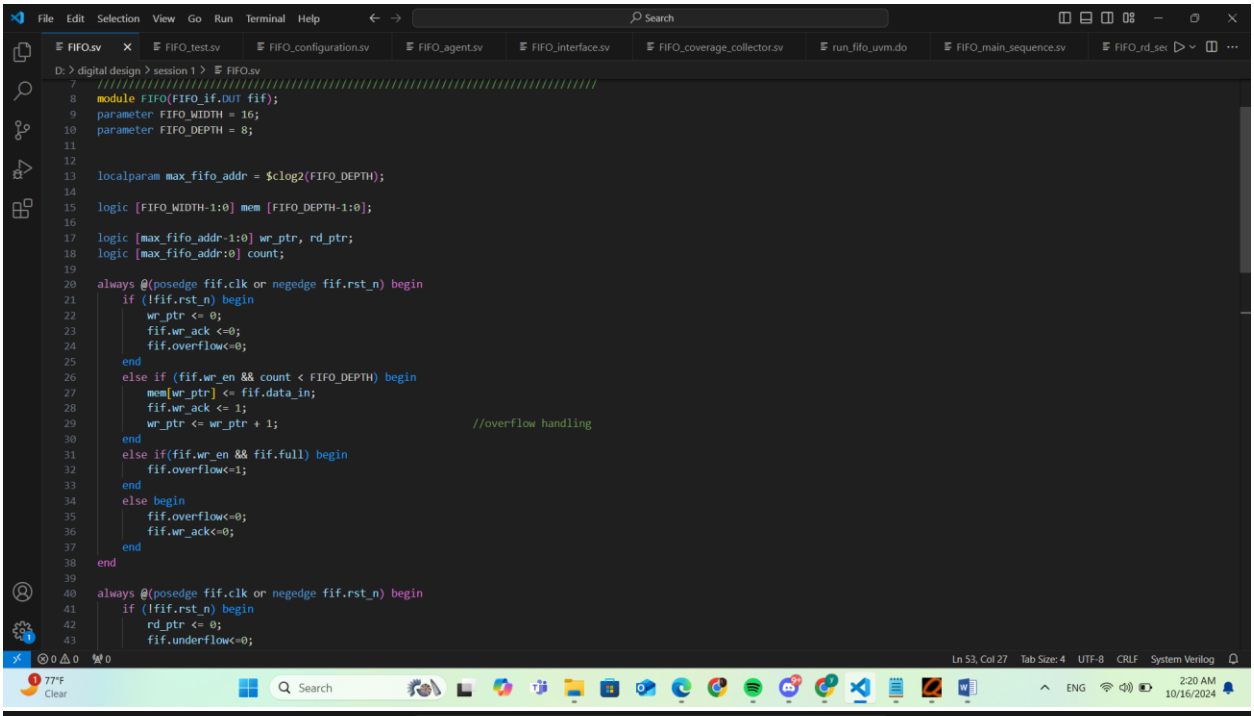
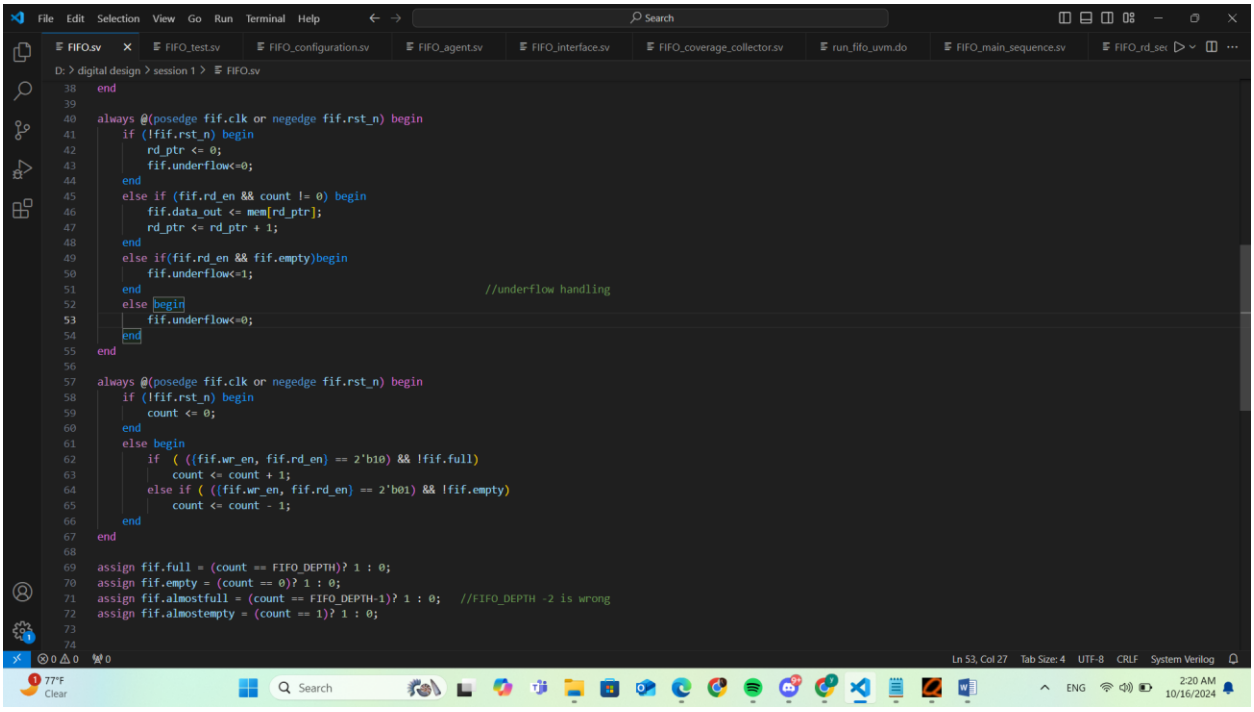
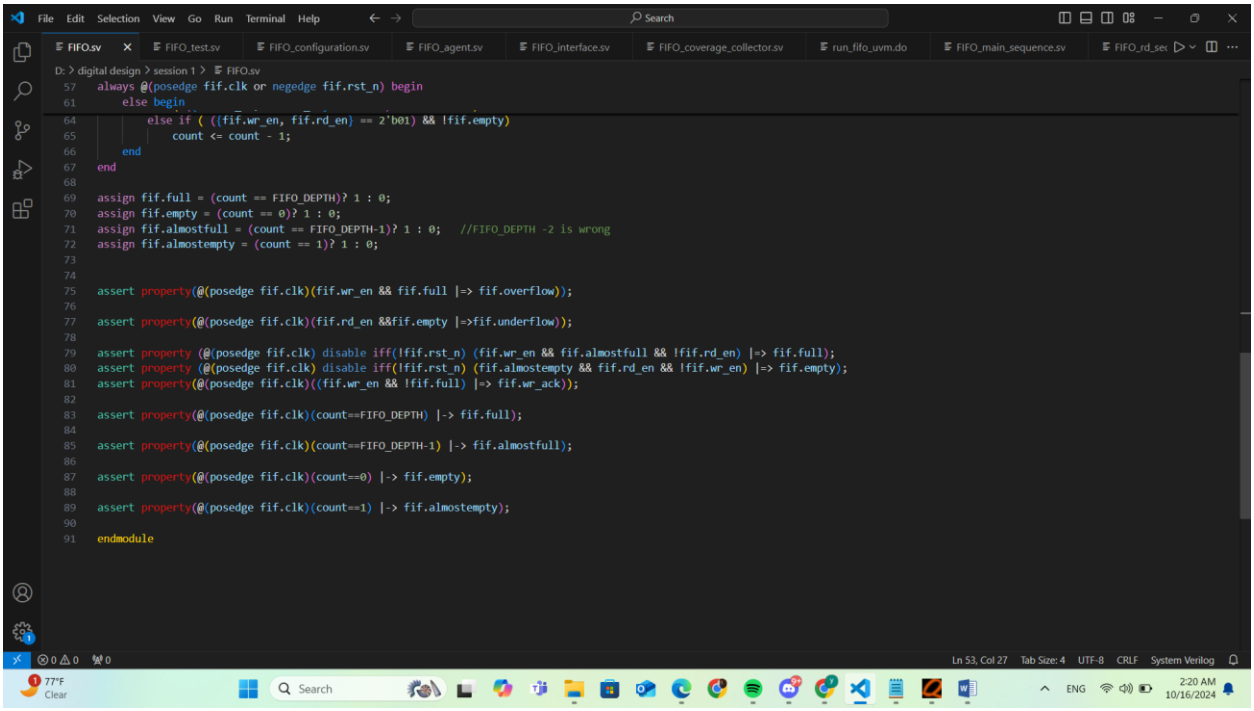
FIFO_design



```
module FIFO(FIFO_if.DUT fif);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;


localparam max_fifo_addr = $clog2(FIFO_DEPTH);

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

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

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

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

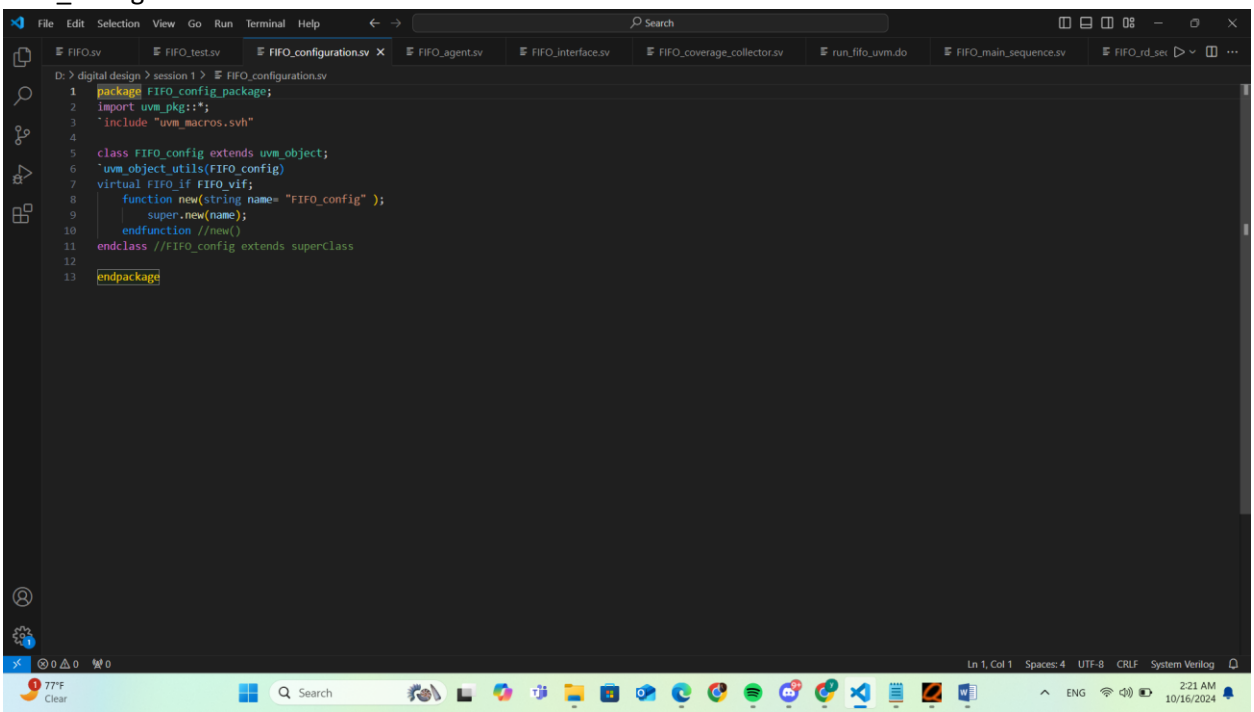


```
end

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

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

assign fif.full = (count == FIFO_DEPTH)? 1 : 0;
assign fif.empty = (count == 0)? 1 : 0;
assign fif.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;   //FIFO_DEPTH -2 is wrong
assign fif.almostempty = (count == 1)? 1 : 0;
```



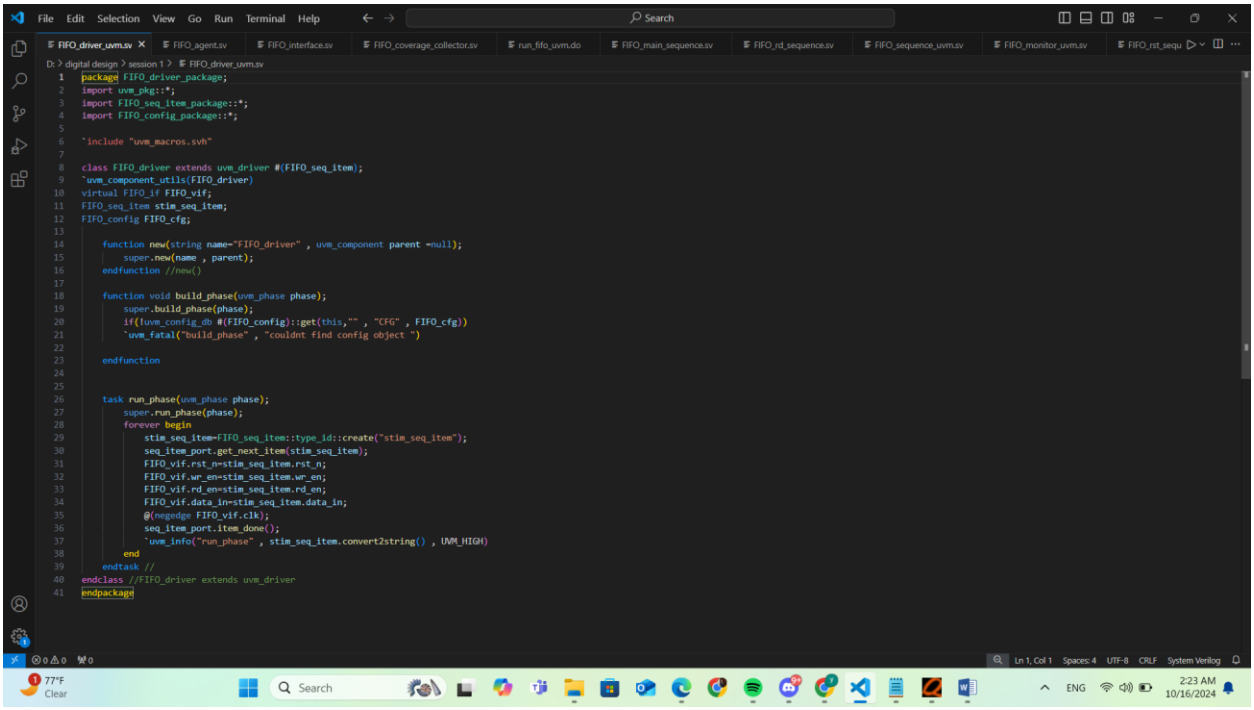
```
always @(posedge fif.clk or negedge fif.rst_n) begin
    else begin
        else if ( ({fif.wr_en, fif.rd_en} == 2'b01) && !fif.empty)
            count <= count - 1;
    end
end

assign fif.full = (count == FIFO_DEPTH)? 1 : 0;
assign fif.empty = (count == 0)? 1 : 0;
assign fif.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;   //FIFO_DEPTH -2 is wrong
assign fif.almostempty = (count == 1)? 1 : 0;


assert property(@(posedge fif.clk)(fif.wr_en && fif.full |=> fif.overflow));

assert property(@(posedge fif.clk)(fif.rd_en &&fif.empty |=>fif.underflow));

assert property (@(posedge fif.clk) disable iff(!fif.rst_n) (fif.wr_en && fif.almostfull && !fif.rd_en) |=> fif.full);
assert property (@(posedge fif.clk) disable iff(!fif.rst_n) (fif.almostempty && fif.rd_en && !fif.wr_en) |=> fif.empty);
assert property(@(posedge fif.clk)((fif.wr_en && !fif.full) |=> fif.wr_ack));

assert property(@(posedge fif.clk)(count==FIFO_DEPTH) |-> fif.full);

assert property(@(posedge fif.clk)(count==FIFO_DEPTH-1) |-> fif.almostfull);

assert property(@(posedge fif.clk)(count==0) |-> fif.empty);

assert property(@(posedge fif.clk)(count==1) |-> fif.almostempty);

endmodule
```

FIFO_configuration



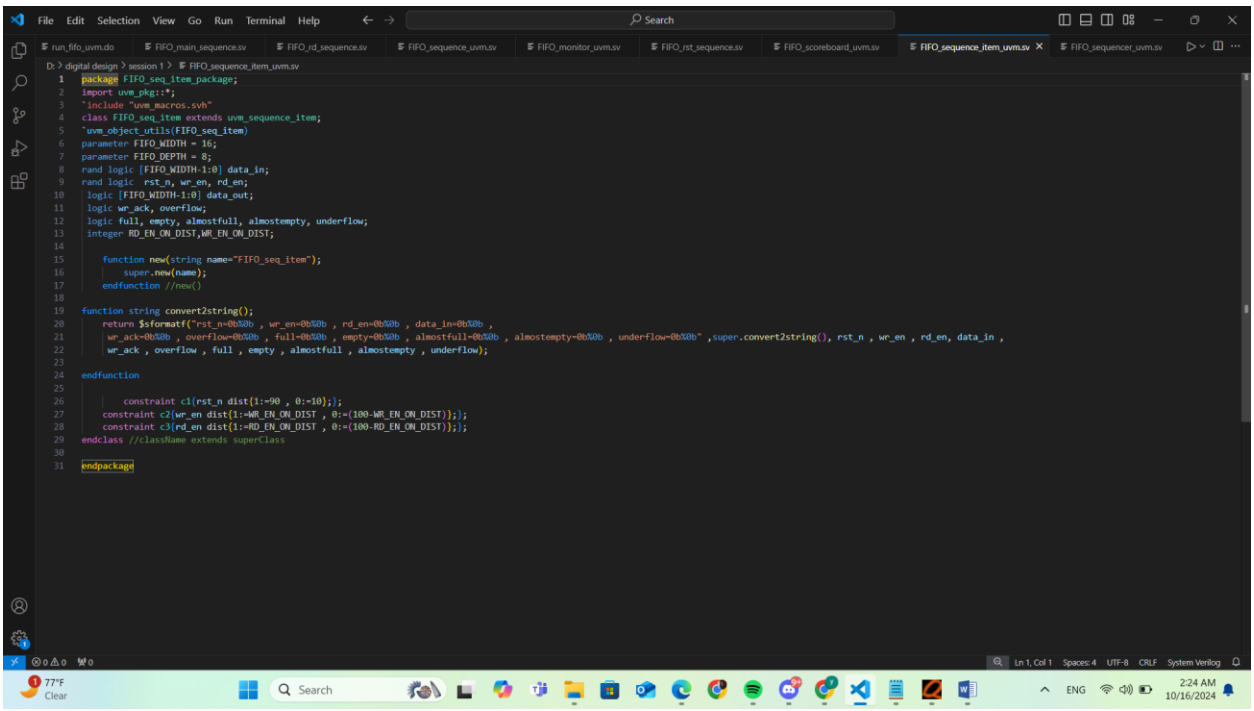
```
package FIFO_config_package;
import uvm_pkg::*;
`include "uvm_macros.svh"

class FIFO_config extends uvm_object;
`uvm_object_utils(FIFO_config)
virtual FIFO_if FIFO_vif;
    function new(string name= "FIFO_config" );
        super.new(name);
    endfunction //new()
endclass //FIFO_config extends superClass

endpackage
```

FIFO_driver



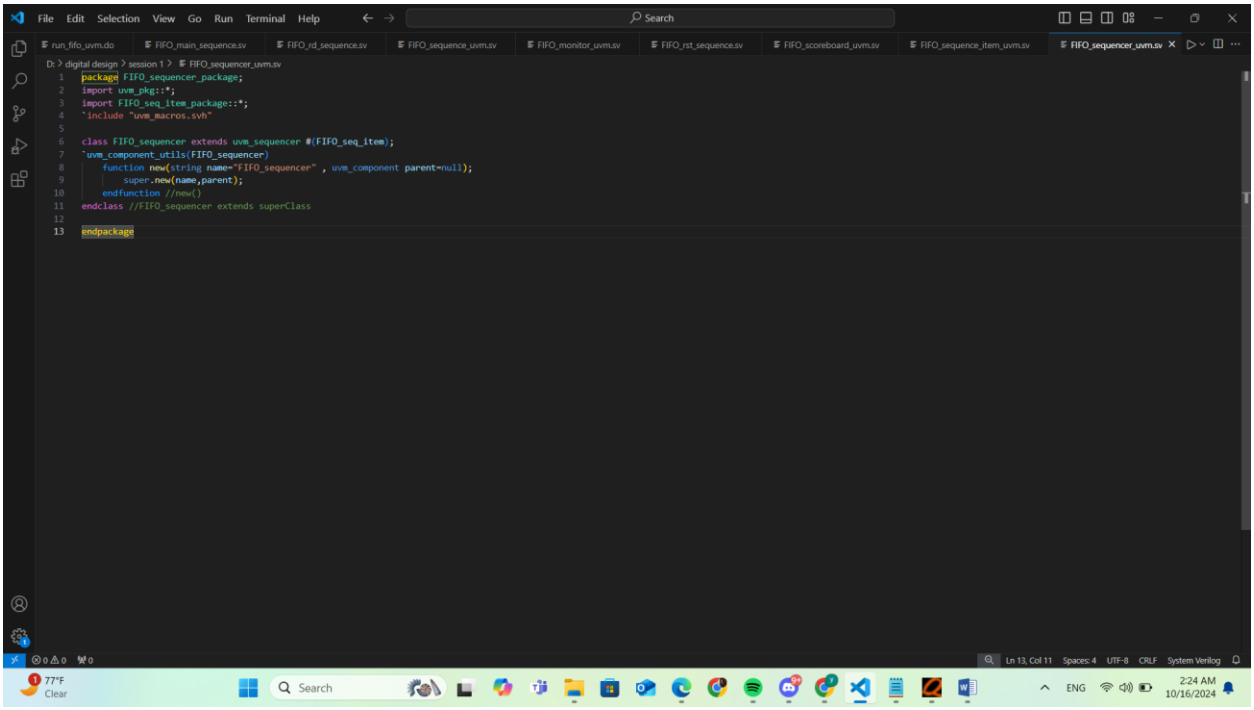
```
package FIFO_driver_package;
import uvm_pkg::*;
import FIFO_seq_item_package::*;
import FIFO_config_package::*;

`include "uvm_macros.svh"

class FIFO_driver extends uvm_driver #(FIFO_seq_item);
`uvm_component_utils(FIFO_driver)
virtual FIFO_if FIFO_vif;
FIFO_seq_item stim_seq_item;
FIFO_config FIFO_cfg;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(FIFO_config)::get(this,"" , "CFG" , FIFO_cfg))
        `uvm_fatal("build_phase" , "couldnt find config object ")

    endfunction


    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            stim_seq_item=FIFO_seq_item::type_id::create("stim_seq_item");
            seq_item_port.get_next_item(stim_seq_item);
            FIFO_vif.rst_n=stim_seq_item.rst_n;
            FIFO_vif.wr_en=stim_seq_item.wr_en;
            FIFO_vif.rd_en=stim_seq_item.rd_en;
            FIFO_vif.data_in=stim_seq_item.data_in;
            @(negedge FIFO_vif.clk);
            seq_item_port.item_done();
            `uvm_info("run_phase" , stim_seq_item.convert2string() , UVM_HIGH)
        end
    endtask //
endclass //FIFO_driver extends uvm_driver
endpackage
```

FIFO_sequence_item

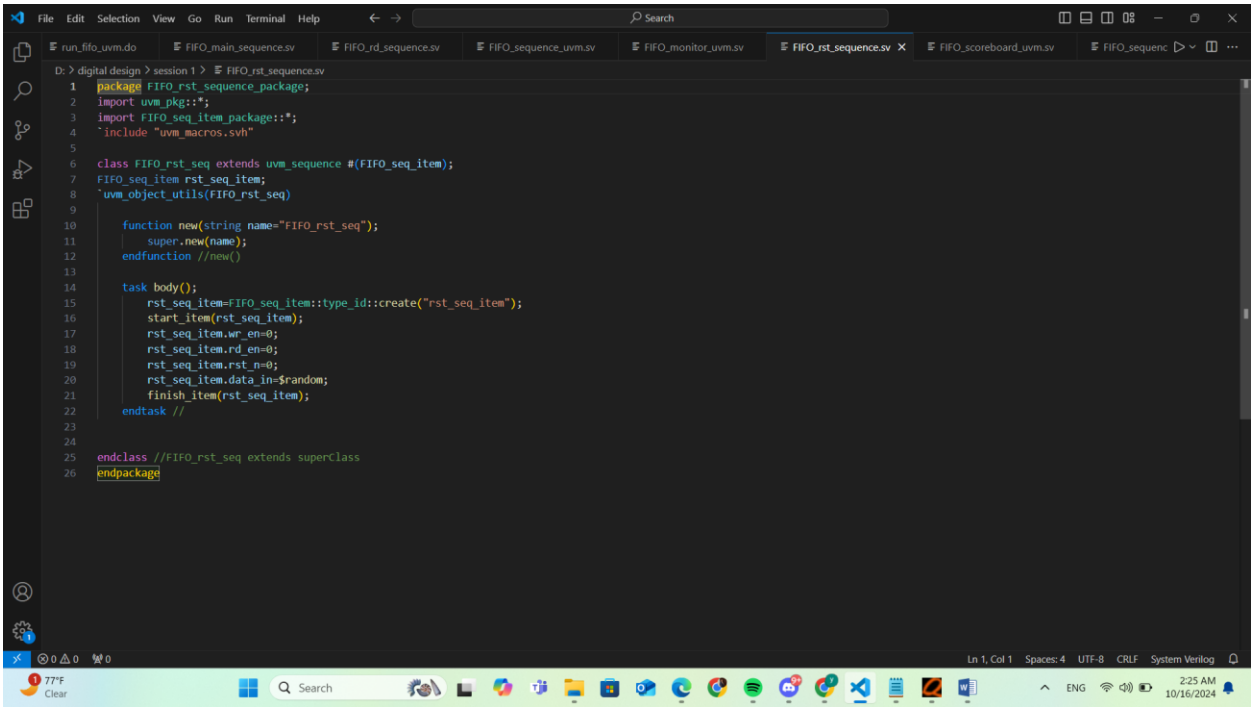


```
package FIFO_seq_item_package;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_seq_item extends uvm_sequence_item;
`uvm_object_utils(FIFO_seq_item)
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
rand logic [FIFO_WIDTH-1:0] data_in;
rand logic  rst_n, wr_en, rd_en;
 logic [FIFO_WIDTH-1:0] data_out;
 logic wr_ack, overflow;
 logic full, empty, almostfull, almostempty, underflow;
 integer RD_EN_ON_DIST,WR_EN_ON_DIST;

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

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

endfunction

        constraint c1{rst_n dist{1:=90 , 0:=10};};
    constraint c2{wr_en dist{1:=WR_EN_ON_DIST , 0:=(100-WR_EN_ON_DIST)};};
    constraint c3{rd_en dist{1:=RD_EN_ON_DIST , 0:=(100-RD_EN_ON_DIST)};};
endclass //className extends superClass

endpackage
```

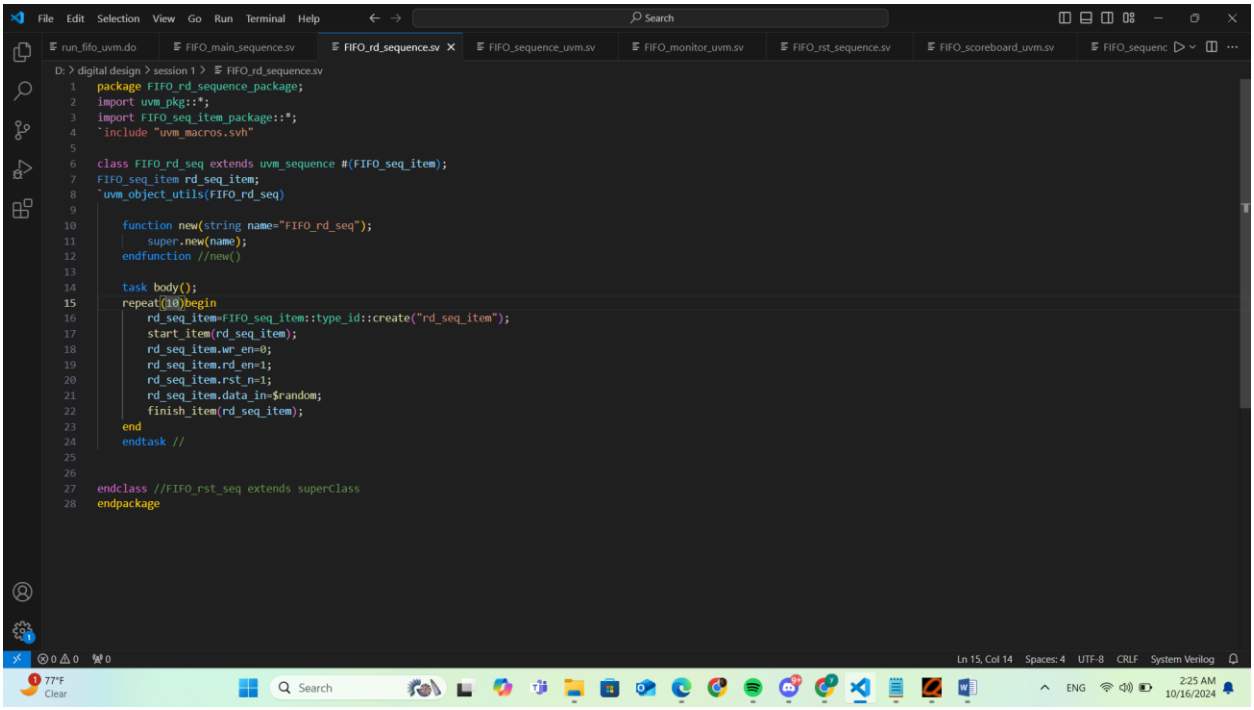
FIFO_sequencer



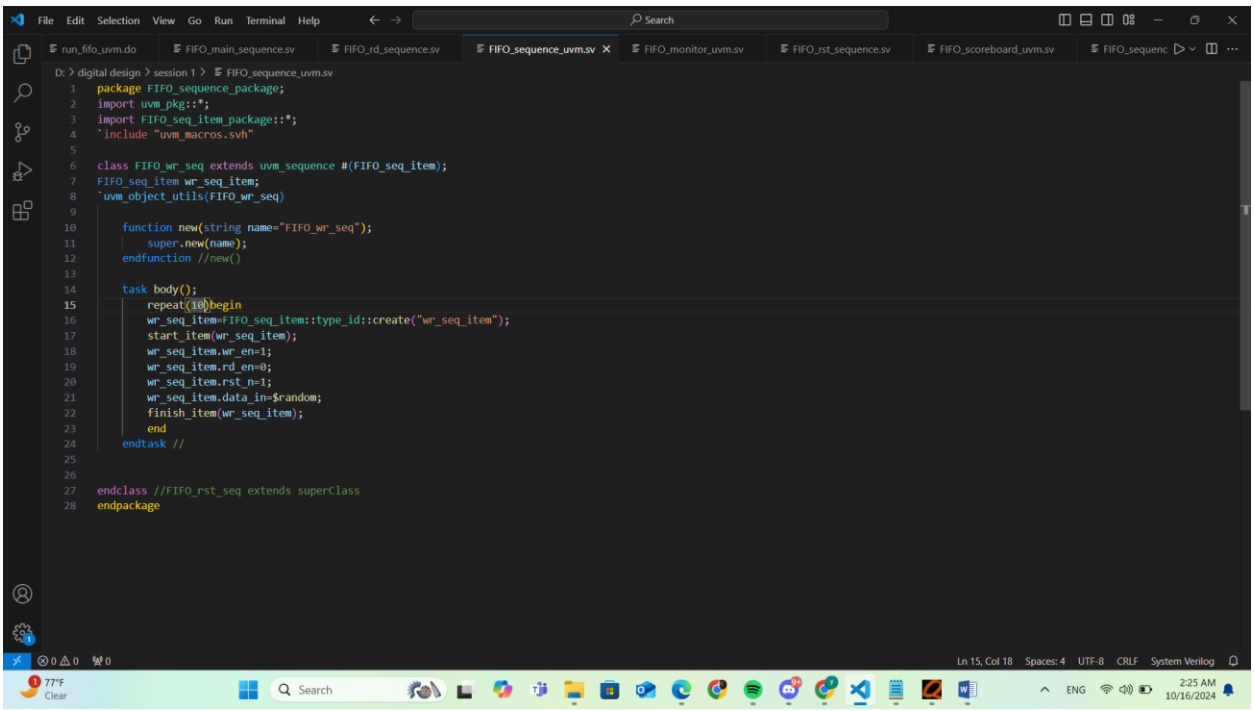
FIFO_reset_sequence



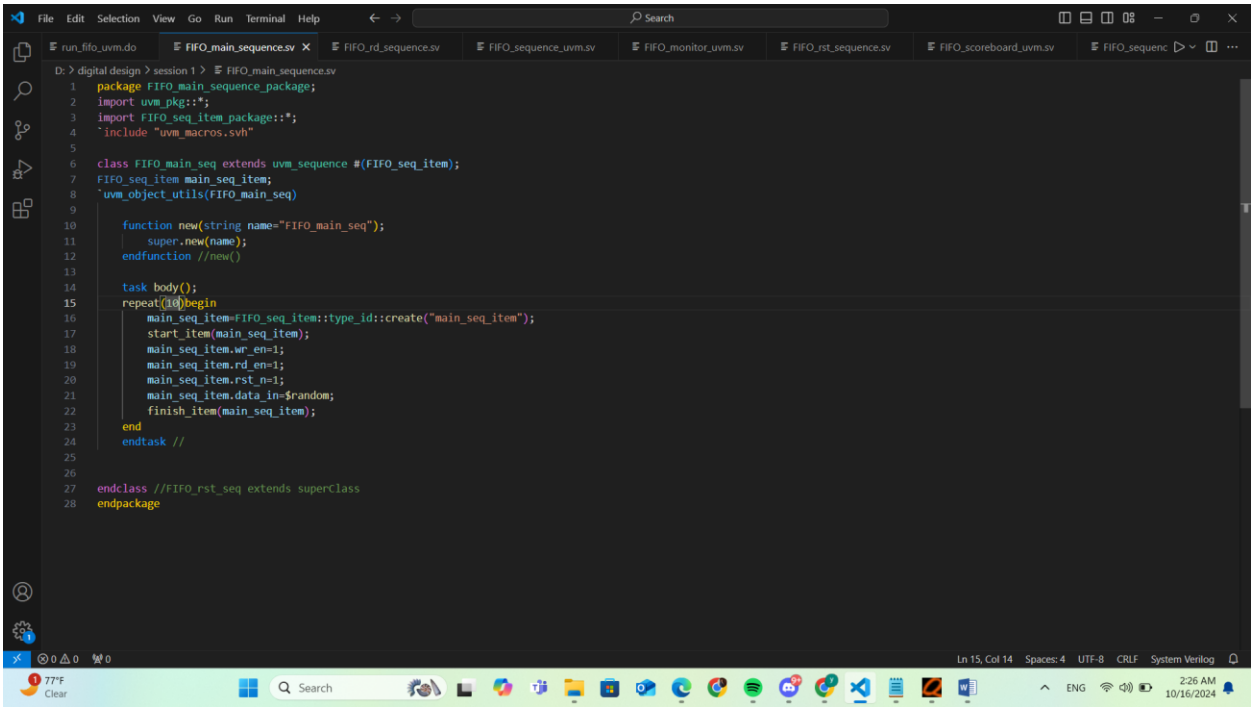
FIFO_read_sequence



FIFO_write_sequence

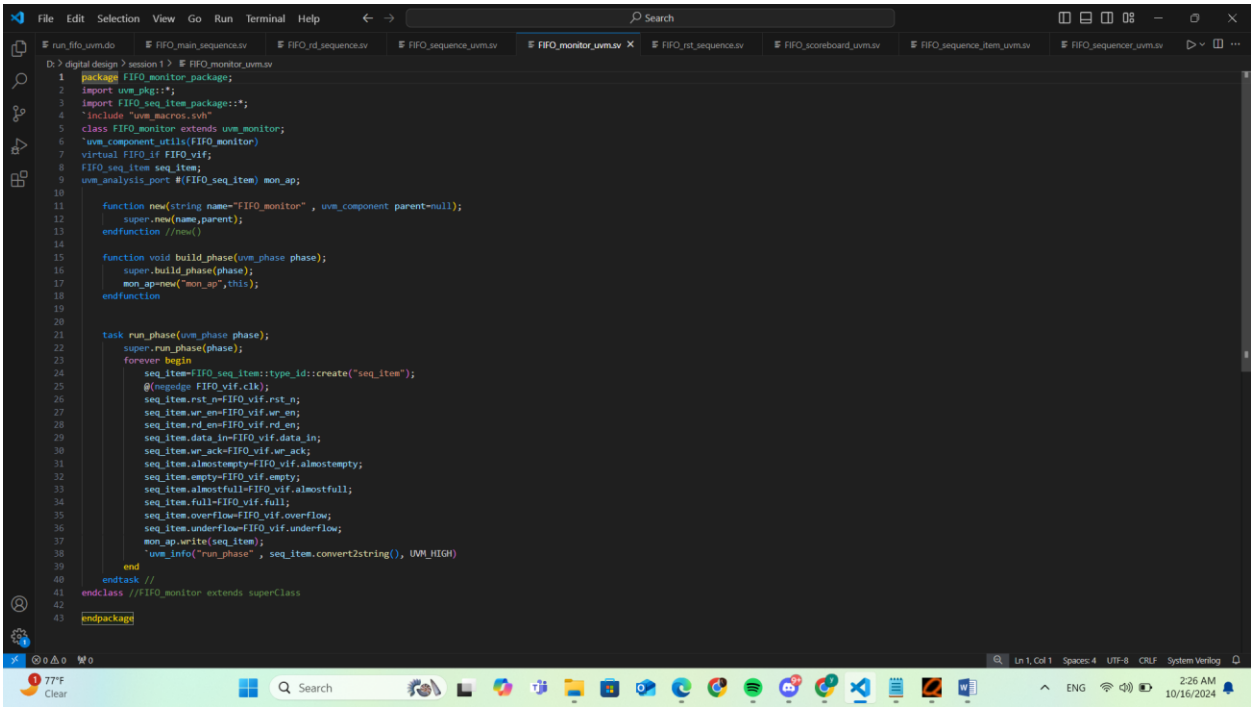


FIFO sequence with both read and write



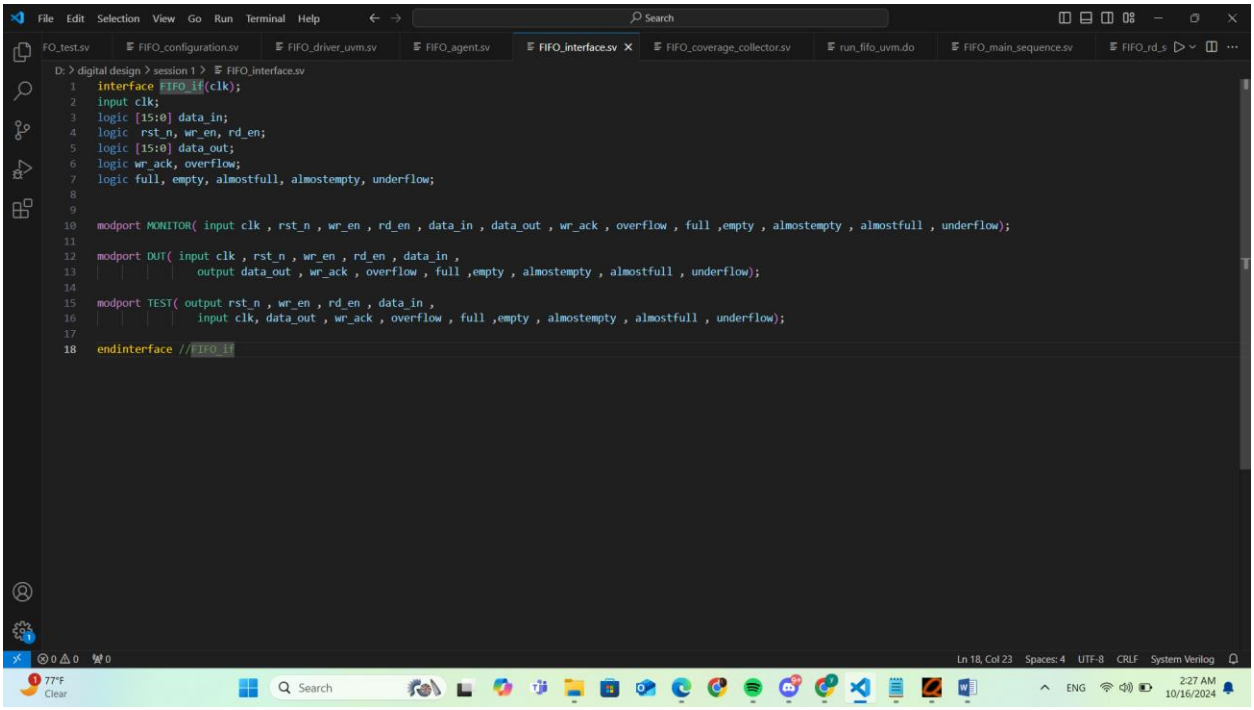
```
package FIFO_main_sequence_package;
import uvm_pkg::*;
import FIFO_seq_item_package::*;
`include "uvm_macros.svh"

class FIFO_main_seq extends uvm_sequence #(FIFO_seq_item);
FIFO_seq_item main_seq_item;
`uvm_object_utils(FIFO_main_seq)

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

    task body();
    repeat(10)begin
        main_seq_item=FIFO_seq_item::type_id::create("main_seq_item");
        start_item(main_seq_item);
        main_seq_item.wr_en=1;
        main_seq_item.rd_en=1;
        main_seq_item.rst_n=1;
        main_seq_item.data_in=$random;
        finish_item(main_seq_item);
    end
    endtask //


endclass //FIFO_rst_seq extends superClass
endpackage
```

FIFO_monitor



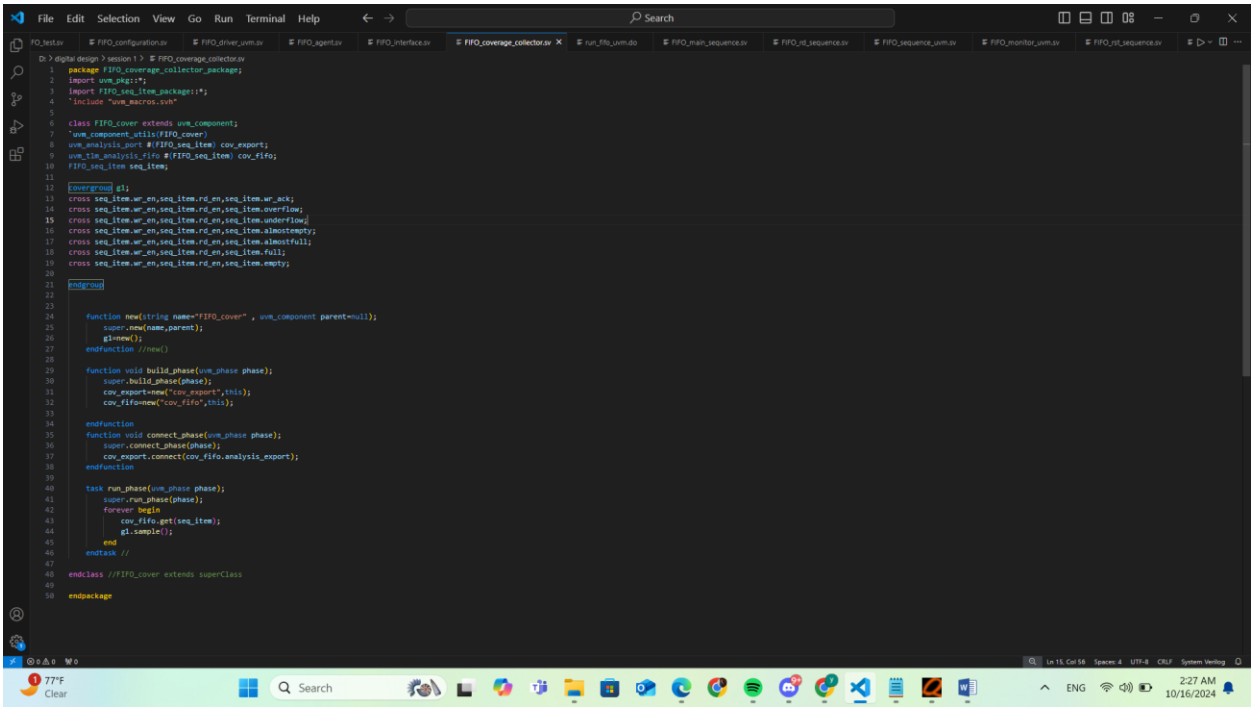
```
package FIFO_monitor_package;
import uvm_pkg::*;
import FIFO_seq_item_package::*;
`include "uvm_macros.svh"
class FIFO_monitor extends uvm_monitor;
`uvm_component_utils(FIFO_monitor)
virtual FIFO_if FIFO_vif;
FIFO_seq_item seq_item;
uvm_analysis_port #(FIFO_seq_item) mon_ap;

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

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


    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            seq_item=FIFO_seq_item::type_id::create("seq_item");
            @(negedge FIFO_vif.clk);
            seq_item.rst_n=FIFO_vif.rst_n;
            seq_item.wr_en=FIFO_vif.wr_en;
            seq_item.rd_en=FIFO_vif.rd_en;
            seq_item.data_in=FIFO_vif.data_in;
            seq_item.wr_ack=FIFO_vif.wr_ack;
            seq_item.almostempty=FIFO_vif.almostempty;
            seq_item.empty=FIFO_vif.empty;
            seq_item.almostfull=FIFO_vif.almostfull;
            seq_item.full=FIFO_vif.full;
            seq_item.overflow=FIFO_vif.overflow;
            seq_item.underflow=FIFO_vif.underflow;
            mon_ap.write(seq_item);
            `uvm_info("run_phase" , seq_item.convert2string(), UVM_HIGH)
        end
    endtask //
endclass //FIFO_monitor extends superClass

endpackage
```

FIFO_interface



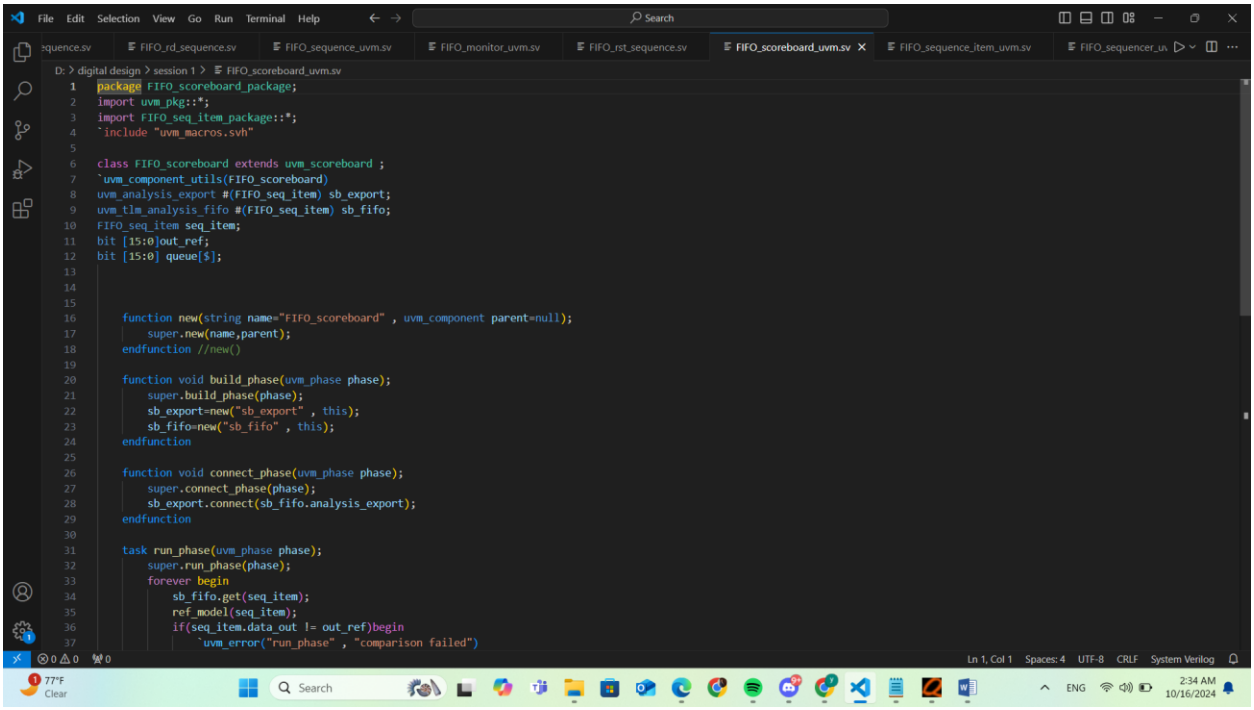
```
interface FIFO_if(clk);
input clk;
logic [15:0] data_in;
logic  rst_n, wr_en, rd_en;
logic [15:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;


modport MONITOR( input clk , rst_n , wr_en , rd_en , data_in , data_out , wr_ack , overflow , full ,empty , almostempty , almostfull , underflow);

modport DUT( input clk , rst_n , wr_en , rd_en , data_in ,
             output data_out , wr_ack , overflow , full ,empty , almostempty , almostfull , underflow);

modport TEST( output rst_n , wr_en , rd_en , data_in ,
              input clk, data_out , wr_ack , overflow , full ,empty , almostempty , almostfull , underflow);

endinterface //FIFO_if
```

FIFO_coverage_collector

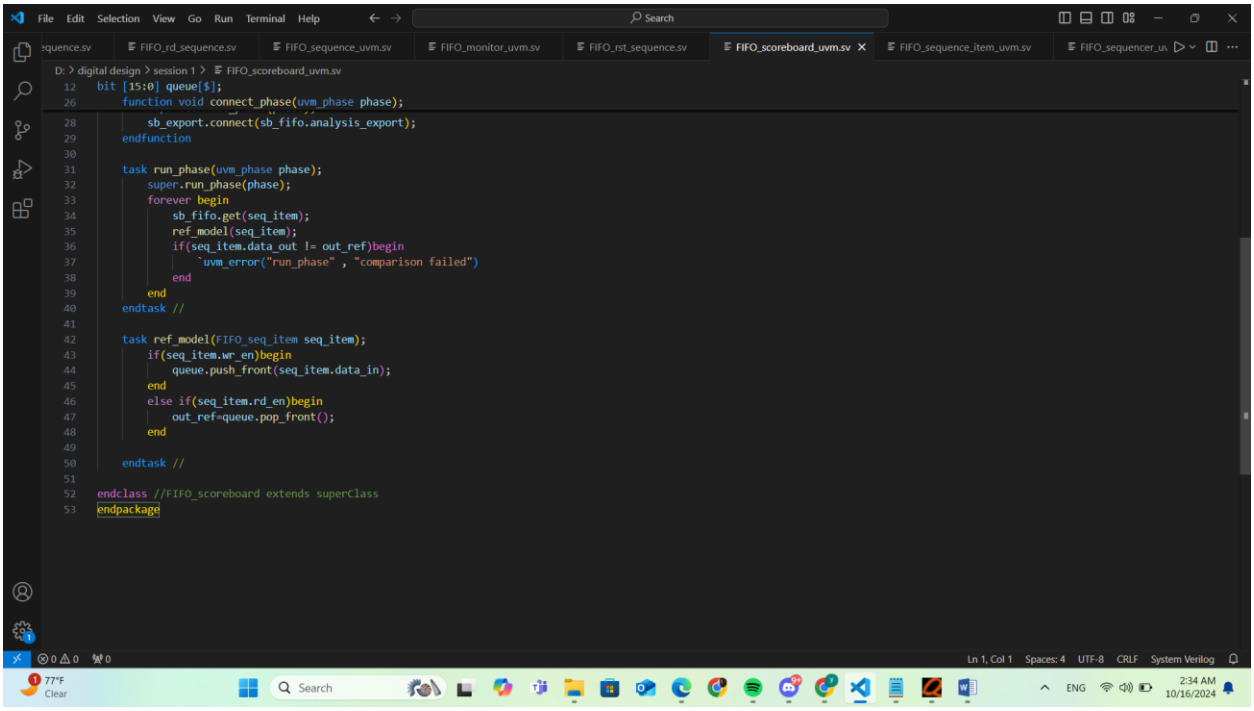


```
package FIFO_coverage_collector_package;
import uvm_pkg::*;
import FIFO_seq_item_package::*;
`include "uvm_macros.svh"

class FIFO_cover extends uvm_component;
`uvm_component_utils(FIFO_cover)
uvm_analysis_port #(FIFO_seq_item) cov_export;
uvm_tlm_analysis_fifo #(FIFO_seq_item) cov_fifo;
FIFO_seq_item seq_item;

covergroup g1;
cross seq_item.wr_en,seq_item.rd_en,seq_item.wr_ack;
cross seq_item.wr_en,seq_item.rd_en,seq_item.overflow;
cross seq_item.wr_en,seq_item.rd_en,seq_item.underflow;
cross seq_item.wr_en,seq_item.rd_en,seq_item.almostempty;
cross seq_item.wr_en,seq_item.rd_en,seq_item.almostfull;
cross seq_item.wr_en,seq_item.rd_en,seq_item.full;
cross seq_item.wr_en,seq_item.rd_en,seq_item.empty;

endgroup


    function new(string name="FIFO_cover" , uvm_component parent=null);
        super.new(name,parent);
        g1=new();
    endfunction //new()

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        cov_export=new("cov_export",this);
        cov_fifo=new("cov_fifo",this);

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            cov_fifo.get(seq_item);
            g1.sample();
        end
    endtask //

endclass //FIFO_cover extends superClass

endpackage
```

FIFO_scoreboard

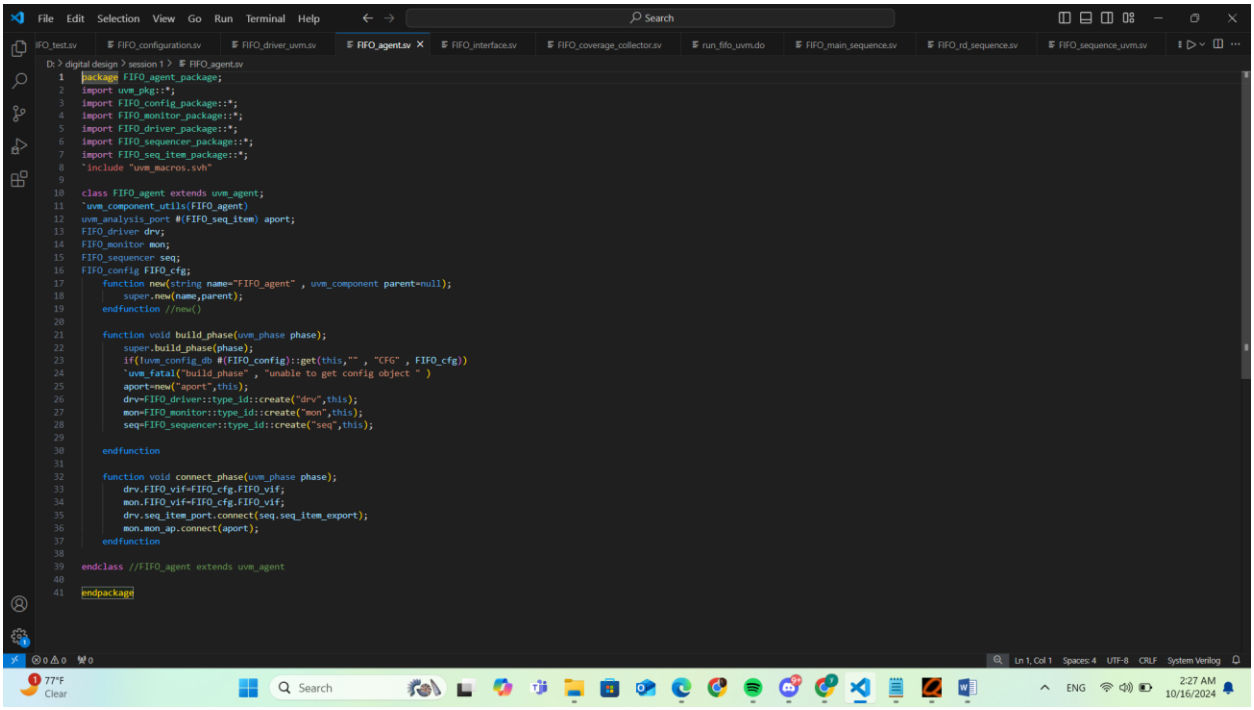


```
package FIFO_scoreboard_package;
import uvm_pkg::*;
import FIFO_seq_item_package::*;
`include "uvm_macros.svh"

class FIFO_scoreboard extends uvm_scoreboard ;
`uvm_component_utils(FIFO_scoreboard)
uvm_analysis_export #(FIFO_seq_item) sb_export;
uvm_tlm_analysis_fifo #(FIFO_seq_item) sb_fifo;
FIFO_seq_item seq_item;
bit [15:0]out_ref;
bit [15:0] queue[$];



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

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(seq_item);
            ref_model(seq_item);
            if(seq_item.data_out != out_ref)begin
                `uvm_error("run_phase" , "comparison failed")
```



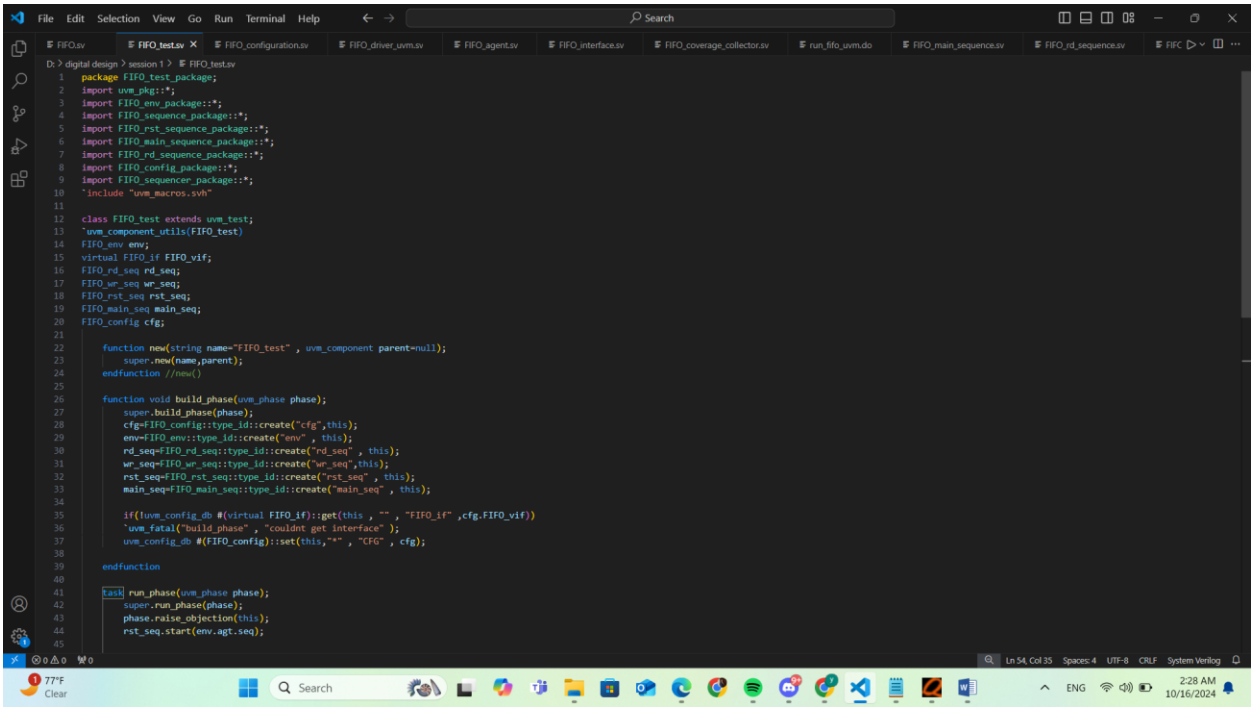
```
bit [15:0] queue[$];
    function void connect_phase(uvm_phase phase);
        sb_export.connect(sb_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(seq_item);
            ref_model(seq_item);
            if(seq_item.data_out != out_ref)begin
                `uvm_error("run_phase" , "comparison failed")
            end
        end
    endtask //

    task ref_model(FIFO_seq_item seq_item);
        if(seq_item.wr_en)begin
            queue.push_front(seq_item.data_in);
        end
        else if(seq_item.rd_en)begin
            out_ref=queue.pop_front();
        end

    endtask //

endclass //FIFO_scoreboard extends superClass
endpackage
```

FIFO_agent

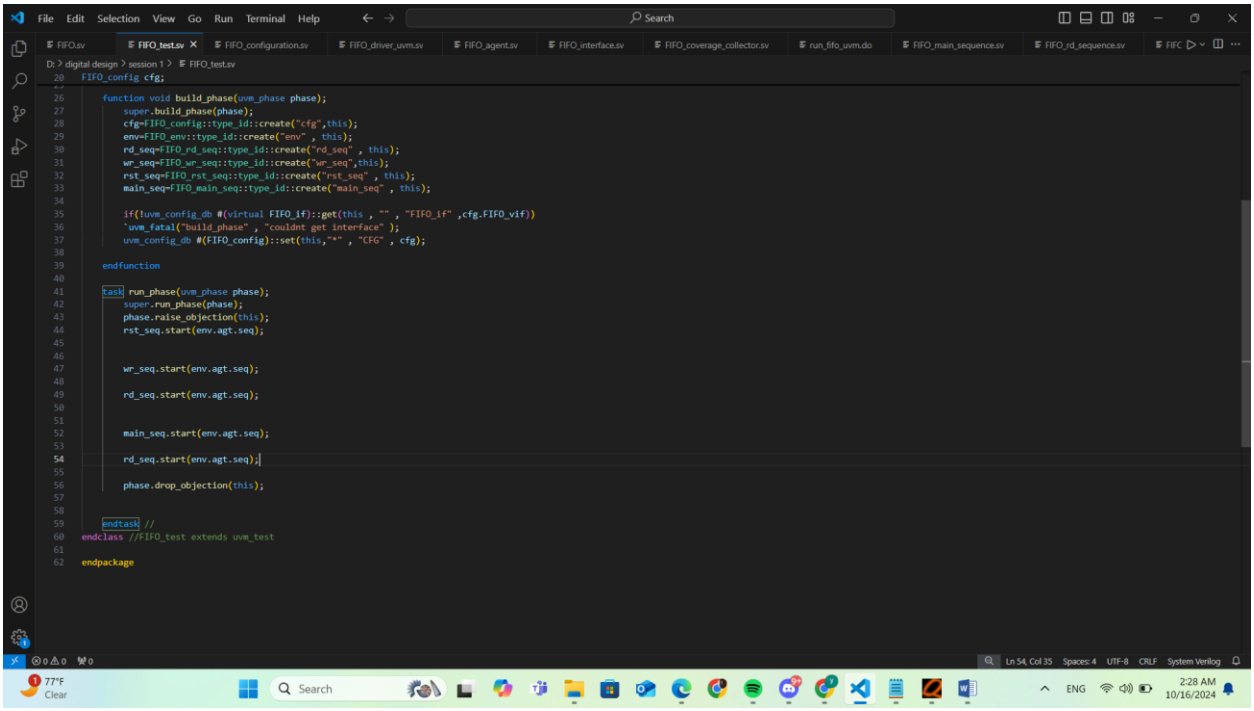


```
package FIFO_agent_package;
import uvm_pkg::*;
import FIFO_config_package::*;
import FIFO_monitor_package::*;
import FIFO_driver_package::*;
import FIFO_sequencer_package::*;
import FIFO_seq_item_package::*;
`include "uvm_macros.svh"

class FIFO_agent extends uvm_agent;
`uvm_component_utils(FIFO_agent)
uvm_analysis_port #(FIFO_seq_item) aport;
FIFO_driver drv;
FIFO_monitor mon;
FIFO_sequencer seq;
FIFO_config FIFO_cfg;
    function new(string name="FIFO_agent" , uvm_component parent=null);
        super.new(name,parent);
    endfunction //new()

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(FIFO_config)::get(this,"" , "CFG" , FIFO_cfg))
        `uvm_fatal("build_phase" , "unable to get config object " )
        aport=new("aport",this);
        drv=FIFO_driver::type_id::create("drv",this);
        mon=FIFO_monitor::type_id::create("mon",this);
        seq=FIFO_sequencer::type_id::create("seq",this);

    endfunction

    function void connect_phase(uvm_phase phase);
        drv.FIFO_vif=FIFO_cfg.FIFO_vif;
        mon.FIFO_vif=FIFO_cfg.FIFO_vif;
        drv.seq_item_port.connect(seq.seq_item_export);
        mon.mon_ap.connect(aport);
    endfunction

endclass //FIFO_agent extends uvm_agent

endpackage
```

FIFO_Test

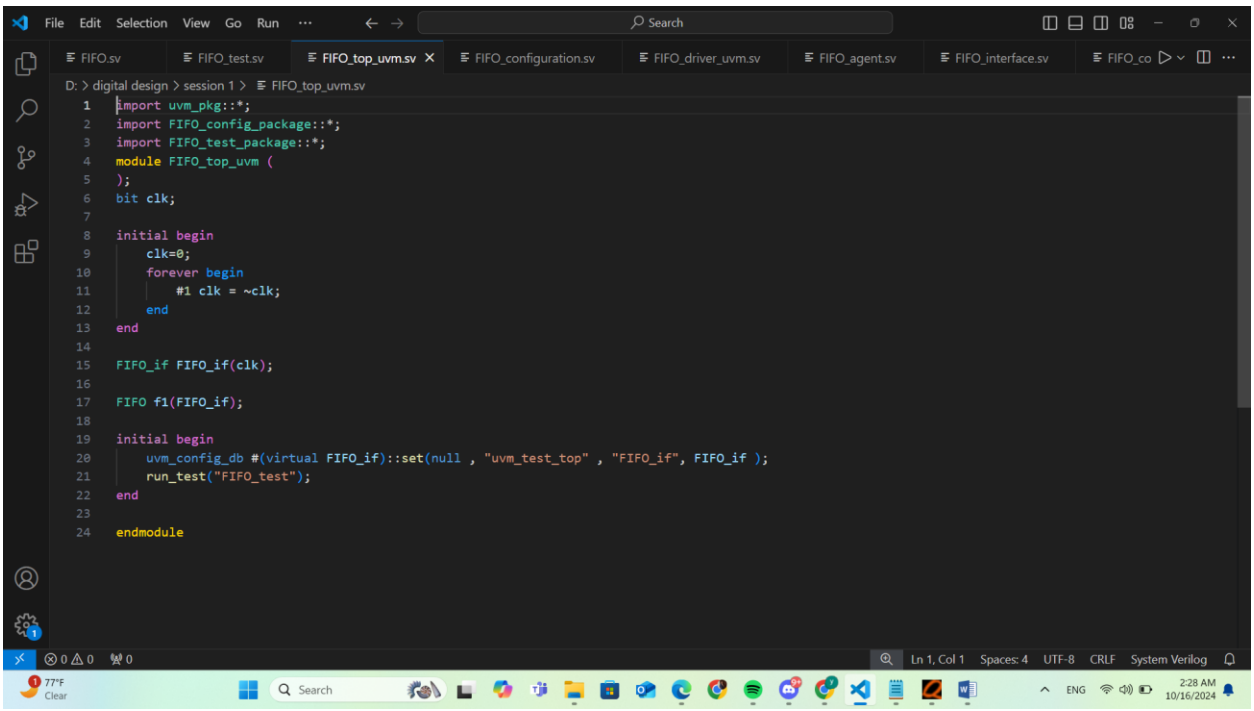


```
package FIFO_test_package;
import uvm_pkg::*;
import FIFO_env_package::*;
import FIFO_sequence_package::*;
import FIFO_rst_sequence_package::*;
import FIFO_main_sequence_package::*;
import FIFO_rd_sequence_package::*;
import FIFO_config_package::*;
import FIFO_sequencer_package::*;
`include "uvm_macros.svh"

class FIFO_test extends uvm_test;
`uvm_component_utils(FIFO_test)
FIFO_env env;
virtual FIFO_if FIFO_vif;
FIFO_rd_seq rd_seq;
FIFO_wr_seq wr_seq;
FIFO_rst_seq rst_seq;
FIFO_main_seq main_seq;
FIFO_config cfg;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        cfg=FIFO_config::type_id::create("cfg",this);
        env=FIFO_env::type_id::create("env" , this);
        rd_seq=FIFO_rd_seq::type_id::create("rd_seq" , this);
        wr_seq=FIFO_wr_seq::type_id::create("wr_seq",this);
        rst_seq=FIFO_rst_seq::type_id::create("rst_seq" , this);
        main_seq=FIFO_main_seq::type_id::create("main_seq" , this);

        if(!uvm_config_db #(virtual FIFO_if)::get(this , "" , "FIFO_if" ,cfg.FIFO_vif))
        `uvm_fatal("build_phase" , "couldnt get interface" );
        uvm_config_db #(FIFO_config)::set(this,"*" , "CFG" , cfg);

    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
        rst_seq.start(env.agt.seq);
```



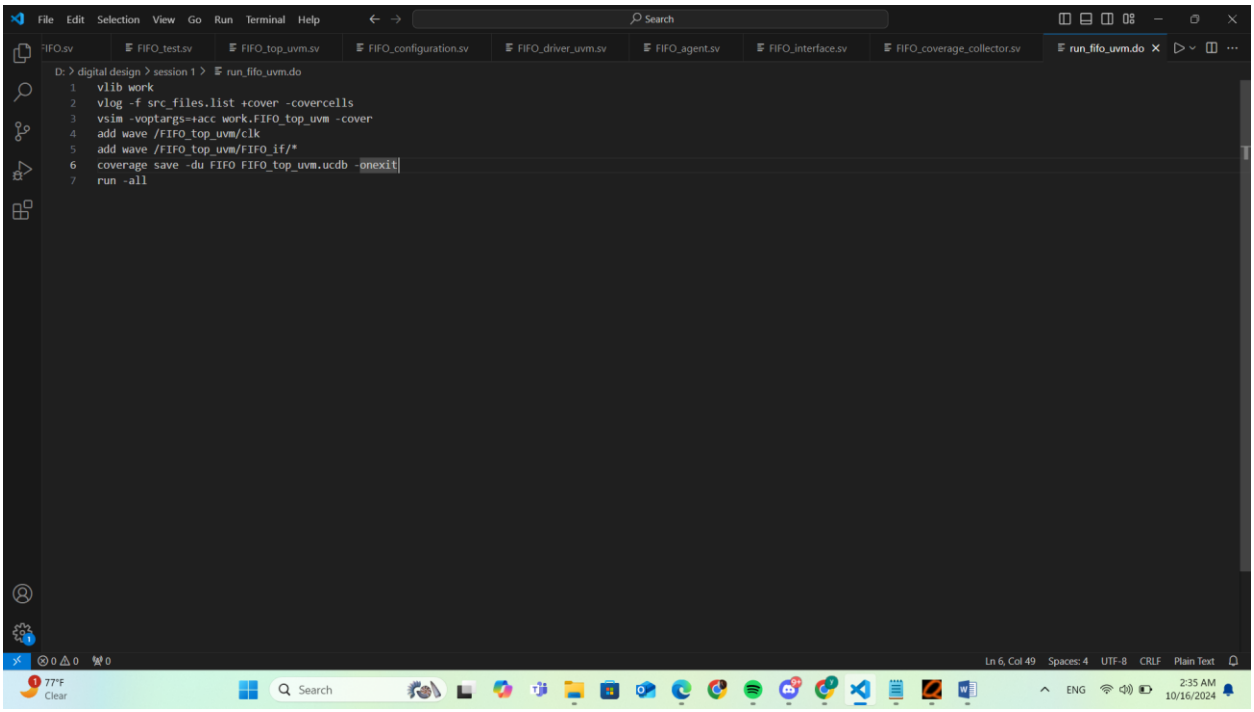
```
FIFO_config cfg;

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        cfg=FIFO_config::type_id::create("cfg",this);
        env=FIFO_env::type_id::create("env" , this);
        rd_seq=FIFO_rd_seq::type_id::create("rd_seq" , this);
        wr_seq=FIFO_wr_seq::type_id::create("wr_seq",this);
        rst_seq=FIFO_rst_seq::type_id::create("rst_seq" , this);
        main_seq=FIFO_main_seq::type_id::create("main_seq" , this);

        if(!uvm_config_db #(virtual FIFO_if)::get(this , "" , "FIFO_if" ,cfg.FIFO_vif))
        `uvm_fatal("build_phase" , "couldnt get interface" );
        uvm_config_db #(FIFO_config)::set(this,"*" , "CFG" , cfg);

    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
        rst_seq.start(env.agt.seq);


        wr_seq.start(env.agt.seq);

        rd_seq.start(env.agt.seq);


        main_seq.start(env.agt.seq);

        rd_seq.start(env.agt.seq);

        phase.drop_objection(this);


    endtask //
endclass //FIFO_test extends uvm_test

endpackage
```

FIFO_top

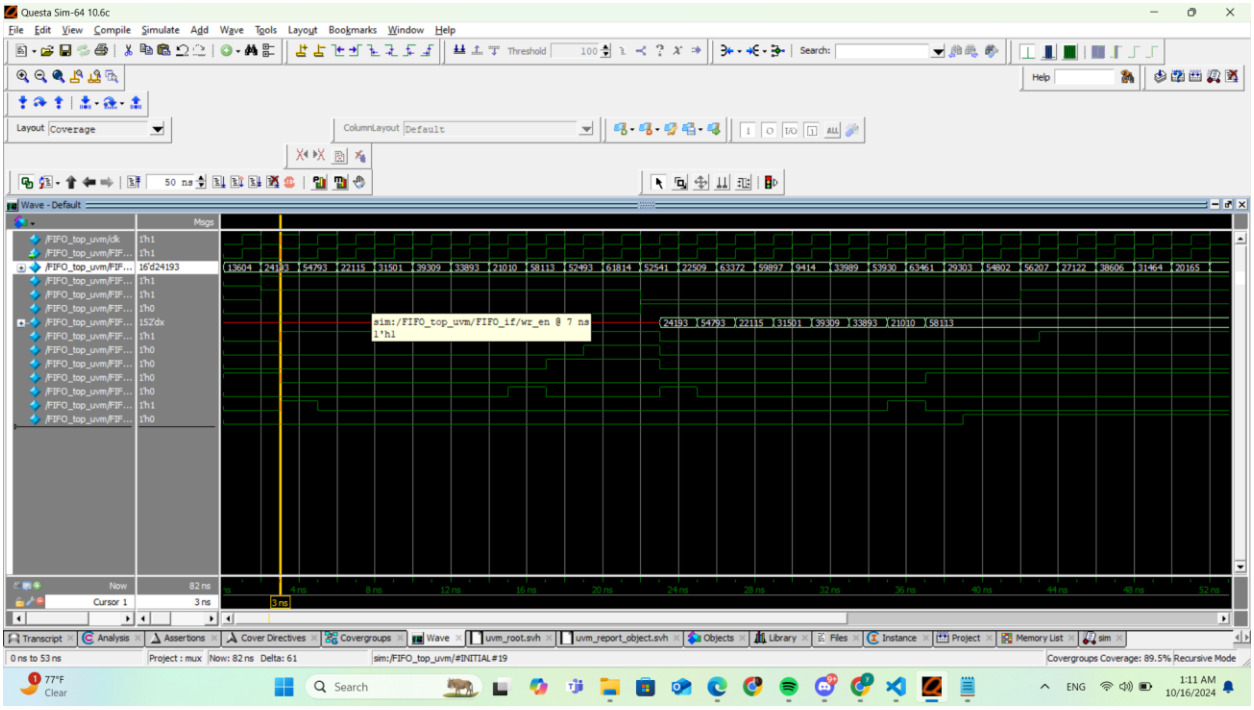


```
import uvm_pkg::*;
import FIFO_config_package::*;
import FIFO_test_package::*;
module FIFO_top_uvm (
);
bit clk;

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

FIFO_if FIFO_if(clk);

FIFO f1(FIFO_if);

initial begin
    uvm_config_db #(virtual FIFO_if)::set(null , "uvm_test_top" , "FIFO_if", FIFO_if );
    run_test("FIFO_test");
end

endmodule
```

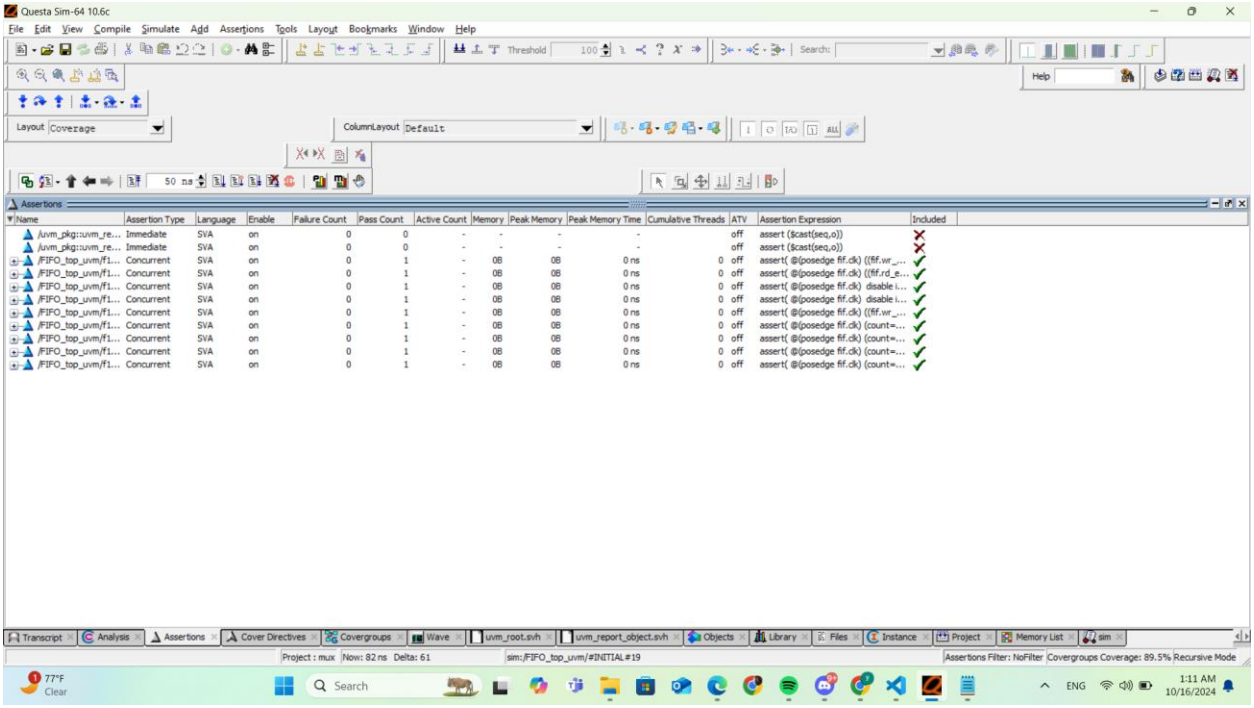
Do_file



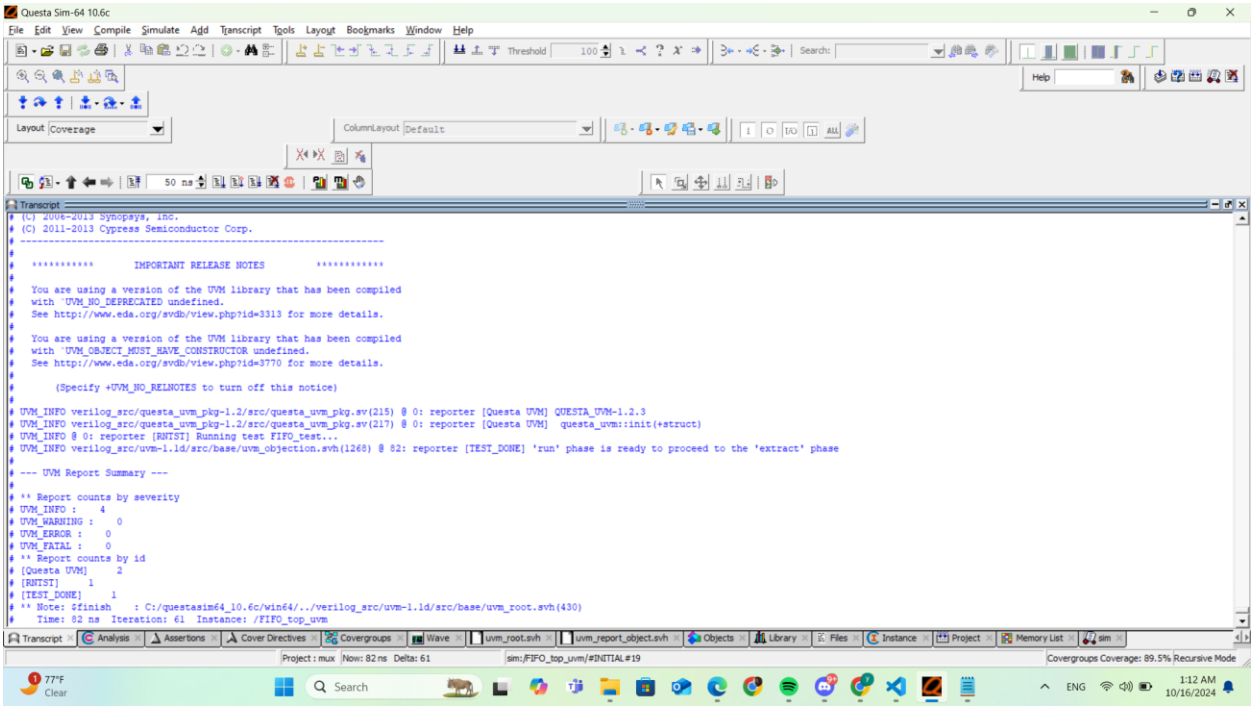
Simulation



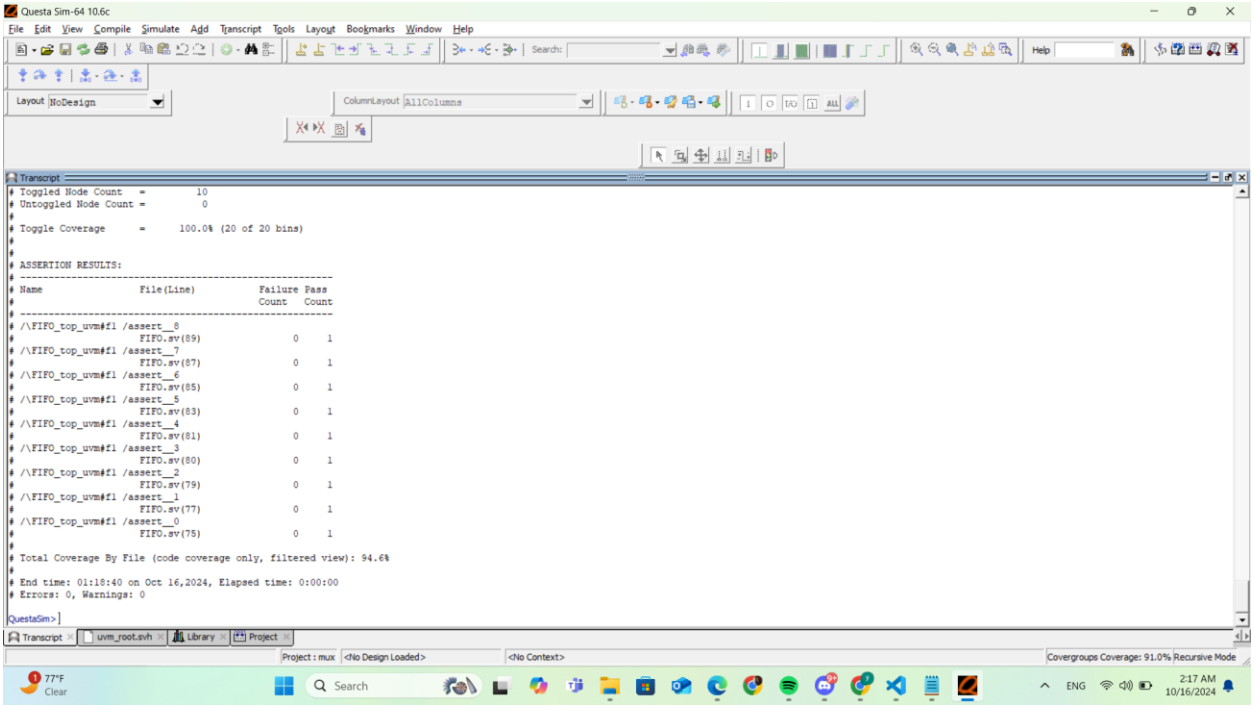
Assertions

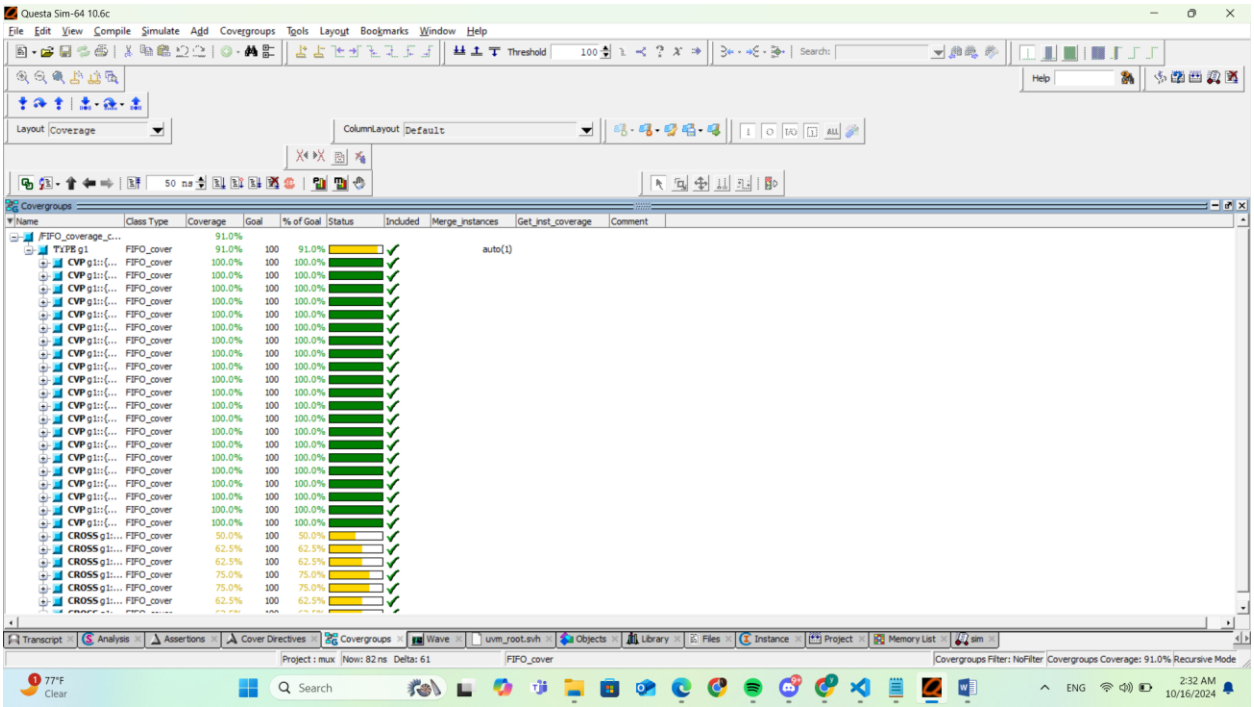


Transcript

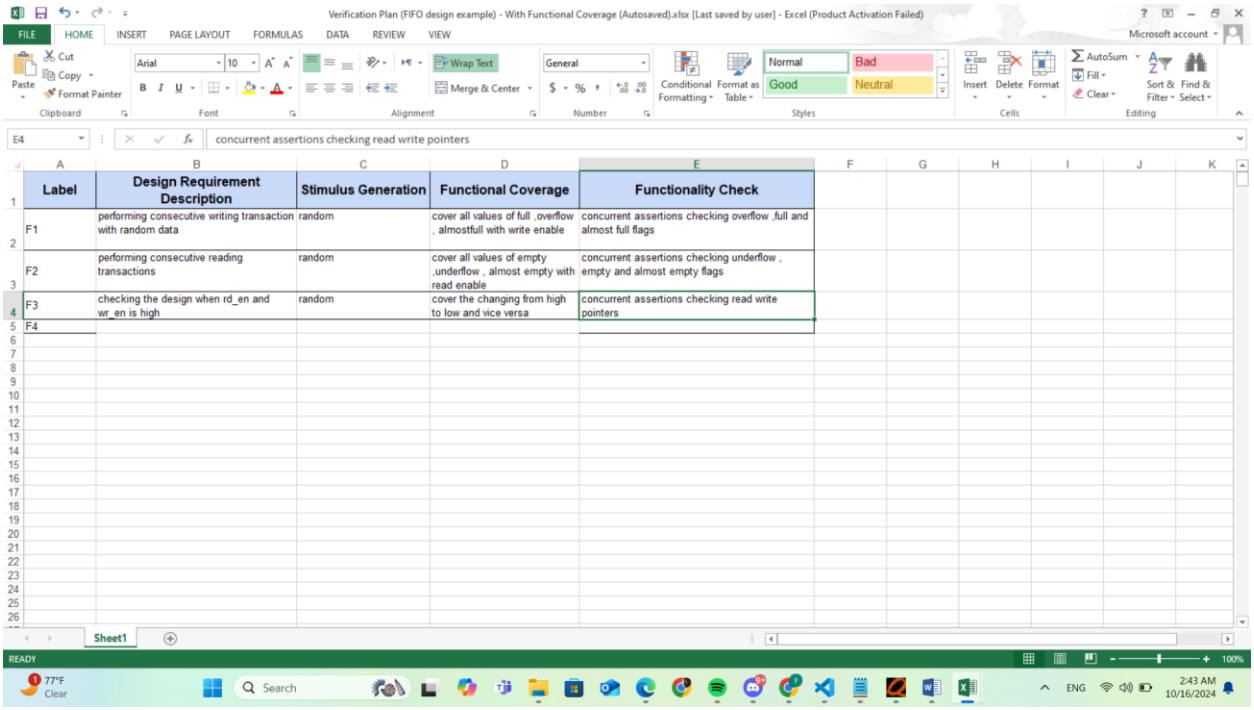


coverage


Questa Sim-64 10.6c
File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help
Transcript
Toggled Node Count = 10
Untoggled Node Count = 0
Toggle Coverage = 100.0% (20 of 20 bins)
ASSERTION RESULTS:
Name File(Line) Failure Count Pass Count
/\FIFO_top_uvm#fl /assert__8 FIFO.sv(89) 0 1
/\FIFO_top_uvm#fl /assert__7 FIFO.sv(87) 0 1
/\FIFO_top_uvm#fl /assert__6 FIFO.sv(85) 0 1
/\FIFO_top_uvm#fl /assert__5 FIFO.sv(83) 0 1
/\FIFO_top_uvm#fl /assert__4 FIFO.sv(81) 0 1
/\FIFO_top_uvm#fl /assert__3 FIFO.sv(80) 0 1
/\FIFO_top_uvm#fl /assert__2 FIFO.sv(79) 0 1
/\FIFO_top_uvm#fl /assert__1 FIFO.sv(77) 0 1
/\FIFO_top_uvm#fl /assert__0 FIFO.sv(75) 0 1
Total Coverage By File (code coverage only, filtered view): 94.6%
End time: 01:18:40 on Oct 16,2024, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
QuestaSim>
Project : mux
<No Design Loaded>
<No Context>
Covergroups Coverage: 91.0% Recursive Mode
77°F Clear
Search
ENG
2:17 AM 10/16/2024


Questa Sim-64 10.6c
File Edit View Compile Simulate Add Covergroups Tools Layout Bookmarks Window Help
Covergroups
Name Class Type Coverage Goal % of Goal Status Included Merge_instances Get_inst_coverage Comment
/FIFO_coverage_c... 91.0%
TYPE g1 FIFO_cover 91.0% 100 91.0% auto(1)
CVP g1::{... FIFO_cover 100.0% 100 100.0%
CROSS g1::... FIFO_cover 50.0% 100 50.0%
CROSS g1::... FIFO_cover 62.5% 100 62.5%
CROSS g1::... FIFO_cover 75.0% 100 75.0%
Project : mux
Now: 82 ns
Delta: 61
FIFO_cover
Covergroups Filter: NoFilter
Covergroups Coverage: 91.0% Recursive Mode
77°F Clear
Search
ENG
2:32 AM 10/16/2024

Verification Plan



Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
F1	performing consecutive writing transaction with random data	random	cover all values of full ,overflow , almostfull with write enable	concurrent assertions checking overflow ,full and almost full flags
F2	performing consecutive reading transactions	random	cover all values of empty ,underflow , almost empty with read enable	concurrent assertions checking underflow , empty and almost empty flags
F3	checking the design when rd_en and wr_en is high	random	cover the changing from high to low and vice versa	concurrent assertions checking read write pointers
F4				

Feature	assertion
Whenever the FIFO is full and there is a write operation the overflow flag becomes high	`assert property(@(posedge fif.clk)(fif.wr_en && fif.full |=> fif.overflow));`
Whenever the FIFO is empty and there is read operation the underflow flag becomes high	`assert property(@(posedge fif.clk)(fif.rd_en &&fif.empty |=>fif.underflow));`
Whenever the FIFO almostfull flag is high and there is write operation the full flag becomes high	`assert property (@(posedge fif.clk) disable iff(!fif.rst_n) (fif.wr_en && fif.almostfull && !fif.rd_en) |=> fif.full);`
Whenever the almost empty flag is high and there is a read operation the empty flag becomes high	`assert property (@(posedge fif.clk) disable iff(!fif.rst_n) (fif.almostempty && fif.rd_en && !fif.wr_en) |=> fif.empty);`
Whenever there is a write operation and the FIFO is not full the wr_ack becomes high	`assert property(@(posedge fif.clk)((fif.wr_en && !fif.full) |=> fif.wr_ack));`
Whenever the count is maximum the FIFO is full	`assert property(@(posedge fif.clk)(count==FIFO_DEPTH) |-> fif.full);`
Whenever the count is 0 the FIFO is empty	`assert property(@(posedge fif.clk)(count==0) |-> fif.empty);`
Whenever the count is 1 the FIFO is almostempty	`assert property(@(posedge fif.clk)(count==1) |-> fif.almostempty);`
Whenever the count is before the maximum the almostfull flag becomes high	`assert property(@(posedge fif.clk)(count==FIFO_DEPTH-1) |-> fif.almostfull);`