VERIFYING SYNCHRONOUS FIFO

USING UVM

Marwan Khaled Mohamed

Contents

1 UVM Architecture

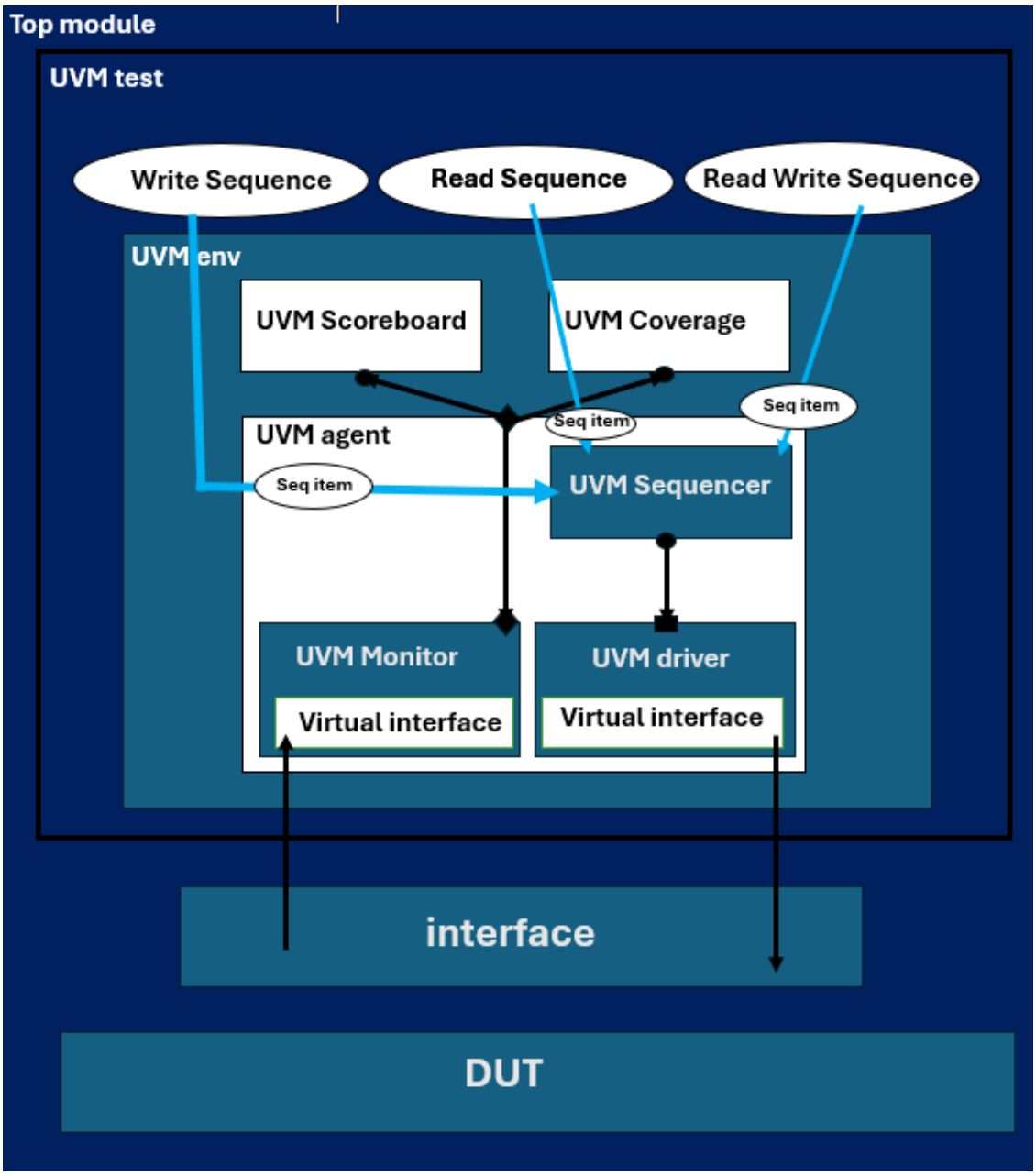

Figure 1:UVM architecture

2 Define Bugs

2.1 Bug1

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

Figure 2:Bug1

```
always @(posedge inter_type.clk or negedge inter_type.rst_n) begin
    if (!inter_type.rst_n) begin
        wr_ptr <= 0;
        /*Bug: in case of reset no defined behavior  for wr_ack , overflow */
        inter_type.wr_ack <= 0;
        inter_type.overflow <= 0;
    end
```

Figure 3:Bug1 fixed

Bug: in case of reset no defined behavior for wr_ack , overflow

2.2 Bug2

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

Figure 4:Bug2

```
/*Bug:this is the implementation for underflow since it sequential not in continuous assignment */
else
begin
    inter_type.data_out<=0;
    if (inter_type.empty && inter_type.rd_en)
        inter_type.underflow <= 1;
    else
        inter_type.underflow <= 0;

end
```

Figure 5:Bug2 fixed

Bug: in case of reset no defined behavior for underflow

2.3 Bug3

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

Figure 6:Bug3

```
/*Bug:this is the implementation for underflow since it sequential not in continuous assignment */
else
begin
    inter_type.data_out<=0;
    if (inter_type.empty && inter_type.rd_en)
        inter_type.underflow <= 1;
    else
        inter_type.underflow <= 0;

end
```

Figure 7:Bug3 Fixed

Bug: this is the implementation for underflow since it sequential not in continuous assignment

2.4 Bug4

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

Figure 8:Bug4

```
always @(posedge inter_type.clk or negedge inter_type.rst_n) begin
    if (!inter_type.rst_n) begin
        count <= 0;
    end
    else begin
        /*Bug: there is no implementation for corner case for empty and full FIFO*/
        if  ( ( ({inter_type.wr_en, inter_type.rd_en} == 2'b10) && !inter_type.full ) || (({inter_type.wr_en, inter_type.rd_en} == 2'b11) && inter_type.empty ) )
            count <= count + 1;
        else if ( ( ({inter_type.wr_en, inter_type.rd_en} == 2'b01) && !inter_type.empty ) || (({inter_type.wr_en, inter_type.rd_en} == 2'b11) && inter_type.full ) )
            count <= count - 1;
    end
end
```

Figure 9:Bug4 Fixed

Bug: there is no implementation for corner cases for empty and full FIFO

3 Code

3.1 Design code

3.1.1 Design Module with assertions

```
module FIFO(interface_FIFO.DUT inter_type);


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

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

always @(posedge inter_type.clk or negedge inter_type.rst_n) begin
    if (!inter_type.rst_n) begin
        wr_ptr <= 0;
        /*Bug: in case of reset no defined behavior  for wr_ack , overflow */
        inter_type.wr_ack <= 0;
        inter_type.overflow <= 0;
    end
    else if (inter_type.wr_en && count < inter_type.FIFO_DEPTH) begin
        mem[wr_ptr] <= inter_type.data_in;
        inter_type.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        inter_type.overflow <= 0;

    end
    else begin
        inter_type.wr_ack <= 0;
        if (inter_type.full && inter_type.wr_en)
            inter_type.overflow <= 1;
        else
            inter_type.overflow <= 0;
    end
end

always @(posedge inter_type.clk or negedge inter_type.rst_n) begin
    if (!inter_type.rst_n)
    begin
        rd_ptr <= 0;
        /*Bug: in case of reset no defined behavior for underflow  */
        inter_type.underflow <= 0;
        inter_type.data_out<=0;
    end
```

Figure 10: Design code Part1

```
    else if (inter_type.rd_en && count != 0)
    begin
        inter_type.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        inter_type.underflow <= 0;
    end
    /*Bug:this is the implementation for underflow since it sequential not in continuous assignment */
    else
    begin
        inter_type.data_out<=0;
        if (inter_type.empty && inter_type.rd_en)
            inter_type.underflow <= 1;
        else
            inter_type.underflow <= 0;

    end

end

always @(posedge inter_type.clk or negedge inter_type.rst_n) begin
    if (!inter_type.rst_n) begin
        count <= 0;
    end
    else begin
        /*Bug: there is no implementation for corner cases for empty and full FIFO*/
        if  ( ( ({inter_type.wr_en, inter_type.rd_en} == 2'b10) && !inter_type.full ) || (({inter_type.wr_en, inter_type.rd_en} == 2'
            count <= count + 1;
        else if ( ( ({inter_type.wr_en, inter_type.rd_en} == 2'b01) && !inter_type.empty ) || (({inter_type.wr_en, inter_type.rd_en}
            count <= count - 1;
    end
end


assign inter_type.full = (count == inter_type.FIFO_DEPTH)? 1 : 0;
assign inter_type.empty = (count == 0)? 1 : 0;
assign inter_type.almostfull = (count == inter_type.FIFO_DEPTH-1)? 1 : 0;  //Bug:in case of almostfull it must be -1 not -2
assign inter_type.almostempty = (count == 1)? 1 : 0;
```

Figure 11: Design Code Part2

```
/*..................................... counter,write,read pointer assertion  ..............................................*/

`ifdef SIM

/*..... wr_en=1 , rd_en=0 then count++ , rd_ptr const , wr_ptr++.........*/
property count1;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.wr_en && ~ inter_type.rd_en && count!=8 |=> count==$past(count)+1 && $stable(rd_ptr) && ( wr_ptr==$past(wr_ptr)+1 || ( $past(wr_ptr)==7 && wr_ptr==0) ) ;
endproperty

/*..... wr_en=0 , rd_en=1 then count-- , rd_ptr ++ , wr_ptr const.........*/
property count2;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 ~inter_type.wr_en && inter_type.rd_en && count!=0  |=> count==$past(count)-1 && $stable(wr_ptr)&& ( rd_ptr==$past(rd_ptr)+1 || ( $past(rd_ptr)==7 && rd_ptr==0) ) ;
endproperty

/*..... wr_en=1 , rd_en=1 , full then count-- , rd_ptr ++ , wr_ptr const.........*/
property count3;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 inter_type.wr_en && inter_type.rd_en && inter_type.full |=> count==$past(count)-1 && $stable(wr_ptr)&& ( rd_ptr==$past(rd_ptr)+1 || ( $past(rd_ptr)==7 && rd_ptr==0) )  ;
endproperty

/*..... wr_en=1 , rd_en=1 then count++ , rd_ptr const , wr_ptr++.........*/
property count4;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 inter_type.wr_en && inter_type.rd_en && inter_type.empty  |=> count==$past(count)+1 && $stable(rd_ptr) &&  ( wr_ptr==$past(wr_ptr)+1 || ( $past(wr_ptr)==7 && wr_ptr==0) ) ;
endproperty
```

Figure 13: Design Code Part3

```
/*..... wr_en=1 , rd_en=1 , NOT full, NOT empty then count const , rd_ptr ++, wr_ptr++.........*/
property count5;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 inter_type.wr_en && inter_type.rd_en && ~inter_type.empty && ~inter_type.full  |=> $stable(count) &&
 ( rd_ptr==$past(rd_ptr)+1 || ( $past(rd_ptr)==7 && rd_ptr==0) ) && ( wr_ptr==$past(wr_ptr)+1 || ( $past(wr_ptr)==7 && wr_ptr==0) )  ;
endproperty

count1_assert: assert property (count1);
count2_assert: assert property (count2);
count3_assert: assert property (count3);
count4_assert: assert property (count4);
count5_assert: assert property (count5);

count1_cover: cover property (count1);
count2_cover: cover property (count2);
count3_cover: cover property (count3);
count4_cover: cover property (count4);
count5_cover: cover property (count5);


/*..................................................................................*/


`endif


endmodule
```

Figure 12: Design Code Part4

3.1.2 Interface

```
interface the_interface (clk);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);
input clk;
logic  [FIFO_WIDTH-1:0] data_in,data_out;
logic rst_n, wr_en, rd_en,wr_ack, overflow,underflow,full, empty, almostfull, almostempty;
modport DUT (input data_in ,clk , rst_n, wr_en, rd_en,output data_out, wr_ack, overflow, underflow, full, empty, almostfull, almostempty );

endinterface
```

Figure 14: interface code

3.2 Testbench Code

3.2.1 Seq item Code

```
package seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh";
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);

    class seq_item extends  uvm_sequence_item;
        `uvm_object_utils(seq_item);
rand logic[FIFO_WIDTH-1:0] data_in;
rand logic  rst_n;
logic wr_en, rd_en;
logic[FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow, underflow, full, empty, almostfull, almostempty;

    function new (string name="seq_item");
            super.new(name);
        endfunction

    function string covert2string();
        return $sformatf("%s data_in=0b%0b , rst_n=0b%0b , wr_en=0b%0b , rd_en=0b%0b ,data_out=0b%0b , wr_ack=0b%0b , overflow=0b%0b, underflow=0b%0b,full=0b%0b,empty=0b%0b,almostfull
    endfunction

    function string covert2string_stimuluts();
        return $sformatf("data_in=0b%0b , rst_n=0b%0b , wr_en=0b%0b , rd_en=0b%0b ",data_in,rst_n,wr_en,rd_en );
    endfunction


    /*.....................................................  constraints ..........................................................................*/
    constraint reset_const {rst_n dist {1:/98 , 0:/2};};


    endclass
endpackage
```

Figure 15:seq item code

3.2.2 Config Code

```
package FIFO_config_pack;
import uvm_pkg::*;
`include "uvm_macros.svh";
    class FIFO_config extends uvm_object;
        `uvm_object_utils(FIFO_config);
        virtual the_interface FIFO_virtual;
        function new (string name="FIFO_config");
            super.new(name);
        endfunction
    endclass

endpackage
```

Figure 16:Config Code

3.2.3 Read Only Sequence

```
package read_seq_pkg;
import seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh";
    class read_only_seq extends uvm_sequence #(seq_item);
        `uvm_object_utils(read_only_seq);
        seq_item read_only_seq_item;

        function new (string name="read_only_seq");
            super.new(name);
        endfunction

        task body;
            read_only_seq_item=seq_item::type_id::create("read_only_seq_item");
            read_only_seq_item.rd_en=1;
            read_only_seq_item.wr_en=0;
            repeat(100)
            begin
            start_item(read_only_seq_item);
            assert(read_only_seq_item.randomize());
            finish_item(read_only_seq_item);
            end



        endtask
    endclass
endpackage
```

Figure 17:Read Only Seq Code

3.2.4 Write Only Seq Code

```
package write_seq_pkg;
import seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh";
    class write_only_seq extends uvm_sequence #(seq_item);
        `uvm_object_utils(write_only_seq);
        seq_item write_only_seq_item;

        function new (string name="write_only_seq");
            super.new(name);
        endfunction

        task body;
            write_only_seq_item=seq_item::type_id::create("write_only_seq_item");
            write_only_seq_item.rd_en=0;
            write_only_seq_item.wr_en=1;
            repeat(100)
            begin
            start_item(write_only_seq_item);
            assert(write_only_seq_item.randomize());
            finish_item(write_only_seq_item);

            end

        endtask
    endclass
endpackage
```

Figure 18:Write Only Seq Code

3.2.5 Write Read Seq Code

```
package write_read_seq_pkg;
import seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh";
    class write_read_seq extends uvm_sequence #(seq_item);
        `uvm_object_utils(write_read_seq);
        seq_item write_read_seq_item;

        function new (string name="write_read_seq");
            super.new(name);
        endfunction

        task body;
            write_read_seq_item=seq_item::type_id::create("write_read_seq_item");
            write_read_seq_item.rd_en=1;
            write_read_seq_item.wr_en=1;
            repeat(100)
            begin
            start_item(write_read_seq_item);
            assert(write_read_seq_item.randomize());
            finish_item(write_read_seq_item);
            end


        endtask
    endclass
endpackage
```

Figure 19:Write Read Seq Code

3.2.6 Sequencer Code

```
package sequencer_pkg;
import seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh";
    class sequencer extends  uvm_sequencer #(seq_item);
        `uvm_component_utils(sequencer);
        function new (string name="sequencer",uvm_component parent=null );
        super.new(name,parent);
        endfunction
    endclass

endpackage
```

Figure 20:Sequencer Code

3.2.7 Monitor Code

```
package monitor_pkg;
import uvm_pkg::*;
import seq_item_pkg::*;
`include "uvm_macros.svh";
    class monitor extends uvm_monitor;
        `uvm_component_utils(monitor);
        virtual the_interface mon_vif;
        seq_item mon_seq;
        uvm_analysis_port #(seq_item) mon_port;

        function new (string name="monitor",uvm_component parent=null);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase );
        super.build_phase(phase);
        mon_port=new("mon_port",this);
        mon_seq=seq_item::type_id::create("mon_seq");
        endfunction

        task run_phase(uvm_phase phase );
            super.run_phase(phase);

            forever begin
            mon_seq=seq_item::type_id::create("mon_seq");
            @(negedge (mon_vif.clk));

            mon_seq.rst_n=mon_vif.rst_n;
            mon_seq.rd_en=mon_vif.rd_en;
            mon_seq.wr_en=mon_vif.wr_en;
            mon_seq.wr_ack=mon_vif.wr_ack;
            mon_seq.data_out=mon_vif.data_out;
            mon_seq.overflow=mon_vif.overflow;
            mon_seq.underflow=mon_vif.underflow;
            mon_seq.almostempty=mon_vif.almostempty;
            mon_seq.almostfull=mon_vif.almostfull;
            mon_seq.data_in=mon_vif.data_in;
            mon_seq.empty=mon_vif.empty;
            mon_seq.full=mon_vif.full;


            mon_port.write(mon_seq);

            end

        endtask

    endclass
endpackage
```

Figure 21:Monitor Code

3.2.8 Driver Code

```
package FIFO_driver_pack;
import uvm_pkg::*;
`include "uvm_macros.svh";
import seq_item_pkg::*;

    class FIFO_driver extends uvm_driver #(seq_item);
        `uvm_component_utils(FIFO_driver);
        seq_item my_seq_item;

        virtual the_interface driver_vif;

        function new (string name="FIFO_driver",uvm_component parent=null );
            super.new(name,parent);
        endfunction



        task run_phase(uvm_phase phase );
            super.run_phase(phase);
            my_seq_item=seq_item::type_id::create("my_seq_item");
            forever begin
            seq_item_port.get_next_item(my_seq_item);
            driver_vif.rd_en=my_seq_item.rd_en;
            driver_vif.wr_en=my_seq_item.wr_en;
            driver_vif.rst_n=my_seq_item.rst_n;
            driver_vif.data_in=my_seq_item.data_in;

            @(negedge (driver_vif.clk));
            seq_item_port.item_done();
            end
        endtask

    endclass

endpackage
```

Figure 22:Driver Code

3.2.9 Agent Code

```
package agent_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh";
import FIFO_driver_pack::*;
import monitor_pkg::*;
import sequencer_pkg::*;
import seq_item_pkg::*;
import FIFO_config_pack::*;


    class agent extends uvm_agent;
        `uvm_component_utils(agent);
        monitor monitor_comp;
        FIFO_driver driver_comp;
        sequencer sequencer_comp;
        FIFO_config agent_config;
        uvm_analysis_port #(seq_item) agent_port;

    function new (string name="agent",uvm_component parent=null );
            super.new(name,parent);
        endfunction

    function void build_phase(uvm_phase phase );
        super.build_phase(phase);
        driver_comp=FIFO_driver::type_id::create("driver_comp",this);
        sequencer_comp=sequencer::type_id::create("sequencer_comp",this);
        monitor_comp=monitor::type_id::create("monitor_comp",this);
        agent_port=new("agent_port",this);
        agent_config=FIFO_config::type_id::create("agent_config",this);
        if(!(uvm_config_db#(FIFO_config)::get(this, "", "CFG", agent_config)))
                `uvm_fatal("build_phase","agent failed to get config object");

        endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        driver_comp.driver_vif=agent_config.FIFO_virtual;
        monitor_comp.mon_vif=agent_config.FIFO_virtual;
        monitor_comp.mon_port.connect(agent_port);
        driver_comp.seq_item_port.connect(sequencer_comp.seq_item_export);
    endfunction
    endclass
endpackage
```

Figure 23:agent code

3.2.10 Scoreboard Code

```
package FIFO_scoreboard_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh";
import seq_item_pkg::*;
    class FIFO_scoreboard extends  uvm_scoreboard;
        `uvm_component_utils(FIFO_scoreboard);
        uvm_analysis_export #(seq_item) scoreboard_export;
        uvm_tlm_analysis_fifo #(seq_item) scoreboard_fifo;
        seq_item score_seq_item;


        parameter FIFO_WIDTH = 16;
        reg[FIFO_WIDTH-1:0] fifo [$];
        int q_size_before;
        logic [FIFO_WIDTH-1:0] data_out_ref;
        int correct_count=0;
        int error_count=0;
        logic overflow,underflow,wr_ack,full,empty,almostfull,almostempty;

    function new (string name="FIFO_scoreboard",uvm_component parent=null );
        super.new(name,parent);
    endfunction

    function void build_phase(uvm_phase phase );
        super.build_phase(phase);
        scoreboard_export=new("scoreboard_export",this);
        scoreboard_fifo=new("scoreboard_fifo",this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        scoreboard_export.connect(scoreboard_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase );
        super.run_phase(phase);
        score_seq_item=seq_item::type_id::create("score_seq_item");
        forever begin
            scoreboard_fifo.get(score_seq_item);
            golden_model(score_seq_item);
            if(score_seq_item.data_out!=data_out_ref && score_seq_item.underflow!=underflow && score_seq_item.overflow!=overflow&& score_seq_item.wr_ack!=wr_ack
             && score_seq_item.full!=full && score_seq_item.almostfull!=almostfull && score_seq_item.empty!=empty && score_seq_item.almostempty!=almostempty)
                begin
                    `uvm_error("run_phase",$sformatf("error at trans %s while the ref 0b%0b",score_seq_item.convert2string(),data_out_ref));
                    error_count++;

                end

            else

                    correct_count++;
            end
        endtask
```

Figure 24: Scoreboard code part1

```
        endtask

function void golden_model(seq_item seq_item_param);

        q_size_before=fifo.size();
        if(seq_item_param.rst_n==0)
          begin
            data_out_ref=0;
            for(int i=0 ; i<q_size_before;i++)
            fifo.pop_front();
            {full,almostfull,empty,almostempty,overflow,underflow,wr_ack}=7'b0000000;
            end
            else
            begin
            case({seq_item_param.wr_en,seq_item_param.rd_en})
            2'b00:
            begin
                data_out_ref=0;

            end
            2'b10:
            begin
                if(fifo.size()!=8)
                    begin

                    fifo.push_back(seq_item_param.data_in);
                    data_out_ref=0;
                    if(fifo.size()==7)
                        begin
                        full=1;
                        almostfull=1;
                        almostempty=0;
                        end

                    else if(fifo.size()==6)
                        begin
                        full=0;
                        almostfull=1;
                        almostempty=0;
                        end


                    else if(fifo.size()==0)
                        begin
                        full=0;
                        almostfull=0;
                        almostempty=1;
                        end

                    {empty,overflow,underflow,underflow,wr_ack}=4'b0001;

                    end
```

Figure 25: Scoreboard code part2

```
            else if(fifo.size()==8)
            begin
                {full,almostfull,empty,almostempty,overflow,underflow,wr_ack}=7'b1000100;

            end
    end

    2'b01:
    begin
        if(fifo.size()==1)
            data_out_ref=fifo.pop_front();
        begin
            if(fifo.size()==1)
            begin
            empty=1;
            almostempty=0;
            underflow=0;
            end

        else if(fifo.size()==8)
            begin
            almostfull=1;
            empty=0;
            almostempty=0;
            underflow=0;
            end
        else if (fifo.size()==0)
            begin
            almostfull=0;
            empty=0;
            almostempty=0;
            underflow=1;
            end
        else

            begin
            almostfull=0;
            empty=0;
            almostempty=0;
            underflow=0;
            end

        {full,overflow,underflow,wr_ack}=4'b000;

            data_out_ref=0;
        end
    end
```

Figure 26: Scoreboard code part3

```
            end

            2'b11:
            begin
            if(fifo.size()==0)
                begin
                    fifo.push_back(seq_item_param.data_in);
                    data_out_ref=0;
                    {full,almostfull,empty,almostempty,overflow,underflow,wr_ack}=7'b0010001;

                end


            else if (fifo.size()==8)
                begin
                    data_out_ref=fifo.pop_front();
                    {full,almostfull,empty,almostempty,overflow,underflow,wr_ack}=7'b0100000;
                end



            else
                begin
                    fifo.push_back(seq_item_param.data_in);
                    data_out_ref=fifo.pop_front();
                    if(fifo.size()==7)
                    {full,almostfull,empty,almostempty,overflow,underflow,wr_ack}=7'b0100001;
                    else if (fifo.size()==1)
                    {full,almostfull,empty,almostempty,overflow,underflow,wr_ack}=7'b0001001;

                end
                    {full,almostfull,empty,almostempty,overflow,underflow,wr_ack}=7'b0000001;

            end

            endcase

            end

endfunction


function void report_phase(uvm_phase phase );
        super.report_phase(phase);
        `uvm_info("report_phase",($sformatf("correct_count=%0d",correct_count)), UVM_MEDIUM );
        `uvm_info("report_phase",($sformatf("error_count=%0d",error_count)),UVM_MEDIUM );
    endfunction


    endclass
endpackage
```

Figure 27: Scoreboard code part4

3.2.11 Coverage code

```
package coverage_pkg;
import uvm_pkg::*;
import seq_item_pkg::*;

`include "uvm_macros.svh";
    class coverage extends  uvm_component;
        `uvm_component_utils(coverage);
        uvm_analysis_export #(seq_item) coverage_export;
        uvm_tlm_analysis_fifo #(seq_item) coverage_fifo;
        seq_item cov_seq_item;

    covergroup cg;

    write:coverpoint cov_seq_item.wr_en
    {
        bins write_0={0};
        bins write_1={1};
    }

    read:coverpoint cov_seq_item.rd_en
    {
        bins read_0={0};
        bins read_1={1};
    }


    full:coverpoint cov_seq_item.full
    {
        bins full_0={0};
        bins full_1={1};
    }


    empty:coverpoint cov_seq_item.empty
    {
        bins empty_0={0};
        bins empty_1={1};
    }


    almostfull:coverpoint cov_seq_item.almostfull
    {
        bins almostfull_0={0};
        bins almostfull_1={1};
    }


    almostempty:coverpoint cov_seq_item.almostempty
    {
        bins almostempty_0={0};
        bins almostempty_1={1};
    }
```

Figure 28: coverage code part1

```
overflow:coverpoint cov_seq_item.overflow
{
    bins overflow_0={0};
    bins overflow_1={1};
}

underflow:coverpoint cov_seq_item.underflow
{
    bins underflow_0={0};
    bins underflow_1={1};
}

wr_ack:coverpoint cov_seq_item.wr_ack
{
    bins wr_ack_0={0};
    bins wr_ack_1={1};
}

cross_full:cross write,read,full
{

    illegal_bins f_000=binsof(write.write_0)&&binsof(read.read_0)&&binsof(full.full_0);
    illegal_bins f_001=binsof(write.write_0)&&binsof(read.read_0)&&binsof(full.full_1);

    illegal_bins f_111=binsof(write.write_1)&&binsof(read.read_1)&&binsof(full.full_1);
    illegal_bins f_011=binsof(write.write_0)&&binsof(read.read_1)&&binsof(full.full_1);
}
cross_empty:cross write,read,empty
{

    illegal_bins e_000=binsof(write.write_0)&&binsof(read.read_0)&&binsof(empty.empty_0);
    illegal_bins e_001=binsof(write.write_0)&&binsof(read.read_0)&&binsof(empty.empty_1);


}

cross_almostfull:cross write,read,almostfull
{

    illegal_bins alfull_000=binsof(write.write_0)&&binsof(read.read_0)&&binsof(almostfull.almostfull_0);
    illegal_bins alfull_001=binsof(write.write_0)&&binsof(read.read_0)&&binsof(almostfull.almostfull_1);

}

cross_almostempty:cross write,read,almostempty
{
    illegal_bins alempty_000=binsof(write.write_0)&&binsof(read.read_0)&&binsof(almostempty.almostempty_0);
    illegal_bins alempty_001=binsof(write.write_0)&&binsof(read.read_0)&&binsof(almostempty.almostempty_1);
```

Figure 29: coverage code part2

```
    cross_overflow:cross write,read,overflow
    {
        illegal_bins ov_000=binsof(write.write_0)&&binsof(read.read_0)&&binsof(overflow.overflow_0);

        illegal_bins ov_001=binsof(write.write_0)&&binsof(read.read_0)&&binsof(overflow.overflow_1);
        illegal_bins ov_011=binsof(write.write_0)&&binsof(read.read_1)&&binsof(overflow.overflow_1);
    }

    cross_underflow:cross write,read,underflow
    {

        illegal_bins uv_000=binsof(write.write_0)&&binsof(read.read_0)&&binsof(underflow.underflow_0);


        illegal_bins uv_101=binsof(write.write_1)&&binsof(read.read_0)&&binsof(underflow.underflow_1);
        illegal_bins uv_001=binsof(write.write_0)&&binsof(read.read_0)&&binsof(underflow.underflow_1);
    }

    cross_wr_ack:cross write,read,wr_ack
    {
        illegal_bins wack_000=binsof(write.write_0)&&binsof(read.read_0)&&binsof(wr_ack.wr_ack_0);
        illegal_bins wack_001=binsof(write.write_0)&&binsof(read.read_0)&&binsof(wr_ack.wr_ack_1);

        illegal_bins wack_011=binsof(write.write_0)&&binsof(read.read_1)&&binsof(wr_ack.wr_ack_1);
    }

    endgroup

    function new (string name="coverage",uvm_component parent=null );
        super.new(name,parent);
        cg=new();
    endfunction

    function void build_phase(uvm_phase phase );
        super.build_phase(phase);
        coverage_export=new("coverage_export",this);
        coverage_fifo=new("coverage_fifo",this);
    endfunction

    function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            coverage_export.connect(coverage_fifo.analysis_export);
        endfunction

    task run_phase(uvm_phase phase );
            super.run_phase(phase);
            cov_seq_item=seq_item::type_id::create("cov_seq_item");
            forever begin
            coverage_fifo.get(cov_seq_item);
            cg.sample();
            end
        endtask
    endclass
endpackage
```

Figure 30:coverage code part3

3.2.12 environment code

```
package env_pack;
import uvm_pkg::*;
import coverage_pkg::*;
import agent_pkg::*;
import FIFO_scoreboard_pkg::*;
`include "uvm_macros.svh"


  class env extends uvm_env;
  `uvm_component_utils(env);
  coverage coverage_env;
  agent agent_env;
  FIFO_scoreboard score_env;

function new (string name="shift_reg_env", uvm_component parent=null);
  super.new(name,parent);

endfunction

function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  agent_env=agent::type_id::create("agent_env",this);
  coverage_env=coverage::type_id::create("coverage_env",this);
  score_env=FIFO_scoreboard::type_id::create("score_env",this);

endfunction



function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
    agent_env.agent_port.connect(coverage_env.coverage_export);
    agent_env.agent_port.connect(score_env.scoreboard_export);
    endfunction
  endclass


endpackage
```

Figure 31: environment code

3.2.13 test code

```
package test_pack;
import uvm_pkg::*;
import env_pack::*;
import FIFO_config_pack::*;
import write_read_seq_pkg::*;
import write_seq_pkg::*;
import read_seq_pkg::*;

`include "uvm_macros.svh";
    class test extends uvm_test;
        `uvm_component_utils(test);
        env env_obj;
        FIFO_config FIFO_object;
        write_read_seq wr_seq;
        write_only_seq w_seq;
        read_only_seq r_seq ;
        int i=0;

        function new (string name="test",uvm_component parent=null);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env_obj=env::type_id::create("env_obj",this);
            FIFO_object=FIFO_config::type_id::create("FIFO_object");
            wr_seq=write_read_seq::type_id::create("wr_seq");
            w_seq=write_only_seq::type_id::create("w_seq");
            r_seq=read_only_seq::type_id::create("r_seq");
            if(!(uvm_config_db#(virtual the_interface)::get(this, "", "interface", FIFO_object.FIFO_virtual)))
            `uvm_fatal("build_phase","test failed to get config object")
            uvm_config_db#(FIFO_config)::set(this, "*", "CFG",FIFO_object );
         endfunction

         task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);

            repeat(100)
            begin
                i++;
            `uvm_info("run_phase",$sformatf("loop number %0d",i),UVM_MEDIUM);
            w_seq.start(env_obj.agent_env.sequencer_comp);
            wr_seq.start(env_obj.agent_env.sequencer_comp);
            r_seq.start(env_obj.agent_env.sequencer_comp);
            w_seq.start(env_obj.agent_env.sequencer_comp);
            r_seq.start(env_obj.agent_env.sequencer_comp);

            end
            phase.drop_objection(this);
         endtask
    endclass
endpackage
```

Figure 32: test code

3.3 Top module code

```
import uvm_pkg::*;
import test_pack::*;

`include "uvm_macros.svh";
module top ();
bit clk;

initial
begin
clk=0;
forever
#1 clk=~clk;
end

the_interface inter_type(clk);
FIFO DUT(inter_type);
bind DUT FIFO_assertion SVA(inter_type);

initial
begin
uvm_config_db#(virtual the_interface)::set(null, "uvm_test_top", "interface", inter_type);
run_test("test");
end
endmodule
```

Figure 33: Top Module code

3.4 Assertion code

```
module FIFO_assertion (the_interface.DUT inter_type);

always_comb
begin
    if(inter_type.rst_n==0)
        assert final (inter_type.empty);
end

property wr_ack;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.full |=> !inter_type.wr_ack ;
endproperty


property full_prop;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.almostfull && inter_type.wr_en && ~ inter_type.rd_en |=> inter_type.full ;
endproperty


property almostfull_prop;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.full && ~inter_type.wr_en &&  inter_type.rd_en |=> inter_type.almostfull ;
endproperty

property empty_prop;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.almostempty && inter_type.rd_en && ~ inter_type.wr_en |=> inter_type.empty ;
endproperty

property almostempty_prop;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.empty && ~inter_type.rd_en &&  inter_type.wr_en |=> inter_type.almostempty ;
endproperty


property overflow_prop;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.full  &&  inter_type.wr_en |=> inter_type.overflow ;
endproperty

property underflow_prop;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.empty  &&  inter_type.rd_en |=> inter_type.underflow ;
endproperty
```

Figure 34: Assertion Code part1

```
wr_ack_cover: cover property (wr_ack);
full_cover: cover property (full_prop);
empty_cover: cover property (empty_prop);
almostfull_cover: cover property (almostfull_prop);
almostempty_cover: cover property (almostempty_prop);
overflow_cover: cover property (overflow_prop);
underflow_cover: cover property (underflow_prop);



property overflow2;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.wr_en  && DUT.count==8 |=> inter_type.overflow ;
endproperty


property underflow2;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
inter_type.rd_en && DUT.count==0 |=> inter_type.underflow ;
endproperty


property wr_ack2;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 inter_type.wr_en  && DUT.count!=8  |=> inter_type.wr_ack ;
endproperty

property wr_ack3;
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 inter_type.wr_en  && DUT.count==8  |=> ~inter_type.wr_ack ;
endproperty


overflow2_assert: assert property (overflow2);
underflow2_assert: assert property (underflow2);
wr_ack2_assert: assert property (wr_ack2);
wr_ack3_assert: assert property (wr_ack3);



overflow1_cover: cover property (overflow2);
underflow1_cover: cover property (underflow2);
wr_ack1_cover: cover property (wr_ack2);
wr_ack2_cover: cover property (wr_ack3);
```

Figure 35:Assertion Code part2

3.5 Do file

```
vlib work
vlog -f src_files.list +define+SIM  +cover -covercells
vsim -voptargs=+acc work.top -cover
add wave /top/inter_type/*
coverage save FIFO.ucdb -onexit
run -all
```

Figure 36: Do file

```
interface.sv
config.sv
FIFO.sv
seq_item.sv
read_only_seq.sv
write_only_seq.sv
write_read_seq.sv
sequencer.sv
monitor.sv
driver.sv
agent.sv
coverage.sv
scoreboard.sv
assertion.sv
env.sv
test.sv
top.sv
```

Figure 37:List File

4 Assertions table

Description	Type	place
Whenever the FIFO in not Full and wr_en is high while rd_en is low then both of counter and write pointer will increase by one and read point doesn't change	Internal signal	design

```
inter_type.wr_en && ~ inter_type.rd_en && count!=8 |=>
count==$past(count)+1 && $stable(rd_ptr) && ( wr_ptr==$past(wr_ptr)+1 || ( $past(wr_ptr)==7 && wr_ptr==0) ) ;
```

Whenever the FIFO in not empty and wr_en is low while rd_en is high then both of counter and read pointer will increase by one and write point doesn't change	Internal signal	design

```
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 ~inter_type.wr_en && inter_type.rd_en && count!=0  |=> count==$past(count)-1 && $stable(wr_ptr)&& ( rd_ptr==$past(rd_ptr)+1 || ( $past(rd_ptr)==7 && rd_ptr==0) )
```

Whenever the FIFO in Full and both wr_en and rd_en is high then only read operation will happen so both counter and rd_ptr will move forward while wr_ptr stay the same	Internal signal	design

```
@(posedge inter_type.clk) disable iff(inter_type.rst_n==0)
 inter_type.wr_en && inter_type.rd_en && inter_type.full |=> count==$past(count)-1 && $stable(wr_ptr)&& ( rd_ptr==$past(rd_ptr)+1 || ( $past(rd_ptr)==7 && rd_ptr==0) )
```

Whenever the FIFO in empty and both wr_en and rd_en is high then only write operation will happen so both counter and wr_ptr will move forward while rd_ptr stay the same	Internal signal	design

```
inter_type.wr_en && inter_type.rd_en && inter_type.empty  |=> count==$past(count)+1 && $stable(rd_ptr) &&  ( wr_ptr==$past(wr_ptr)+1 || ( $past(wr_ptr)==7 && wr_ptr==0) )
```

Whenever both wr_en and rd_en is low then counter and pointer stay the same	Internal signal	design

```
inter_type.wr_en && inter_type.rd_en && ~inter_type.empty && ~inter_type.full  |=> $stable(count) &&
( rd_ptr==$past(rd_ptr)+1 || ( $past(rd_ptr)==7 && rd_ptr==0) ) && ( wr_ptr==$past(wr_ptr)+1 || ( $past(wr_ptr)==7 && wr_ptr==0) )  ;
```

Whenever the FIFO in not empty and not full and both wr_en and rd_en is high then both read and write operation will happen so counter stay constant while wr_ptr and rd_ptr moving forward	Internal signal	design

```
inter_type.wr_en && inter_type.rd_en && ~inter_type.empty && ~inter_type.full  |=> $stable(count) &&
( rd_ptr==$past(rd_ptr)+1 || ( $past(rd_ptr)==7 && rd_ptr==0) ) && ( wr_ptr==$past(wr_ptr)+1 || ( $past(wr_ptr)==7 && wr_ptr==0) )  ;
```

Description	Type	place
Whenever the reset is asserted then empty flag must be high	output signal	Assertion module

```
always_comb
begin
    if(inter_type.rst_n==0)
        assert final (inter_type.empty);
end
```

Description	Type	place
Whenever the FIFO Full then wr_ack flag must be low	output signal	Assertion module

```
inter_type.full |=> !inter_type.wr_ack ;
```

Description	Type	place
Whenever the FIFO is almost full and wr_en is high while rd_en is low then in next cycle full flag must be 1	output signal	Assertion module

```
inter_type.almostfull && inter_type.wr_en && ~ inter_type.rd_en |=> inter_type.full ;
```

Description	Type	place
Whenever the FIFO is full and wr_en is low while rd_en is high then in next cycle almost full flag must be 1	output signal	Assertion module

```
inter_type.full && ~inter_type.wr_en &&  inter_type.rd_en |=> inter_type.almostfull ;
```

Description	Type	place
Whenever the FIFO is almost empty and wr_en is low while rd_en is high then in next cycle empty flag must be 1	output signal	Assertion module

```
inter_type.almostempty && inter_type.rd_en && ~ inter_type.wr_en |=> inter_type.empty ;
```

Description	Type	place
Whenever the FIFO is empty and wr_en is high while rd_en is low then in next cycle almost empty flag must be 1	output signal	Assertion module

```
inter_type.empty && ~inter_type.rd_en &&  inter_type.wr_en |=> inter_type.almostempty ;
```

Description	Type	place
Whenever the FIFO is full and wr_en is high then in next cycle overflow will happen	output signal	Assertion module

```
inter_type.full  &&  inter_type.wr_en |=> inter_type.overflow ;
```

Whenever the FIFO is empty and rd_en is high then in next cycle underflow will happen	output signal	Assertion module
`inter_type.empty && inter_type.rd_en |=> inter_type.underflow ;`		

5 Waveform

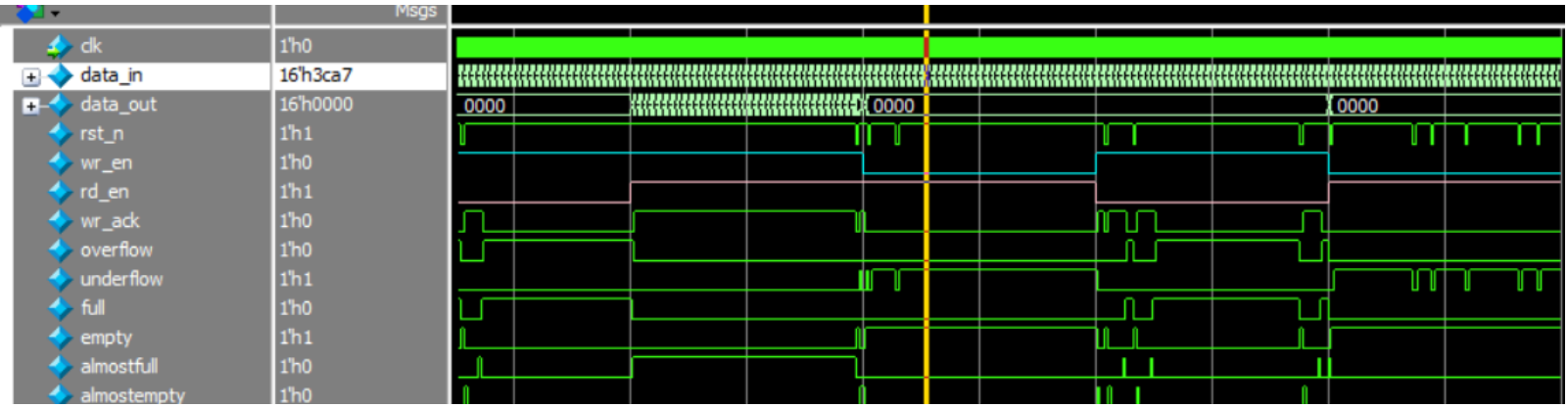

Figure 38: Waveform zoomed version

6 Coverage

6.1 Code coverage

6.1.1 Statement coverage

Figure 39:statement coverage

6.1.2 Branch coverage

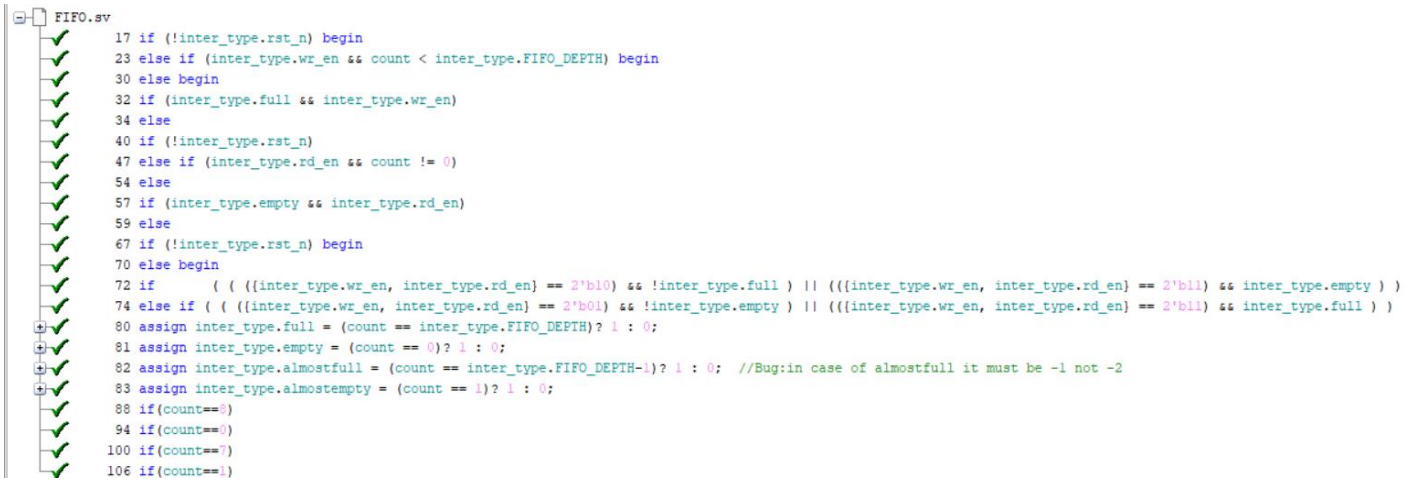

Figure 40:Branch coverage

6.1.3 Toggle coverage

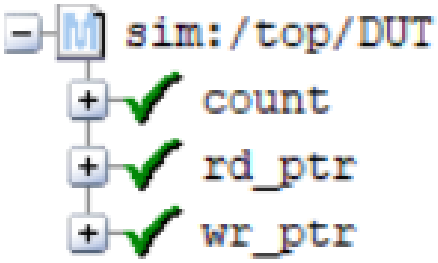

Figure 41: Toggle coverage

6.2 Functional Coverage

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/coverage_pkg/coverage		100.00%				
TYPE cg		100.00%	100	100.00...		✓
CVP cg::write		100.00%	100	100.00...		✓
CVP cg::read		100.00%	100	100.00...		✓
CVP cg::full		100.00%	100	100.00...		✓
CVP cg::empty		100.00%	100	100.00...		✓
CVP cg::almostfull		100.00%	100	100.00...		✓
CVP cg::almostempty		100.00%	100	100.00...		✓
CVP cg::overflow		100.00%	100	100.00...		✓
CVP cg::underflow		100.00%	100	100.00...		✓
CVP cg::wr_ack		100.00%	100	100.00...		✓
CROSS cg::cross_full		100.00%	100	100.00...		✓
CROSS cg::cross_empty		100.00%	100	100.00...		✓
CROSS cg::cross_almostfull		100.00%	100	100.00...		✓
CROSS cg::cross_almostempty		100.00%	100	100.00...		✓
CROSS cg::cross_overflow		100.00%	100	100.00...		✓
CROSS cg::cross_underflow		100.00%	100	100.00...		✓
CROSS cg::cross_wr_ack		100.00%	100	100.00...		✓

Figure 42:functional coverage

6.3 Assertion Coverage

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
/top/DUT/count1_cover	SVA	✓	Off	4168	1	Unli...	1	100%		✓
/top/DUT/count2_cover	SVA	✓	Off	821	1	Unli...	1	100%		✓
/top/DUT/count3_cover	SVA	✓	Off	80	1	Unli...	1	100%		✓
/top/DUT/count4_cover	SVA	✓	Off	177	1	Unli...	1	100%		✓
/top/DUT/count5_cover	SVA	✓	Off	9387	1	Unli...	1	100%		✓
/top/DUT/SVA/wr_ack_cover	SVA	✓	Off	15192	1	Unli...	1	100%		✓
/top/DUT/SVA/full_cover	SVA	✓	Off	473	1	Unli...	1	100%		✓
/top/DUT/SVA/empty_cover	SVA	✓	Off	182	1	Unli...	1	100%		✓
/top/DUT/SVA/almostfull_cover	SVA	✓	Off	84	1	Unli...	1	100%		✓
/top/DUT/SVA/almostempty_cover	SVA	✓	Off	567	1	Unli...	1	100%		✓
/top/DUT/SVA/overflow_cover	SVA	✓	Off	15108	1	Unli...	1	100%		✓
/top/DUT/SVA/underflow_cover	SVA	✓	Off	18583	1	Unli...	1	100%		✓
/top/DUT/SVA/overflow1_cover	SVA	✓	Off	15108	1	Unli...	1	100%		✓
/top/DUT/SVA/underflow1_cover	SVA	✓	Off	18583	1	Unli...	1	100%		✓
/top/DUT/SVA/wr_ack1_cover	SVA	✓	Off	13732	1	Unli...	1	100%		✓
/top/DUT/SVA/wr_ack2_cover	SVA	✓	Off	15108	1	Unli...	1	100%		✓

Figure 43: assertion coverage

6.4 Assertion Report

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/read_seq_pkg::read_only_seq::body/#ublk#240000375#18/...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/write_seq_pkg::write_only_seq::body/#ublk#123194887#18...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/write_read_seq_pkg::write_read_seq::body/#ublk#1450362...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/top/DUT/count1_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/count2_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/count3_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/count4_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/count5_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/#ublk#306607#87/immed__89	Immediate	SVA	on	0	1	-	-	-	-		off	assert (inter_type.full)	✓
/top/DUT/#ublk#306607#93/immed__95	Immediate	SVA	on	0	1	-	-	-	-		off	assert (inter_type.empty)	✓
/top/DUT/#ublk#306607#99/immed__101	Immediate	SVA	on	0	1	-	-	-	-		off	assert (inter_type.almostfull)	✓
/top/DUT/#ublk#306607#105/immed__107	Immediate	SVA	on	0	1	-	-	-	-		off	assert (inter_type.almostempty)	✓
/top/DUT/SVA/wr_ack_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/full_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/empty_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/almostfull_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/almostempty_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/overflow_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/underflow_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/overflow2_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/underflow2_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/wr_ack2_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/wr_ack3_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge inter_type.clk) ...	✓
/top/DUT/SVA/#ublk#75297886#7/immed__9	Immediate	SVA	on	0	1	-	-	-	-		off	assert (inter_type.empty)	✓

Figure 44:assertion Report