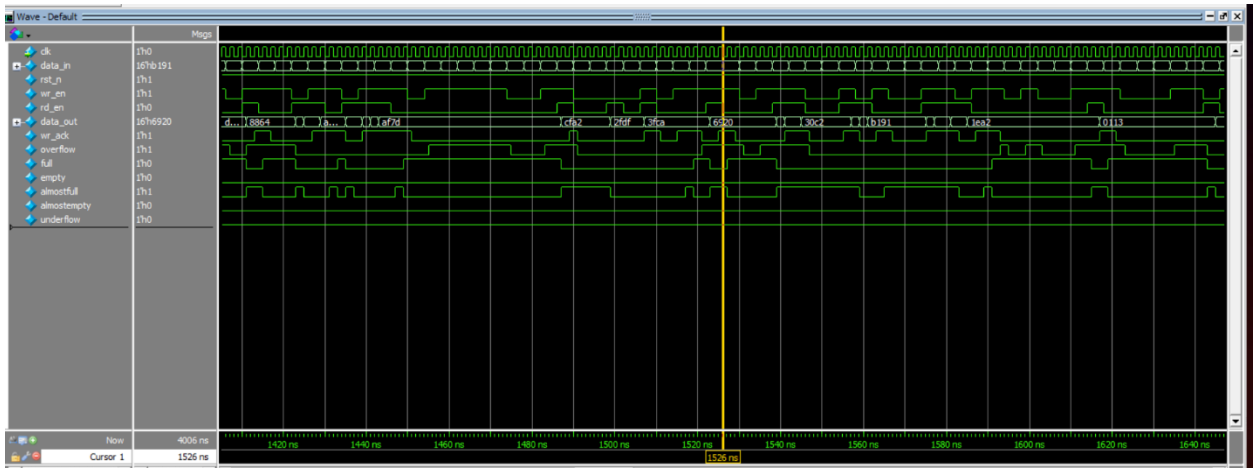
Assertions | Covergroups | Wave | FIFO.sv | FIFO_MONITOR.sv

Assertion coverage:

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_top/DUT/reset_cover	SVA	✓	Off	49	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/wr_ack_cover	SVA	✓	Off	712	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/overflow_cover	SVA	✓	Off	616	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/underflow_cover	SVA	✓	Off	14	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/empty_cover	SVA	✓	Off	70	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/full_cover	SVA	✓	Off	851	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/almostfull_cover	SVA	✓	Off	459	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/almostempty_cover	SVA	✓	Off	59	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/wraparound_wr_cover	SVA	✓	Off	79	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/wraparound_rd_cover	SVA	✓	Off	60	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/threshold_cover	SVA	✓	Off	1954	1	Unli...	1	100%		✓	0	0	0 ns	0

Wave and transcript:




```
/SIM 2> run -all
# correct_count= 2003,error_count=0
# ** Note: $stop    : FIFO_MONITOR.sv(40)
#    Time: 4006 ns  Iteration: 1  Instance: /FIFO_top/MON
# Break in Module FIFO_MONITOR at FIFO_MONITOR.sv line 40
```