SV Project - Synchronous FIFO

By: Amr Ayman Mohamed Abdo

Contents

Verification Plan

1. Test writing operation by performing writes more often than reads. **(Only_Write)**
2. Test reading operation by performing reads more often than writes. **(Only_Read)**
3. Test the FIFO with multiple concurrent writers and readers to ensure data integrity and proper flag behavior. **(Concurrent_Write_Read)**
4. Test simultaneous write-read operation by performing reads and writes with equal distribution probability. **(Simultaneous_Read_Write)**
5. Fill the FIFO to its maximum capacity and attempt more writes to ensure the `full` and `almsotfull` flags are set and `overflow` is triggered. **(Write_Full)**
6. Completely empty the FIFO and attempt more reads to check if the `empty` and `almostempty` flags are set and `underflow` is triggered. **(Read_Empty)**
7. With a full FIFO, perform a simultaneous write and read to check if the FIFO correctly updates the `full` flag and does not set the `overflow`. **(Simultaneous_Read_Write_Full)**
8. With an empty FIFO, perform a simultaneous read and write and verify the correct behavior of the `empty` flag and does not set the `underflow`. **(Simultaneous_Read_Write_Full)**
9. On reset, verify that the FIFO is cleared and all flags are in the correct initial state. **(Reset)**
10. Perform random sequences of reads and writes to test the FIFO's robustness. **(Read_Write_Randomized)**
11. Test flags stability when neither reading nor writing is taking place. **(Flags_Stability)**

Verification Requirements Document Snippet

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
Only_Write	Write unique values to the FIFO while reading is disabled	constraint rd_en to be zero while randomizing wr_en to be high most of the time. Constraint data_in to have unique values.	Included in **write_seq** that ensures that 8 consecutive writes have taken place	Outputs are checked against reference model + **assert_full, assert_almostfull & assert_wr_ack**
Only_Read	Read the previously written values to the FIFO while writing is disabled	constraint wr_en to be zero while randomizing rd_en to be high most of the time.	Included in **read_seq** that ensures that 8 consecutive reads have taken place	Outputs are checked against reference model + **assert_empty & assert_almostempty**
Concurrent_Write_Read	Loop containing a sequence of writes followed by another sequence of reads	constraint rd_en to be zero while randomizing wr_en to be high most of the time while writing and vice versa	Included in **empty_to_not** and **full_to_not** that ensure that a writing sequence took place after a reading sequence and vice versa	Outputs are checked against reference model
Simultaneous_Read_Write	Read and Write simultaneously and check for the flags remaining low.	constraint wr_en and rd_en to be high simultaneously most of the time.	Included in **write_read_simlt** cross bin that covers read cross write	Outputs are checked against reference model
Write_Full	Write to a full FIFO and check for the overflow to be deasserted when rd_en is asserted	When the FIFO is full constraint wr_en to be high most of the time and rd_en to be low most of the time	Included in **write_full** cross coverage bin that covers write cross full	Outputs are checked against reference model + **assert_overflow**
Read_Empty	Read from an empty FIFO and check for the underflow to be deasserted when rd_en is asserted	When the FIFO is empty constraint rd_en to be high most of the time and wr_en to be low most of the time	Included in **read_empty** cross coverage bin that covers read cross empty	Outputs are checked against reference model + **assert_underflow**
Reset	In reset cases check for the FIFO to be cleared and empty is asserted and full is deasserted	Constraint reset to be activated more frequently while randomizing the other inputs	Included in **full_to_empty** bin that covers the transition from a full FIFO to an empty FIFO (possible only if reset is activated)	Outputs are checked against reference model + **assert_count_rst, assert_wr_ptr_rst & assert_rd_ptr_rst**
Simultaneous_Read_Write_Full	When the FIFO is full and reading and writing are taking place simultaneously, then overflow must remain low	When the FIFO is full, constraint wr_en and rd_en to be high simultaneously most of the time.	Included in **write_read_simlt_full** cross coverage between **write_read_simlt** and **full**	Outputs are checked against reference model + **assert_overflow**
Simultaneous_Read_Write_Empty	When the FIFO is empty and reading and writing are taking place simultaneously, then underflow must remain low	When the FIFO is empty, constraint wr_en and rd_en to be high simultaneously most of the time.	Included in **write_read_simlt_empty** cross coverage between **write_read_simlt** and **empty**	Outputs are checked against reference model + **assert_underflow**
Read_Write_Randomized	Perform random sequences of reads and writes to test the FIFO's robustness	Randomize inputs without constraints		Outputs are checked against reference model
Flags_Stability	When neither reading nor writing is taking place, flags must remain stable	Constraint rd_en and wr_en to be low most of the time	Included in **write_nor_read** that covers that write and read are deactivated simultaneously	Outputs are checked against reference model + **assert_full, assert_empty, assert_almostfull, assert_almostempty, assert_overflow, assert_underflow, assert_wr_ack**

Code Snippets

FIFO_transaction_pkg

```
package FIFO_transaction_pkg;

    class FIFO_transaction;
        parameter FIFO_WIDTH = 16;
        parameter FIFO_DEPTH = 8;

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

        int WR_EN_ON_DIST, RD_EN_ON_DIST, RST_ON_DIST;

        constraint Reset_Constraint {
            rst_n dist {1:=(100-RST_ON_DIST), 0:=RST_ON_DIST};
        }

        constraint General_Constraints {
            wr_en dist {1:=WR_EN_ON_DIST, 0:=(100-WR_EN_ON_DIST)};
            rd_en dist {1:=RD_EN_ON_DIST, 0:=(100-RD_EN_ON_DIST)};
            (full == 1)  -> wr_en dist {1:=90, 0:=10};
            (empty == 1) -> rd_en dist {1:=90, 0:=10};
            unique {data_in};
        }

        constraint Simultaneous_Read_Write {
            (wr_en && rd_en) dist {1:=90, 0:=10};
            (full == 1)  -> wr_en dist {1:=90, 0:=10};
            (empty == 1) -> rd_en dist {1:=90, 0:=10};
            unique {data_in};
        }

        function new(int RST_ON_DIST = 5, int WR_EN_ON_DIST = 70, int RD_EN_ON_DIST = 30);
            this.RST_ON_DIST    = RST_ON_DIST;
            this.WR_EN_ON_DIST  = WR_EN_ON_DIST;
            this.RD_EN_ON_DIST  = RD_EN_ON_DIST;
        endfunction : new
    endclass : FIFO_transaction

endpackage : FIFO_transaction_pkg
```

FIFO_coverage_pkg

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

    class FIFO_coverage;
        FIFO_transaction F_cvg_txn = new();

        covergroup FIFO_cvg_gp;
            //Labeling coverpoints to be visible in the functional coverage report
            write_cp:                   coverpoint F_cvg_txn.wr_en              {bins wr_en[]               = {[0:1]};}
            read_cp:                    coverpoint F_cvg_txn.rd_en              {bins rd_en[]               = {[0:1]};}
            full_cp:                    coverpoint F_cvg_txn.full               {bins full[]                = {[0:1]};}
            empty_cp:                   coverpoint F_cvg_txn.empty              {bins empty[]               = {[0:1]};}
            almostfull_cp:              coverpoint F_cvg_txn.almostfull         {bins almostfull[]          = {[0:1]};}
            almostempty_cp:             coverpoint F_cvg_txn.almostempty        {bins almostempty[]         = {[0:1]};}
            overflow_cp:                coverpoint F_cvg_txn.overflow           {bins overflow[]            = {[0:1]};}
            underflow_cp:               coverpoint F_cvg_txn.underflow          {bins underflow[]           = {[0:1]};}
            wr_ack_cp:                  coverpoint F_cvg_txn.wr_ack             {bins wr_ack[]              = {[0:1]};}
            not_to_empty_cp:            coverpoint F_cvg_txn.empty              {bins not_to_empty          = (0=>1);}


            write_seq_cp:               coverpoint F_cvg_txn.wr_en              {bins write_seq             = (0=>1[*8]=>0);}
            read_seq_cp:                coverpoint F_cvg_txn.rd_en              {bins read_seq              = (0=>1[*8]=>0);}
            empty_to_not_cp:            coverpoint F_cvg_txn.empty              {bins empty_to_not          = (1=>0);}
            full_to_not_cp:             coverpoint F_cvg_txn.full               {bins full_to_not           = (1=>0);}
            write_read_simlt_cp:        cross write_cp, read_cp                 {bins write_read_simlt      = binsof(write_cp.wr_en[1]) && binsof(read_cp.rd_en[1]); option.cross_auto_bin_max = 0;}
            full_to_empty_cp:           cross not_to_empty_cp, full_to_not_cp   {bins full_to_empty         = binsof(full_to_not_cp) && binsof(not_to_empty_cp); option.cross_auto_bin_max = 0;}
            write_read_simlt_full_cp:   cross write_read_simlt_cp, full_cp      {bins write_read_simlt_full = binsof(write_read_simlt_cp) && binsof(full_cp.full[1]); option.cross_auto_bin_max = 0;}
            write_read_simlt_empty_cp:  cross write_read_simlt_cp, empty_cp     {bins write_read_simlt_empty= binsof(write_read_simlt_cp) && binsof(empty_cp.empty[1]); option.cross_auto_bin_max = 0;}
            write_nor_read_cp:          cross write_cp, read_cp                 {bins write_read_simlt      = binsof(write_cp.wr_en[0]) && binsof(read_cp.rd_en[0]); option.cross_auto_bin_max = 0;}

            write_cross_states: cross write_cp, full_cp, empty_cp, almostfull_cp, almostempty_cp, overflow_cp, underflow_cp, wr_ack_cp {
                bins write_full =       binsof(write_cp) && binsof(full_cp);
                bins write_empty =      binsof(write_cp) && binsof(empty_cp);
                bins write_almostfull = binsof(write_cp) && binsof(almostfull_cp);
                bins write_almostempty= binsof(write_cp) && binsof(almostempty_cp);
                bins write_overflow =   binsof(write_cp) && binsof(overflow_cp);
                bins write_underflow =  binsof(write_cp) && binsof(underflow_cp);
                bins write_wr_ack =     binsof(write_cp) && binsof(wr_ack_cp);
            }

            read_cross_states: cross read_cp, full_cp, empty_cp, almostfull_cp, almostempty_cp, overflow_cp, underflow_cp, wr_ack_cp {
                bins read_full =        binsof(read_cp) && binsof(full_cp);
                bins read_empty =       binsof(read_cp) && binsof(empty_cp);
                bins read_almostfull =  binsof(read_cp) && binsof(almostfull_cp);
                bins read_almostempty = binsof(read_cp) && binsof(almostempty_cp);
                bins read_overflow =    binsof(read_cp) && binsof(overflow_cp);
                bins read_underflow =   binsof(read_cp) && binsof(underflow_cp);
                bins read_wr_ack =      binsof(read_cp) && binsof(wr_ack_cp);
            }

        endgroup : FIFO_cvg_gp

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

        function new();
            FIFO_cvg_gp = new();
        endfunction : new
    endclass : FIFO_coverage
endpackage : FIFO_coverage_pkg
```

FIFO_monitor

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

module FIFO_monitor (FIFO_if FIFOif);
    FIFO_transaction FIFO_tr = new();
    FIFO_scoreboard FIFO_sb = new();
    FIFO_coverage FIFO_cvg = new();

    initial begin
        forever @(negedge FIFOif.clk) begin
            //Assigning inputs
            FIFO_tr.data_in = FIFOif.data_in;
            FIFO_tr.wr_en = FIFOif.wr_en;
            FIFO_tr.rd_en = FIFOif.rd_en;
            FIFO_tr.rst_n = FIFOif.rst_n;

            //Assigning outputs
            FIFO_tr.full = FIFOif.full;
            FIFO_tr.empty = FIFOif.empty;
            FIFO_tr.almostfull = FIFOif.almostfull;
            FIFO_tr.almostempty = FIFOif.almostempty;
            FIFO_tr.wr_ack = FIFOif.wr_ack;
            FIFO_tr.overflow = FIFOif.overflow;
            FIFO_tr.underflow = FIFOif.underflow;
            FIFO_tr.data_out = FIFOif.data_out;

            //Assigning Reference Outputs
            FIFO_sb.full_ref = FIFOif.full_ref;
            FIFO_sb.empty_ref = FIFOif.empty_ref;
            FIFO_sb.almostfull_ref = FIFOif.almostfull_ref;
            FIFO_sb.almostempty_ref = FIFOif.almostempty_ref;
            FIFO_sb.wr_ack_ref = FIFOif.wr_ack_ref;
            FIFO_sb.overflow_ref = FIFOif.overflow_ref;
            FIFO_sb.underflow_ref = FIFOif.underflow_ref;
            FIFO_sb.data_out_ref = FIFOif.data_out_ref;

            fork
                begin
                    FIFO_cvg.sample_data(FIFO_tr);
                end

                begin
                    FIFO_sb.check_data(FIFO_tr);
                end
            join

            if(test_finished) begin
                $display("FIFO's test has been successfuly finished\nerror_count = %0d\ncorrect_count = %0d", error_count, correct_count);
                $stop;
            end

        end
    end
endmodule : FIFO_monitor
```

FIFO_scoreboard_pkg

```
1   package FIFO_scoreboard_pkg;
2       import FIFO_transaction_pkg::*;
3       import shared_pkg::*;
4
5       class FIFO_scoreboard;
6           FIFO_transaction F_sb_txn = new();
7
8           parameter FIFO_WIDTH = 16;
9           parameter FIFO_DEPTH = 8;
10
11          bit [FIFO_WIDTH-1:0] data_out_ref;
12          bit wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
13
14          bit [FIFO_WIDTH-1:0] FIFO_model[$]; //golden model for the FIFO
15
16          task check_data(FIFO_transaction F_txn);
17              F_sb_txn = F_txn;
18
19              if (F_sb_txn.data_out == data_out_ref)
20                  correct_count++;
21              else begin
22                  $error("Mismatch: data_out (%0d) != data_out_ref (%0d)", F_sb_txn.data_out, data_out_ref);
23                  error_count++;
24              end
25
26              if (F_sb_txn.full == full_ref)
27                  correct_count++;
28              else begin
29                  $error("Mismatch: full (%0d) != full_ref (%0d)", F_sb_txn.full, full_ref);
30                  error_count++;
31              end
32
33              if (F_sb_txn.empty == empty_ref)
34                  correct_count++;
35              else begin
36                  $error("Mismatch: empty (%0d) != empty_ref (%0d)", F_sb_txn.empty, empty_ref);
37                  error_count++;
38              end
39
40              if (F_sb_txn.almostfull == almostfull_ref)
41                  correct_count++;
42              else begin
43                  $error("Mismatch: almostfull (%0d) != almostfull_ref (%0d)", F_sb_txn.almostfull, almostfull_ref);
44                  error_count++;
45              end
46
47              if (F_sb_txn.almostempty == almostempty_ref)
48                  correct_count++;
49              else begin
50                  $error("Mismatch: almostempty (%0d) != almostempty_ref (%0d)", F_sb_txn.almostempty, almostempty_ref);
51                  error_count++;
52              end
53
54              if (F_sb_txn.overflow == overflow_ref)
55                  correct_count++;
56              else begin
57                  $error("Mismatch: overflow (%0d) != overflow_ref (%0d)", F_sb_txn.overflow, overflow_ref);
58                  error_count++;
59              end
60
61              if (F_sb_txn.underflow == underflow_ref)
62                  correct_count++;
63              else begin
64                  $error("Mismatch: underflow (%0d) != underflow_ref (%0d)", F_sb_txn.underflow, underflow_ref);
65                  error_count++;
66              end
67
68              if (F_sb_txn.wr_ack == wr_ack_ref)
69                  correct_count++;
70              else begin
71                  $error("Mismatch: wr_ack (%0d) != wr_ack_ref (%0d)", F_sb_txn.wr_ack, wr_ack_ref);
72                  error_count++;
73              end
74          endtask : check_data
75
76      endclass : FIFO_scoreboard
77
78  endpackage : FIFO_scoreboard_pkg
```

FIFO_ref (reference model)

```
module FIFO_ref(FIFO_if.REF FIFOif);
    logic [FIFOif.FIFO_WIDTH-1:0] mem[$];
    bit wr_ack_next, wr_ack_next1;

    always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
        if(~FIFOif.rst_n) begin
            mem.delete();
            FIFOif.overflow_ref <= 0;
            FIFOif.wr_ack_ref <= 0;
        end
        else begin
            if(FIFOif.wr_en) begin
                if(!FIFOif.full_ref) begin
                    mem.push_front(FIFOif.data_in);
                    FIFOif.wr_ack_ref <= 1;
                    FIFOif.overflow_ref <= 0;
                end else begin
                    FIFOif.wr_ack_ref <= 0;
                    FIFOif.overflow_ref <= 1;
                end
            end else begin
                FIFOif.wr_ack_ref <= 0;
                FIFOif.overflow_ref <= 0;
            end
        end
    end

    always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
        if(~FIFOif.rst_n) begin
            mem.delete();
            FIFOif.data_out_ref <= 0;
            FIFOif.underflow_ref <= 0;
        end
        else begin
            if(FIFOif.rd_en) begin
                if(!FIFOif.empty_ref) begin
                    FIFOif.data_out_ref <= mem.pop_back();
                    FIFOif.underflow_ref <= 0;
                end else begin
                    FIFOif.underflow_ref <= 1;
                end
            end else
                FIFOif.underflow_ref <= 0;
        end
    end

    always_comb begin
        case(mem.size())
            0:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 1;
                FIFOif.almostempty_ref = 0;
            end
            1:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 1;
            end
            FIFOif.FIFO_DEPTH-1:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 1;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 0;
            end
            FIFOif.FIFO_DEPTH:
            begin
                FIFOif.full_ref = 1;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 0;
            end
            default:
            begin
                FIFOif.full_ref = 0;
                FIFOif.almostfull_ref = 0;
                FIFOif.empty_ref = 0;
                FIFOif.almostempty_ref = 0;
            end
        endcase // mem.size()
    end

endmodule
```

FIFO_tb

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

module FIFO_tb(FIFO_if.TEST FIFOif);
    FIFO_transaction FIFO_tr = new();

    //Sequence Items
    FIFO_transaction tr_wr = new(5, 95, 5);     //Only_Write
    FIFO_transaction tr_rd = new(5, 5, 95);     //Only_Read
    FIFO_transaction tr_sm = new(5, 90, 90);    //Simultaneous_Read_Write
    FIFO_transaction tr_wf = new(5, 90, 10);    //Write_Full
    FIFO_transaction tr_re = new(5, 10, 90);    //Read_Empty
    FIFO_transaction tr_rs = new(50, 70, 30);   //Reset
    FIFO_transaction tr_sf = new(5, 100, 100);  //Simultaneous_Read_Write_Full


    initial begin
        //Customizing the constraints for each test case
        prepare_items();

        //Initialize the design
        reset();

        //Only_Write
        repeat(500) begin
            randomize_then_assign(tr_wr);
            @(negedge FIFOif.clk);
        end

        //Only_Read
        repeat(500) begin
            randomize_then_assign(tr_rd);
            @(negedge FIFOif.clk);
        end

        //Concurrent_Write_Read
        repeat(100) begin
            repeat(10) begin
                randomize_then_assign(tr_wr);
                @(negedge FIFOif.clk);
            end
            repeat(10) begin
                randomize_then_assign(tr_rd);
                @(negedge FIFOif.clk);
            end
        end

        //Simultaneous_Read_Write
        repeat(500) begin
            randomize_then_assign(tr_sm);
            @(negedge FIFOif.clk);
        end

        //Reset
        repeat(500) begin
            randomize_then_assign(tr_rs);
            @(negedge FIFOif.clk);
        end

        test_finished = 1;
    end

    task reset();
        FIFOif.rst_n = 0;
        @(negedge FIFOif.clk);
        FIFOif.rst_n = 1;
    endtask : reset
```

```
68
69          function void randomize_then_assign(FIFO_transaction tr);
70              FIFO_tr = tr;
71              assert(FIFO_tr.randomize());
72              //Assigning inputs
73              FIFOif.data_in = FIFO_tr.data_in;
74              FIFOif.wr_en = FIFO_tr.wr_en;
75              FIFOif.rd_en = FIFO_tr.rd_en;
76              FIFOif.rst_n = FIFO_tr.rst_n;
77          endfunction : randomize_then_assign
78
79          task fill_FIFO();
80              FIFOif.rst_n = 1;
81              FIFOif.wr_en = 1;
82              FIFOif.rd_en = 0;
83              repeat(10) begin
84                  FIFOif.data_in = $random();
85                  @(negedge FIFOif.clk);
86              end
87          endtask : fill_FIFO
88
89          task empty_FIFO();
90              FIFOif.rst_n = 1;
91              FIFOif.wr_en = 0;
92              FIFOif.rd_en = 1;
93              repeat(10) begin
94                  FIFOif.data_in = $random();
95                  @(negedge FIFOif.clk);
96              end
97          endtask : empty_FIFO
98
99          task prepare_items();
100             tr_wr.constraint_mode(0);
101             tr_wr.Reset_Constraint.constraint_mode(1);
102             tr_wr.General_Constraints.constraint_mode(1);
103
104             tr_rd.constraint_mode(0);
105             tr_rd.Reset_Constraint.constraint_mode(1);
106             tr_rd.General_Constraints.constraint_mode(1);
107
108             tr_sm.constraint_mode(0);
109             tr_sm.Reset_Constraint.constraint_mode(1);
110             tr_sm.Simultaneous_Read_Write.constraint_mode(1);
111
112             tr_rs.constraint_mode(0);
113             tr_rs.Reset_Constraint.constraint_mode(1);
114             tr_rs.General_Constraints.constraint_mode(1);
115         endtask : prepare_items
116
117     endmodule : FIFO_tb
```

FIFO_if

```
1  interface FIFO_if #(
2      parameter FIFO_WIDTH = 16,
3      parameter FIFO_DEPTH = 8)(input bit clk);
4
5      Logic [FIFO_WIDTH-1:0] data_in;
6      Logic rst_n, wr_en, rd_en;
7      Logic [FIFO_WIDTH-1:0] data_out;
8      Logic wr_ack, overflow;
9      Logic full, empty, almostfull, almostempty, underflow;
10
11     Logic [FIFO_WIDTH-1:0] data_out_ref;
12     Logic wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
13
14     modport DUT (
15         input data_in, clk, rst_n, wr_en, rd_en,
16         output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
17         );
18
19     modport REF (
20         input data_in, clk, rst_n, wr_en, rd_en,
21         output data_out_ref, wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref
22         );
23
24     modport TEST (
25         output data_in, rst_n, wr_en, rd_en,
26         input clk, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
27         );
28
29     modport MONITOR (input data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
30 endinterface : FIFO_if
```

FIFO_top

```
1  module FIFO_top ();
2      bit clk;
3
4      initial begin
5          forever #10 clk = ~clk;
6      end
7
8      FIFO_if FIFOif(clk);
9      FIFO DUT(FIFOif);
10     FIFO_ref REF(FIFOif);
11     FIFO_tb TEST(FIFOif);
12     FIFO_monitor MONITOR(FIFOif);
13 endmodule : FIFO_top
```

run_FIFO (Do File)

```
vlib work
vlog -f src_files.list -mfcu +define+SIM +cover
vsim -voptargs=+acc work.FIFO_top -cover
add wave -position 1   -color red   sim:/FIFO_top/FIFOif/clk
add wave -position 2                sim:/FIFO_top/FIFOif/rst_n
add wave -position 3                sim:/FIFO_top/FIFOif/wr_en
add wave -position 4                sim:/FIFO_top/FIFOif/rd_en
add wave -position 5                sim:/FIFO_top/FIFOif/data_in
add wave -position 6   -color cyan  sim:/FIFO_top/FIFOif/data_out
add wave -position 7   -color gold  sim:/FIFO_top/FIFOif/data_out_ref
add wave -position 8   -color cyan  sim:/FIFO_top/FIFOif/wr_ack
add wave -position 9   -color gold  sim:/FIFO_top/FIFOif/wr_ack_ref
add wave -position 10  -color cyan  sim:/FIFO_top/FIFOif/full
add wave -position 11  -color gold  sim:/FIFO_top/FIFOif/full_ref
add wave -position 12  -color cyan  sim:/FIFO_top/FIFOif/empty
add wave -position 13  -color gold  sim:/FIFO_top/FIFOif/empty_ref
add wave -position 14  -color cyan  sim:/FIFO_top/FIFOif/almostfull
add wave -position 15  -color gold  sim:/FIFO_top/FIFOif/almostfull_ref
add wave -position 16  -color cyan  sim:/FIFO_top/FIFOif/almostempty
add wave -position 17  -color gold  sim:/FIFO_top/FIFOif/almostempty_ref
add wave -position 18  -color cyan  sim:/FIFO_top/FIFOif/overflow
add wave -position 19  -color gold  sim:/FIFO_top/FIFOif/overflow_ref
add wave -position 20  -color cyan  sim:/FIFO_top/FIFOif/underflow
add wave -position 21  -color gold  sim:/FIFO_top/FIFOif/underflow_ref
add wave -position 22  -color blue  sim:/FIFO_top/DUT/mem
add wave -position 23  -color blue  sim:/FIFO_top/DUT/count
add wave -position 24  -color blue  sim:/FIFO_top/DUT/wr_ptr
add wave -position 25  -color blue  sim:/FIFO_top/DUT/rd_ptr
add wave -position 26               sim:/FIFO_top/DUT/assert_full
add wave -position 27               sim:/FIFO_top/DUT/assert_empty
add wave -position 28               sim:/FIFO_top/DUT/assert_almostfull
add wave -position 29               sim:/FIFO_top/DUT/assert_almostempty
add wave -position 30               sim:/FIFO_top/DUT/assert_overflow
add wave -position 31               sim:/FIFO_top/DUT/assert_underflow
add wave -position 32               sim:/FIFO_top/DUT/assert_wr_ack
add wave -position 33               sim:/FIFO_top/DUT/assert_count_incr
add wave -position 34               sim:/FIFO_top/DUT/assert_count_decr
add wave -position 35               sim:/FIFO_top/DUT/assert_wr_ptr_incr
add wave -position 36               sim:/FIFO_top/DUT/assert_rd_ptr_incr
add wave -position 37               sim:/FIFO_top/DUT/assert_count_rst
add wave -position 38               sim:/FIFO_top/DUT/assert_wr_ptr_rst
add wave -position 39               sim:/FIFO_top/DUT/assert_rd_ptr_rst
.vcop Action toggleleafnames
coverage save FIFO_tb.ucdb -onexit -du FIFO
run -all
coverage report -detail -cvg -directive -comments -output fcover_report.txt {}
quit -sim
vcover report FIFO_tb.ucdb -details -annotate -all -output coverage_rpt.txt
```

src_files.list

```
shared_pkg.sv
FIFO_transaction.sv
FIFO_coverage.sv
FIFO_scoreboard.sv
FIFO_if.sv
FIFO_monitor.sv
FIFO_tb.sv
FIFO_top.sv
FIFO.sv
FIFO_ref.sv
```

Detected Bugs

1. **Extended count bus by 1-bit to allow it to reach FIFO_DEPTH**

Before:

```
22   reg [max_fifo_addr:0] count;
```

After:

```
15   reg [max_fifo_addr+1:0] count;
```

2. **Added this line to deassert overflow incase of writing**

Before:

```
28       else if (wr_en && count < FIFO_DEPTH) begin
29           mem[wr_ptr] <= data_in;
30           wr_ack <= 1;
31           wr_ptr <= wr_ptr + 1;
32       end
```

After:

```
21       else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
22           mem[wr_ptr] <= FIFOif.data_in;
23           FIFOif.wr_ack <= 1;
24           wr_ptr <= wr_ptr + 1;
25           FIFOif.overflow <= 0;
26       end
```

3. **Added this line to deassert underflow incase of reading**

Before:

```
46       else if (rd_en && count != 0) begin
47           data_out <= mem[rd_ptr];
48           rd_ptr <= rd_ptr + 1;
49       end
```

After:

```
40       else if (FIFOif.rd_en && count != 0) begin
41           FIFOif.data_out <= mem[rd_ptr];
42           rd_ptr <= rd_ptr + 1;
43           FIFOif.underflow <= 0;
44       end
```

4. **Removed this line since underflow is a sequential output not combinational**

Before:

```
66   assign underflow = (empty && rd_en)? 1 : 0;
```

After:

```
73   //assign FIFOif.underflow = (FIFOif.empty && FIFOif.rd_en)? 1 : 0;
```

5. **Added the sequential implementation of the underflow**

Before:

```
42    always @(posedge clk or negedge rst_n) begin
43        if (!rst_n) begin
44            rd_ptr <= 0;
45        end
46        else if (rd_en && count != 0) begin
47            data_out <= mem[rd_ptr];
48            rd_ptr <= rd_ptr + 1;
49        end
50    end
```

After:

```
36    always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
37        if (!FIFOif.rst_n) begin
38            rd_ptr <= 0;
39        end
40        else if (FIFOif.rd_en && count != 0) begin
41            FIFOif.data_out <= mem[rd_ptr];
42            rd_ptr <= rd_ptr + 1;
43            FIFOif.underflow <= 0;
44        end
45        else begin
46            if (FIFOif.empty & FIFOif.rd_en)
47                FIFOif.underflow <= 1;
48            else
49                FIFOif.underflow <= 0;
50        end
51    end
```

6. **Modified this line as FIFO is almostfull when count reaches FIFO_DEPTH-1**

Before:

```
67    assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
```

After:

```
74    assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH-1)? 1 : 0;
```

7. **Added this line as data_out must asynchronously be resetted (sequential)**

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        rd_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;
        rd_ptr <= 0;
        FIFOif.underflow <= 0;
```

8. Added this line as overflow must asynchronously be resetted (sequential)

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
        FIFOif.overflow <= 0;
```

9. Added this line as underflow must asynchronously be resetted (sequential)

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        rd_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;
        rd_ptr <= 0;
        FIFOif.underflow <= 0;
```

10. Added this line as wr_ack must asynchronously be resetted (sequential)

Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
```

After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
        FIFOif.overflow <= 0;
        FIFOif.wr_ack <= 0;
    end
```

DUT after fixing bugs and adding assertions

```
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+1:0] count;                                  //modified_code(1)      /*to allow count to reach FIFO_DEPTH*/

always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        wr_ptr <= 0;
        FIFOif.overflow <= 0;                                   //added_code (1)        /*overflow must asynchronously be resetted (sequential)*/
        FIFOif.wr_ack <= 0;                                     //added_code (2)        /*overflow must asynchronously be resetted (sequential)*/
    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;                                   //added_code (3)        /*overflow should be deasserted in the case of writing*/
    end
    else begin
        FIFOif.wr_ack <= 0;
        if (FIFOif.full & FIFOif.wr_en)
            FIFOif.overflow <= 1;
        else
            FIFOif.overflow <= 0;
    end
end

always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;                                   //added_code (4)        /*data_out must asynchronously be resetted (sequential)*/
        rd_ptr <= 0;
        FIFOif.underflow <= 0;                                  //added_code (5)        /*underflow must asynchronously be resetted (sequential)*/
    end
    else if (FIFOif.rd_en && count != 0) begin
        FIFOif.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        FIFOif.underflow <= 0;                                  //added_code (6)        /*underflow should be deasserted in the case of reading*/
    end
    else begin
        if (FIFOif.empty & FIFOif.rd_en)                        //added_code (7) begin  /*added the sequential implementation of underflow*/
            FIFOif.underflow <= 1;                              //******************//
        else                                                    //******************//
            FIFOif.underflow <= 0;                              //******************//
    end                                                         //added_code (7) 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   //added_code (8) begin  /*added the part that handles simultaneous read-write*/
            case ({FIFOif.full, FIFOif.empty})                  //******************//
                2'b01: count <= count + 1;                      //******************//
                2'b10: count <= count - 1;                      //******************//
            endcase                                             //******************//
        end                                                     //added_code (8) end
    end
end


assign FIFOif.full = (count == FIFOif.FIFO_DEPTH)? 1 : 0;
assign FIFOif.empty = (count == 0)? 1 : 0;
//assign FIFOif.underflow = (FIFOif.empty && FIFOif.rd_en)? 1 : 0;  //removed_code          /*underflow is sequential and not combinational*/
assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH-1)? 1 : 0;   //modified_code (2)     /*FIFO is almostfull when count reaches FIFO_DEPTH-1 not FIFO_DEPTH-2 as 0 is not included*/
assign FIFOif.almostempty = (count == 1)? 1 : 0;

`ifdef SIM

    //Assertions

    //Assertions to the combinational flags
    always_comb begin
        if(count == FIFOif.FIFO_DEPTH) begin
            assert_full:  assert final (FIFOif.full == 1) else $error("Mismatch: full_tb (%0d) != full_ref (1)", FIFOif.full);
            cover_full:  cover final (FIFOif.full == 1);
        end

        if(count == 0) begin
            assert_empty:  assert final (FIFOif.empty == 1) else $error("Mismatch: empty_tb (%0d) != empty_ref (1)", FIFOif.empty);
            cover_empty:  cover final (FIFOif.empty == 1);
        end

        if(count == FIFOif.FIFO_DEPTH-1) begin
            assert_almostfull:  assert final (FIFOif.almostfull == 1) else $error("Mismatch: almostfull_tb (%0d) != almostfull_ref (1)", FIFOif.almostfull);
            cover_almostfull:  cover final (FIFOif.almostfull == 1);
        end

        if(count == 1) begin
            assert_almostempty:  assert final (FIFOif.almostempty == 1) else $error("Mismatch: almostempty_tb (%0d) != almostempty_ref (1)", FIFOif.almostempty);
            cover_almostempty:  cover final (FIFOif.almostempty == 1);
        end
    end


    //Assertions to the sequential flags
    assert_overflow: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFOif.full) |=> (FIFOif.overflow == 1));
    cover_overflow: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFOif.full) |=> (FIFOif.overflow == 1));

    assert_underflow: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFOif.empty) |=> (FIFOif.underflow == 1));
    cover_underflow: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFOif.empty) |=> (FIFOif.underflow == 1));

    assert_wr_ack: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |=> (FIFOif.wr_ack == 1));
    cover_wr_ack: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |=> (FIFOif.wr_ack == 1));

    property count_plus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.wr_en && !FIFOif.full && !FIFOif.rd_en) || (FIFOif.wr_en && !FIFOif.full && FIFOif.rd_en && FIFOif.empty) |=> (count == $past(count) + 1);
    endproperty

    property count_minus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.rd_en && !FIFOif.empty && !FIFOif.wr_en) || (FIFOif.rd_en && !FIFOif.empty && FIFOif.wr_en && FIFOif.full) |=> (count == $past(count) - 1);
    endproperty

    property wr_ptr_plus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.wr_en && !FIFOif.full) |=> (wr_ptr == $past(wr_ptr) + 1) || (wr_ptr == 0);
    endproperty

    property rd_ptr_plus;
        @(posedge FIFOif.clk)
        disable iff (!FIFOif.rst_n)
        (FIFOif.rd_en && !FIFOif.empty) |=> (rd_ptr == $past(rd_ptr) + 1) || (rd_ptr == 0);
    endproperty

    property write_pointer_stable;
        @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
        FIFOif.full && FIFOif.wr_en && !FIFOif.rd_en |=> $stable(wr_ptr);
    endproperty

    property read_pointer_stable;
        @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
        FIFOif.empty && FIFOif.rd_en && !FIFOif.wr_en |=> $stable(rd_ptr);
    endproperty

    //Assertions to the counters
    assert_count_incr: assert property (count_plus);
    cover_count_incr: cover property (count_plus);

    assert_count_decr: assert property (count_minus);
    cover_count_decr: cover property (count_minus);

    assert_wr_ptr_incr: assert property (wr_ptr_plus);
    cover_wr_ptr_incr: cover property (wr_ptr_plus);

    assert_rd_ptr_incr: assert property (rd_ptr_plus);
    cover_rd_ptr_incr: cover property (rd_ptr_plus);

    assert_write_pointer_stable: assert property (write_pointer_stable);
    cover_write_pointer_stable: cover property (write_pointer_stable);

    assert_read_pointer_stable: assert property (read_pointer_stable);
    cover_read_pointer_stable: cover property (read_pointer_stable);

    always_comb begin
        if(~FIFOif.rst_n) begin
            assert_count_rst: assert final (count == 0);
            cover_count_rst: cover final (count == 0);

            assert_wr_ptr_rst: assert final (wr_ptr == 0);
            cover_wr_ptr_rst: cover final (wr_ptr == 0);

            assert_rd_ptr_rst: assert final (rd_ptr == 0);
            cover_rd_ptr_rst: cover final (rd_ptr == 0);
        end
    end

`endif
endmodule
```

QuestaSim Snippets

Wave colour guidelines

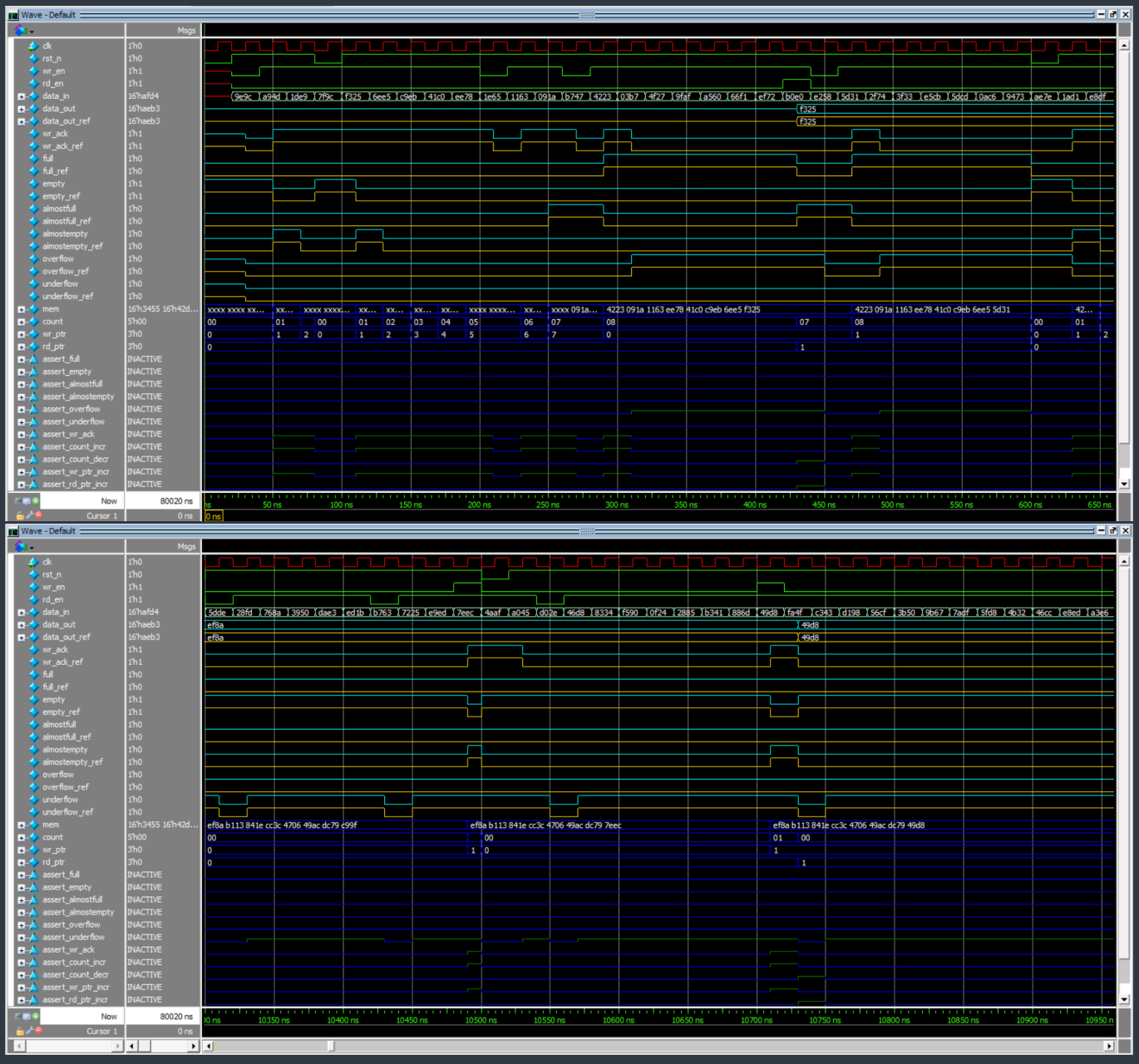
1. Cyan for the DUT outputs
2. Gold for the reference outputs
3. Red for the clock
4. Green for the inputs
5. Blue for the internal signals

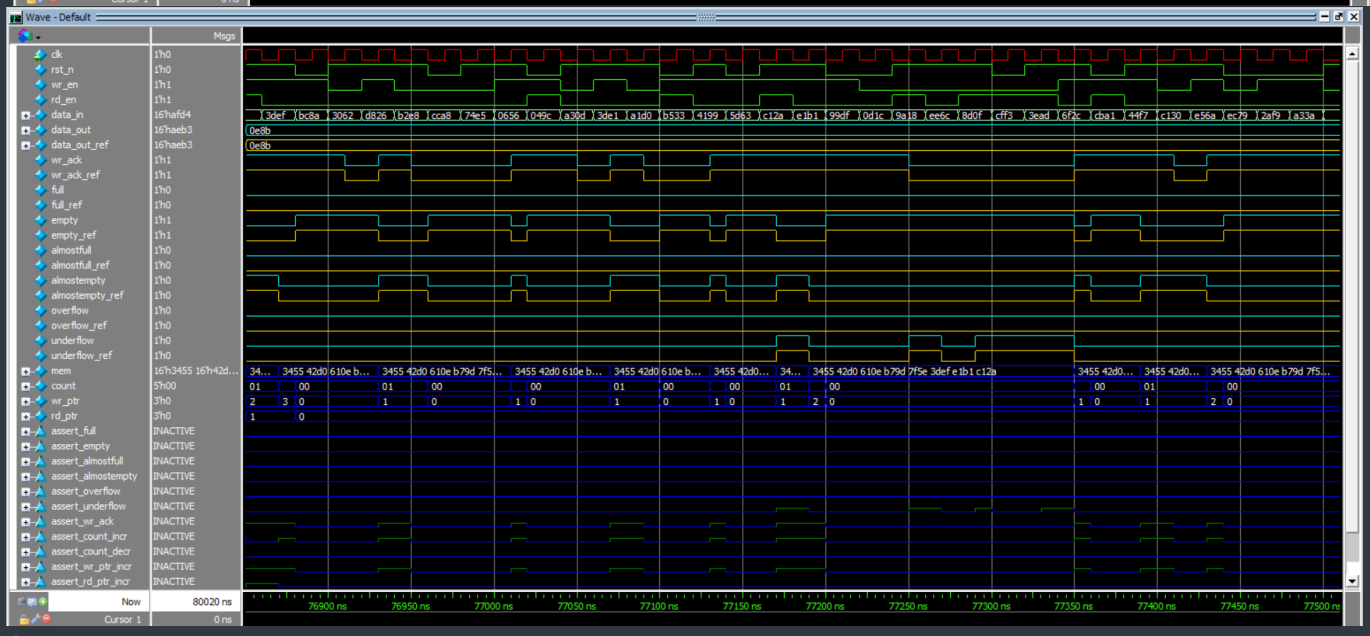
Snippets

Transcript

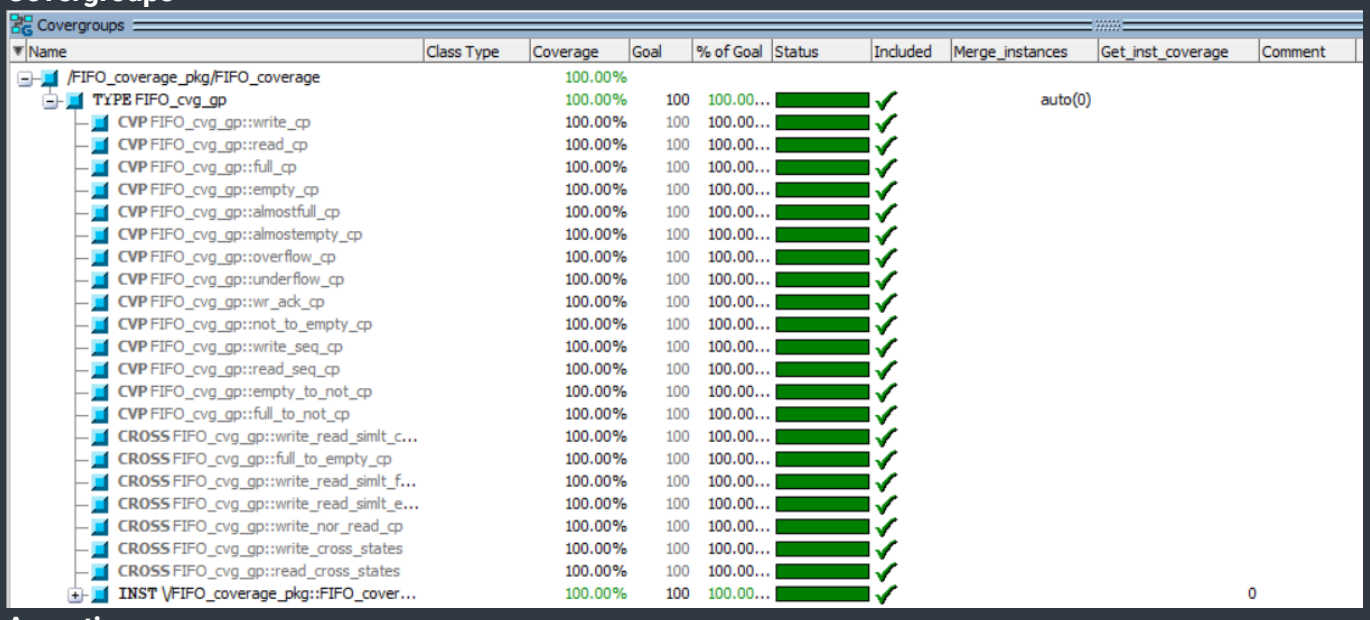
```
# FIFO's test has been successfuly finished
# error_count = 0
# correct_count = 32008
# ** Note: $stop    : FIFO_monitor.sv(51)
#    Time: 80020 ns  Iteration: 1  Instance: /FIFO_top/MONITOR
# Break in Module FIFO_monitor at FIFO_monitor.sv line 51

VSIM 3>
```

Waveform

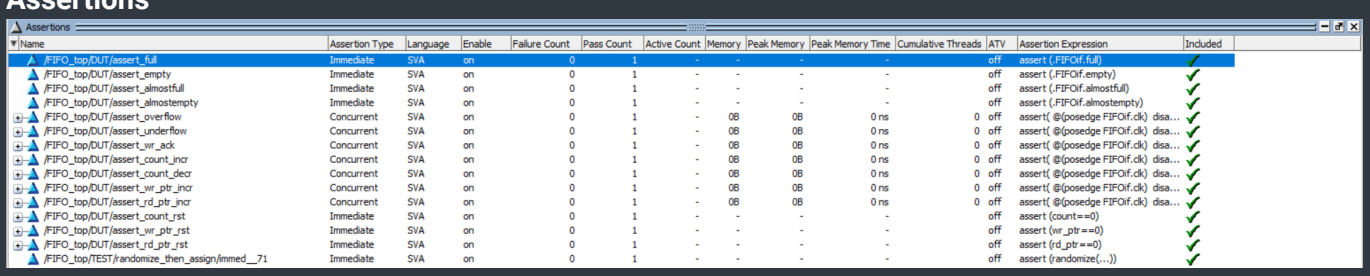




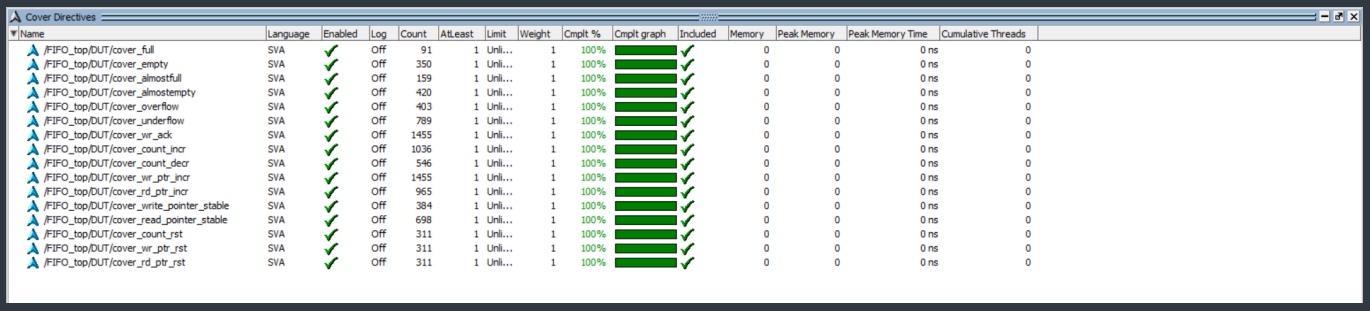
Covergroups



Assertions



Cover Directives



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/cover_full	SVA	✓	Off	91	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_empty	SVA	✓	Off	350	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_almostfull	SVA	✓	Off	159	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_almostempty	SVA	✓	Off	420	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_overflow	SVA	✓	Off	403	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_underflow	SVA	✓	Off	789	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ack	SVA	✓	Off	1455	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_count_incr	SVA	✓	Off	1036	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_count_decr	SVA	✓	Off	546	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ptr_incr	SVA	✓	Off	1455	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_rd_ptr_incr	SVA	✓	Off	965	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_write_pointer_stable	SVA	✓	Off	384	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_read_pointer_stable	SVA	✓	Off	698	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_count_rst	SVA	✓	Off	311	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_wr_ptr_rst	SVA	✓	Off	311	1	Uni...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/cover_rd_ptr_rst	SVA	✓	Off	311	1	Uni...	1	100%		✓	0	0	0 ns	0

Coverage Reports

Code Coverage Report

Note: Conditional Coverage didn't reach 100% as the the two conditions (!FIFOif.full && FIFOif.wr_en) and (!FIFOif.empty && FIFOif.rd_en) would never happen if else was entered; because in case one of them happened else statement wouldn't be entered. As a result, Total Coverage By Instance dropped to 96.51%.

```
Condition Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----    -------    ------  --------
    Conditions                      24         22         2    91.66%

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

Condition Coverage for instance /\FIFO_top#DUT  --

  File FIFO.sv
----------------Focused Condition View-------------------
Line       21 Item    1  (FIFOif.wr_en && (count < FIFOif.FIFO_DEPTH))
Condition totals: 2 of 2 input terms covered = 100.00%

                    Input Term   Covered  Reason for no coverage   Hint
                   -----------  --------  ----------------------  --------------
                  FIFOif.wr_en         Y
  (count < FIFOif.FIFO_DEPTH)          Y

     Rows:       Hits  FEC Target                    Non-masking condition(s)
 ---------  ---------  --------------------          -------------------------
  Row   1:          1  FIFOif.wr_en_0                -
  Row   2:          1  FIFOif.wr_en_1                (count < FIFOif.FIFO_DEPTH)
  Row   3:          1  (count < FIFOif.FIFO_DEPTH)_0  FIFOif.wr_en
  Row   4:          1  (count < FIFOif.FIFO_DEPTH)_1  FIFOif.wr_en

----------------Focused Condition View-------------------
Line       29 Item    1  (FIFOif.full & FIFOif.wr_en)
Condition totals: 1 of 2 input terms covered = 50.00%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
   FIFOif.full         N  '_0' not hit             Hit '_0'
  FIFOif.wr_en         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:     ***0***  FIFOif.full_0         FIFOif.wr_en
  Row   2:          1  FIFOif.full_1         FIFOif.wr_en
  Row   3:          1  FIFOif.wr_en_0        FIFOif.full
  Row   4:          1  FIFOif.wr_en_1        FIFOif.full

----------------Focused Condition View-------------------
Line       40 Item    1  (FIFOif.rd_en && (count != 0))
Condition totals: 2 of 2 input terms covered = 100.00%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
  FIFOif.rd_en         Y
  (count != 0)         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  FIFOif.rd_en_0        -
  Row   2:          1  FIFOif.rd_en_1        (count != 0)
  Row   3:          1  (count != 0)_0        FIFOif.rd_en
  Row   4:          1  (count != 0)_1        FIFOif.rd_en

----------------Focused Condition View-------------------
Line       46 Item    1  (FIFOif.empty & FIFOif.rd_en)
Condition totals: 1 of 2 input terms covered = 50.00%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
  FIFOif.empty         N  '_0' not hit             Hit '_0'
  FIFOif.rd_en         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:     ***0***  FIFOif.empty_0        FIFOif.rd_en
  Row   2:          1  FIFOif.empty_1        FIFOif.rd_en
  Row   3:          1  FIFOif.rd_en_0        FIFOif.empty
  Row   4:          1  FIFOif.rd_en_1        FIFOif.empty
```

```
21  else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
22      mem[wr_ptr] <= FIFOif.data_in;
23      FIFOif.wr_ack <= 1;
24      wr_ptr <= wr_ptr + 1;
25      FIFOif.overflow <= 0;
26  end
27  else begin
28      FIFOif.wr_ack <= 0;
29      if (FIFOif.full & FIFOif.wr_en)
30          FIFOif.overflow <= 1;
31      else
32          FIFOif.overflow <= 0;
33  end
```

```
40  else if (FIFOif.rd_en && count != 0) begin
41      FIFOif.data_out <= mem[rd_ptr];
42      rd_ptr <= rd_ptr + 1;
43      FIFOif.underflow <= 0;
44  end
45  else begin
46      if (FIFOif.empty & FIFOif.rd_en)
47          FIFOif.underflow <= 1;
48      else
49          FIFOif.underflow <= 0;
50  end
```

Note: Toggle coverage didn't reach 100% as count is 4-bits while it resets to zero when reaching 8

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         22        20         2    90.90%

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

Toggle Coverage for instance /\FIFO_top#DUT  --

                                                      Node      1H->0L      0L->1H  "Coverage"
                                                      ----------------------------------------
                                                  count[4]           0           0        0.00
                                                count[3-0]           1           1      100.00
                                               rd_ptr[2-0]           1           1      100.00
                                               wr_ptr[2-0]           1           1      100.00

Total Node Count      =         11
Toggled Node Count    =         10
Untoggled Node Count  =          1

Toggle Coverage       =      90.90% (20 of 22 bins)
```

Functional Coverage Report

```
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        bin empty_to_not                          349         1         -    Covered
    Coverpoint full_to_not_cp                 100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        bin full_to_not                            91         1         -    Covered
    Cross write_read_simlt_cp                 100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt                 4001         1         -    Covered
    Cross full_to_empty_cp                    100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin full_to_empty                      17         1         -    Covered
    Cross write_read_simlt_full_cp            100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt_full            4001         1         -    Covered
    Cross write_read_simlt_empty_cp           100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt_empty           4001         1         -    Covered
    Cross write_nor_read_cp                   100.00%       100         -    Covered
        covered/total bins:                         1         1         -
        missing/total bins:                         0         1         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_read_simlt                 4001         1         -    Covered
    Cross write_cross_states                  100.00%       100         -    Covered
        covered/total bins:                         7         7         -
        missing/total bins:                         0         7         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin write_full                       4001         1         -    Covered
            bin write_empty                      4001         1         -    Covered
            bin write_almostfull                 4001         1         -    Covered
            bin write_almostempty                4001         1         -    Covered
            bin write_overflow                   4001         1         -    Covered
            bin write_underflow                  4001         1         -    Covered
            bin write_wr_ack                     4001         1         -    Covered
    Cross read_cross_states                   100.00%       100         -    Covered
        covered/total bins:                         7         7         -
        missing/total bins:                         0         7         -
        % Hit:                                100.00%       100         -
        Auto, Default and User Defined Bins:
            bin read_full                        4001         1         -    Covered
            bin read_empty                       4001         1         -    Covered
            bin read_almostfull                  4001         1         -    Covered
            bin read_almostempty                 4001         1         -    Covered
            bin read_overflow                    4001         1         -    Covered
            bin read_underflow                   4001         1         -    Covered
            bin read_wr_ack                      4001         1         -    Covered

TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

Total Coverage By Instance (filtered view): 100.00%
```

Sequential Domain Coverage Report

```
    Cross read_cross_states                          100.00%        100          -    Covered
        covered/total bins:                                7          7          -
        missing/total bins:                                0          7          -
        % Hit:                                        100.00%        100          -
        Auto, Default and User Defined Bins:
            bin read_full                                 4001          1          -    Covered
            bin read_empty                                4001          1          -    Covered
            bin read_almostfull                           4001          1          -    Covered
            bin read_almostempty                          4001          1          -    Covered
            bin read_overflow                             4001          1          -    Covered
            bin read_underflow                            4001          1          -    Covered
            bin read_wr_ack                               4001          1          -    Covered

TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                     Design Design   Lang File(Line)      Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/DUT/cover_full                 FIFO   Verilog  SVA  FIFO.sv(89)       91 Covered
/FIFO_top/DUT/cover_empty                FIFO   Verilog  SVA  FIFO.sv(94)      350 Covered
/FIFO_top/DUT/cover_almostfull           FIFO   Verilog  SVA  FIFO.sv(99)      159 Covered
/FIFO_top/DUT/cover_almostempty          FIFO   Verilog  SVA  FIFO.sv(104)     420 Covered
/FIFO_top/DUT/cover_overflow             FIFO   Verilog  SVA  FIFO.sv(111)     403 Covered
/FIFO_top/DUT/cover_underflow            FIFO   Verilog  SVA  FIFO.sv(114)     789 Covered
/FIFO_top/DUT/cover_wr_ack               FIFO   Verilog  SVA  FIFO.sv(117)    1455 Covered
/FIFO_top/DUT/cover_count_incr           FIFO   Verilog  SVA  FIFO.sv(155)    1036 Covered
/FIFO_top/DUT/cover_count_decr           FIFO   Verilog  SVA  FIFO.sv(158)     546 Covered
/FIFO_top/DUT/cover_wr_ptr_incr          FIFO   Verilog  SVA  FIFO.sv(161)    1455 Covered
/FIFO_top/DUT/cover_rd_ptr_incr          FIFO   Verilog  SVA  FIFO.sv(164)     965 Covered
/FIFO_top/DUT/cover_write_pointer_stable FIFO   Verilog  SVA  FIFO.sv(167)     384 Covered
/FIFO_top/DUT/cover_read_pointer_stable  FIFO   Verilog  SVA  FIFO.sv(170)     698 Covered
/FIFO_top/DUT/cover_count_rst            FIFO   Verilog  SVA  FIFO.sv(175)     311 Covered
/FIFO_top/DUT/cover_wr_ptr_rst           FIFO   Verilog  SVA  FIFO.sv(178)     311 Covered
/FIFO_top/DUT/cover_rd_ptr_rst           FIFO   Verilog  SVA  FIFO.sv(181)     311 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 16

Total Coverage By Instance (filtered view): 100.00%
```