SYNCHROUNOUS FIFO

Supervisor: ENG. Kareem Waseem

MARWAN MOHAMED ELSAYED

[DATE]

[COMPANY NAME]

[Company address]

1-Introduction

A FIFO (First-In, First-Out) is a type of data buffer or queue that follows a principle where the first data to be written into the buffer is the first to be read out. Essentially, it works like a line at a checkout counter—those who arrive first are served first.

What is FIFO Used For ?

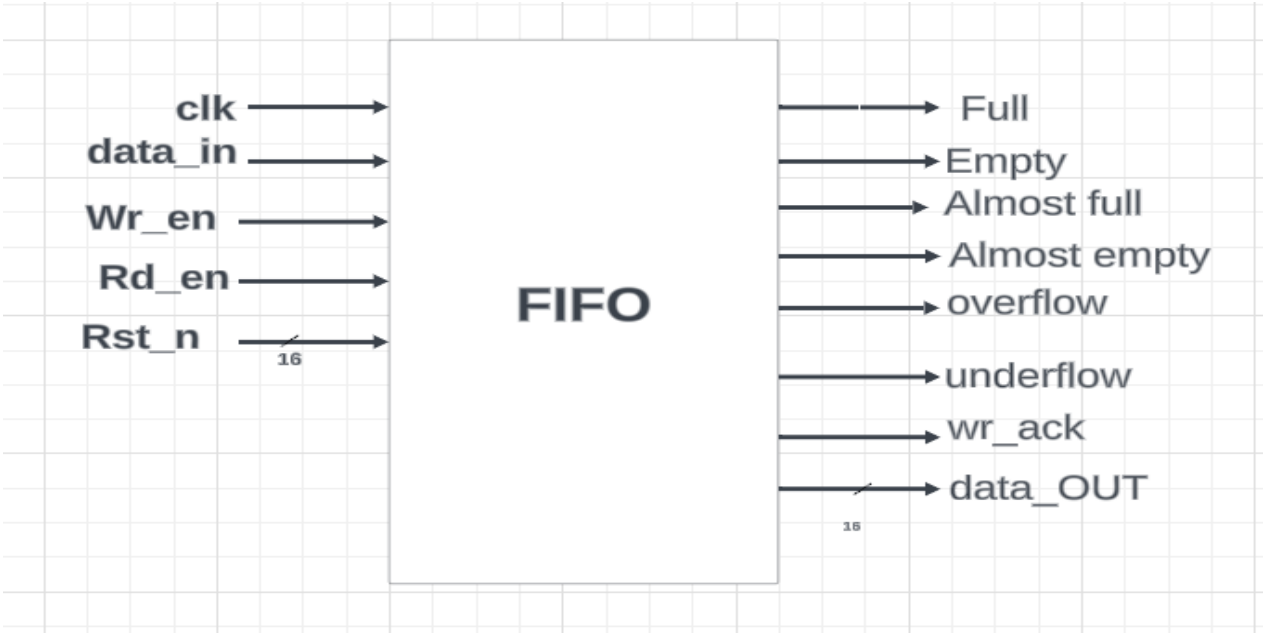
FIFOs are commonly used in digital circuits and systems where data needs to be temporarily stored before being processed, ensuring that it is read in the same order it was written. Some common uses include:

1. Data Flow Control: FIFOs help manage data flow between different clock domains or modules with different processing speeds, ensuring that data is passed smoothly without loss.
2. Communication Buffers: In communication systems, FIFOs buffer incoming data so it can be processed later without overflowing the receiver.
3. Synchronization: FIFOs are used for clock domain crossing (CDC), where data is passed between two different clock domains in a safe and orderly manner.
4. Pipeline Systems: In processors, FIFOs act as buffers in pipeline stages to manage the flow of instructions or data, ensuring smooth execution.

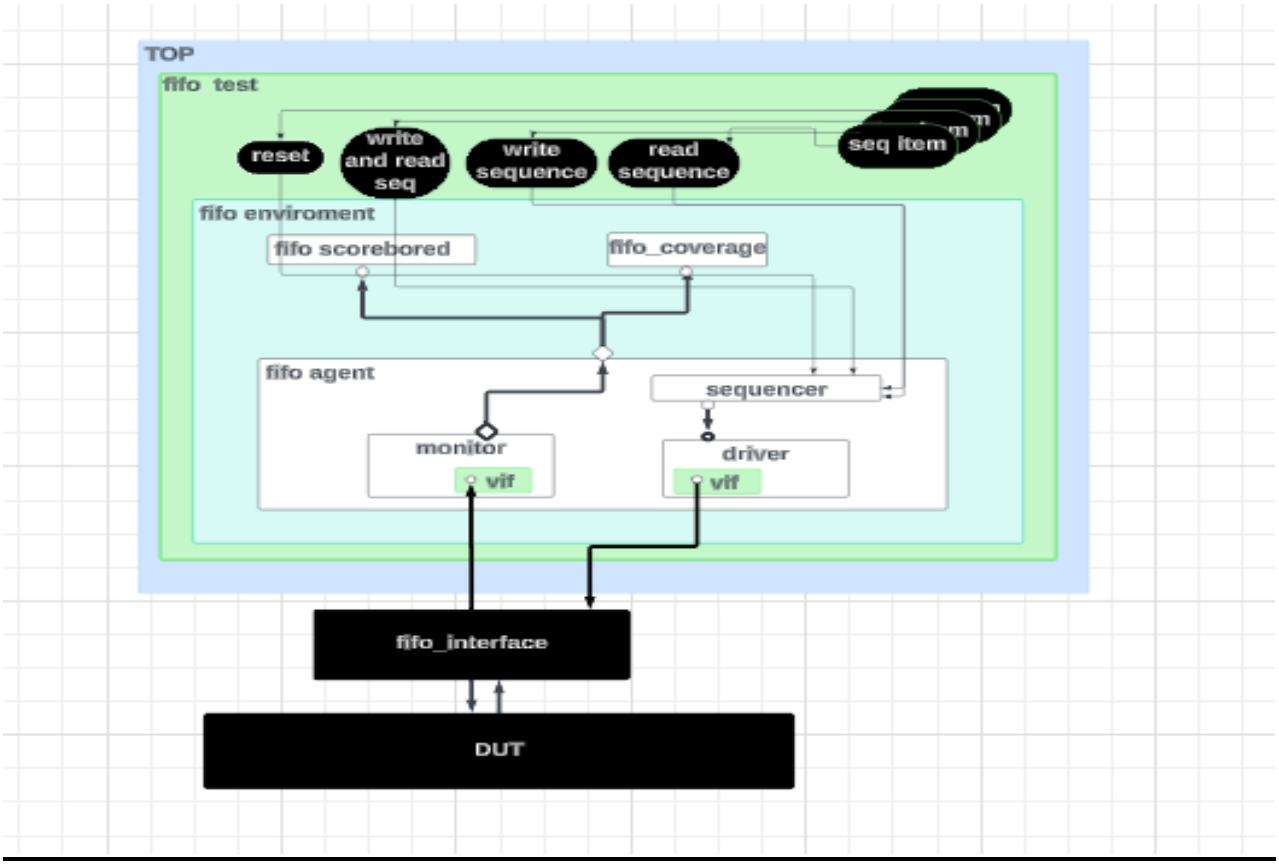
2-Verification plan

Label	Design Requirement Description	Stimulus Generation	Functionality Check	Functional coverage
write read	we need to check when we write more than size of fifo overwrite	we write 10 values on fifo by random	check by golden model	
read	we need to check when we read we read all values right	we read 10 values on fifo by random	check by golden model	
Reset_feature	we assert active low Rst_t to get ideal state	we directed at start of simulation then we randomize under constraint that is should be deasserted most of simulation	check with immediate final Assertion for asyn reset function	we cover the values output with right values when reset asserted
Full_feature	we check when Fifo is full if output flag is high or not	we randomize on simulation	we check with immediate final Assertion because its combintional it should be high when count =depth	we cover all values of full with wr_en and read _en with illegal bins check on logic
empty_feature	we check when Fifo is empty if output flag is high or not	we randomize on simulation	we check with immediate final Assertion because its combintional it should be high when size is zero and reset	we cover all values of full with wr_en and read _en with illegal bins check on logic
almostFull_feature	we check when Fifo is full if output flag is high or not	we randomize on simulation	we check with immediate final Assertion because its combintional it should be high when count = depth -1	we cover all values of full with wr_en and read _en
almostEmpty_feature	we check when Fifo is full if output flag is high or not	we randomize on simulation	we check with immediate final Assertion because its combintional it should be high when count 1	we cover all values of full with wr_en and read _en
overflow_feature	we check when Fifo is full if output flag is high or not	we randomize on simulation	check this property with concurrent assertions that check if flag is high when count =depth and write enaple is high	we cover all values of full with wr_en and read _en with illegal bins check on logic
underflow_feature	we check when Fifo is full if output flag is high or not	we randomize on simulation	check this property with concurrent assertions that check if flag is high when count =depth and write enaple is high	we cover all values of full with wr_en and read _en with illegal bins check on logic
wrAck_feature	we check when Fifo is full if output flag is high or not	we randomize on simulation	check this property with concurrent assertions that check if flag is high when count < depth and write enaple is high so it can write successfully	we cover all values of full with wr_en and read _en with illegal bins check on logic
internal_counter	we check on internal counter that it work right on write and read	we randomize on simulation	check this property with concurrent assertions that check counter increase if write is activated and decrement when read is activated and no change if write and read activated or deactivated At the same time	-
Internal_write_read ptr	we check on write and read pointer if working right	we randomize on simulation	check this property with concurrent assertions that check write ptr increase if write is activated and read ptr increment when read is activated	-

3-block diagram



4-ENVIROMENT



Description

In this UVM-based verification environment, four sequences—write, read, reset, and a combined write-read—are designed to simulate the FIFO's core operations.

1. Sequences:

Write Sequence: Simulates writing data to the FIFO by generating a transaction representing a write operation.

Read Sequence: Simulates reading data from the FIFO by generating a transaction for a read operation.

Reset Sequence: Issues a reset transaction to bring the FIFO into a known, initialized state.

Write-Read Sequence: A combination of write and read operations, allowing for a full verification of data flow in and out of the FIFO.

Each sequence creates the relevant sequence item and sends it to the sequencer for scheduling.

2. Sequencer: The sequencer acts as the coordinator between the sequences and the driver. It controls the flow of transactions,.

3. Driver: The driver receives transactions from the sequencer and converts these high-level transactioninto pin-level signals that interact directly with the interface.

4. Monitor: The monitor passively observes the interface, capturing the signals generated by the driver at every negative clock edge and send to scoreboard and coverage collector

5.**Scoreboard**: Receives the monitored data and checks the functional correctness of the FIFO by comparing the observed behavior with the expected behavior.

6.Coverage Analysis: Receives the monitored data to track functional coverage, ensuring that all test cases and scenarios are exercised during the verification process.

At the beginning of the test, the testbench initiates each sequence by sending it to the sequencer. The sequencer schedules the sequence items and forwards them to the driver. The driver then converts the transactions into pin-level signals and drives them onto the interface on every negative clock edge. The monitor captures the response from the interface, also at each negative clock edge, and forwards this data to the scoreboard and coverage analysis component. The scoreboard checks functional correctness, while the coverage component analyzes the test coverage to ensure all test conditions are met.

5-Design codes

Modified Design

```
module FIFO(fifo_if if_t);

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

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

always @(posedge if_t.clk or negedge if_t.rst_n) begin
    if (!if_t.rst_n) begin
        wr_ptr <= 0;
        if_t.overflow <= 0; // we need to assign zero to this var because if it was high in cycle so in next cycle if rst it should be low but it still high
        if_t.wr_ack   <= 0; // we need to assign zero to this var because if it was high in cycle so in next cycle if rst it should be low but it still high
    end
    else if (if_t.wr_en  && count < if_t.FIFO_DEPTH) begin
        mem[wr_ptr] <= if_t.data_in;
        if_t.wr_ack <= 1;
        if_t.overflow <= 0;// we need to assign zero to this var because if it was high in cycle so in next cycle if write it need to be zero
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        if_t.wr_ack <= 0;
        if(if_t.full && if_t.wr_en ) begin
        //  we should handle if there is read also
            if_t.overflow <= 1;
        end
        else begin
            if_t.overflow <= 0;
        end
    end
end
```

```
always @(posedge if_t.clk or negedge if_t.rst_n) begin
    if (!if_t.rst_n) begin
        rd_ptr <= 0;
        if_t.underflow <= 0; // we need to assign zero to this var because if it was high in cycle so in next cycle if rst it should be low but it still high
    end
    else if (if_t.rd_en && count != 0) begin
        if_t.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        if_t.underflow <= 0;// we need to assign zero to this var because if it was high in cycle so in next cycle if read it need to be zero
    end
    else begin
        if(if_t.empty && if_t.rd_en )
            if_t.underflow <= 1;
        else
            if_t.underflow <= 0;
    end
end
```

```
always @(posedge if_t.clk or negedge if_t.rst_n) begin
    if (!if_t.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({if_t.wr_en, if_t.rd_en} == 2'b10) && !if_t.full)
            count <= count + 1;
        else if ( ({if_t.wr_en, if_t.rd_en} == 2'b01) && !if_t.empty)
            count <= count - 1;
        else if ( ({if_t.wr_en, if_t.rd_en} == 2'b11) && if_t.full)//we need to handle counter in write and read when full
            count <= count - 1;
        else if ( ({if_t.wr_en, if_t.rd_en} == 2'b11) && if_t.empty)//we need to handle counter in write and read when empty
            count <= count + 1;
    end
end

assign if_t.full = (count == if_t.FIFO_DEPTH)? 1 : 0;
assign if_t.empty = (count == 0)? 1 : 0;
assign if_t.almostfull = (count == if_t.FIFO_DEPTH-1)? 1 : 0; // here it should be 1 not 2
assign if_t.almostempty = (count == 1)? 1 : 0;

endmodule
```

Interface

```
interface fifo_if(clk);

input bit clk;

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

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

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

endinterface
```

Top module

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_test_pkg::*;
module top();

bit clk=0;

always #100 clk=!clk;


fifo_if if_t(clk);
FIFO dut(if_t);
bind FIFO Assertions AS(if_t);

initial begin
uvm_config_db #(virtual fifo_if)::set(null,"*","virtualIf",if_t);
run_test("fifo_test");
end



endmodule
```

Test

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

import fifo_sequence_pkg::*;
import fifo_env_pkg::*;
import fifo_config_pkg::*;

    class fifo_test extends uvm_test;
        `uvm_component_utils(fifo_test);
        reset rst_seq;
        write_sequence  write_seq;
        read_sequence read_seq;
        write_read  write_read_seq;
        fifo_env env;
        fifo_config cfg;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            rst_seq=reset::type_id::create("rst_n");
            write_seq=write_sequence::type_id::create("write_sequences");
            read_seq=read_sequence::type_id::create("read_seq");
            write_read_seq=write_read::type_id::create("write_read");
            env=fifo_env::type_id::create("env",this);
            cfg=fifo_config::type_id::create("config");

            if(! uvm_config_db #(virtual fifo_if)::get(this,"","virtualIf",cfg.vif))
                `uvm_fatal("build_phase","cant get virtual interface");


            uvm_config_db #(fifo_config)::set(this,"*","CFG",cfg);
        endfunction

        task run_phase(uvm_phase phase);
            super.build_phase(phase);
            phase.raise_objection(this);
            `uvm_info("run_phase","reset sequence has started now",UVM_MEDIUM);
            rst_seq.start(env.agt.sqr);
            `uvm_info("run_phase","write sequence has started now",UVM_MEDIUM);
            write_seq.start(env.agt.sqr);
            `uvm_info("run_phase","read sequence has started now",UVM_MEDIUM);
            read_seq.start(env.agt.sqr);
            `uvm_info("run_phase","write and read sequence has started now",UVM_MEDIUM);
            write_read_seq.start(env.agt.sqr);
            `uvm_info("run_phase","reset sequence has started again",UVM_MEDIUM);
            rst_seq.start(env.agt.sqr);
            `uvm_info("run_phase","write and read sequence has started again for second time",UVM_MEDIUM);
            write_read_seq.start(env.agt.sqr);
            phase.drop_objection(this);
        endtask


    endclass

endpackage
```

environment

```
package fifo_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
    import fifo_coverage_pkg::*;
    import fifo_scoreboard_pkg::*;
    import fifo_agent_pkg::*;

    class fifo_env extends uvm_component ;
        `uvm_component_utils(fifo_env);
        fifo_agent agt;
        fifo_cov cov;
        fifo_scoreboard sb;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb=fifo_scoreboard::type_id::create("scoreboard",this);
            cov=fifo_cov::type_id::create("coverage collector",this);
            agt=fifo_agent::type_id::create("Agent",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_p.connect(sb.sb_ep);
            agt.agt_p.connect(cov.cov_ep);
        endfunction
    endclass
endpackage
```

Sequences

```
import fifo_seq_item_pkg::*;
import shared_pkg::*;
    class reset extends uvm_sequence #(seq_item);
        `uvm_object_utils(reset);
        seq_item item;
        function new(string name = "RESET");
            super.new(name);
        endfunction
        task body();
                item=seq_item::type_id::create("res") ;
                start_item(item);
                item.rst_n=0;
                finish_item(item);
        endtask
    endclass

    class write_sequence extends uvm_sequence #(seq_item);
        `uvm_object_utils(write_sequence);
        seq_item item;
        function new(string name = "MAIN");
            super.new(name);
        endfunction

        task body();
            repeat(10)begin
                item=new("writing",50,50);
                start_item(item);
                assert(item.randomize() with { rst_n==1;rd_en==0;wr_en==1; });
                finish_item(item);
            end
        endtask
    endclass
```

```
class read_sequence extends uvm_sequence #(seq_item);
    `uvm_object_utils(read_sequence);
    seq_item item;
    function new(string name = "MAIN");
        super.new(name);
    endfunction

    task body();
        repeat(10)begin
            item=new("reading");
            start_item(item);
            assert(item.randomize() with { rst_n==1;rd_en==1;wr_en==0;});
            finish_item(item);
        end
    endtask
endclass

class write_read extends uvm_sequence #(seq_item);
    `uvm_object_utils(write_read);
    seq_item item;
    function new(string name = "MAIN");
        super.new(name);
    endfunction

    task body();
        repeat(500)begin
            item=new("same",50,50);
            start_item(item);
            assert(item.randomize() with {rst_n==1;});
            finish_item(item);
        end
    endtask
```

Sequence item

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

    class seq_item extends uvm_sequence_item ;
        `uvm_object_utils(seq_item);
        parameter FIFO_WIDTH = 16;
        parameter FIFO_DEPTH = 8;

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

        localparam max_fifo_addr = $clog2(FIFO_DEPTH);

        int RD_EN_ON_DIST,WR_EN_ON_DIST;

        function new(string name = "fifo_sequence_item",int RD_EN_ON_DIST_t=30 , int WR_EN_ON_DIST_t=70);
            super.new(name);
            RD_EN_ON_DIST=RD_EN_ON_DIST_t;
            WR_EN_ON_DIST=WR_EN_ON_DIST_t;
        endfunction

        function string convert2string();
            return $sformatf("%s  reset =%0b data_in=%0b ,wr_en=%0b ,rd_en=%0b ,data_out=%0b ,wr_ack=%0b,overflow=%0b ,full=%0b ,empty=%0b
            ,almostfull=%0b,almostempty=%0b ,underflow=%0b ",
            super.convert2string(),rst_n,data_in,wr_en,rd_en,data_out,wr_ack,overflow,full,empty,almostfull,almostempty,underflow);
        endfunction
```

```
        function string convert2string_stimulus();
            return $sformatf("%s  reset =%0b data_in=%0b ,wr_en=%0b ,rd_en=%0b  ",
            super.convert2string(),rst_n,data_in,wr_en,rd_en);
        endfunction

        constraint x{
            rst_n dist { 0:= 5 , 1:= 95 };
            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};
        }
```

Agent

```
class fifo_agent extends uvm_agent;
    `uvm_component_utils(fifo_agent);
    fifo_driver drv;
    fifo_sequencer sqr;
    fifo_monitor mon;
    uvm_analysis_port #(seq_item) agt_p;
    fifo_config cfg;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        drv=fifo_driver::type_id::create("driver",this);
        mon=fifo_monitor::type_id::create("monitor",this);
        sqr=fifo_sequencer::type_id::create("sequencer",this);
        agt_p=new("agent analysis port",this);

        if(!uvm_config_db #(fifo_config)::get(this,"*","CFG",cfg))
            `uvm_fatal("build_phase","cant get configuration object");
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        drv.seq_item_port.connect(sqr.seq_item_export);
        mon.mon_p.connect(agt_p);
        mon.vif=cfg.vif;
        drv.vif=cfg.vif;
    endfunction
```

driver

```
package fifo_driver_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_seq_item_pkg::*;

  class fifo_driver extends uvm_driver #(seq_item);
      `uvm_component_utils(fifo_driver);
      virtual fifo_if vif;
      seq_item item;

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

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

          forever begin
              item=seq_item::type_id::create("item");
              seq_item_port.get_next_item(item);
              vif.data_in=item.data_in;
              vif.rst_n= item.rst_n;
              vif.wr_en=item.wr_en;
              vif.rd_en=item.rd_en;
              @(negedge vif.clk);
              seq_item_port.item_done();
              `uvm_info("run_phase",item.convert2string(),UVM_HIGH);
          end
      endtask
  endclass
endpackage
```

Monitor

```
package fifo_monitor_pkg;

import uvm_pkg::*;
    `include "uvm_macros.svh"
import fifo_seq_item_pkg::*;

class  fifo_monitor extends uvm_monitor;
    `uvm_component_utils(fifo_monitor);
    virtual fifo_if vif;
    seq_item item;
    uvm_analysis_port #(seq_item) mon_p;

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            item=seq_item::type_id::create("monitor item") ;
            @(negedge vif.clk);
            item.data_in=vif.data_in;
            item.rst_n =vif.rst_n;
            item.wr_en =vif.wr_en;
            item.rd_en =vif.rd_en;
            // OUTPUT
            item.data_out   =vif.data_out    ;
            item.wr_ack     =vif.wr_ack   ;
            item.overflow   =vif.overflow   ;
            item.full       =vif.full   ;
            item.empty      =vif.empty    ;
            item.almostfull =vif.almostfull   ;
            item.almostempty=vif.almostempty   ;
            item.underflow  =vif.underflow   ;
            mon_p.write(item);
            `uvm_info("run_phase",item.convert2string(),UVM_HIGH);
        end
    endtask
endclass


endpackage
```

Sequencer

```
package fifo_sequencer_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
    import fifo_seq_item_pkg::*;
    class fifo_sequencer extends uvm_sequencer #(seq_item);
        `uvm_component_utils(fifo_sequencer);

        function new(string name="sequencer",uvm_component phase);
            super.new(name,phase);
        endfunction
    endclass


endpackage
```

Configuration object

```
package fifo_config_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
    class fifo_config extends uvm_object ;
        `uvm_object_utils(fifo_config);
        virtual fifo_if vif;
        function new(string name="config_object");
            super.new(name);
        endfunction
    endclass
endpackage
```

Scoreboard

```
import uvm_pkg::*;
    `include "uvm_macros.svh"
import fifo_seq_item_pkg::*;
import shared_pkg::*;
class fifo_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(fifo_scoreboard);

    uvm_tlm_analysis_fifo #(seq_item) sb_fifo;
    uvm_analysis_export #(seq_item) sb_ep;
    seq_item seq_item_tb;
    logic [seq_item::FIFO_WIDTH-1:0] data_out_ref;
    bit [seq_item::FIFO_WIDTH-1 :0] fifo_modeling[$];;

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

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

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        sb_ep.connect(sb_fifo.analysis_export);
    endfunction
```

```
task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
        sb_fifo.get(seq_item_tb);
        reference_model(seq_item_tb);
        if(data_out_ref != seq_item_tb.data_out )begin
        `uvm_error("run_phase",$sformatf("failed check stimulus %s while ref out=%0b",seq_item_tb.convert2string,data_out_ref));
        error_count++;
        end
        else correct_count++;

    end
endtask

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

```
function void reference_model(input seq_item FI_tr);
    if(!FI_tr.rst_n )begin
        fifo_modeling.delete();
    end
    else begin
        if(FI_tr.wr_en && FI_tr.rd_en && fifo_modeling.size() == 0)begin
            fifo_modeling.push_back(FI_tr.data_in);
        end
        else if(FI_tr.wr_en && FI_tr.rd_en && fifo_modeling.size() == seq_item_tb.FIFO_DEPTH)begin
            data_out_ref=fifo_modeling.pop_front();
        end
        else if(FI_tr.wr_en && FI_tr.rd_en)begin
            fifo_modeling.push_back(FI_tr.data_in);
            data_out_ref=fifo_modeling.pop_front();
        end
        else if(FI_tr.wr_en && fifo_modeling.size() != seq_item_tb.FIFO_DEPTH)begin
            fifo_modeling.push_back(FI_tr.data_in);
        end
        else if(FI_tr.rd_en && fifo_modeling.size() != 0)begin
            data_out_ref=fifo_modeling.pop_front();
        end
    end
endfunction

endclass
```

Coverage collector

```
package fifo_coverage_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_seq_item_pkg::*;

    class fifo_cov extends uvm_component;
        `uvm_component_utils(fifo_cov);

        seq_item item;
        uvm_tlm_analysis_fifo #(seq_item) cov_fifo;
        uvm_analysis_export  #(seq_item) cov_ep;

        //covergorup
        covergroup check_state ;
            Reset:coverpoint item.rst_n{
                bins low={0};
                bins high={1};
            }
            Writing:coverpoint item.wr_en{
                bins write_disaple={0};
                bins write_enaple={1};
                option.weight=0;
            }
            Reading:coverpoint item.rd_en{
                bins read_disaple={0};
                bins read_enaple={1};
                option.weight=0;
            }
```

```
            ack_BIN:coverpoint item.wr_ack {
                bins ack_low={0};
                bins ack_high={1};
                option.weight=0;
            }
            OF_BIN:coverpoint item.overflow {
                bins OF_low={0};
                bins OF_high={1};
                option.weight=0;
            }
            FULL_BIN:coverpoint item.full {
                bins FULL_low={0};
                bins FULL_high={1};
                option.weight=0;
            }
            EMPTY_BIN:coverpoint item.empty {
                bins EMPTY_low={0};
                bins EMPTY_high={1};
                option.weight=0;
            }
            ALMOSTFULL_BIN:coverpoint item.almostfull {
                bins AF_low={0};
                bins AF_high={1};
                option.weight=0;
            }
            ALMOSTEMPTY_BIN:coverpoint item.almostempty {
                bins AE_low={0};
                bins AE_high={1};
                option.weight=0;
            }
```

```
        UNDERFLOW_BIN:coverpoint item.underflow {
            bins UF_low={0};
            bins UF_high={1};
            option.weight=0;
        }

        ACKNOWLDGE: cross Writing,Reading,ack_BIN{
            illegal_bins Ack1= binsof(ack_BIN) intersect {1} && binsof(Writing) intersect {0} ;
        }
        OVERFLOW:   cross Writing,Reading,OF_BIN{
            illegal_bins OF1= binsof(OF_BIN)  intersect {1} && binsof(Writing) intersect {0};
            illegal_bins OF2= binsof(OF_BIN)  intersect {1} && binsof(Writing) intersect {1} && binsof(Reading) intersect {1};
        }
        FULL:       cross Writing,Reading,FULL_BIN {
            illegal_bins full_1= binsof(FULL_BIN)  intersect {1} && binsof(Reading) intersect {1} ;
        }
        EMPTY:      cross Writing,Reading,EMPTY_BIN{
            illegal_bins empty_1= binsof(EMPTY_BIN)  intersect {1} && binsof(Writing) intersect {1} iff (item.rst_n);
        }
        ALMOSTFULL: cross Writing,Reading,ALMOSTFULL_BIN;
        ALMOSTEMPTY:cross Writing,Reading,ALMOSTEMPTY_BIN;
        UNDERFLOW:  cross Writing,Reading,UNDERFLOW_BIN{
            illegal_bins UNDERFLOW1=binsof(UNDERFLOW_BIN) intersect {1} && binsof(Writing) intersect {1};
            illegal_bins UNDERFLOW2=binsof(UNDERFLOW_BIN) intersect {1} && binsof(Writing) intersect {0} && binsof(Reading) intersect {0};
        }
        reset_check:cross Reset,ack_BIN,OF_BIN,FULL_BIN,EMPTY_BIN,ALMOSTFULL_BIN,ALMOSTEMPTY_BIN,UNDERFLOW_BIN{
            bins reset_ch=binsof(Reset.low) && binsof(ack_BIN.ack_low) && binsof(OF_BIN.OF_low) && binsof(FULL_BIN.FULL_low)
                        && binsof(EMPTY_BIN.EMPTY_high) && binsof(ALMOSTFULL_BIN.AF_low) && binsof(ALMOSTEMPTY_BIN.AE_low) && binsof(UNDERFLOW_BIN.UF_low)
            option.cross_auto_bin_max=0;
        }
    endgroup
```

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
          cov_fifo=new("covarage fifo",this);
          cov_ep=new("coverage export",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_ep.connect(cov_fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
            cov_fifo.get(item);
            check_state.sample();
            end

        endtask

    endclass
endpackage
```

Sharedpkg

```
package shared_pkg;
    int error_count, correct_count;
endpackage
```

Assertions

```
module Assertions (fifo_if if_t);
    always_comb begin
    if(!if_t.rst_n) begin
        rst_check:assert final(dut.wr_ptr==0 && dut.rd_ptr==0 && dut.count==0 && if_t.empty &&  !if_t.full && !if_t.almostfull && !if_t.almostempty );
    end
    if(dut.count==if_t.FIFO_DEPTH )begin
        full_check:assert final(if_t.full);
    end

    if(dut.count==if_t.FIFO_DEPTH -1)begin
        almost_full_check:assert final(if_t.almostfull);
    end
    if(dut.count==0)begin
        empty_check:assert final(if_t.empty);
    end

    if(dut.count==1)begin
        almost_empty_check:assert final(if_t.almostempty);
    end
  end


property overflow_check;
@(posedge if_t.clk)  disable iff(!if_t.rst_n) (dut.count == if_t.FIFO_DEPTH && if_t.wr_en && ! if_t.rd_en) |=> (if_t.overflow) ;
endproperty

property underflow_check;
@(posedge if_t.clk)  disable iff(!if_t.rst_n) (dut.count == 0 && ! if_t.wr_en && if_t.rd_en) |=> (if_t.underflow);
endproperty
```

```
31  property WrAck_check;
32  @(posedge if_t.clk) disable iff(!if_t.rst_n) (if_t.wr_en && dut.count != if_t.FIFO_DEPTH) |=> (if_t.wr_ack);
33  endproperty
34
35  property internal_WR_check;
36  @(posedge if_t.clk) disable iff(!if_t.rst_n) (if_t.wr_en && !if_t.rd_en && dut.count < if_t.FIFO_DEPTH) |=> ( ( dut.count == $past(dut.count) + 1'b1) && ( dut.wr_ptr==$past(dut.wr_ptr) + 1'b1) );
37  endproperty
38
39  property internal_RE_check;
40  @(posedge if_t.clk) disable iff(!if_t.rst_n) (if_t.rd_en && !if_t.wr_en && dut.count > 0) |=> ( ( dut.count == $past(dut.count) - 1'b1) && ( dut.rd_ptr==$past(dut.rd_ptr) + 1'b1) );
41  endproperty
42
43  property internal_WR_RE_full;
44  @(posedge if_t.clk) disable iff(!if_t.rst_n) ( if_t.rd_en && if_t.wr_en && if_t.full)  |=>  ( dut.count == $past(dut.count) -1'b1 && (dut.rd_ptr==$past(dut.rd_ptr) + 1'b1) );
45  endproperty
46
47  property internal_WR_RE_empty;
48  @(posedge if_t.clk) disable iff(!if_t.rst_n) ( if_t.rd_en && if_t.wr_en && if_t.empty)  |=>  ( dut.count == $past(dut.count) + 1'b1 && (dut.wr_ptr==$past(dut.wr_ptr) + 1'b1) );
49  endproperty
50
51  property internal_WR_RE;
52  @(posedge if_t.clk) disable iff(!if_t.rst_n) ( if_t.rd_en && if_t.wr_en && !if_t.full && !if_t.empty )  |=>  ( dut.count == $past(dut.count) && (dut.rd_ptr==$past(dut.rd_ptr) + 1'b1) && ( dut.wr_ptr
53  endproperty
54
55  OF_check:assert property(overflow_check);
56  UF_check:assert property(underflow_check);
57  ACK_check:assert property(WrAck_check);
58  intWr_check:assert property(internal_WR_check);
59  intRD_check:assert property(internal_RE_check);
60  intWR_RD_WR:assert property(internal_WR_RE);
61  intWR_RD_WR_full:assert property(internal_WR_RE_full);
62  intWR_RD_WR_empty:assert property(internal_WR_RE_empty);
63
64  OF_cover:cover property(overflow_check);
65  UF_cover:cover property(underflow_check);
66  ACK_cover:cover property(WrAck_check);
67  intWr_cover:cover property(internal_WR_check);
68  intRD_cover:cover property(internal_RE_check);
69  intRD_WR_cover:cover property(internal_WR_RE);
```

6-Coverage report

Code coverage

```
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           27        27         0     100.0

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

Statement Coverage for file FIFO.sv --

    1                                       ////////////////////////////////////////////////////////////////////////////////
    2                                       // Author: Kareem Waseem
    3                                       // Course: Digital Verification using SV & UVM
    4                                       //
    5                                       // Description: FIFO Design
    6                                       //
    7                                       ////////////////////////////////////////////////////////////////////////////////
    8
    9
    10
    11                                      module FIFO(fifo_if if_t);
    12
    13                                      reg [if_t.FIFO_WIDTH-1:0] mem [if_t.FIFO_DEPTH-1:0];
    14
    15                                      reg [if_t.max_fifo_addr-1:0] wr_ptr, rd_ptr;
    16                                      reg [if_t.max_fifo_addr:0] count;
    17
    18              1                  1024     always @(posedge if_t.clk or negedge if_t.rst_n) begin
    19                                             if (!if_t.rst_n) begin
    20              1                     4                wr_ptr <= 0;
    21              1                     4                if_t.overflow <= 0; // we need to assign zero to this var because if it was high in cycle so in next cy
    22              1                     4                if_t.wr_ack   <= 0; // we need to assign zero to this var because if it was high in cycle so in next cy
    23                                             end
    24                                             else if (if_t.wr_en  && count < if_t.FIFO_DEPTH) begin
    25              1                   461                mem[wr_ptr] <= if_t.data_in;
    26              1                   461                if t.wr ack <= 1;
```

```
Branch Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Branches                        25        25         0     100.0

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

Branch Coverage for file FIFO.sv --

------------------------------------IF Branch------------------------------------
    19                                      1024     Count coming in to IF
    19              1                          4        if (!if_t.rst_n) begin
    24              1                        461        else if (if_t.wr_en  && count < if_t.FIFO_DEPTH) begin
    29              1                        559        else begin
Branch totals: 3 hits of 3 branches = 100.0%

------------------------------------IF Branch------------------------------------
    31                                      559     Count coming in to IF
    31              1                          9                if(if_t.full && if_t.wr_en && ! if_t.rd_en ) begin
    35              1                        550                else begin
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    47                                      1024     Count coming in to IF
    47              1                          4        if (!if_t.rst_n) begin
    51              1                        453        else if (if_t.rd_en && count != 0) begin
    55              1                        567        else begin
Branch totals: 3 hits of 3 branches = 100.0%

------------------------------------IF Branch------------------------------------
    56                                      567     Count coming in to IF
    56              1                         18                if(if_t.empty && !if_t.wr_en && if_t.rd_en )
    58              1                        549                else
Branch totals: 2 hits of 2 branches = 100.0%
```

```
================================Toggle Details================================

Toggle Coverage for File FIFO.sv --

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

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

Toggle Coverage      =      100.0% (20 of 20 bins)

================================================================================
--- File: agent.sv
```

Functional coverage

```
        bin <write_enaple,read_enaple,AF_low>              199          1    Covered
        bin <write_disaple,read_disaple,AF_high>           26           1    Covered
        bin <write_enaple,read_disaple,AF_high>            31           1    Covered
        bin <write_disaple,read_enaple,AF_high>            13           1    Covered
        bin <write_enaple,read_enaple,AF_high>             38           1    Covered
    Cross check_state::ALMOSTEMPTY                      100.0%        100    Covered
        covered/total bins:                                 8          8
        missing/total bins:                                 0          8
        % Hit:                                          100.0%        100
        bin <write_disaple,read_disaple,AE_low>            238          1    Covered
        bin <write_enaple,read_disaple,AE_low>             222          1    Covered
        bin <write_disaple,read_enaple,AE_low>             233          1    Covered
        bin <write_enaple,read_enaple,AE_low>              183          1    Covered
        bin <write_disaple,read_disaple,AE_high>           46           1    Covered
        bin <write_enaple,read_disaple,AE_high>            21           1    Covered
        bin <write_disaple,read_enaple,AE_high>            23           1    Covered
        bin <write_enaple,read_enaple,AE_high>             54           1    Covered
    Cross check_state::UNDERFLOW                        100.0%        100    Covered
        covered/total bins:                                 5          5
        missing/total bins:                                 0          5
        % Hit:                                          100.0%        100
        bin <write_disaple,read_disaple,UF_low>            284          1    Covered
        bin <write_disaple,read_enaple,UF_low>             238          1    Covered
        bin <write_disaple,read_enaple,UF_high>            18           1    Covered
        bin <write_enaple,read_disaple,UF_low>             243          1    Covered
        bin <write_enaple,read_enaple,UF_low>              237          1    Covered
        illegal_bin UNDERFLOW1                               0               ZERO
        illegal_bin UNDERFLOW2                               0               ZERO
    Cross check_state::reset_check                      100.0%        100    Covered
        covered/total bins:                                 1          1
        missing/total bins:                                 0          1
        % Hit:                                          100.0%        100
        bin reset_ch                                        2          1    Covered
 CLASS fifo_cov

TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 1

DIRECTIVE COVERAGE:
```

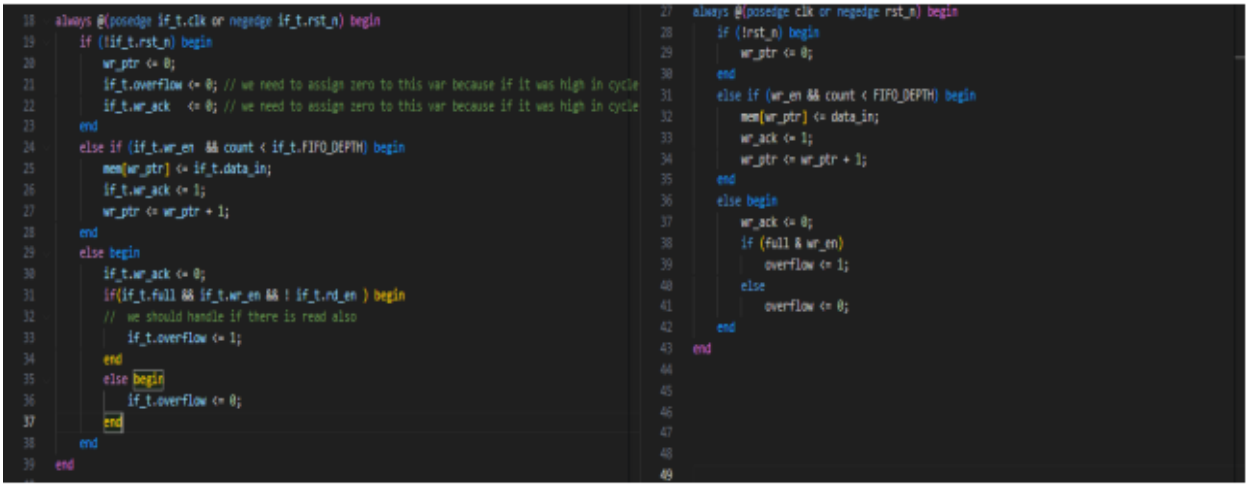
Assertions coverage

```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Count Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/top/dut/AS/OF_cover                    Assertions Verilog  SVA  Assertions.sv(64)
                                                                                9 Covered
/top/dut/AS/UF_cover                    Assertions Verilog  SVA  Assertions.sv(65)
                                                                               18 Covered
/top/dut/AS/ACK_cover                   Assertions Verilog  SVA  Assertions.sv(66)
                                                                              460 Covered
/top/dut/AS/intWr_cover                 Assertions Verilog  SVA  Assertions.sv(67)
                                                                              234 Covered
/top/dut/AS/intRD_cover                 Assertions Verilog  SVA  Assertions.sv(68)
                                                                              237 Covered
/top/dut/AS/intRD_WR_cover              Assertions Verilog  SVA  Assertions.sv(69)
                                                                              204 Covered
/top/dut/AS/intWR_RD_WR_empty_cover     Assertions Verilog  SVA  Assertions.sv(70)
                                                                               22 Covered
/top/dut/AS/intWR_RD_WR_full_cover      Assertions Verilog  SVA  Assertions.sv(71)
                                                                               10 Covered

TOTAL DIRECTIVE COVERAGE: 100.0%  COVERS: 8

ASSERTION RESULTS:
---------------------------------------------------------
```

7-Detected bugs

1-we need to add overflow and Write Acknowledge to reset because if overflow raise to high and then reset asserted so Overflow still high and same for Acknowledge so we need to put it in Reset

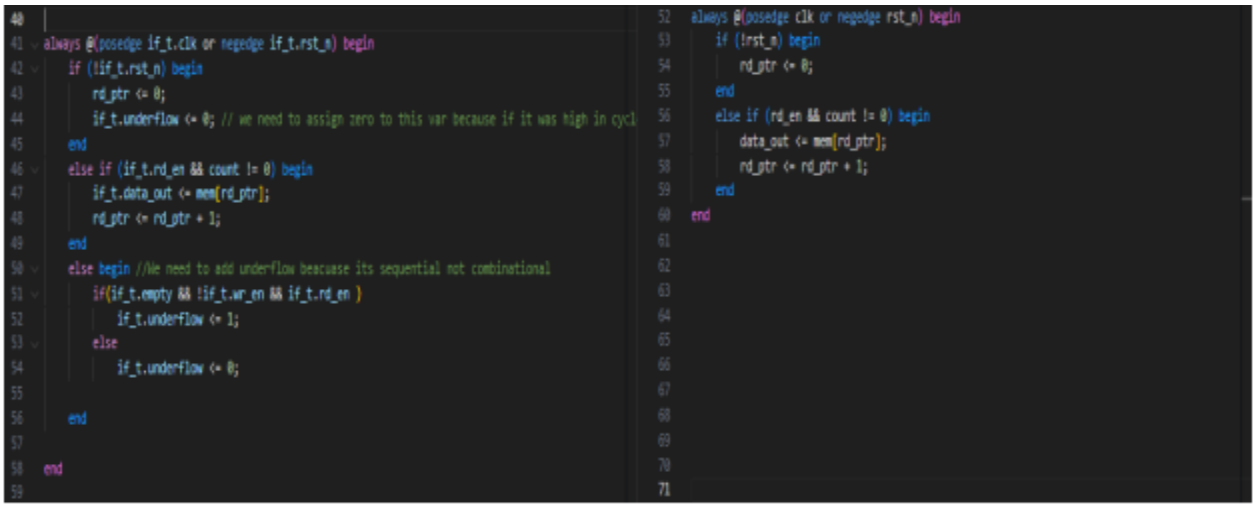
2- we need to put in condition statement in overflow that there is no reading with Writing because if full and write and read are asserted in same time Reading is executed



```
always @(posedge if_t.clk or negedge if_t.rst_n) begin
    if (!if_t.rst_n) begin
        wr_ptr <= 0;
        if_t.overflow <= 0; // we need to assign zero to this var because if it was high in cycle
        if_t.wr_ack   <= 0; // we need to assign zero to this var because if it was high in cycle
    end
    else if (if_t.wr_en  && count < if_t.FIFO_DEPTH) begin
        mem[wr_ptr] <= if_t.data_in;
        if_t.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        if_t.wr_ack <= 0;
        if(if_t.full && if_t.wr_en && ! if_t.rd_en ) begin
        //  we should handle if there is read also
            if_t.overflow <= 1;
        end
        else begin
            if_t.overflow <= 0;
        end
    end
end
```

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

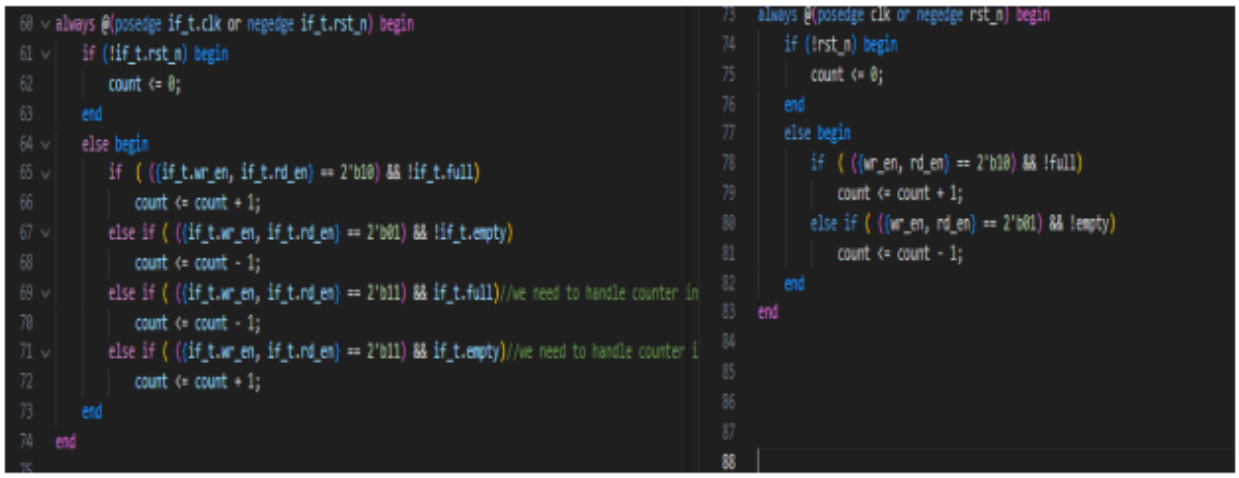
3-we need to add underflow flag in Always block because in spec it s sequential and add underflow in reset



```
always @(posedge if_t.clk or negedge if_t.rst_n) begin
    if (!if_t.rst_n) begin
        rd_ptr <= 0;
        if_t.underflow <= 0; // we need to assign zero to this var because if it was high in cycl
    end
    else if (if_t.rd_en && count != 0) begin
        if_t.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else begin //We need to add underflow because its sequential not combinational
        if(if_t.empty && !if_t.wr_en && if_t.rd_en )
            if_t.underflow <= 1;
        else
            if_t.underflow <= 0;

    end
end
```

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

4-we need to cover additional states that when writing and reading enabled or disabled at same time

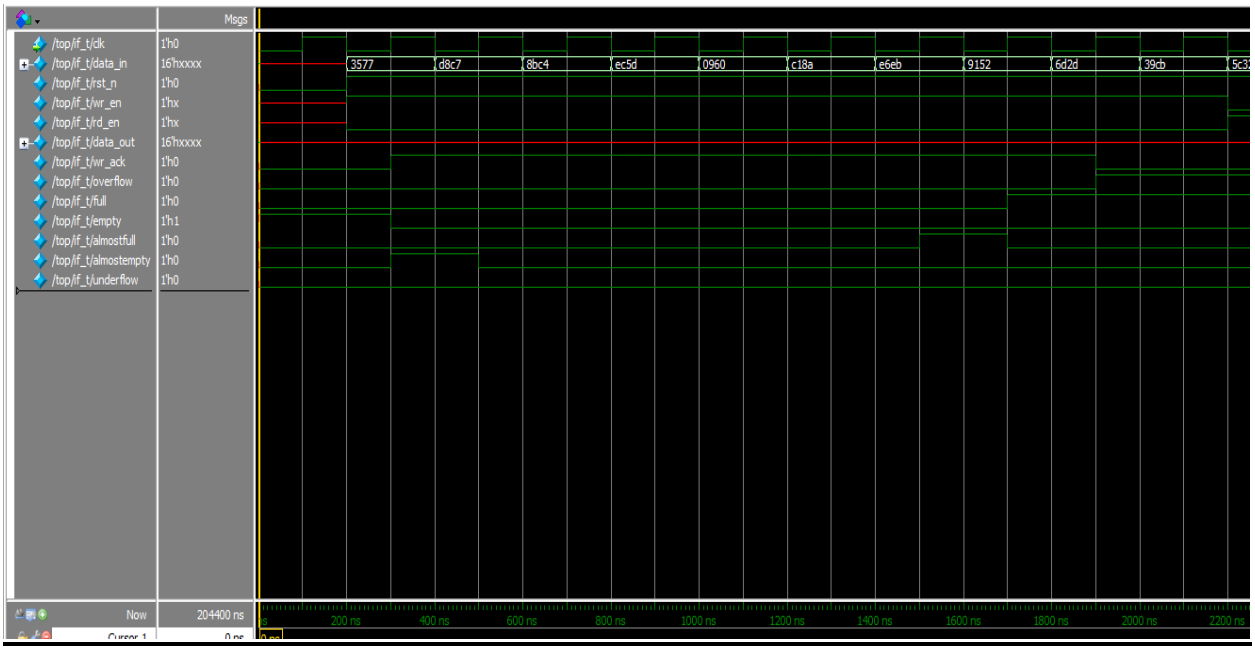


```
always @(posedge if_t.clk or negedge if_t.rst_n) begin
    if (!if_t.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ((if_t.wr_en, if_t.rd_en) == 2'b10) && !if_t.full)
            count <= count + 1;
        else if ( ((if_t.wr_en, if_t.rd_en) == 2'b01) && !if_t.empty)
            count <= count - 1;
        else if ( ((if_t.wr_en, if_t.rd_en) == 2'b11) && if_t.full)//we need to handle counter in
            count <= count - 1;
        else if ( ((if_t.wr_en, if_t.rd_en) == 2'b11) && if_t.empty)//we need to handle counter i
            count <= count + 1;
    end
end
```

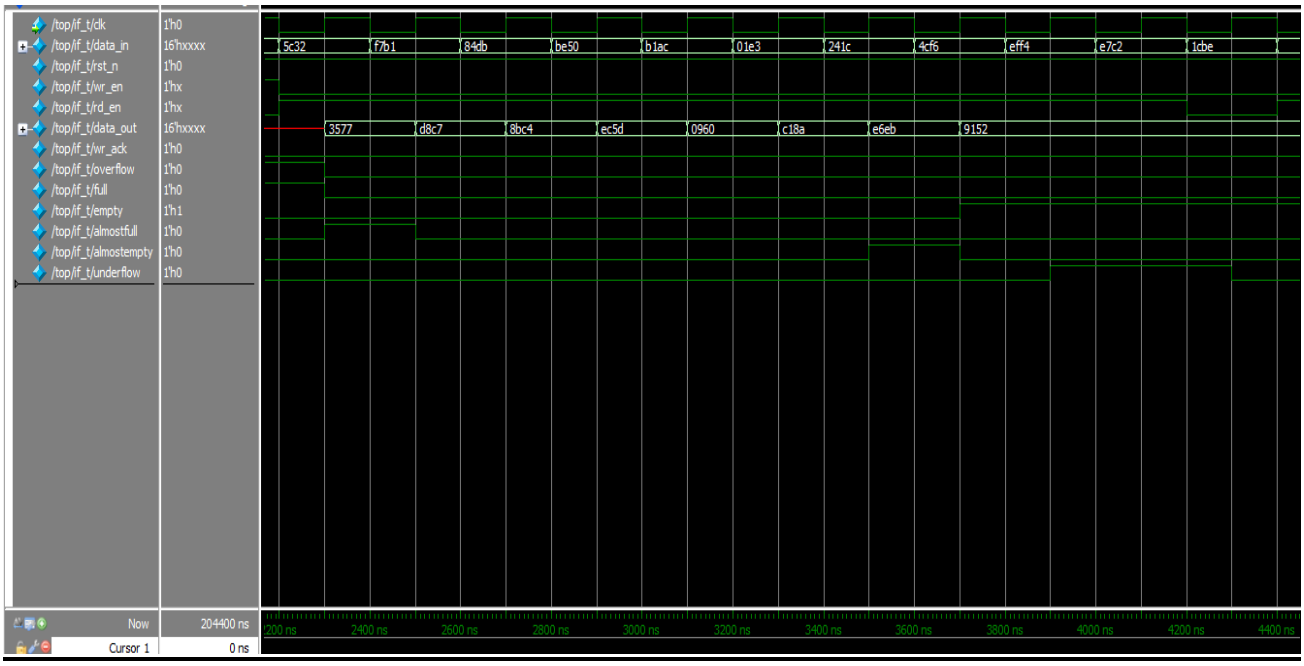
```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ((wr_en, rd_en) == 2'b10) && !full)
            count <= count + 1;
        else if ( ((wr_en, rd_en) == 2'b01) && !empty)
            count <= count - 1;
    end
end
```

8-Questa snippet

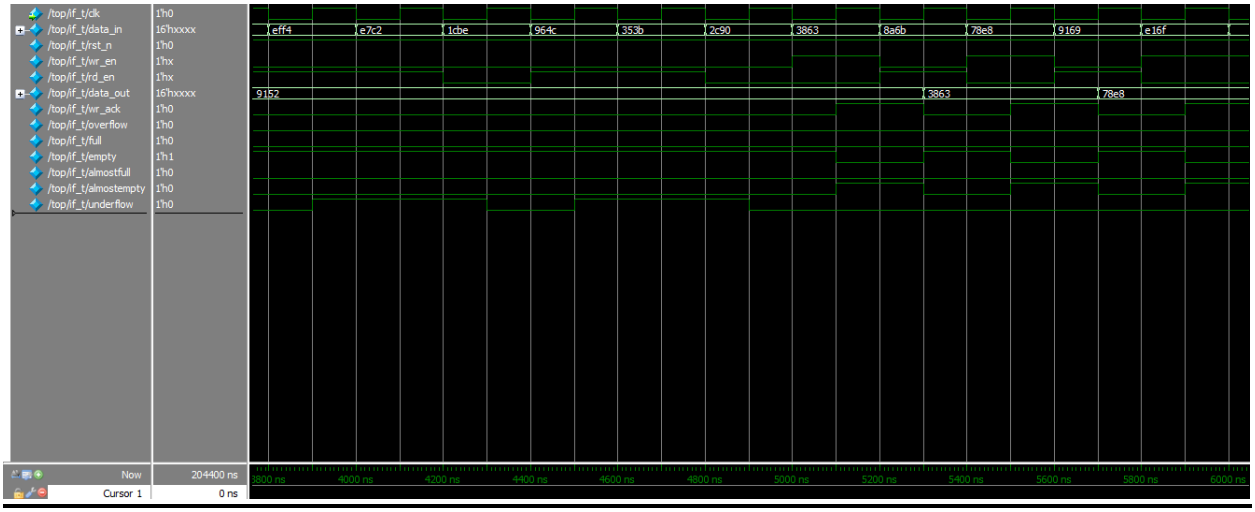
Reset and write sequence

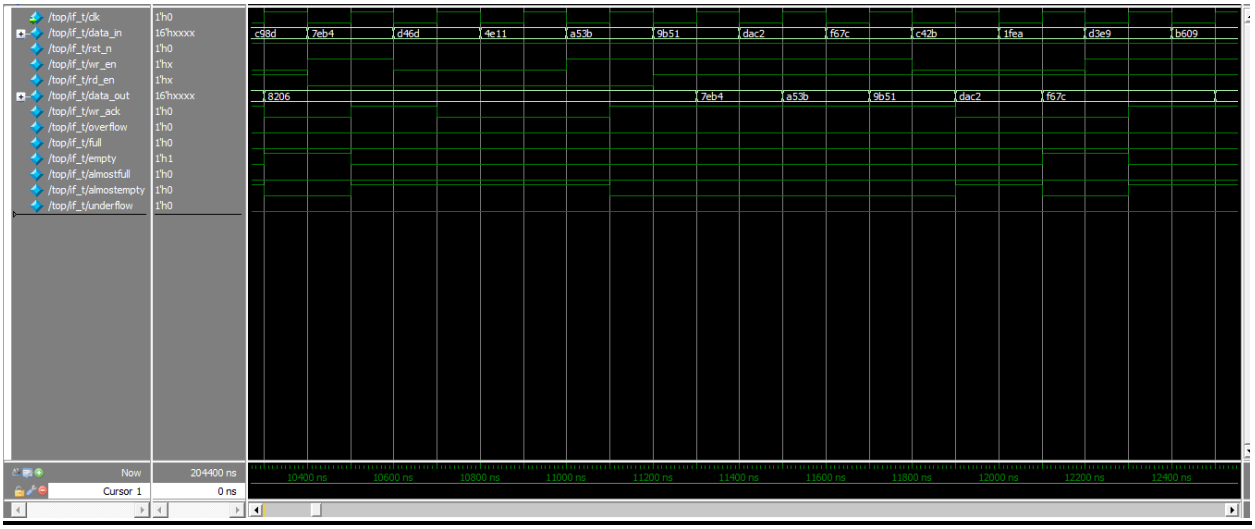


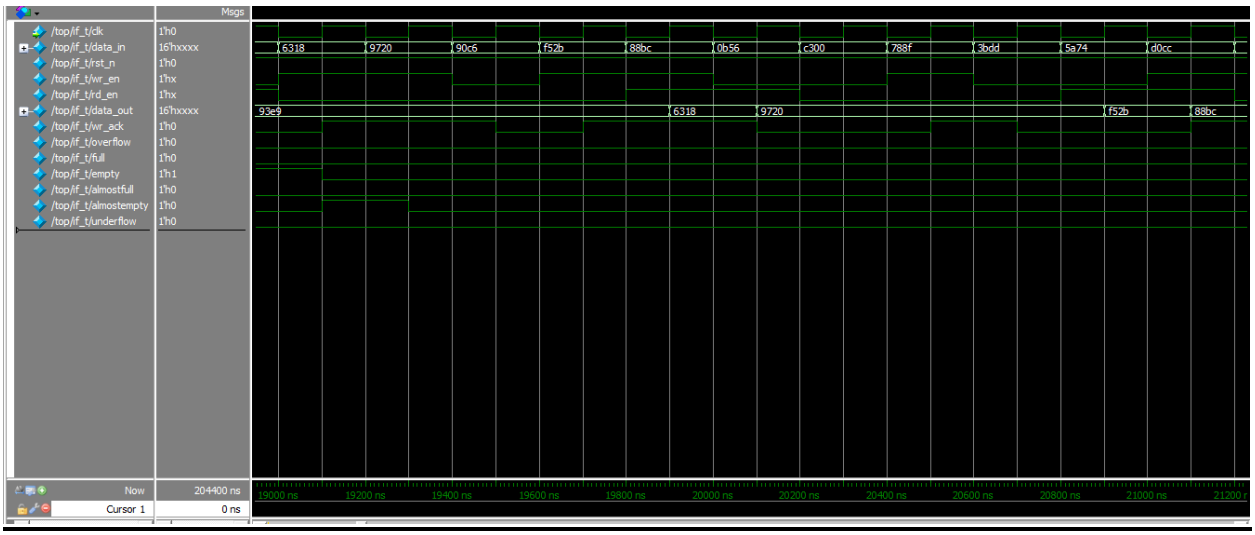
Read sequence



Write and read sequence







9-ASSERTIONS

Feature	Assertion
Check on flags and internal signal on reset	if(!rst_n) assert final(wr_ptr==0 && rd_ptr==0 && count==0 && empty && !full && !almost full && !almost empty);
When fifo size = depth Full flag is high	if(count== FIFO_DEPTH) assert final(full);
When fifo size = 0 empty flag is high	if(count==0) assert final(empty)
When fifo size = 1 almostempty flag is high	if(count==1) assert final(almost empty)
When fifo size = depth-1 almostfull flag is high	if(count== FIFO_DEPTH -1) assert final(almos tfull)
When fifo size = depth and there is write overflow flag is high	@(posedge clk) disable iff(!rst_n) (count == FIFO_DEPTH && wr_en) |=> (overflow)
When fifo size = 0 and there is read underflow flag is high	@(posedge clk) disable iff(!rst_n) (count == 0 && rd_en) |=> (underflow)
When fifo is not full wrAck is always high	@(posedge clk) disable iff(!rst_n) (wr_en && count != FIFO_DEPTH) |=> (wr_ack)
If there is write and no read and count != depth so count increase and wr ptr also	@(posedge clk) disable iff(!rst_n) (wr_en && !rd_en &&count < FIFO_DEPTH) |=> ((count == $past(count) + 1'b1) && (wr_ptr==$past(wr_ptr) + 1'b1))

If there is read and no write and count != 0 so count decrease and read ptr increase	@(posedge clk) disable iff(!rst_n) (rd_en && ! wr_en && count > 0) |=> (count == $past(count) - 1'b1) && (rd_ptr==$past(rd_ptr) + 1'b1));
If there is read and write and full flag high, count decrease and read ptr also	@(posedge clk) disable iff(!if_t.rst_n) (rd_en && wr_en && full) |=> (count == $past(count) -1'b1 && (rd_ptr==$past(rd_ptr) + 1'b1))
If there is read and write and empty high ,count increase and wr ptr also	@(posedge clk) disable iff(!rst_n) (rd_en && wr_en && empty) |=> (count == $past(count) + 1'b1 && (wr_ptr==$past(dut.wr_ptr) + 1'b1));
If there is read and write and empty flag low and full flag low count is constant and wr pointer increase same as read ptr	@(posedge clk) disable iff(!rst_n) (rd_en && wr_en && !full && !empty) |=> (count == $past(count) && (rd_ptr==$past(rd_ptr) + 1'b1) && (wr_ptr==$past(wr_ptr) + 1'b1));