Project

Class-Based Verification for a Synchronous FIFO USING UVM

By: Mohamed Sayed Mohamed Soliman

Contents

1. Introduction to the Synchronous FIFO

A Synchronous FIFO (First-In-First-Out) : is a fundamental hardware component used in digital systems for data buffering and transferring between different subsystems. It ensures that data is read in the same order as it was written, and both write and read operations are synchronized to a common clock. This makes it particularly useful for applications where data flow needs to be managed in a controlled and sequential manner, often between two devices or subsystems operating at different speeds but synchronized to the same clock.

Key Features of a Synchronous FIFO:

1. **Single Clock Domain**: Both read and write operations are synchronized to the same clock signal, eliminating the need for complex clock domain crossing techniques.
2. **Data Storage**: The FIFO stores data in an internal memory (typically implemented as a register array or RAM). The depth (number of entries) and width (bits per entry) of the FIFO can be parameterized to fit the specific requirements of the application.
3. **Control Flags**:
    - **Full**: Indicates the FIFO is full and cannot accept more write operations until space is freed by reading.
    - **Empty**: Indicates the FIFO is empty and no data is available for reading.
    - **Almost Full/Almost Empty**: Optional flags that provide an early warning before the FIFO becomes completely full or empty.
4. **Read/Write Pointers**: The FIFO internally maintains pointers for tracking the current positions for reading and writing, ensuring data is managed in a circular buffer fashion.

Applications of Synchronous FIFO:

- **Data Rate Matching**: Used between subsystems with different data rates but synchronized clocks (e.g., between a CPU and a memory controller).
- **Buffering**: Temporary storage for bursty data streams or packet data, smoothing out fluctuations in data rates.
- **Pipeline Processing**: Helps in decoupling stages of a digital pipeline, allowing one stage to write data while the next stage reads it at a different rate.

Operational Overview:

- **Write Operation**: Data is written to the FIFO at the location pointed to by the write pointer. After a successful write, the write pointer increments, and the **full** flag is updated if necessary.
- **Read Operation**: Data is read from the FIFO at the location pointed to by the read pointer. The read pointer increments after a read operation, and the **empty** flag is updated accordingly.

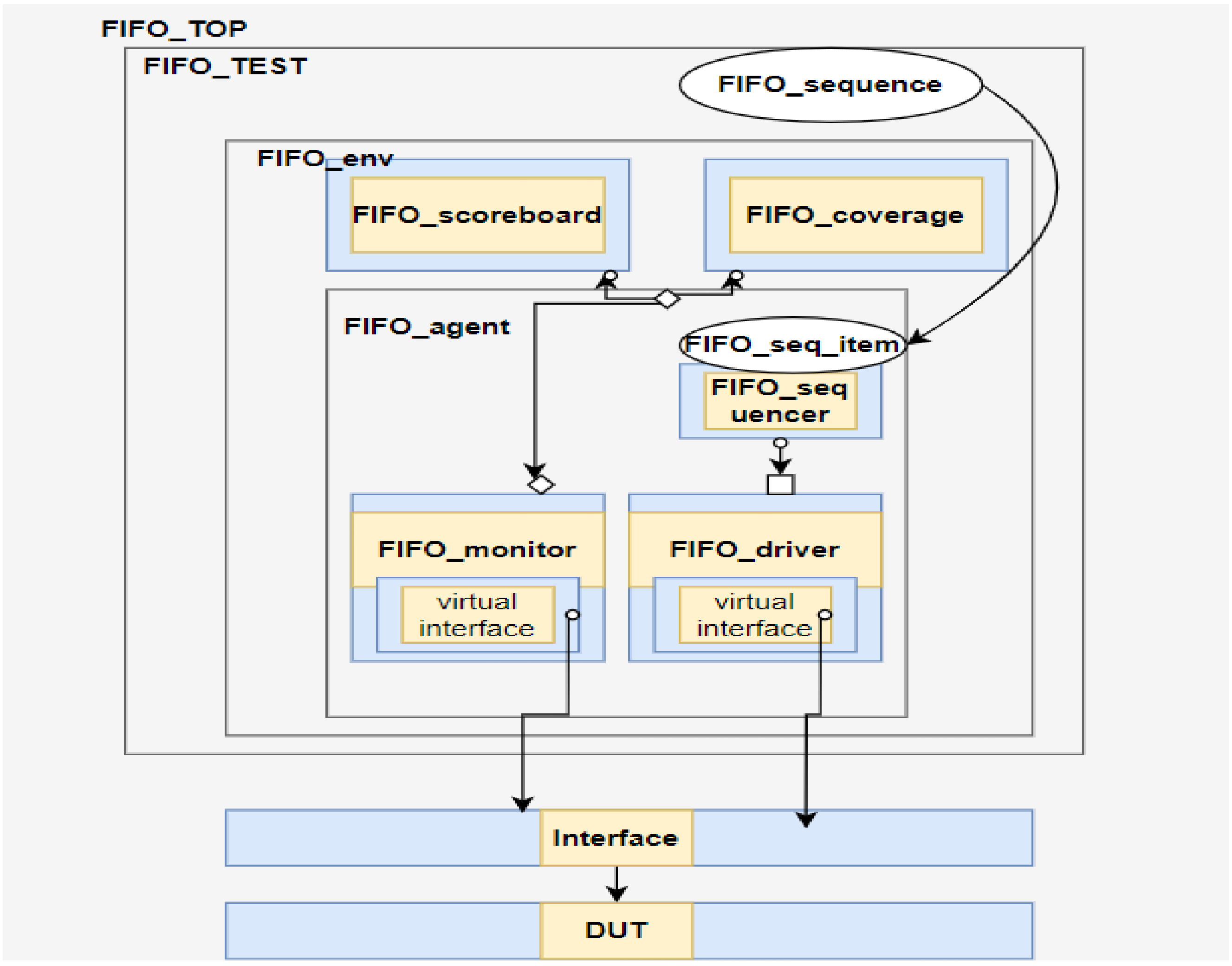
The FIFO must manage these operations without data loss or overwriting, maintaining data integrity across various conditions.

Design Considerations:

- **Depth**: The size of the FIFO (number of entries) is chosen based on the maximum burst length or data buffering requirements of the system.
- **Width**: The data width is parameterized, depending on the size of the data being transferred (e.g., 8-bit, 16-bit, or 32-bit).
- **Latency**: The delay between a write or read request and when the data becomes available is minimal in a well-designed FIFO.

Synchronous FIFOs are widely used in digital designs, including networking hardware, processors, and communication systems, where reliable and ordered data transfer is critical.

2. UVM testbench showing the UVM structure



In the UVM testbench you've shared, the structure of the verification environment for a FIFO (First-In-First-Out) design follows a modular and hierarchical approach. I'll now describe how this UVM testbench works, based on your diagram, starting from the top module to driving the interface, monitoring the outputs, and analyzing the results.

---

1. Top Module (FIFO_TOP and Test Instantiation)

At the highest level, the **FIFO_TOP** module includes:

- **DUT** (Design Under Test): The FIFO module you're verifying.
- **Interface**: The interface connects the DUT to the testbench. This interface contains all the signal connections like the inputs (data, write_enable, read_enable) and outputs (data_out, full, empty).

The interface simplifies passing signals between the DUT and the verification environment.

2. Driving the Interface (FIFO Driver and Sequence)

In this verification flow, input stimulus is generated by the UVM sequence and is driven onto the DUT through the driver.

- **FIFO_seq_item**: This represents the individual transaction (or data item) that is applied to the DUT. For example, a write or read operation could be a transaction in the FIFO sequence item.
- **FIFO_sequence**: This generates sequences of these FIFO_seq_items. The sequence could contain various write and read operations or random patterns to test the FIFO's functionality.
- **FIFO_driver**: The driver receives transactions from the sequencer and converts them into low-level signals that drive the interface of the DUT. The driver ensures that the correct signals (e.g., data_in, write_enable, read_enable) are driven onto the interface pins in synchronization with the clock.
- **Sequencer**: The **FIFO_sequencer** manages the flow of the sequence. It picks the sequence items from the FIFO_sequence and forwards them to the driver. This ensures that the input stimulus is applied in the correct order and at the appropriate times.

3. Monitoring (FIFO Monitor and Virtual Interface)

- **FIFO_monitor**: This component passively monitors the signals on the interface. It captures both the input transactions (e.g., write commands) and output responses (e.g., data read from the FIFO). The monitor converts these signal-level interactions back into transaction-level representations.
- The monitor does not interfere with the signal flow but extracts the necessary information to ensure that the DUT is behaving as expected.
- The monitor also works with the **virtual interface**, which provides the necessary connections to observe the signal activities without physically modifying the interface.

4. Analyzing the Output (Scoreboard and Coverage)

- **FIFO_scoreboard**: This is where the actual checking occurs. The scoreboard compares the monitored transactions against expected results. For example, if a write operation is performed, the scoreboard ensures that the data is stored in the FIFO and can be read out in the correct order. If any mismatches occur (such as reading incorrect data), the scoreboard raises an error.

- **FIFO_coverage**: This component tracks how much of the design has been exercised during the test. Coverage ensures that all functional aspects of the FIFO (like full, empty, overflow, and underflow conditions) have been tested. It records the various operations (write, read, overflow, etc.) and checks whether all scenarios have been covered.

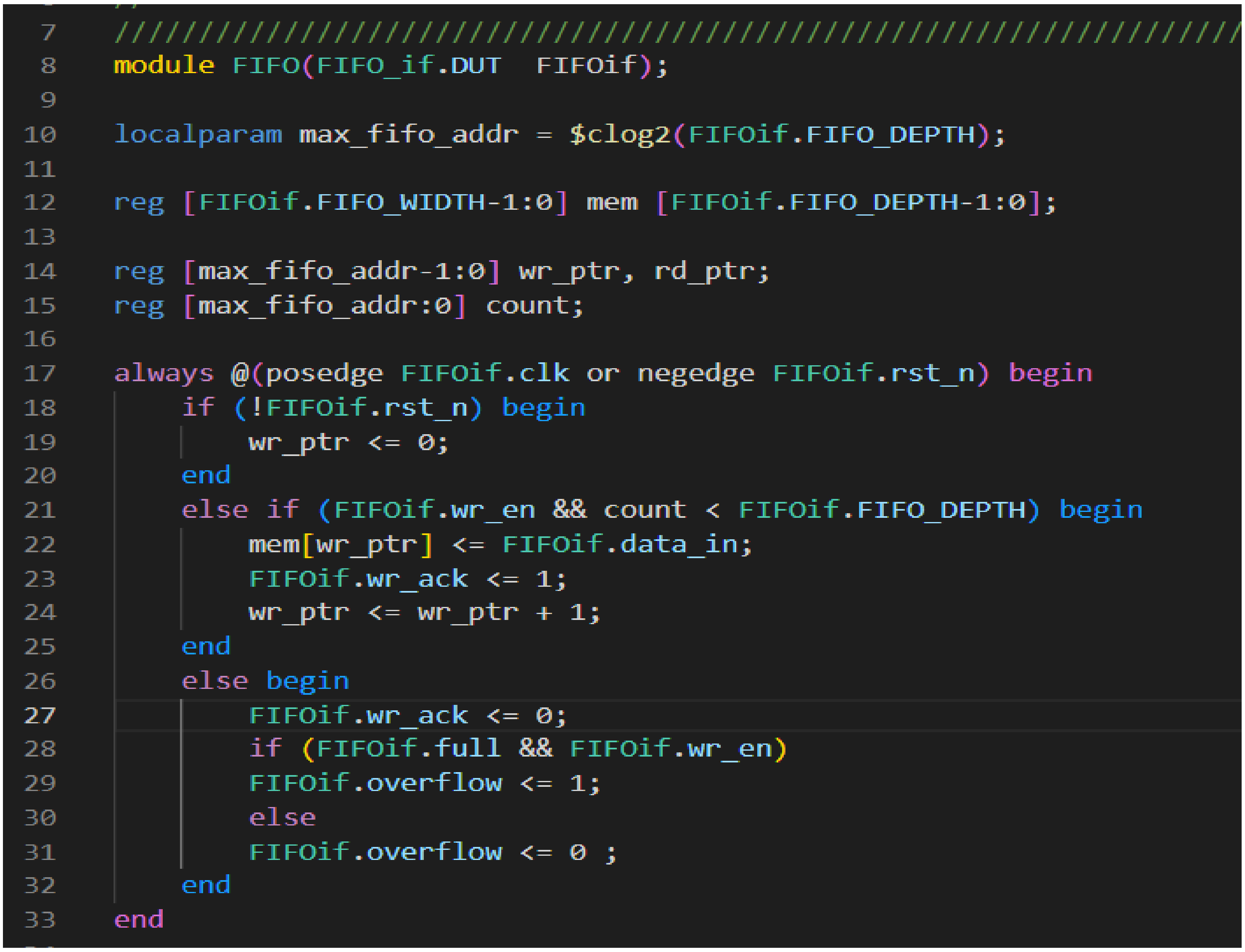
5. Overall Test Flow

- The test starts at the **FIFO_TEST** level, where the testbench environment, including the scoreboard, monitor, driver, and coverage, is instantiated.
- The sequencer triggers the sequences, which generate the transactions for the driver to send to the DUT.
- The monitor observes the interactions and forwards them to the scoreboard for checking.
- Coverage collects statistics on which parts of the design have been exercised.

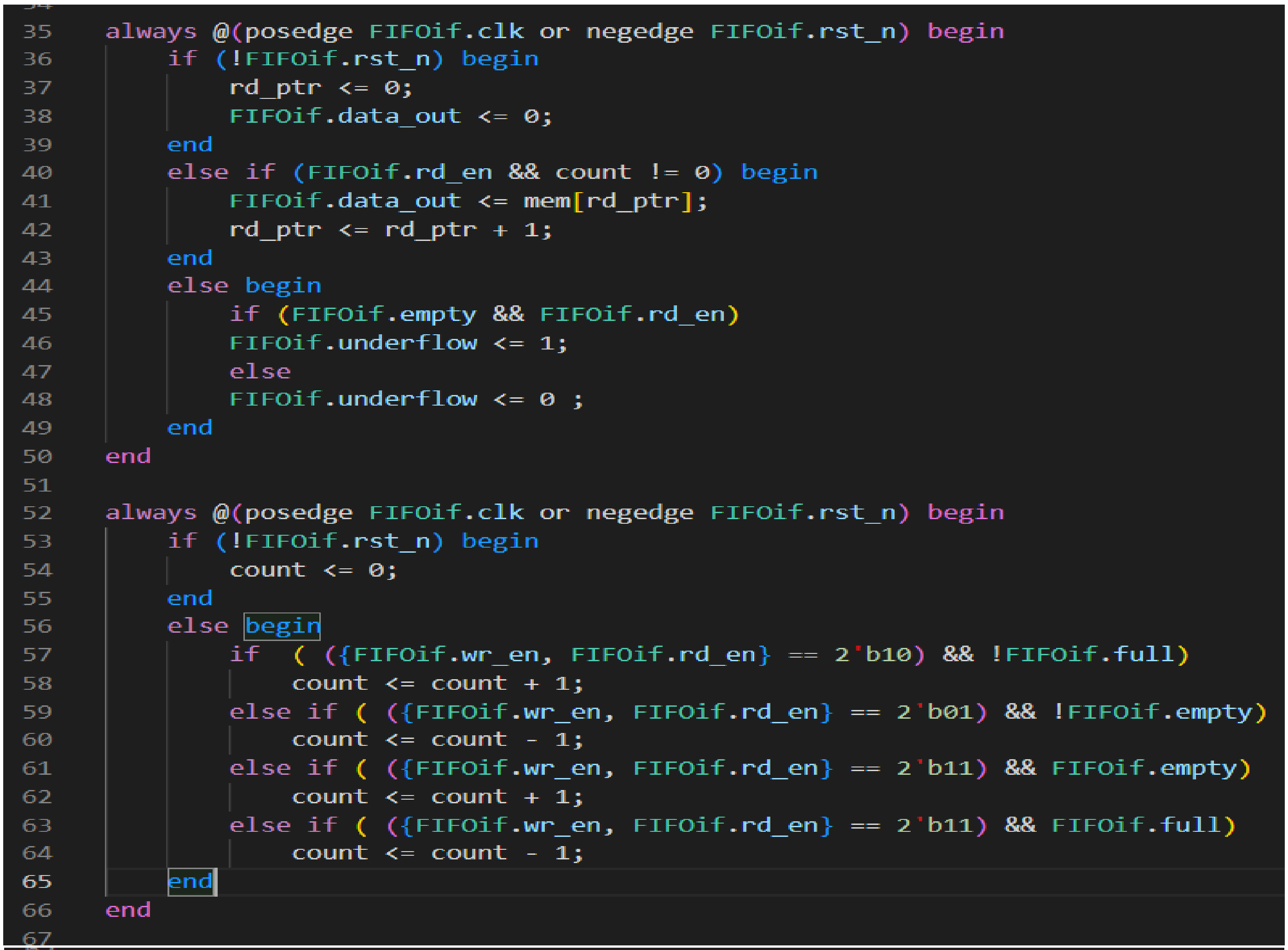
Conclusion

This UVM testbench follows a structured approach to verifying the FIFO design by modularizing the components for stimulus generation, driving the interface, monitoring signals, and analyzing the output. The modular structure makes the testbench reusable, configurable, and scalable for different scenarios and complexities in design verification.

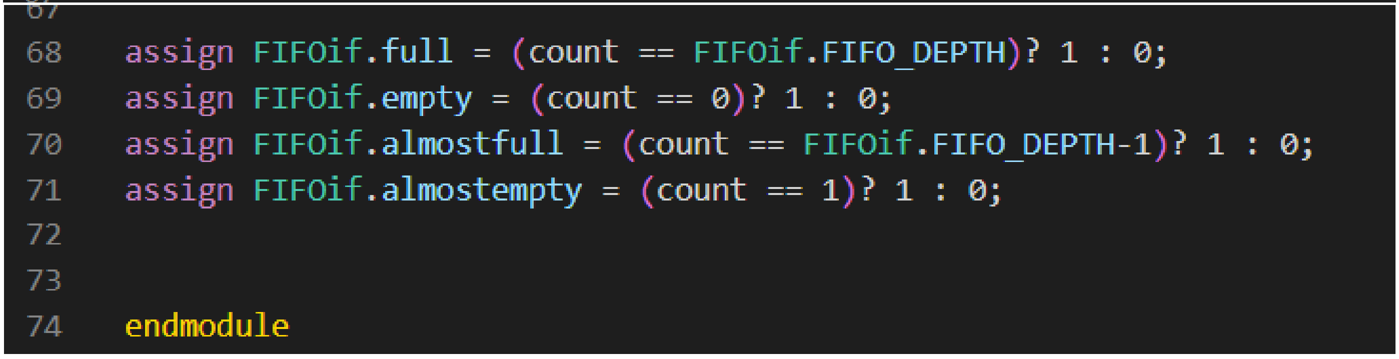
3. Design RTL code



```
/////////////////////////////////////////////////////////////////////
module FIFO(FIFO_if.DUT  FIFOif);

localparam max_fifo_addr = $clog2(FIFOif.FIFO_DEPTH);

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

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

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

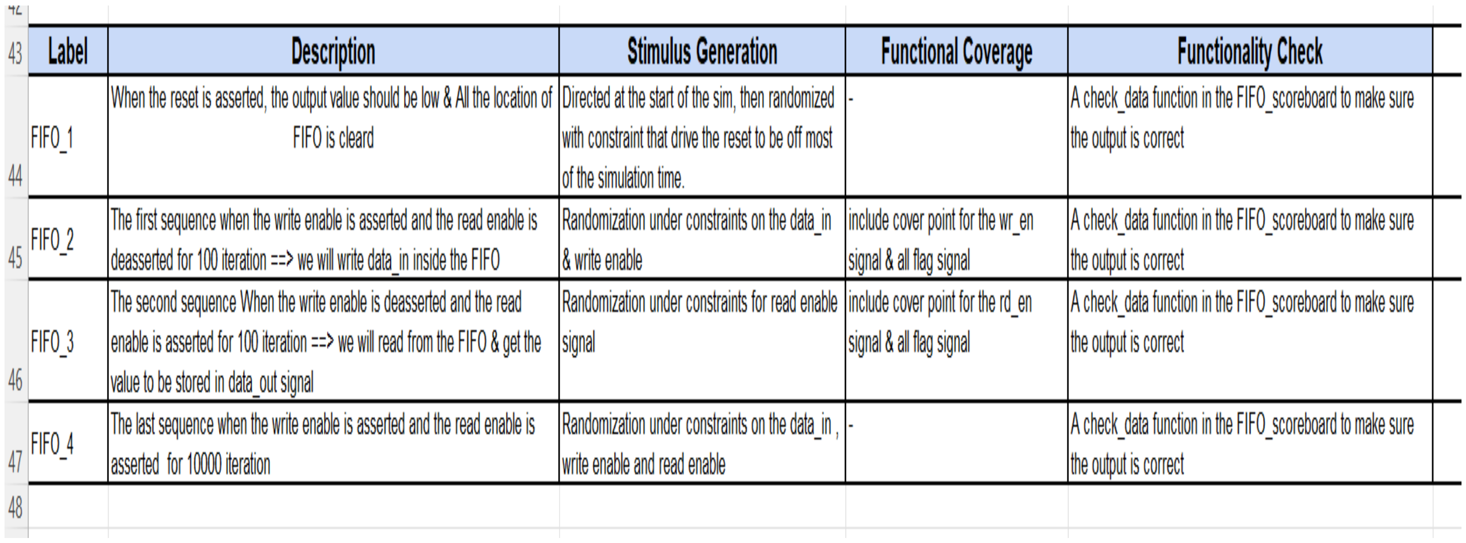


```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        rd_ptr <= 0;
        FIFOif.data_out <= 0;
    end
    else if (FIFOif.rd_en && count != 0) begin
        FIFOif.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else begin
        if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

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



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


endmodule
```

4. Verification plan



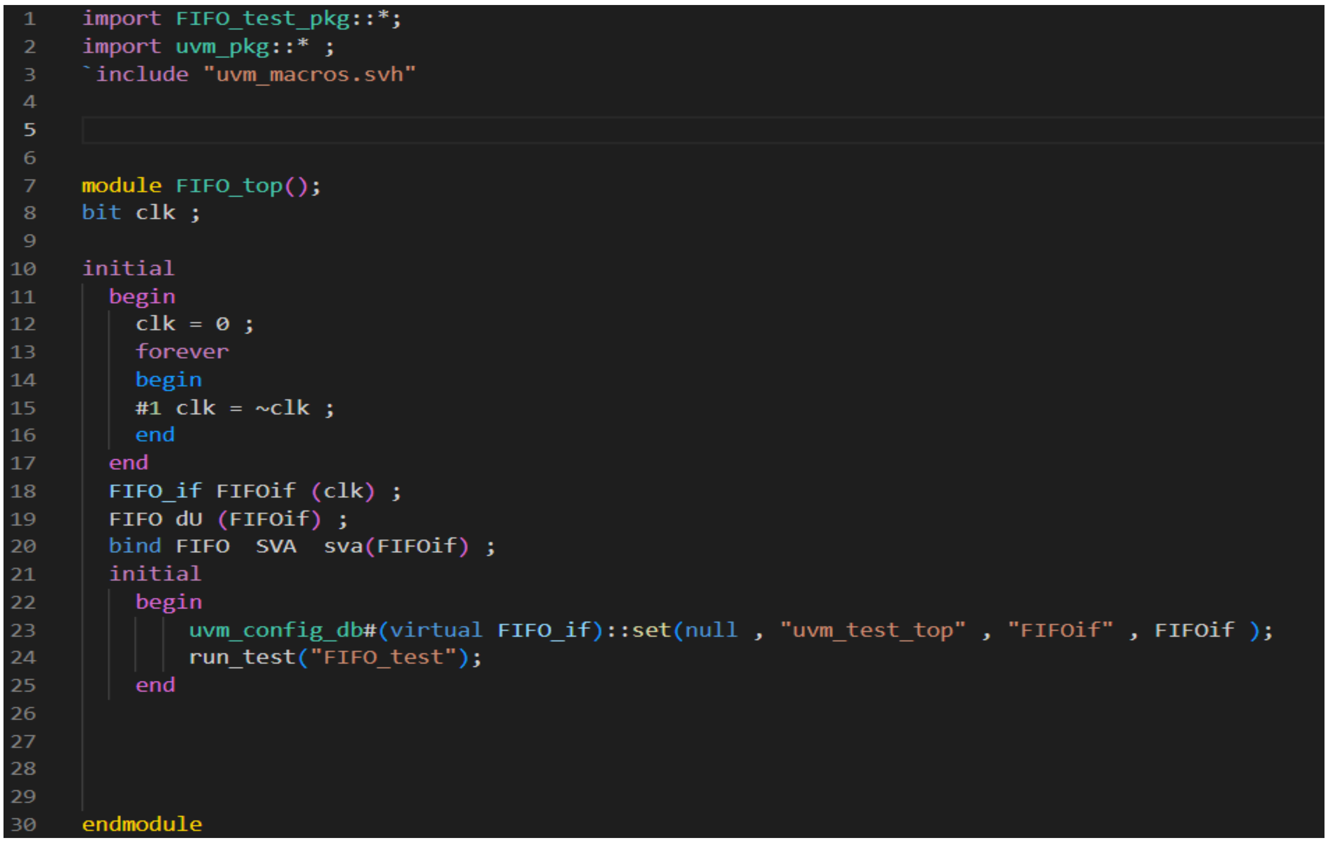
Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, the output value should be low & All the location of FIFO is cleard	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the simulation time.	-	A check_data function in the FIFO_scoreboard to make sure the output is correct
FIFO_2	The first sequence when the write enable is asserted and the read enable is deasserted for 100 iteration ==> we will write data_in inside the FIFO	Randomization under constraints on the data_in & write enable	include cover point for the wr_en signal & all flag signal	A check_data function in the FIFO_scoreboard to make sure the output is correct
FIFO_3	The second sequence When the write enable is deasserted and the read enable is asserted for 100 iteration ==> we will read from the FIFO & get the value to be stored in data_out signal	Randomization under constraints for read enable signal	include cover point for the rd_en signal & all flag signal	A check_data function in the FIFO_scoreboard to make sure the output is correct
FIFO_4	The last sequence when the write enable is asserted and the read enable is asserted for 10000 iteration	Randomization under constraints on the data_in , write enable and read enable	-	A check_data function in the FIFO_scoreboard to make sure the output is correct

5. Table to the assertion

Feature	Assertion
Whenever the FIFO is not full and wr_en is high, wr_ack is high	@(posedge clk) (wr_en && count < FIFO_DEPTH) |=> wr_ack
Whenever the FIFO is full and wr_en is high, overflow signal is asserted	@(posedge FIFOif.clk) (FIFOif.full && FIFOif.wr_en) |=> overflow
When count == 0, FIFO is empty, not full, almostempty and almostfull signals are also cleared	assert (empty && !full && !almostempty && !almostfull)
When count == 1, FIFO is not empty, not full, almost empty is set, and almost full is clear	assert (!empty && !full && almostempty && !almostfull)
When count == FIFO_DEPTH - 1, FIFO is not empty or full, but almost full is asserted	assert (!empty && !full && !almostempty && almostfull)
When count == FIFO_DEPTH, FIFO is full, not empty, and almost signals are clear	assert (!empty && full && !almostempty && !almostfull)
Whenever FIFO is empty and rd_en is high, underflow signal is asserted	@(posedge clk) (empty && rd_en) |=> underflow
When wr_en is high , rd_en is low and FIFO is not full , count increments by 1	@(posedge clk)(wr_en&&!rd_en&& !full) |=>(count === ($past(count) + 1)
When rd_en is high ,wr_en is low and FIFO is not empty,count decrements by 1	@(posedge clk)(!wr_en&&rd_en&& !empty) |=>(count === ($past(count) - 1)
When both wr_en and rd_en are high and FIFO is empty, count increments by 1	@(posedge clk)(wr_en&&rd_en&& empty) |=>(count === ($past(count) + 1)
When both wr_en and rd_en are high and FIFO is full, count decrements by 1	@(posedge clk)(wr_en&&rd_en&& full) |=>(count === ($past(count) - 1)

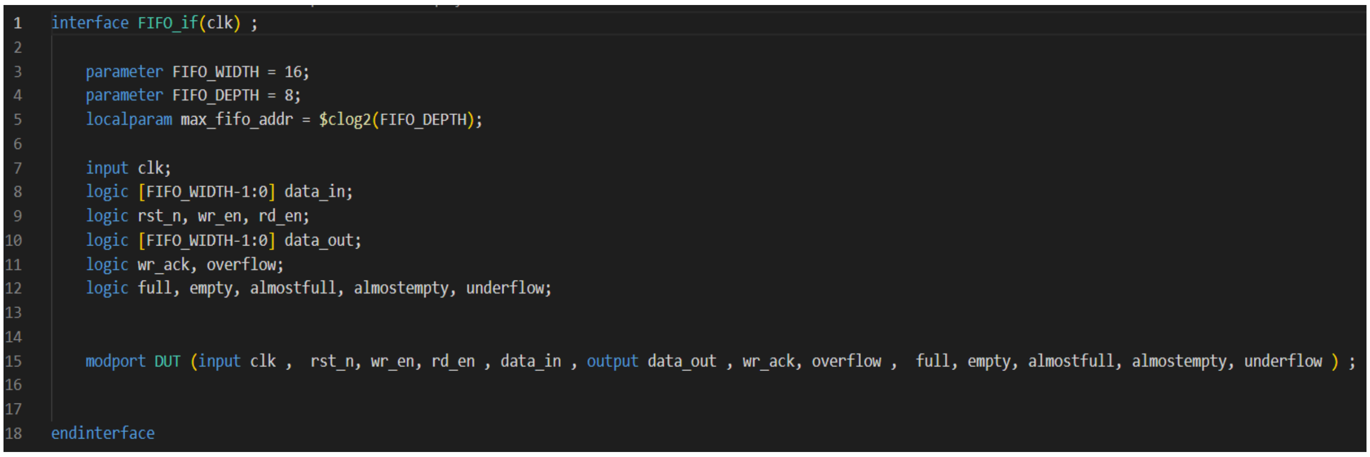
6. Verification code

6.1. top module



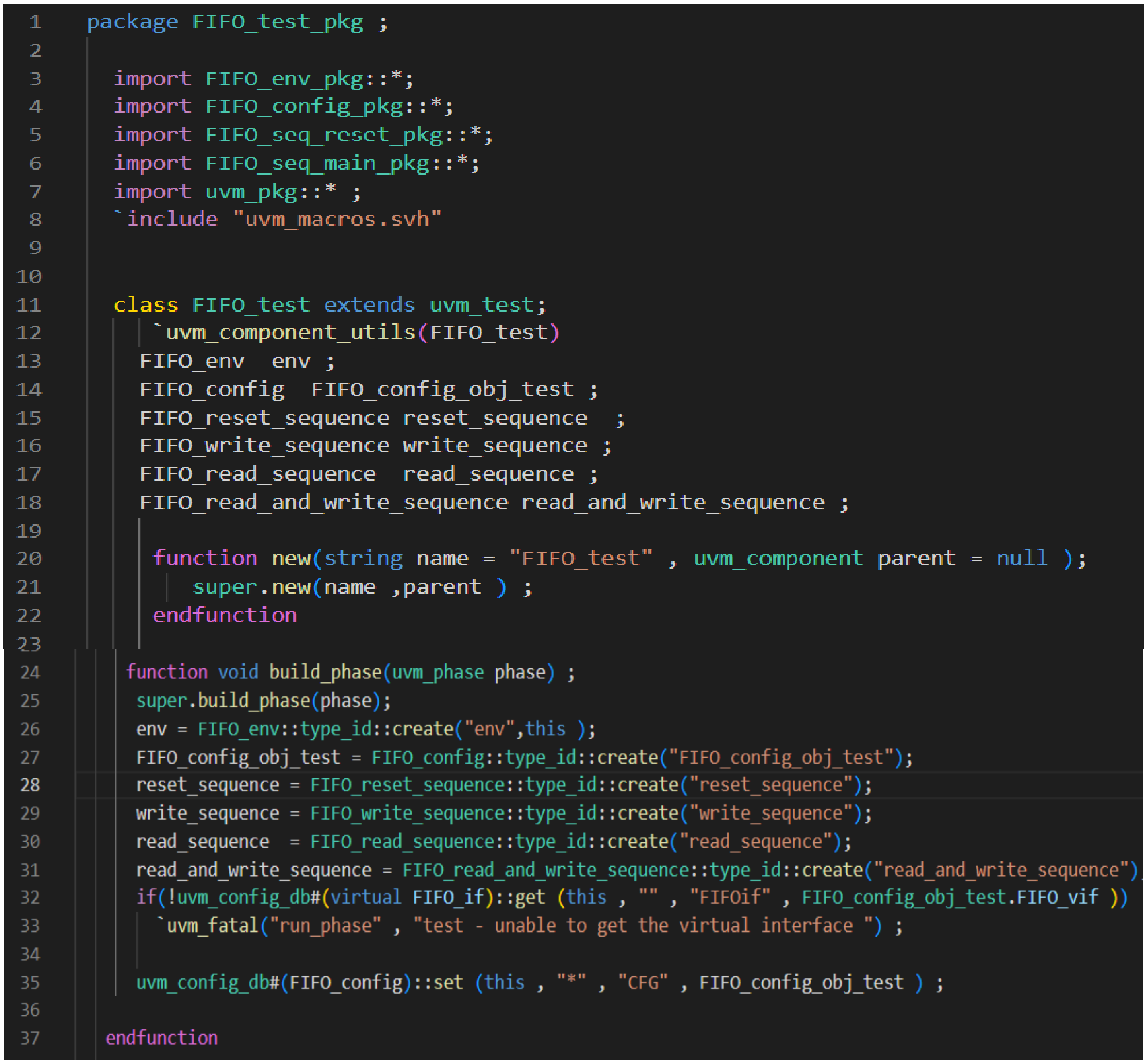
```
import FIFO_test_pkg::*;
import uvm_pkg::* ;
`include "uvm_macros.svh"



module FIFO_top();
bit clk ;

initial
  begin
    clk = 0 ;
    forever
    begin
    #1 clk = ~clk ;
    end
  end
  FIFO_if FIFOif (clk) ;
  FIFO dU (FIFOif) ;
  bind FIFO  SVA  sva(FIFOif) ;
  initial
    begin
        uvm_config_db#(virtual FIFO_if)::set(null , "uvm_test_top" , "FIFOif" , FIFOif );
        run_test("FIFO_test");
    end




endmodule
```

6.2. interface

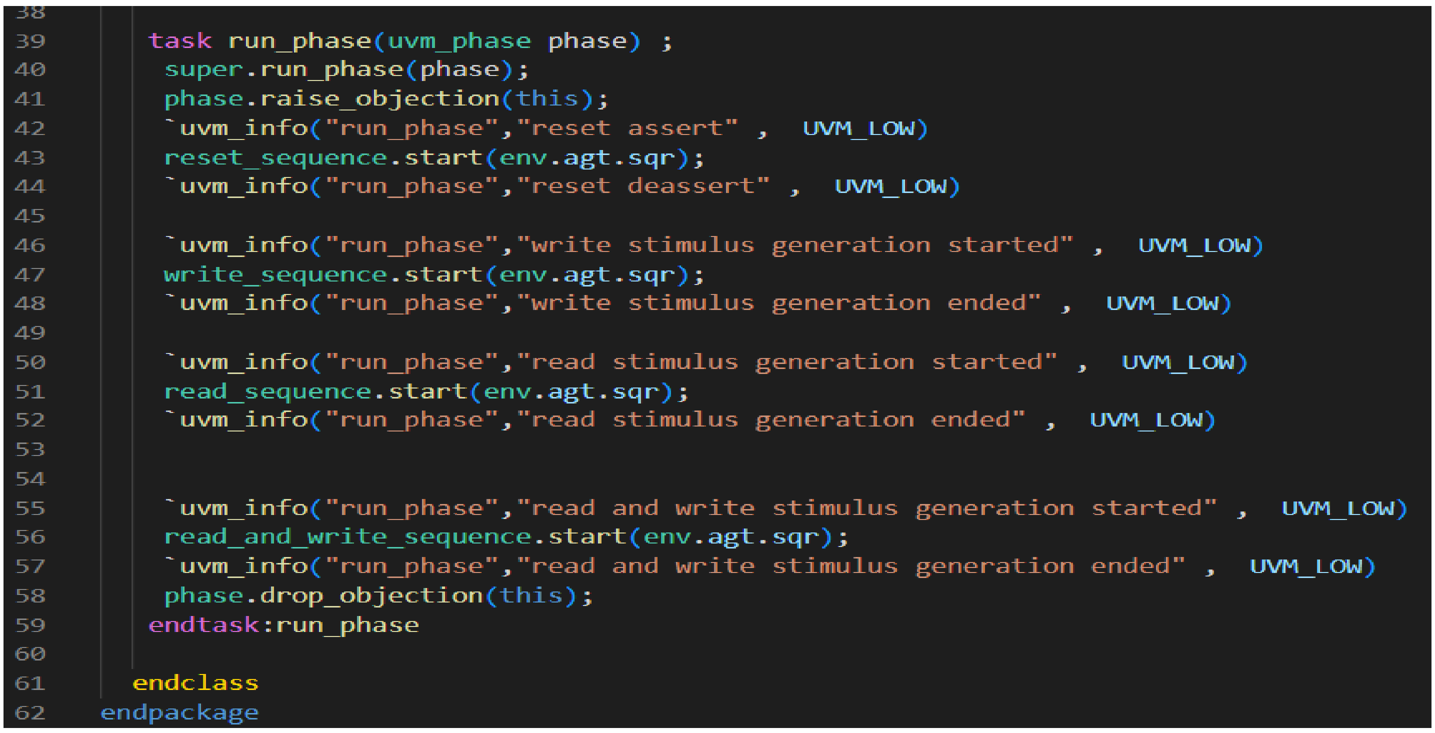


```
interface FIFO_if(clk) ;

    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    localparam max_fifo_addr = $clog2(FIFO_DEPTH);

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


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


endinterface
```

6.3. FIFO test

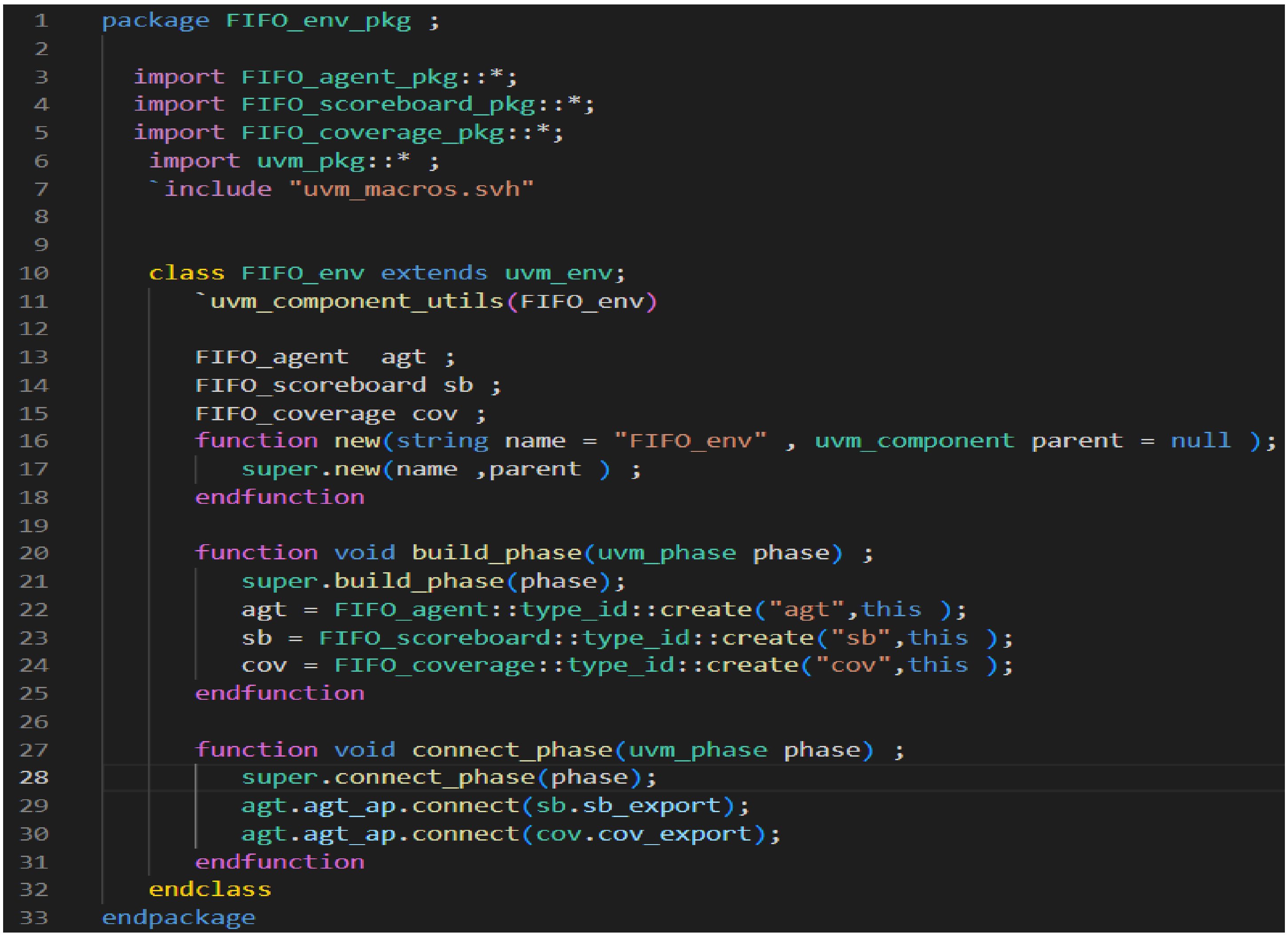


```
package FIFO_test_pkg ;

  import FIFO_env_pkg::*;
  import FIFO_config_pkg::*;
  import FIFO_seq_reset_pkg::*;
  import FIFO_seq_main_pkg::*;
  import uvm_pkg::* ;
  `include "uvm_macros.svh"


  class FIFO_test extends uvm_test;
     `uvm_component_utils(FIFO_test)
    FIFO_env  env ;
    FIFO_config  FIFO_config_obj_test ;
    FIFO_reset_sequence reset_sequence  ;
    FIFO_write_sequence write_sequence ;
    FIFO_read_sequence  read_sequence ;
    FIFO_read_and_write_sequence read_and_write_sequence ;

     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);
     env = FIFO_env::type_id::create("env",this );
     FIFO_config_obj_test = FIFO_config::type_id::create("FIFO_config_obj_test");
     reset_sequence = FIFO_reset_sequence::type_id::create("reset_sequence");
     write_sequence = FIFO_write_sequence::type_id::create("write_sequence");
     read_sequence  = FIFO_read_sequence::type_id::create("read_sequence");
     read_and_write_sequence = FIFO_read_and_write_sequence::type_id::create("read_and_write_sequence")
     if(!uvm_config_db#(virtual FIFO_if)::get (this , "" , "FIFOif" , FIFO_config_obj_test.FIFO_vif ))
       `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;

     uvm_config_db#(FIFO_config)::set (this , "*" , "CFG" , FIFO_config_obj_test ) ;

  endfunction
```



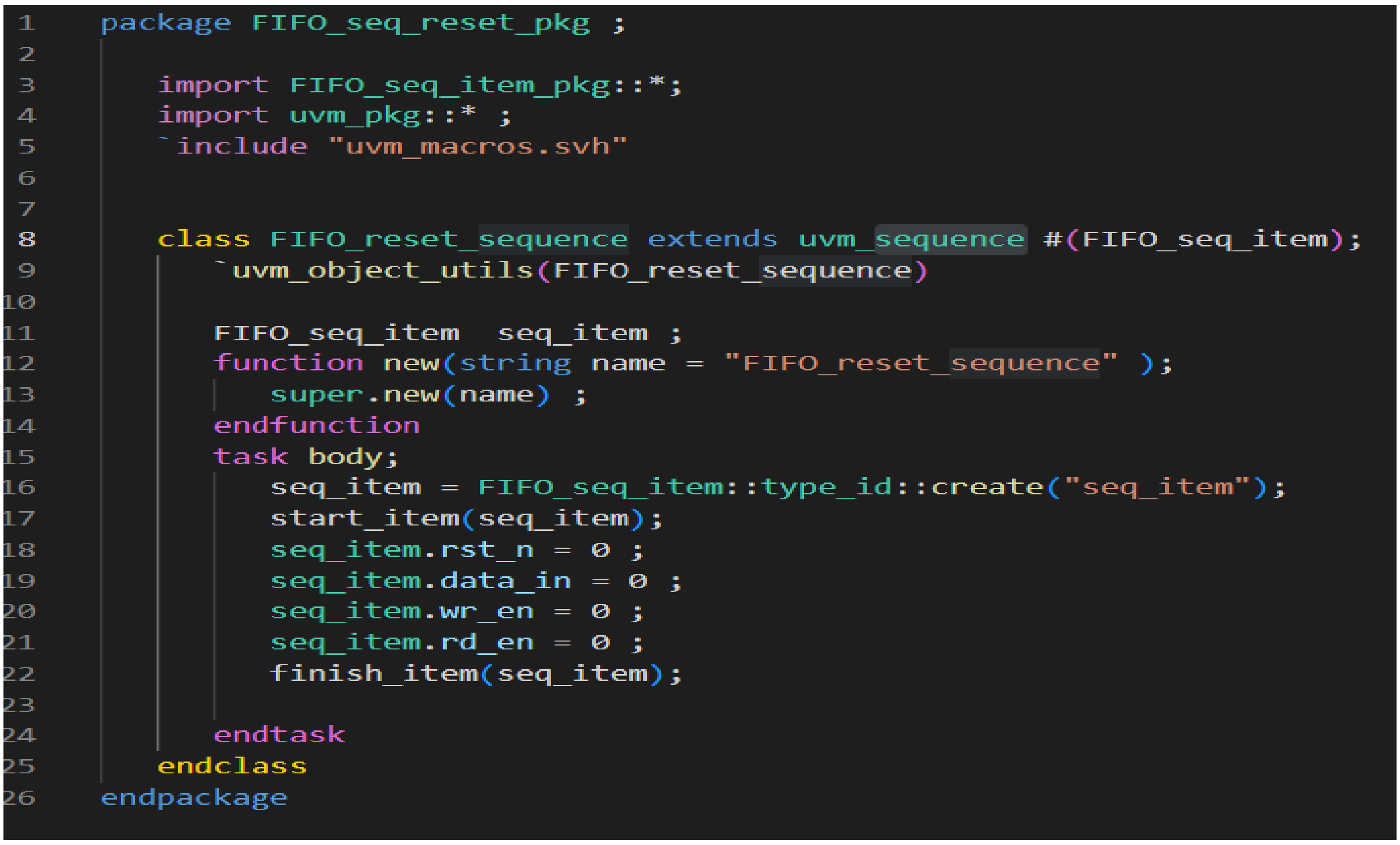
```
    task run_phase(uvm_phase phase) ;
     super.run_phase(phase);
     phase.raise_objection(this);
     `uvm_info("run_phase","reset assert" ,  UVM_LOW)
     reset_sequence.start(env.agt.sqr);
     `uvm_info("run_phase","reset deassert" ,  UVM_LOW)

     `uvm_info("run_phase","write stimulus generation started" ,  UVM_LOW)
     write_sequence.start(env.agt.sqr);
     `uvm_info("run_phase","write stimulus generation ended" ,  UVM_LOW)

     `uvm_info("run_phase","read stimulus generation started" ,  UVM_LOW)
     read_sequence.start(env.agt.sqr);
     `uvm_info("run_phase","read stimulus generation ended" ,  UVM_LOW)


     `uvm_info("run_phase","read and write stimulus generation started" ,  UVM_LOW)
     read_and_write_sequence.start(env.agt.sqr);
     `uvm_info("run_phase","read and write stimulus generation ended" ,  UVM_LOW)
     phase.drop_objection(this);
    endtask:run_phase

  endclass
endpackage
```

6.4. FIFO env



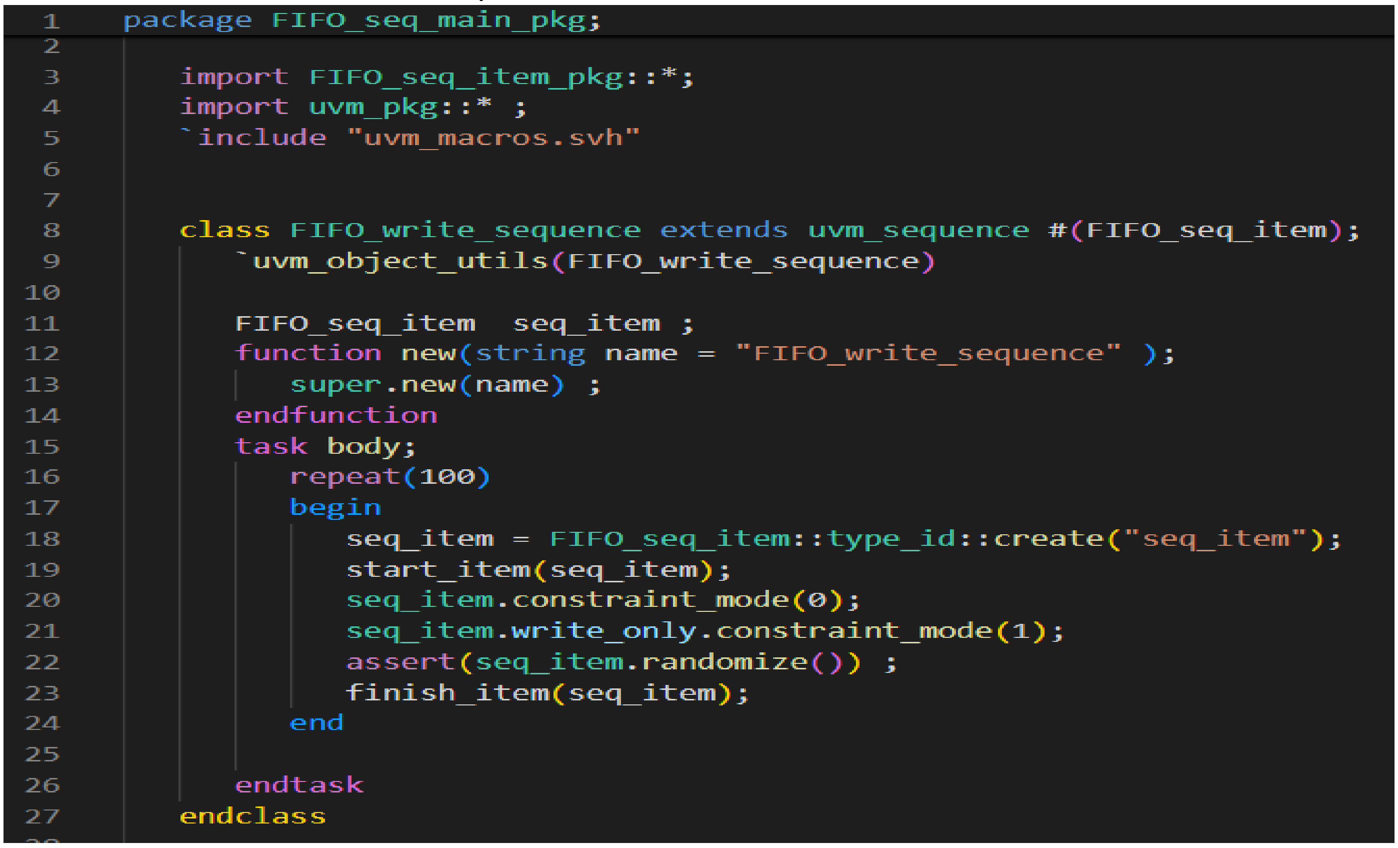
```
package FIFO_env_pkg ;

  import FIFO_agent_pkg::*;
  import FIFO_scoreboard_pkg::*;
  import FIFO_coverage_pkg::*;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"


   class FIFO_env extends uvm_env;
      `uvm_component_utils(FIFO_env)

      FIFO_agent  agt ;
      FIFO_scoreboard sb ;
      FIFO_coverage cov ;
      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);
         agt = FIFO_agent::type_id::create("agt",this );
         sb = FIFO_scoreboard::type_id::create("sb",this );
         cov = FIFO_coverage::type_id::create("cov",this );
      endfunction

      function void connect_phase(uvm_phase phase) ;
         super.connect_phase(phase);
         agt.agt_ap.connect(sb.sb_export);
         agt.agt_ap.connect(cov.cov_export);
      endfunction
   endclass
endpackage
```

6.5. FIFO reset sequence

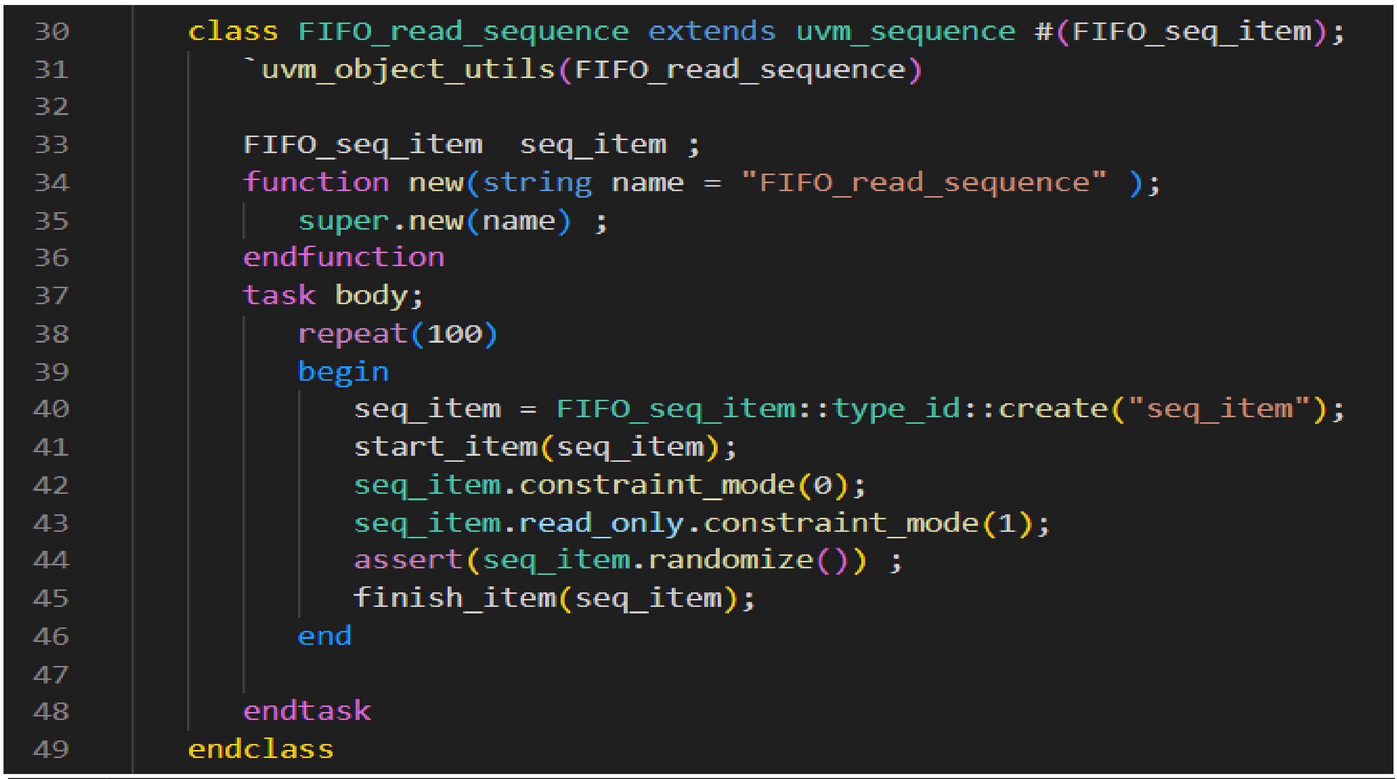


```
package FIFO_seq_reset_pkg ;

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


    class FIFO_reset_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_reset_sequence)

        FIFO_seq_item  seq_item ;
        function new(string name = "FIFO_reset_sequence" );
            super.new(name) ;
        endfunction
        task body;
            seq_item = FIFO_seq_item::type_id::create("seq_item");
            start_item(seq_item);
            seq_item.rst_n = 0 ;
            seq_item.data_in = 0 ;
            seq_item.wr_en = 0 ;
            seq_item.rd_en = 0 ;
            finish_item(seq_item);

        endtask
    endclass
endpackage
```

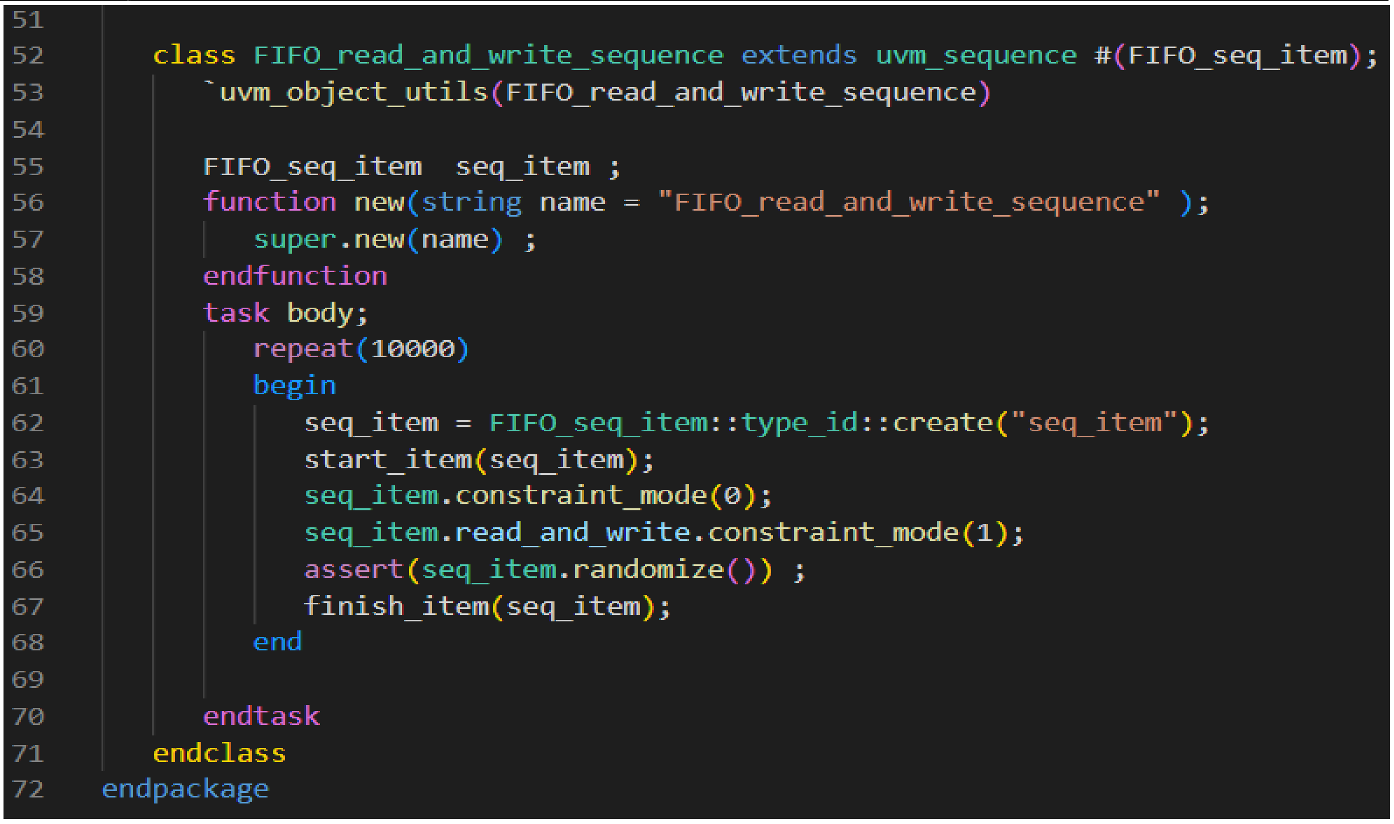
6.6. FIFO main sequence



```
package FIFO_seq_main_pkg;

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


    class FIFO_write_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_write_sequence)

        FIFO_seq_item  seq_item ;
        function new(string name = "FIFO_write_sequence" );
            super.new(name) ;
        endfunction
        task body;
            repeat(100)
            begin
                seq_item = FIFO_seq_item::type_id::create("seq_item");
                start_item(seq_item);
                seq_item.constraint_mode(0);
                seq_item.write_only.constraint_mode(1);
                assert(seq_item.randomize()) ;
                finish_item(seq_item);
            end

        endtask
    endclass
```

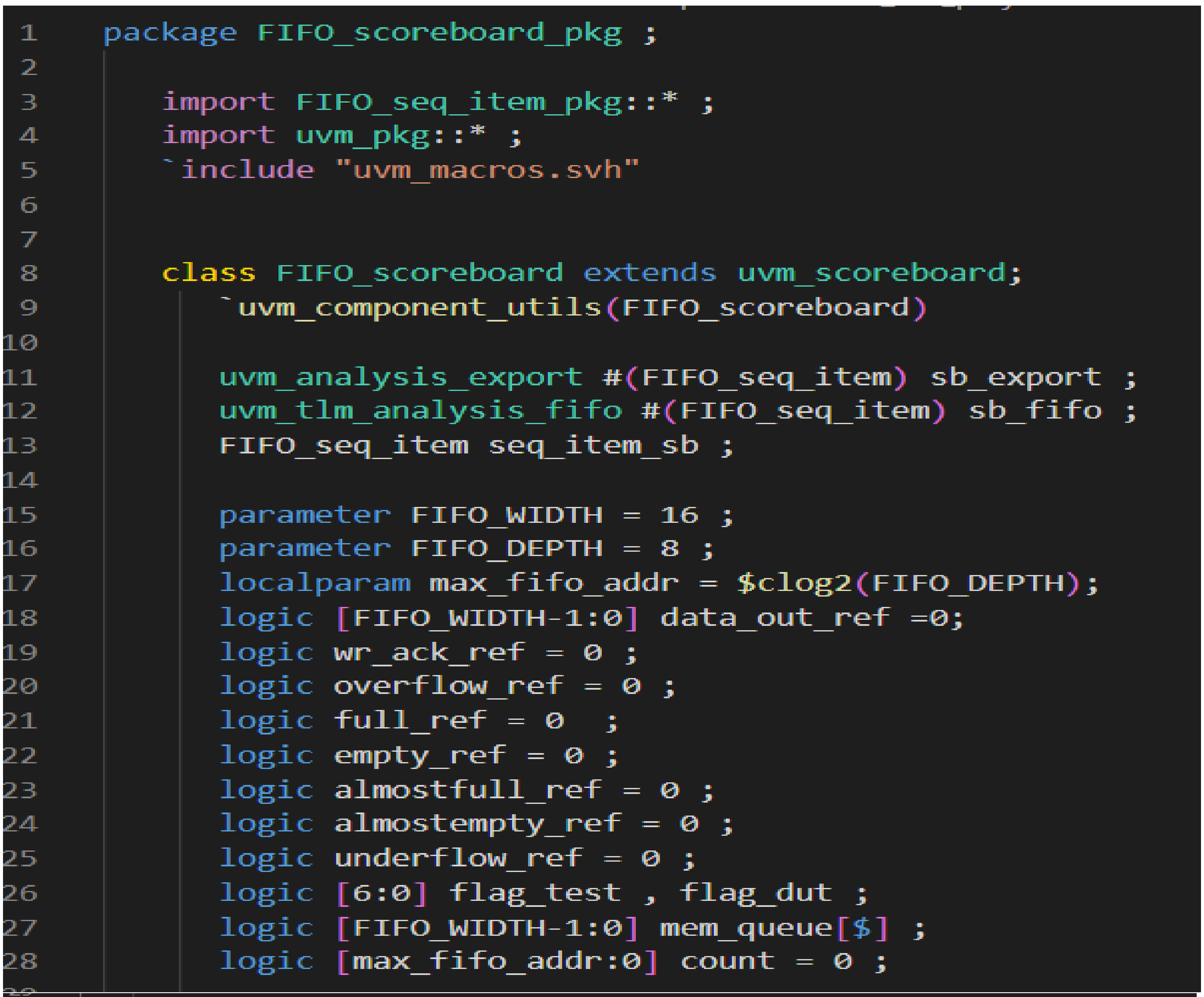


```systemverilog
    class FIFO_read_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_read_sequence)

        FIFO_seq_item  seq_item ;
        function new(string name = "FIFO_read_sequence" );
            super.new(name) ;
        endfunction
        task body;
            repeat(100)
            begin
                seq_item = FIFO_seq_item::type_id::create("seq_item");
                start_item(seq_item);
                seq_item.constraint_mode(0);
                seq_item.read_only.constraint_mode(1);
                assert(seq_item.randomize()) ;
                finish_item(seq_item);
            end

        endtask
    endclass
```

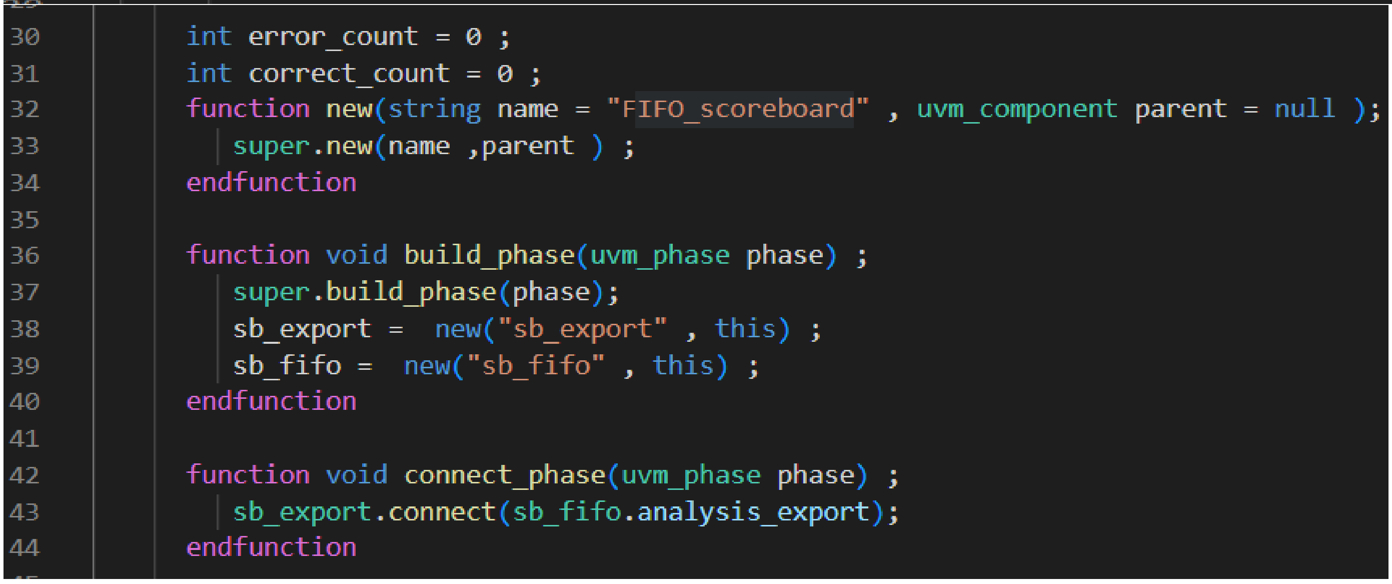


```systemverilog

    class FIFO_read_and_write_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_read_and_write_sequence)

        FIFO_seq_item  seq_item ;
        function new(string name = "FIFO_read_and_write_sequence" );
            super.new(name) ;
        endfunction
        task body;
            repeat(10000)
            begin
                seq_item = FIFO_seq_item::type_id::create("seq_item");
                start_item(seq_item);
                seq_item.constraint_mode(0);
                seq_item.read_and_write.constraint_mode(1);
                assert(seq_item.randomize()) ;
                finish_item(seq_item);
            end

        endtask
    endclass
endpackage
```

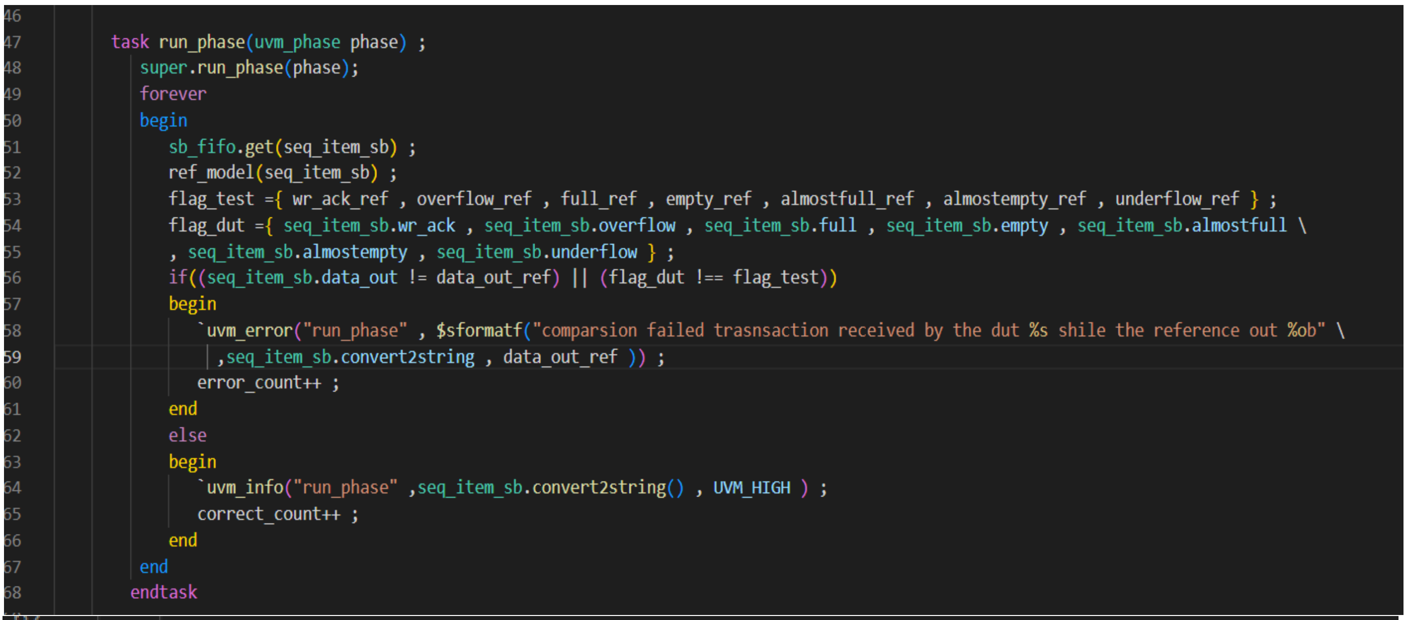
6.7. FIFO scoreboard



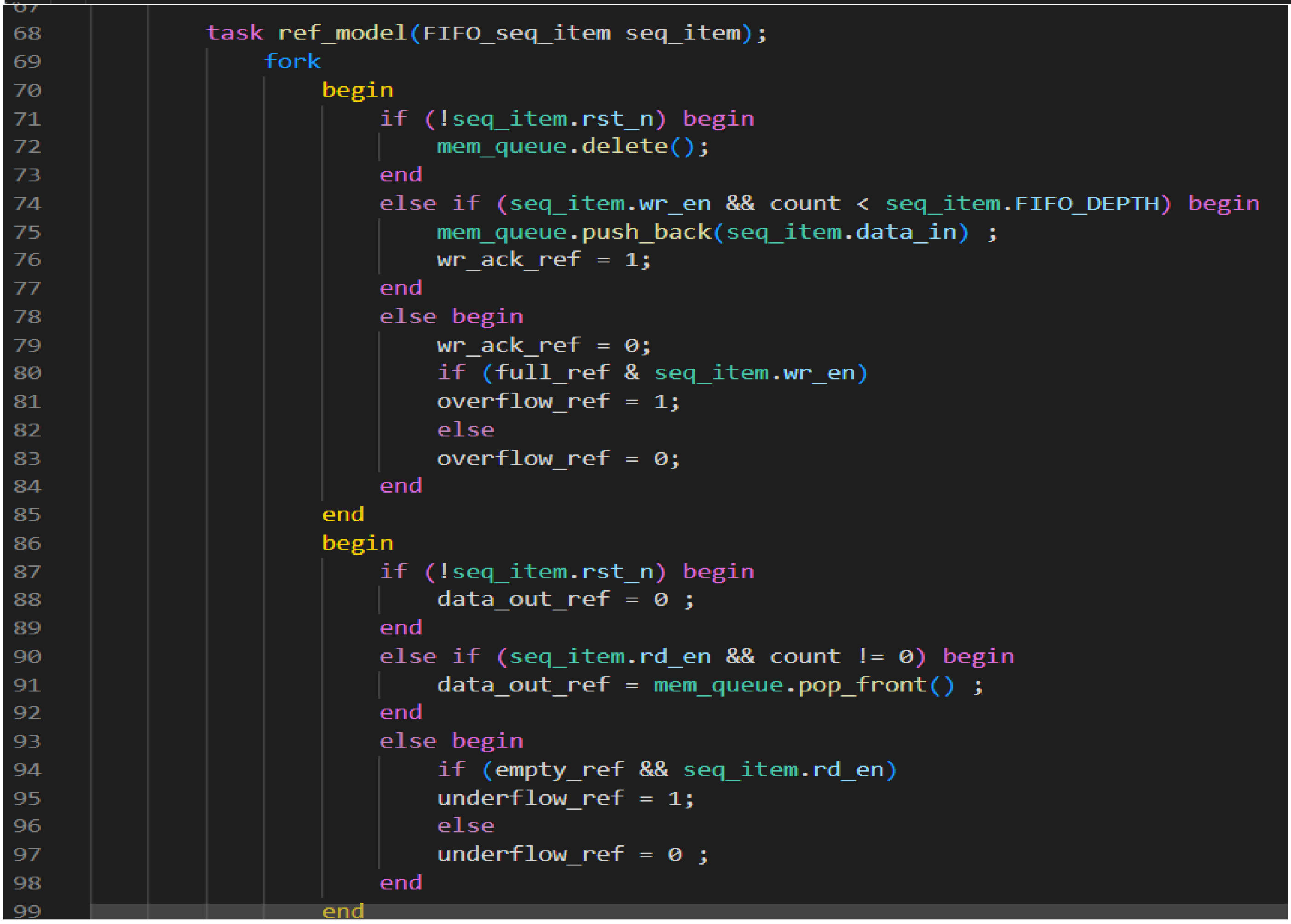
```
package FIFO_scoreboard_pkg ;

   import FIFO_seq_item_pkg::* ;
   import uvm_pkg::* ;
   `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_sb ;

      parameter FIFO_WIDTH = 16 ;
      parameter FIFO_DEPTH = 8 ;
      localparam max_fifo_addr = $clog2(FIFO_DEPTH);
      logic [FIFO_WIDTH-1:0] data_out_ref =0;
      logic wr_ack_ref = 0 ;
      logic overflow_ref = 0 ;
      logic full_ref = 0  ;
      logic empty_ref = 0 ;
      logic almostfull_ref = 0 ;
      logic almostempty_ref = 0 ;
      logic underflow_ref = 0 ;
      logic [6:0] flag_test , flag_dut ;
      logic [FIFO_WIDTH-1:0] mem_queue[$] ;
      logic [max_fifo_addr:0] count = 0 ;
```



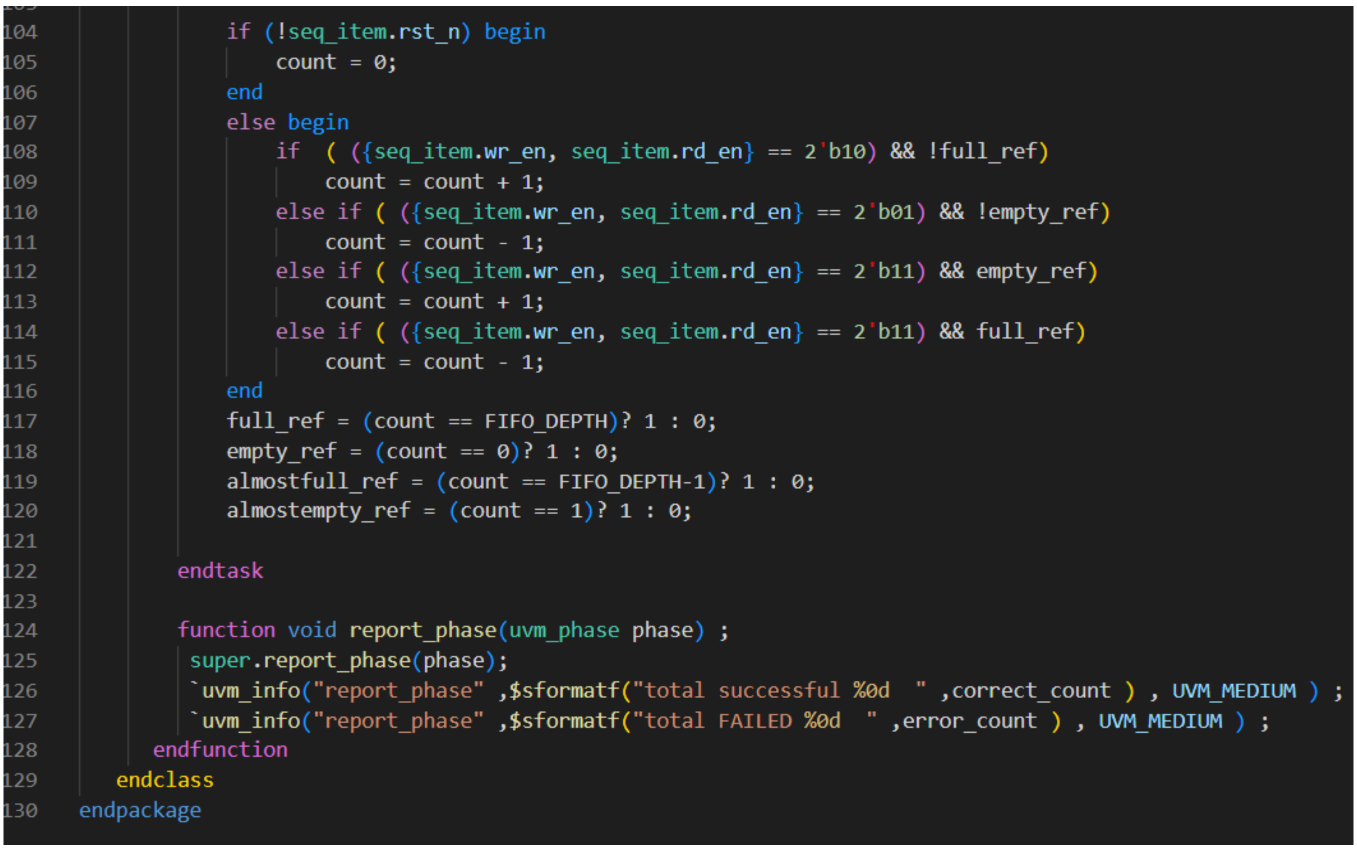
```
      int error_count = 0 ;
      int correct_count = 0 ;
      function new(string name = "FIFO_scoreboard" , uvm_component parent = null );
         super.new(name ,parent ) ;
      endfunction

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

      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_sb) ;
        ref_model(seq_item_sb) ;
        flag_test ={ wr_ack_ref , overflow_ref , full_ref , empty_ref , almostfull_ref , almostempty_ref , underflow_ref } ;
        flag_dut ={ seq_item_sb.wr_ack , seq_item_sb.overflow , seq_item_sb.full , seq_item_sb.empty , seq_item_sb.almostfull \
        , seq_item_sb.almostempty , seq_item_sb.underflow } ;
        if((seq_item_sb.data_out != data_out_ref) || (flag_dut !== flag_test))
        begin
            `uvm_error("run_phase" , $sformatf("comparsion failed trasnsaction received by the dut %s shile the reference out %0b" \
            ,seq_item_sb.convert2string , data_out_ref )) ;
            error_count++ ;
        end
        else
        begin
            `uvm_info("run_phase" ,seq_item_sb.convert2string() , UVM_HIGH ) ;
            correct_count++ ;
        end
    end
  endtask
```

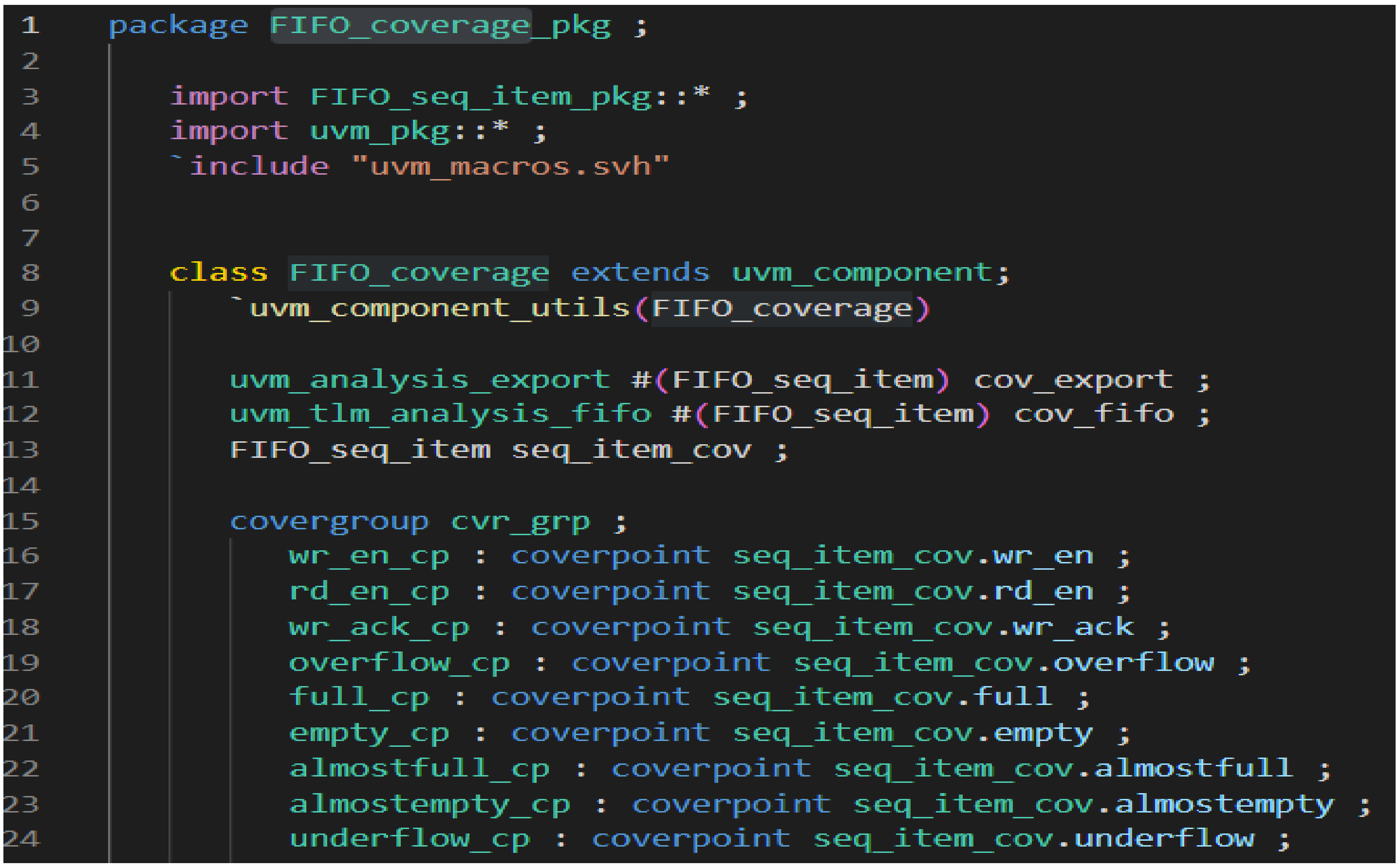


```
task ref_model(FIFO_seq_item seq_item);
    fork
        begin
            if (!seq_item.rst_n) begin
                mem_queue.delete();
            end
            else if (seq_item.wr_en && count < seq_item.FIFO_DEPTH) begin
                mem_queue.push_back(seq_item.data_in) ;
                wr_ack_ref = 1;
            end
            else begin
                wr_ack_ref = 0;
                if (full_ref & seq_item.wr_en)
                overflow_ref = 1;
                else
                overflow_ref = 0;
            end
        end
        begin
            if (!seq_item.rst_n) begin
                data_out_ref = 0 ;
            end
            else if (seq_item.rd_en && count != 0) begin
                data_out_ref = mem_queue.pop_front() ;
            end
            else begin
                if (empty_ref && seq_item.rd_en)
                underflow_ref = 1;
                else
                underflow_ref = 0 ;
            end
        end
```

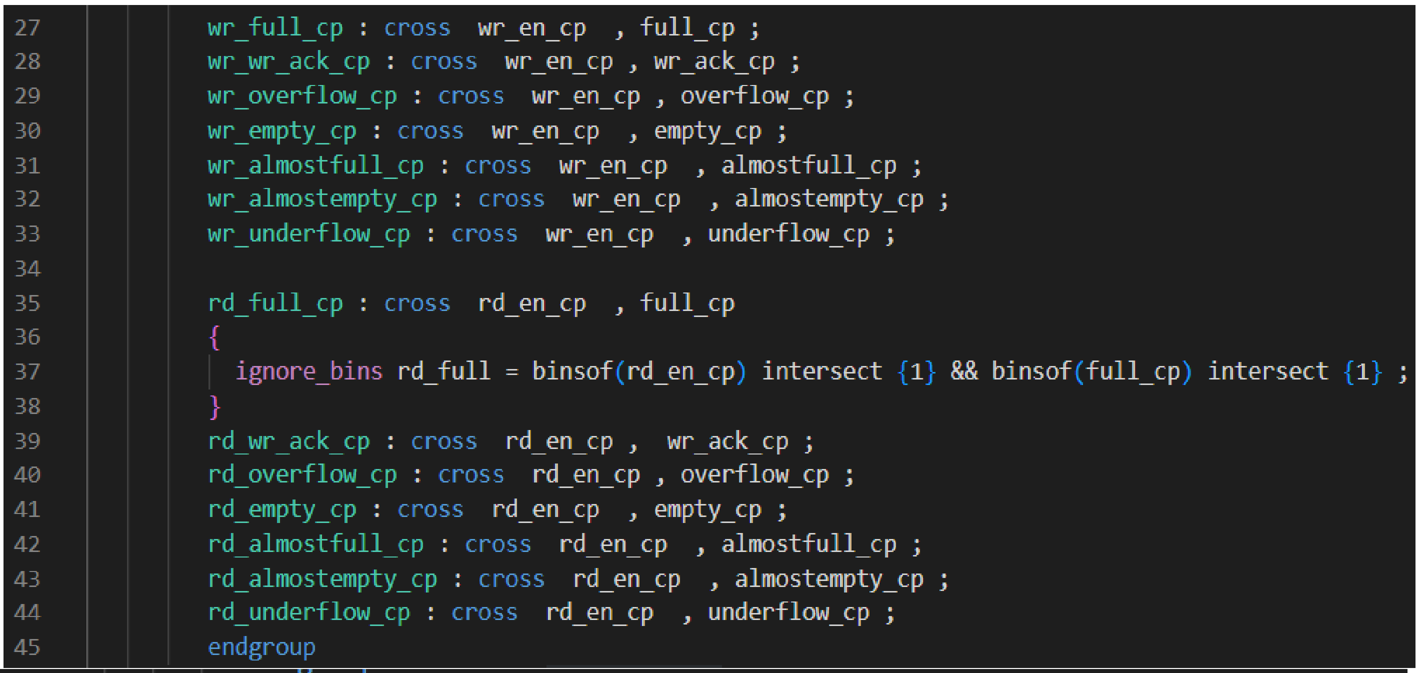


```
            if (!seq_item.rst_n) begin
                count = 0;
            end
            else begin
                if  ( ({seq_item.wr_en, seq_item.rd_en} == 2'b10) && !full_ref)
                    count = count + 1;
                else if ( ({seq_item.wr_en, seq_item.rd_en} == 2'b01) && !empty_ref)
                    count = count - 1;
                else if ( ({seq_item.wr_en, seq_item.rd_en} == 2'b11) && empty_ref)
                    count = count + 1;
                else if ( ({seq_item.wr_en, seq_item.rd_en} == 2'b11) && full_ref)
                    count = count - 1;
            end
            full_ref = (count == FIFO_DEPTH)? 1 : 0;
            empty_ref = (count == 0)? 1 : 0;
            almostfull_ref = (count == FIFO_DEPTH-1)? 1 : 0;
            almostempty_ref = (count == 1)? 1 : 0;

        endtask

        function void report_phase(uvm_phase phase) ;
         super.report_phase(phase);
         `uvm_info("report_phase" ,$sformatf("total successful %0d  " ,correct_count ) , UVM_MEDIUM ) ;
         `uvm_info("report_phase" ,$sformatf("total FAILED %0d  " ,error_count ) , UVM_MEDIUM ) ;
      endfunction
   endclass
endpackage
```

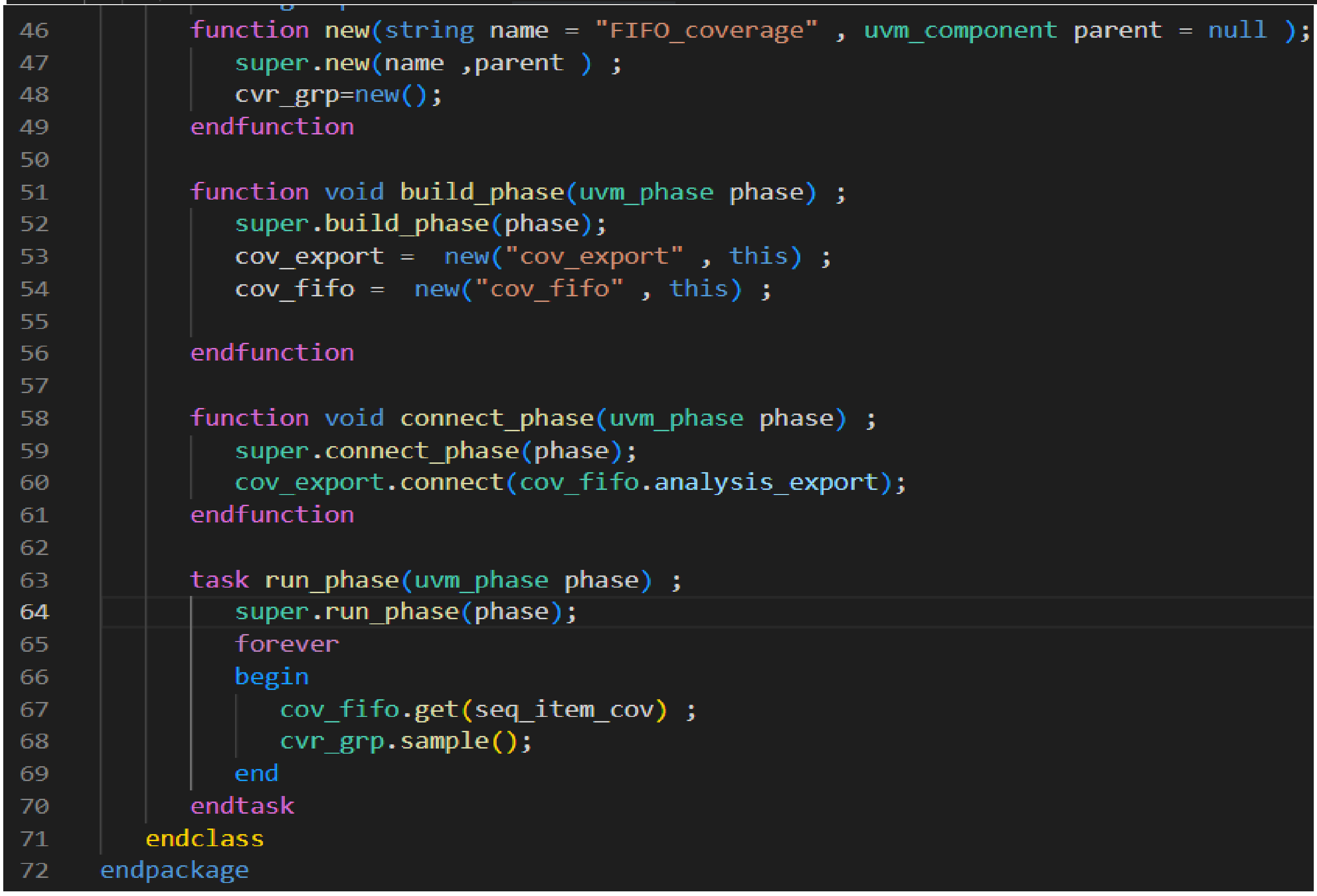
6.8. FIFO coverage



```
package FIFO_coverage_pkg ;

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


   class FIFO_coverage extends uvm_component;
      `uvm_component_utils(FIFO_coverage)

      uvm_analysis_export #(FIFO_seq_item) cov_export ;
      uvm_tlm_analysis_fifo #(FIFO_seq_item) cov_fifo ;
      FIFO_seq_item seq_item_cov ;

      covergroup cvr_grp ;
         wr_en_cp : coverpoint seq_item_cov.wr_en ;
         rd_en_cp : coverpoint seq_item_cov.rd_en ;
         wr_ack_cp : coverpoint seq_item_cov.wr_ack ;
         overflow_cp : coverpoint seq_item_cov.overflow ;
         full_cp : coverpoint seq_item_cov.full ;
         empty_cp : coverpoint seq_item_cov.empty ;
         almostfull_cp : coverpoint seq_item_cov.almostfull ;
         almostempty_cp : coverpoint seq_item_cov.almostempty ;
         underflow_cp : coverpoint seq_item_cov.underflow ;
```

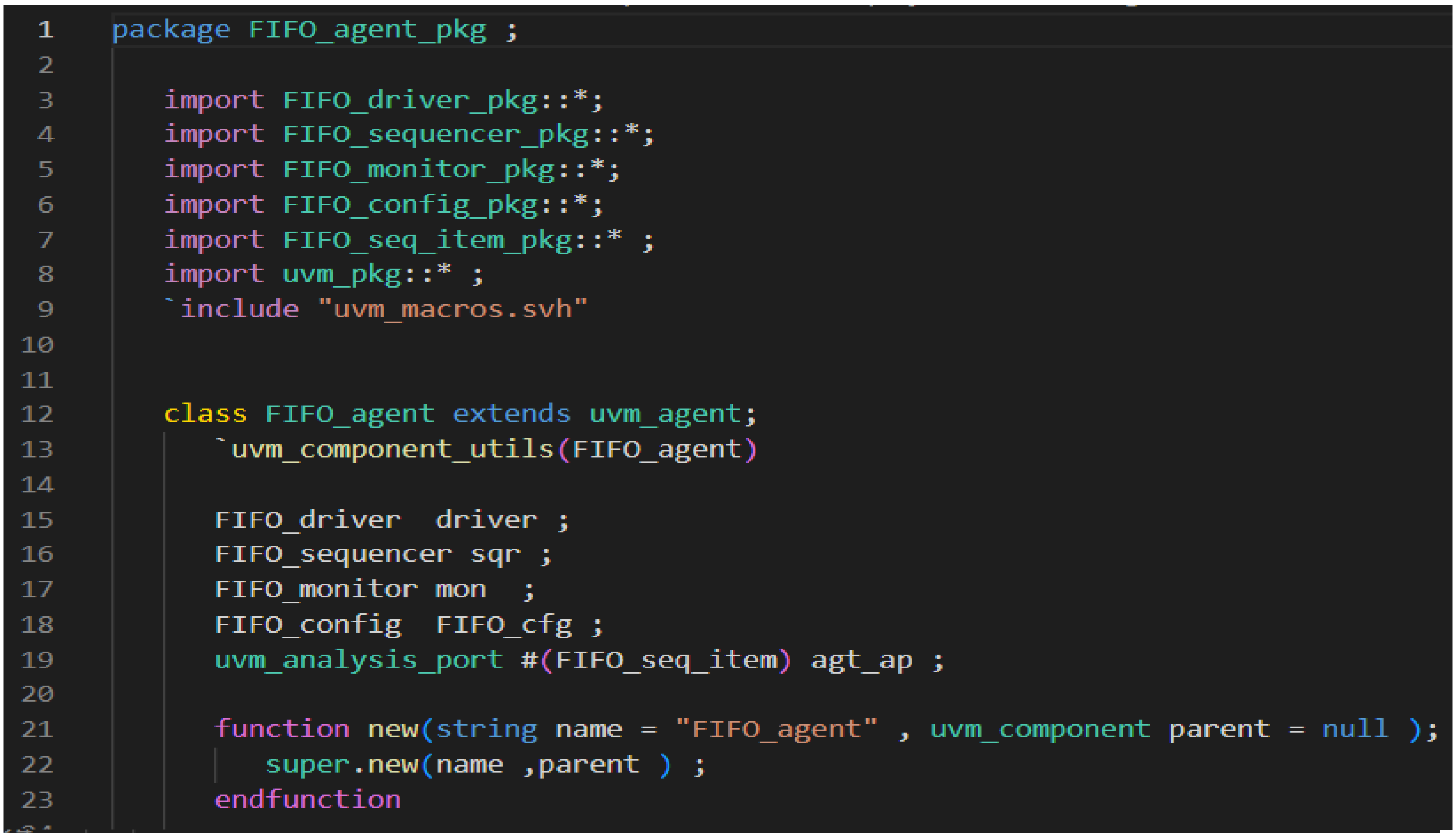


```
            wr_full_cp : cross  wr_en_cp  , full_cp ;
            wr_wr_ack_cp : cross  wr_en_cp , wr_ack_cp ;
            wr_overflow_cp : cross  wr_en_cp , overflow_cp ;
            wr_empty_cp : cross  wr_en_cp  , empty_cp ;
            wr_almostfull_cp : cross  wr_en_cp  , almostfull_cp ;
            wr_almostempty_cp : cross  wr_en_cp  , almostempty_cp ;
            wr_underflow_cp : cross  wr_en_cp  , underflow_cp ;

            rd_full_cp : cross  rd_en_cp  , full_cp
            {
              ignore_bins rd_full = binsof(rd_en_cp) intersect {1} && binsof(full_cp) intersect {1} ;
            }
            rd_wr_ack_cp : cross  rd_en_cp ,  wr_ack_cp ;
            rd_overflow_cp : cross  rd_en_cp , overflow_cp ;
            rd_empty_cp : cross  rd_en_cp  , empty_cp ;
            rd_almostfull_cp : cross  rd_en_cp  , almostfull_cp ;
            rd_almostempty_cp : cross  rd_en_cp  , almostempty_cp ;
            rd_underflow_cp : cross  rd_en_cp  , underflow_cp ;
            endgroup
```

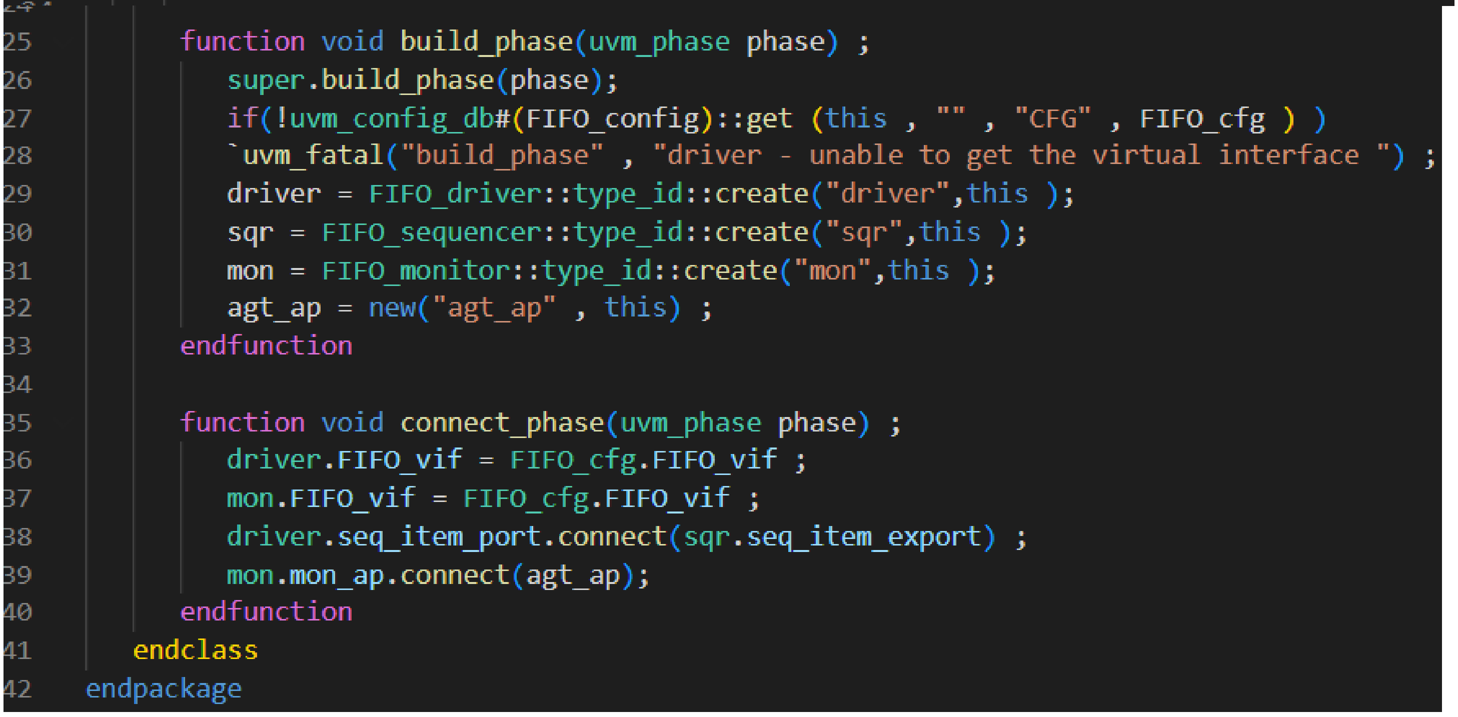


```
        function new(string name = "FIFO_coverage" , uvm_component parent = null );
            super.new(name ,parent ) ;
            cvr_grp=new();
        endfunction

        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_cov) ;
                cvr_grp.sample();
            end
        endtask
    endclass
endpackage
```

6.9. FIFO Agent

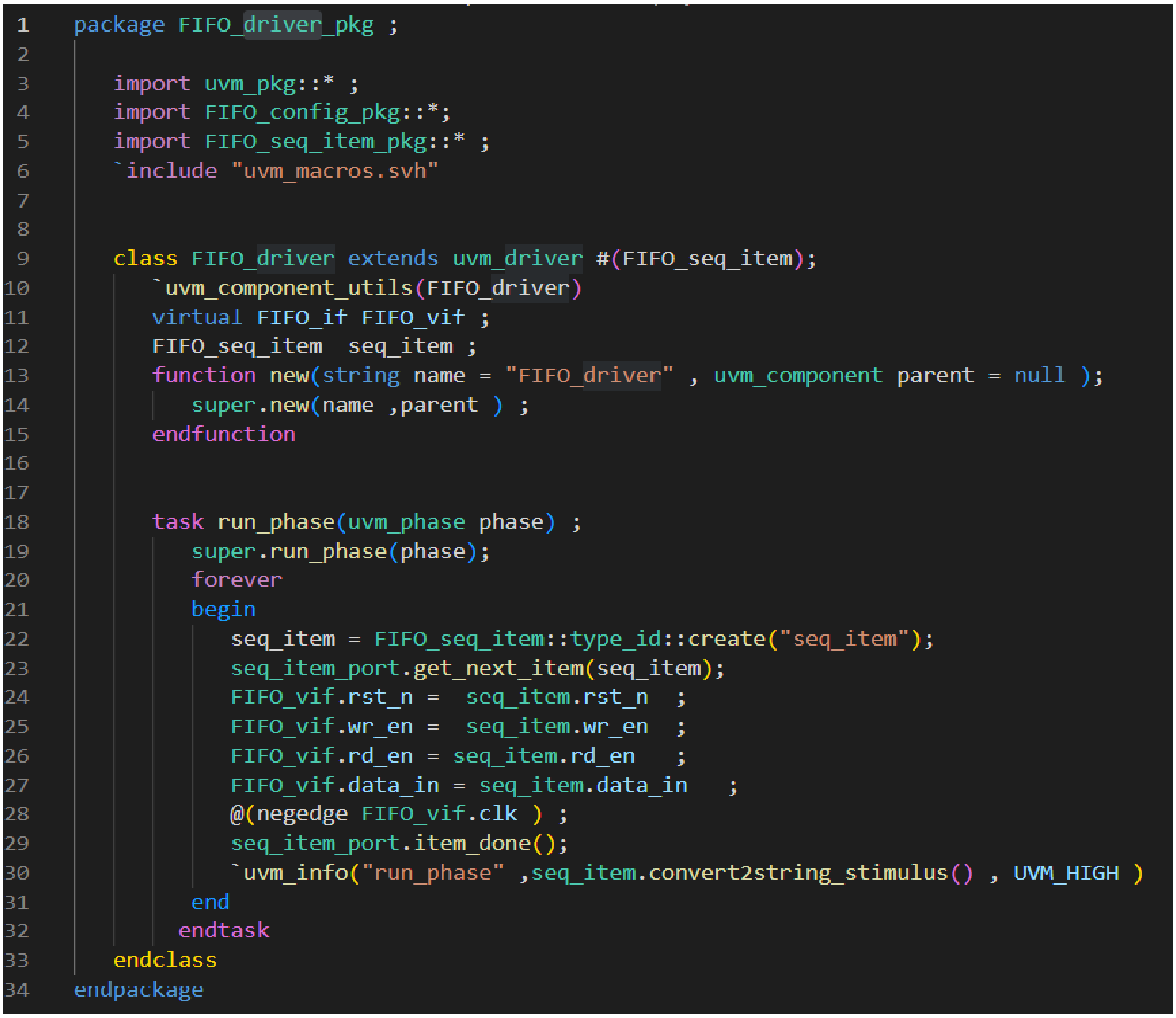


```
package FIFO_agent_pkg ;

   import FIFO_driver_pkg::*;
   import FIFO_sequencer_pkg::*;
   import FIFO_monitor_pkg::*;
   import FIFO_config_pkg::*;
   import FIFO_seq_item_pkg::* ;
   import uvm_pkg::* ;
   `include "uvm_macros.svh"


   class FIFO_agent extends uvm_agent;
      `uvm_component_utils(FIFO_agent)

      FIFO_driver  driver ;
      FIFO_sequencer sqr ;
      FIFO_monitor mon  ;
      FIFO_config  FIFO_cfg ;
      uvm_analysis_port #(FIFO_seq_item) agt_ap ;

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



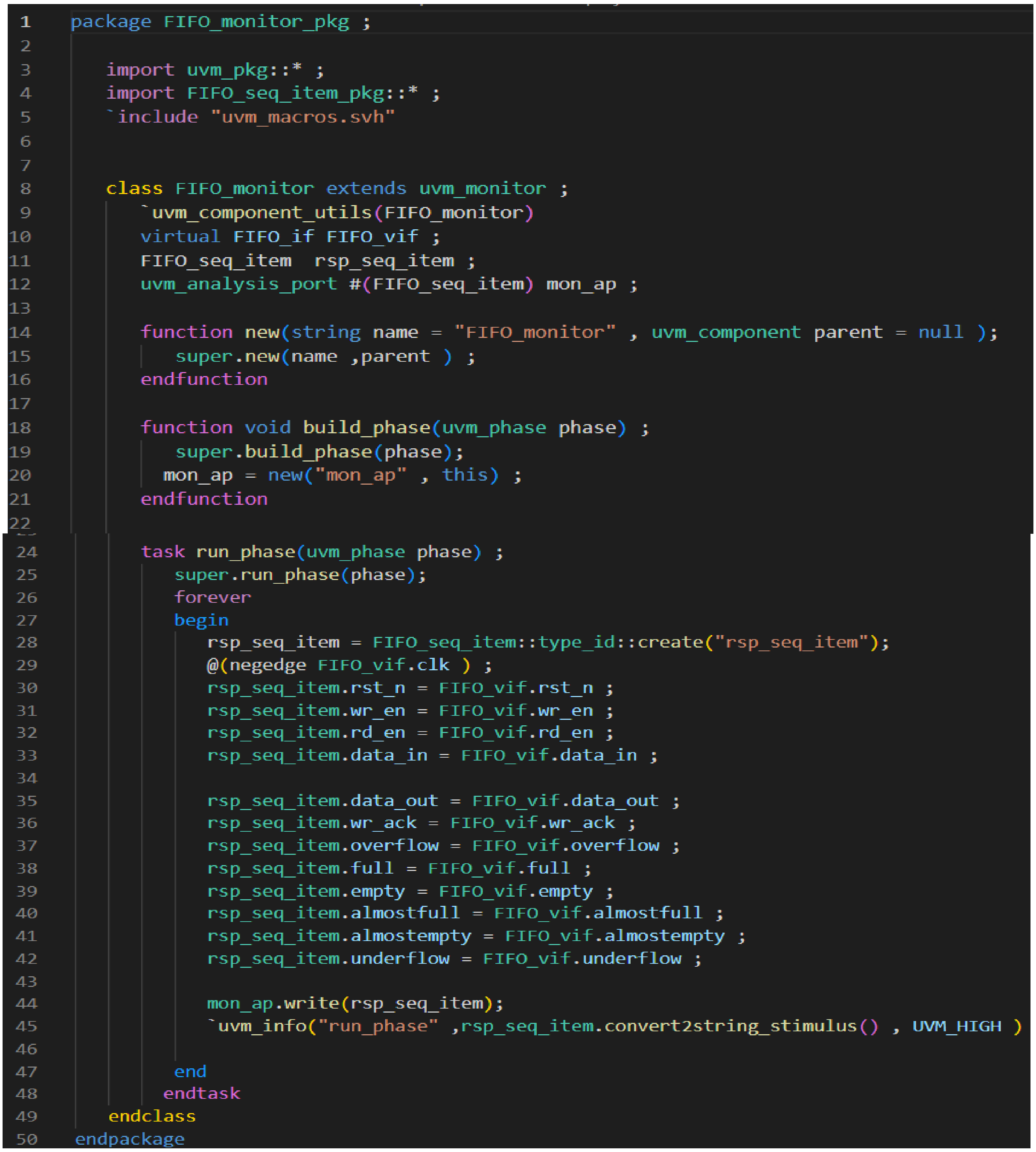
```
      function void build_phase(uvm_phase phase) ;
         super.build_phase(phase);
         if(!uvm_config_db#(FIFO_config)::get (this , "" , "CFG" , FIFO_cfg ) )
         `uvm_fatal("build_phase" , "driver - unable to get the virtual interface ") ;
         driver = FIFO_driver::type_id::create("driver",this );
         sqr = FIFO_sequencer::type_id::create("sqr",this );
         mon = FIFO_monitor::type_id::create("mon",this );
         agt_ap = new("agt_ap" , this) ;
      endfunction

      function void connect_phase(uvm_phase phase) ;
         driver.FIFO_vif = FIFO_cfg.FIFO_vif ;
         mon.FIFO_vif = FIFO_cfg.FIFO_vif ;
         driver.seq_item_port.connect(sqr.seq_item_export) ;
         mon.mon_ap.connect(agt_ap);
      endfunction
   endclass
endpackage
```

6.10. Driver



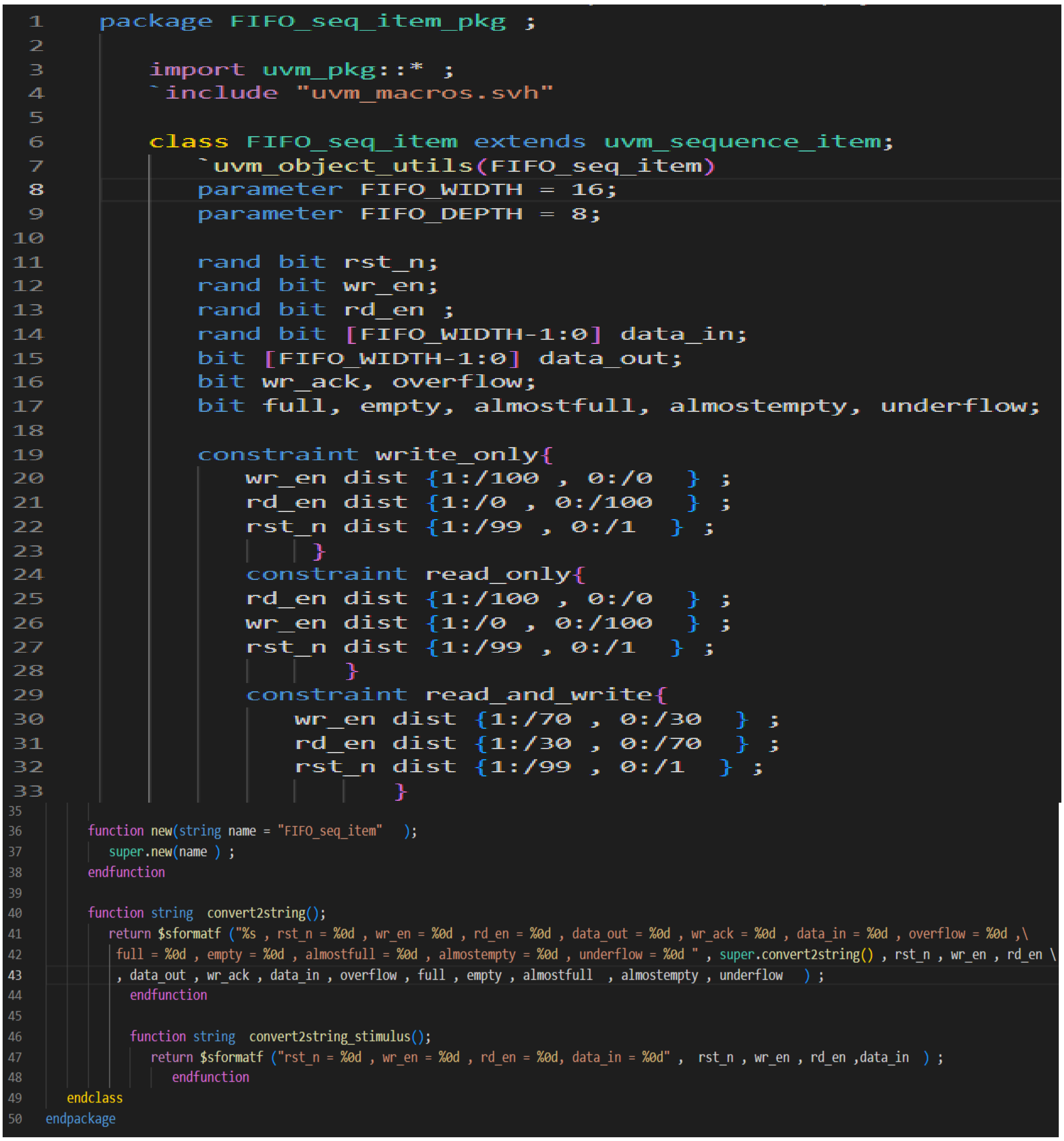
```
package FIFO_driver_pkg ;

   import uvm_pkg::* ;
   import FIFO_config_pkg::*;
   import FIFO_seq_item_pkg::* ;
   `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  seq_item ;
      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);
         forever
         begin
            seq_item = FIFO_seq_item::type_id::create("seq_item");
            seq_item_port.get_next_item(seq_item);
            FIFO_vif.rst_n =  seq_item.rst_n  ;
            FIFO_vif.wr_en =  seq_item.wr_en  ;
            FIFO_vif.rd_en = seq_item.rd_en   ;
            FIFO_vif.data_in = seq_item.data_in   ;
            @(negedge FIFO_vif.clk ) ;
            seq_item_port.item_done();
            `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
         end
        endtask
   endclass
endpackage
```

6.11. Monitor



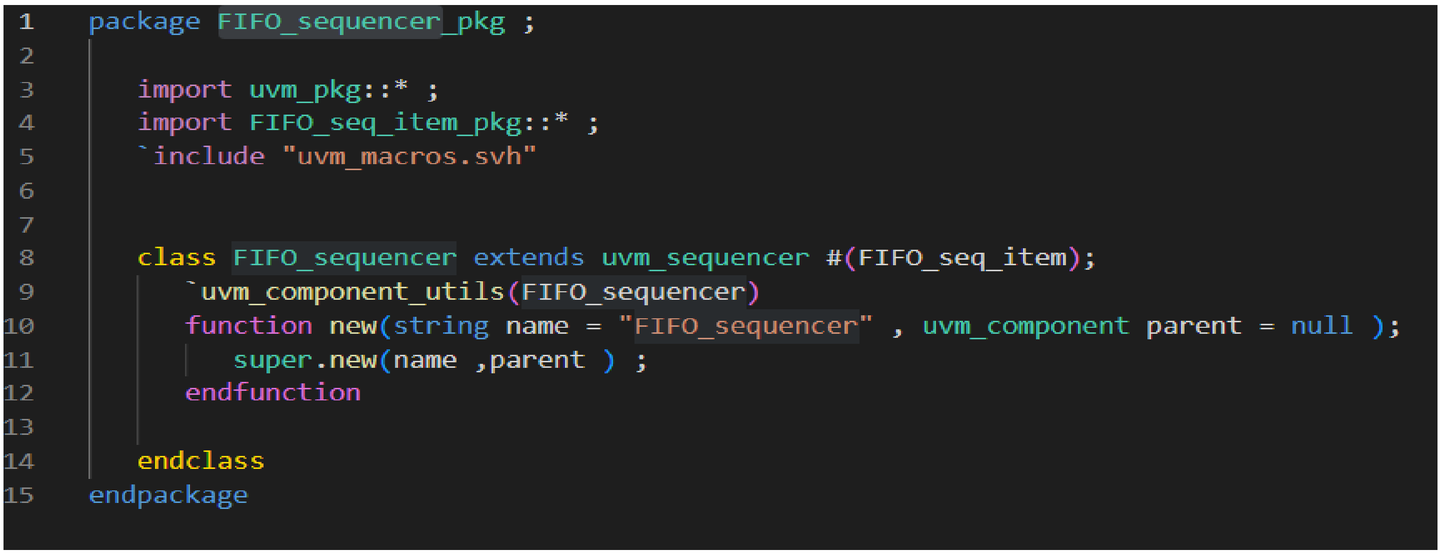
```
package FIFO_monitor_pkg ;

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


    class FIFO_monitor extends uvm_monitor ;
        `uvm_component_utils(FIFO_monitor)
        virtual FIFO_if FIFO_vif ;
        FIFO_seq_item  rsp_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

        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
                rsp_seq_item = FIFO_seq_item::type_id::create("rsp_seq_item");
                @(negedge FIFO_vif.clk ) ;
                rsp_seq_item.rst_n = FIFO_vif.rst_n ;
                rsp_seq_item.wr_en = FIFO_vif.wr_en ;
                rsp_seq_item.rd_en = FIFO_vif.rd_en ;
                rsp_seq_item.data_in = FIFO_vif.data_in ;

                rsp_seq_item.data_out = FIFO_vif.data_out ;
                rsp_seq_item.wr_ack = FIFO_vif.wr_ack ;
                rsp_seq_item.overflow = FIFO_vif.overflow ;
                rsp_seq_item.full = FIFO_vif.full ;
                rsp_seq_item.empty = FIFO_vif.empty ;
                rsp_seq_item.almostfull = FIFO_vif.almostfull ;
                rsp_seq_item.almostempty = FIFO_vif.almostempty ;
                rsp_seq_item.underflow = FIFO_vif.underflow ;

                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase" ,rsp_seq_item.convert2string_stimulus() , UVM_HIGH )

            end
          endtask
    endclass
endpackage
```

6.12. FIFO sequence item

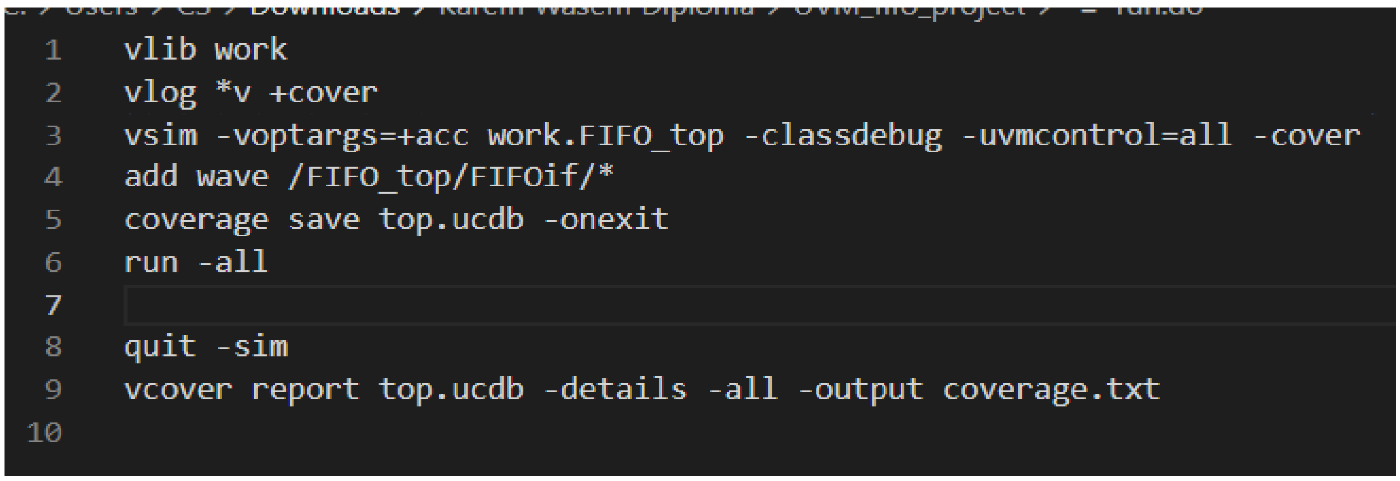


```
package FIFO_seq_item_pkg ;

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

      constraint write_only{
         wr_en dist {1:/100 , 0:/0  } ;
         rd_en dist {1:/0 , 0:/100  } ;
         rst_n dist {1:/99 , 0:/1  } ;
            }
         constraint read_only{
         rd_en dist {1:/100 , 0:/0  } ;
         wr_en dist {1:/0 , 0:/100  } ;
         rst_n dist {1:/99 , 0:/1  } ;
               }
         constraint read_and_write{
            wr_en dist {1:/70 , 0:/30  } ;
            rd_en dist {1:/30 , 0:/70  } ;
            rst_n dist {1:/99 , 0:/1  } ;
               }

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

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

          function string  convert2string_stimulus();
            return $sformatf ("rst_n = %0d , wr_en = %0d , rd_en = %0d, data_in = %0d" ,  rst_n , wr_en , rd_en ,data_in  ) ;
               endfunction
   endclass
endpackage
```

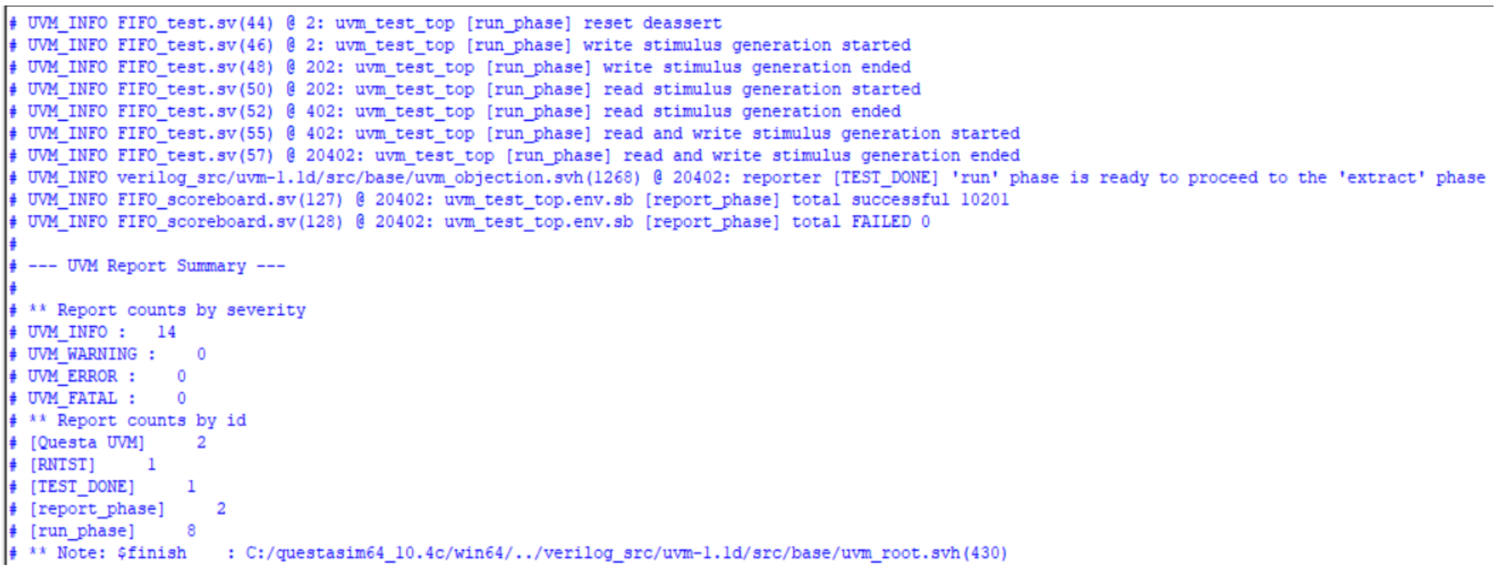
6.13. FIFO sequencer


```
package FIFO_sequencer_pkg ;

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


  class FIFO_sequencer extends uvm_sequencer #(FIFO_seq_item);
    `uvm_component_utils(FIFO_sequencer)
    function new(string name = "FIFO_sequencer" , uvm_component parent = null );
      super.new(name ,parent ) ;
    endfunction

  endclass
endpackage
```

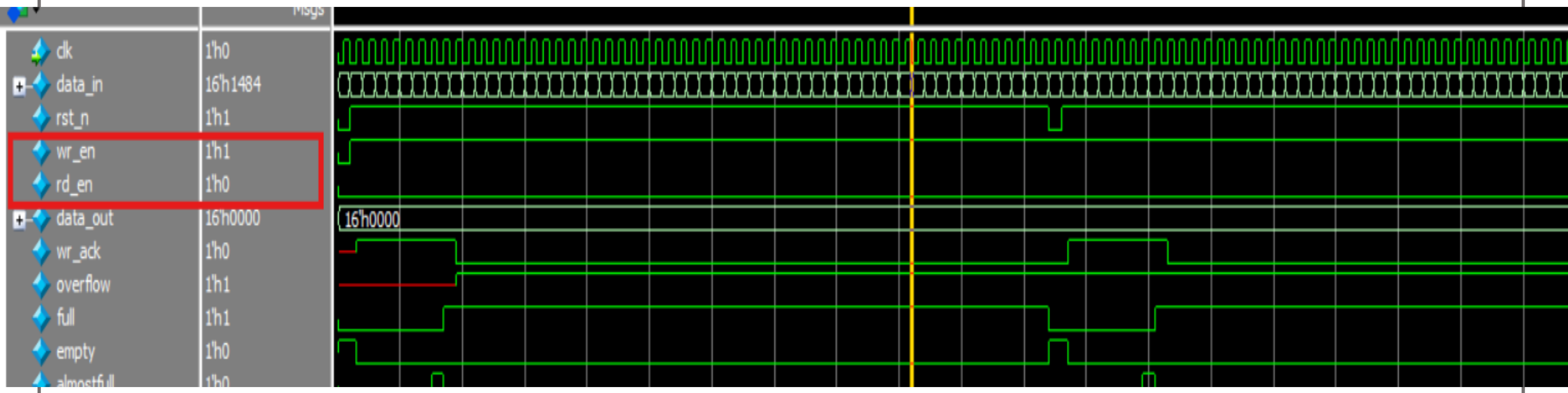
7. Do file


```
vlib work
vlog *v +cover
vsim -voptargs=+acc work.FIFO_top -classdebug -uvmcontrol=all -cover
add wave /FIFO_top/FIFOif/*
coverage save top.ucdb -onexit
run -all

quit -sim
vcover report top.ucdb -details -all -output coverage.txt

```

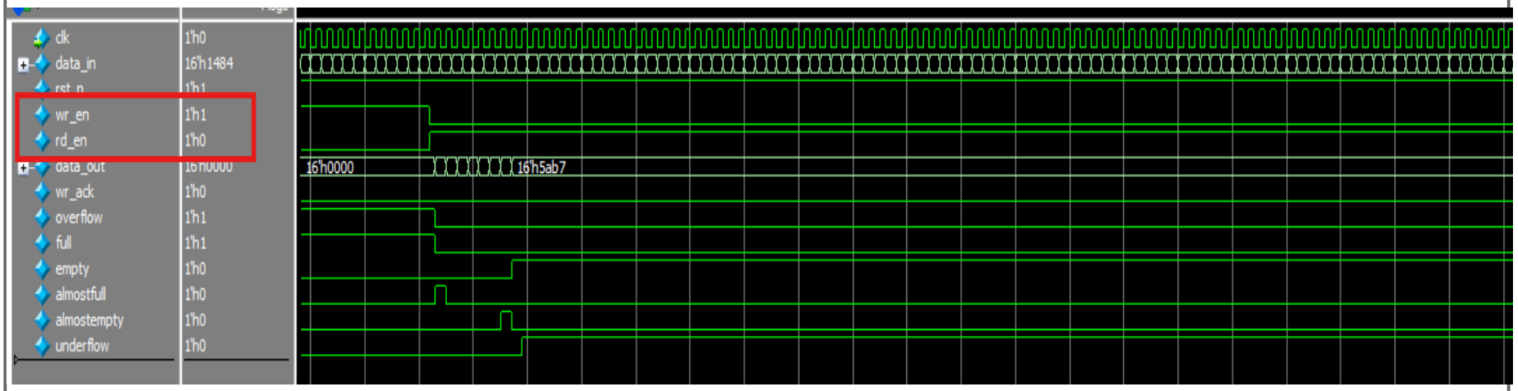
8. Transcript


```
# UVM_INFO FIFO_test.sv(44) @ 2: uvm_test_top [run_phase] reset deassert
# UVM_INFO FIFO_test.sv(46) @ 2: uvm_test_top [run_phase] write stimulus generation started
# UVM_INFO FIFO_test.sv(48) @ 202: uvm_test_top [run_phase] write stimulus generation ended
# UVM_INFO FIFO_test.sv(50) @ 202: uvm_test_top [run_phase] read stimulus generation started
# UVM_INFO FIFO_test.sv(52) @ 402: uvm_test_top [run_phase] read stimulus generation ended
# UVM_INFO FIFO_test.sv(55) @ 402: uvm_test_top [run_phase] read and write stimulus generation started
# UVM_INFO FIFO_test.sv(57) @ 20402: uvm_test_top [run_phase] read and write stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 20402: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(127) @ 20402: uvm_test_top.env.sb [report_phase] total successful 10201
# UVM_INFO FIFO_scoreboard.sv(128) @ 20402: uvm_test_top.env.sb [report_phase] total FAILED 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   14
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_10.4c/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
```

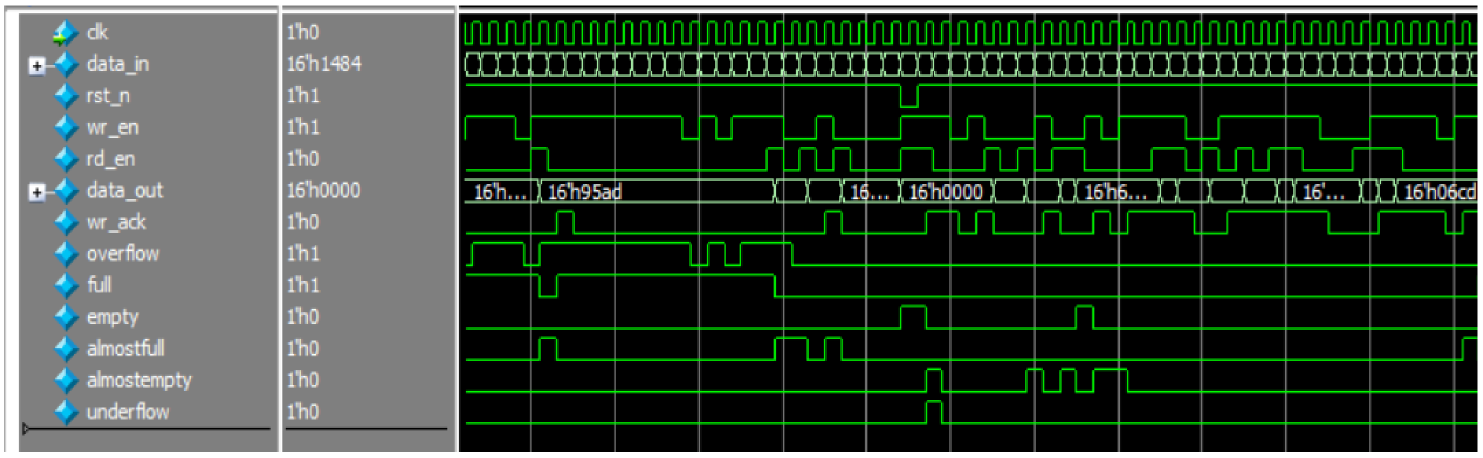
9. Waveform write sequence



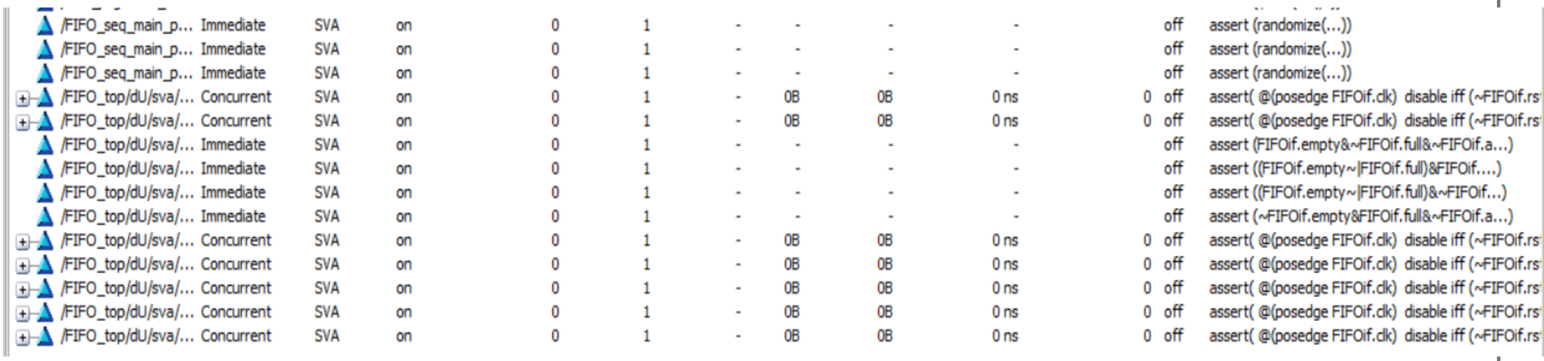
10. Waveform read sequence



11. Waveform write and read sequence

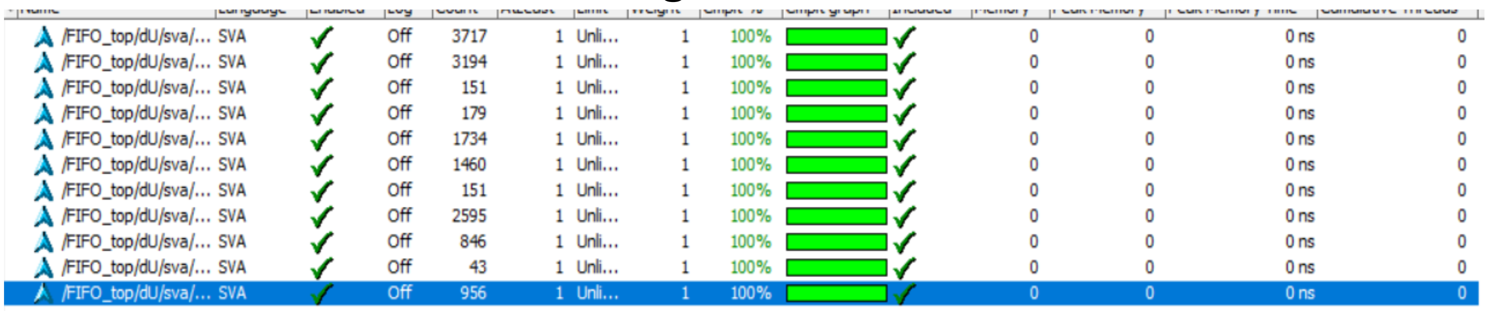


12. Assertion



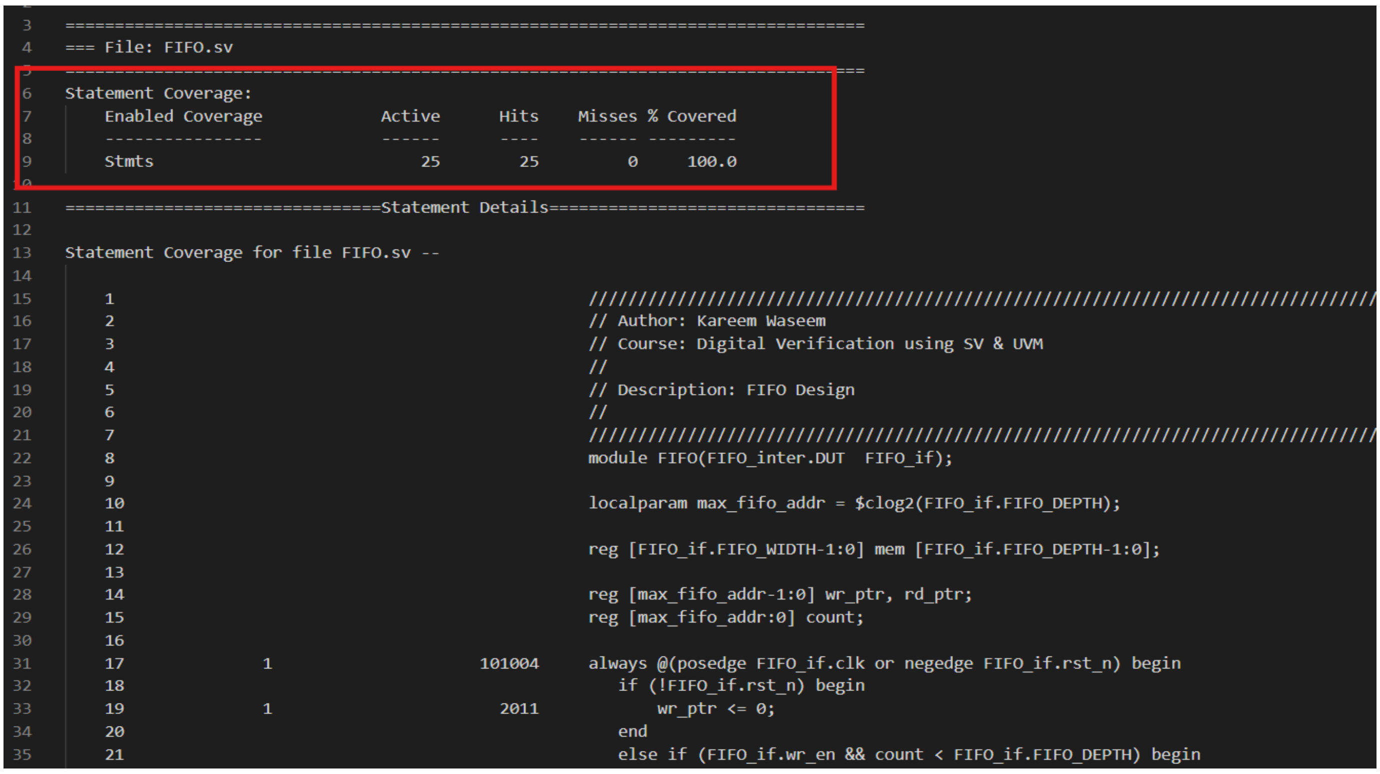
/FIFO_seq_main_p...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/FIFO_seq_main_p...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/FIFO_seq_main_p...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/FIFO_top/dU/sva/...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disable iff (~FIFOif.rs
/FIFO_top/dU/sva/...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disable iff (~FIFOif.rs
/FIFO_top/dU/sva/...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFOif.empty&~FIFOif.full&~FIFOif.a...)
/FIFO_top/dU/sva/...	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((FIFOif.empty~|FIFOif.full)&FIFOif....)
/FIFO_top/dU/sva/...	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((FIFOif.empty~|FIFOif.full)&~FIFOif...)
/FIFO_top/dU/sva/...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFOif.empty&FIFOif.full&~FIFOif.a...)
/FIFO_top/dU/sva/...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disable iff (~FIFOif.rs
/FIFO_top/dU/sva/...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disable iff (~FIFOif.rs
/FIFO_top/dU/sva/...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disable iff (~FIFOif.rs
/FIFO_top/dU/sva/...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disable iff (~FIFOif.rs
/FIFO_top/dU/sva/...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFOif.clk) disable iff (~FIFOif.rs

13. Assertion coverage

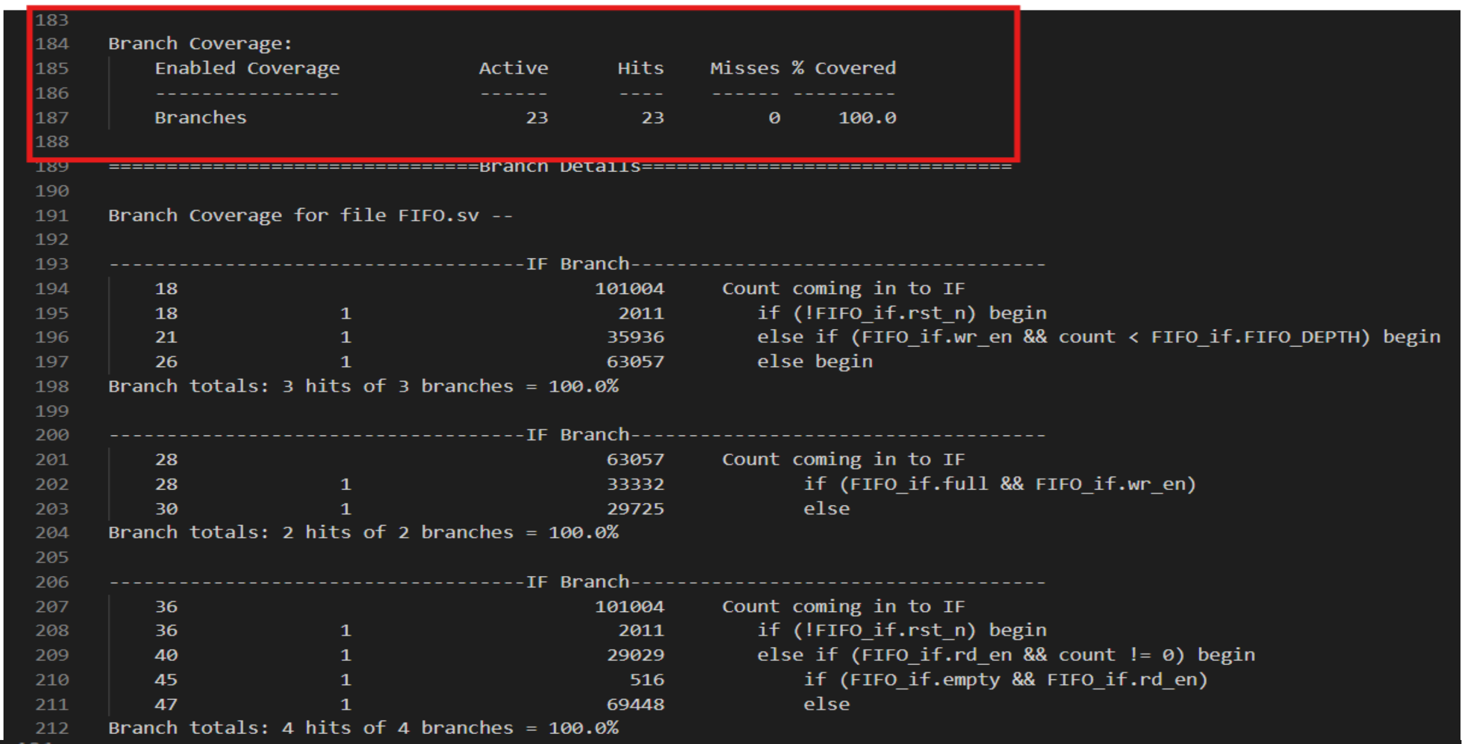


Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_top/dU/sva/...	SVA	✓	Off	3717	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	3194	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	151	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	179	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	1734	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	1460	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	151	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	2595	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	846	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	43	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/dU/sva/...	SVA	✓	Off	956	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0

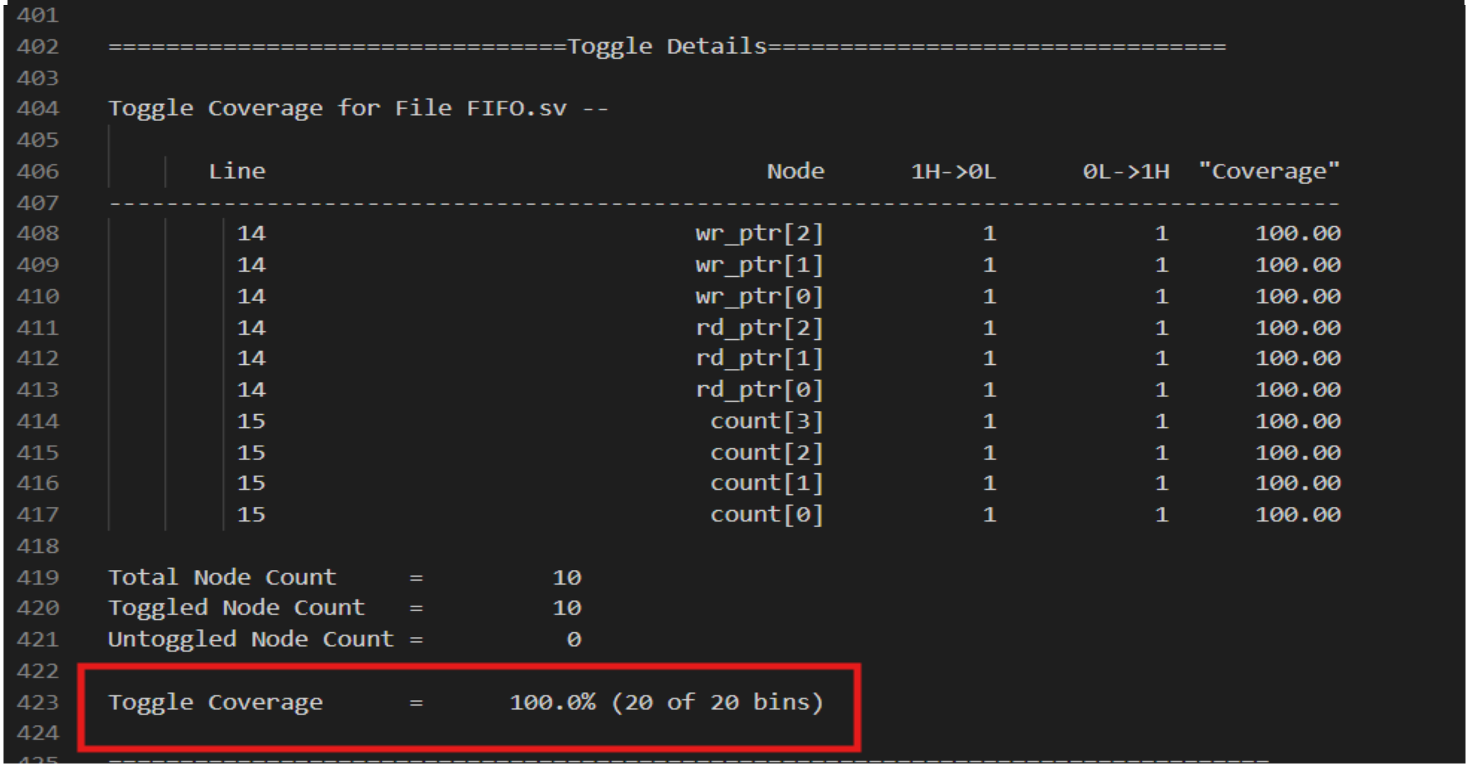
14. Code Coverage



```
=================================================================================
=== File: FIFO.sv
=================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           25        25         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                                                    module FIFO(FIFO_inter.DUT  FIFO_if);
    9
    10                                                   localparam max_fifo_addr = $clog2(FIFO_if.FIFO_DEPTH);
    11
    12                                                   reg [FIFO_if.FIFO_WIDTH-1:0] mem [FIFO_if.FIFO_DEPTH-1:0];
    13
    14                                                   reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
    15                                                   reg [max_fifo_addr:0] count;
    16
    17              1                      101004        always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    18                                                      if (!FIFO_if.rst_n) begin
    19              1                        2011               wr_ptr <= 0;
    20                                                      end
    21                                                      else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
```

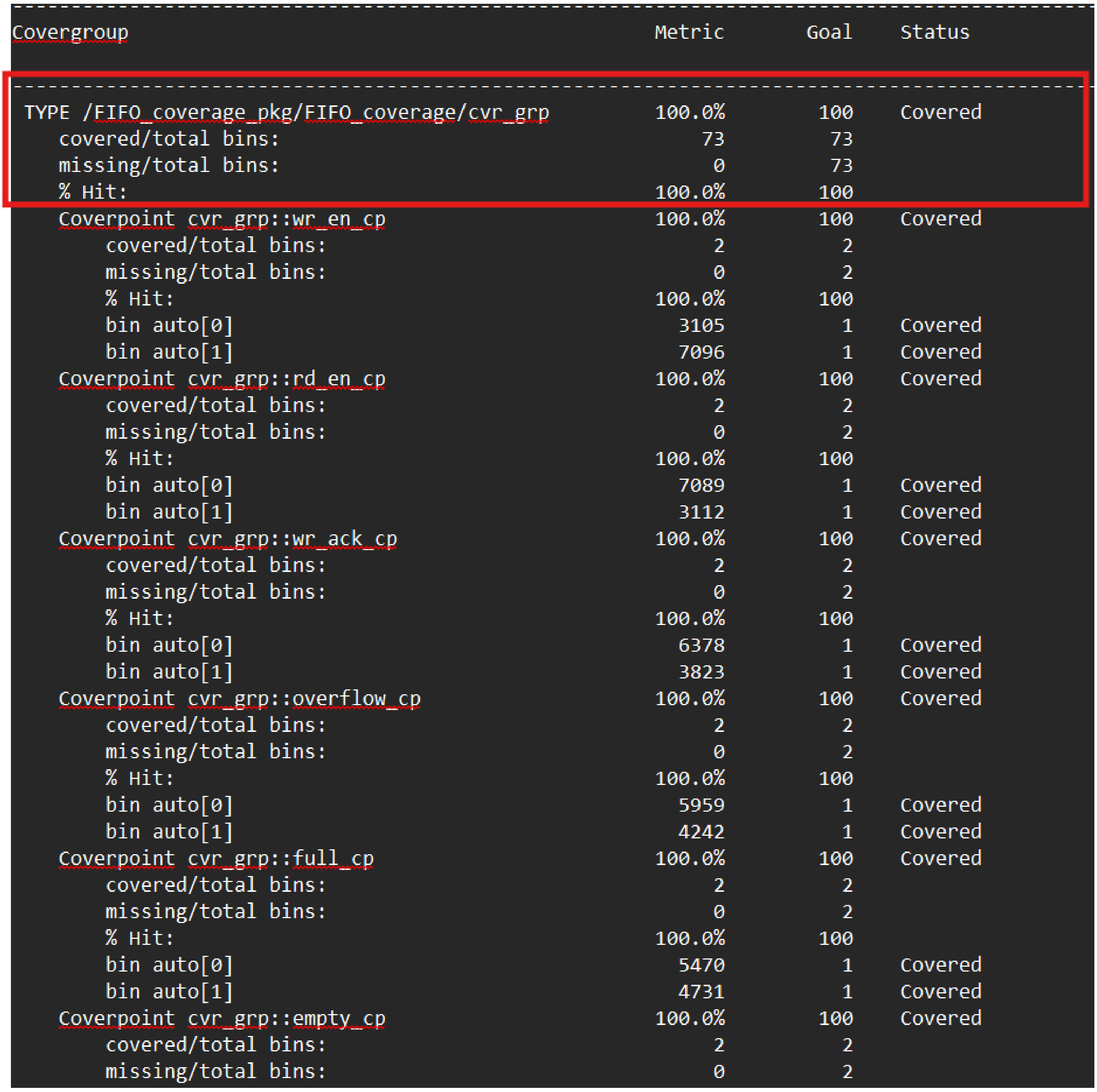


```

Branch Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Branches                          23        23         0     100.0

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

Branch Coverage for file FIFO.sv --

------------------------------------IF Branch------------------------------------
    18                                101004     Count coming in to IF
    18              1                   2011        if (!FIFO_if.rst_n) begin
    21              1                  35936        else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
    26              1                  63057        else begin
Branch totals: 3 hits of 3 branches = 100.0%

------------------------------------IF Branch------------------------------------
    28                                 63057     Count coming in to IF
    28              1                  33332            if (FIFO_if.full && FIFO_if.wr_en)
    30              1                  29725            else
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    36                                101004     Count coming in to IF
    36              1                   2011        if (!FIFO_if.rst_n) begin
    40              1                  29029        else if (FIFO_if.rd_en && count != 0) begin
    45              1                    516            if (FIFO_if.empty && FIFO_if.rd_en)
    47              1                  69448            else
Branch totals: 4 hits of 4 branches = 100.0%
```

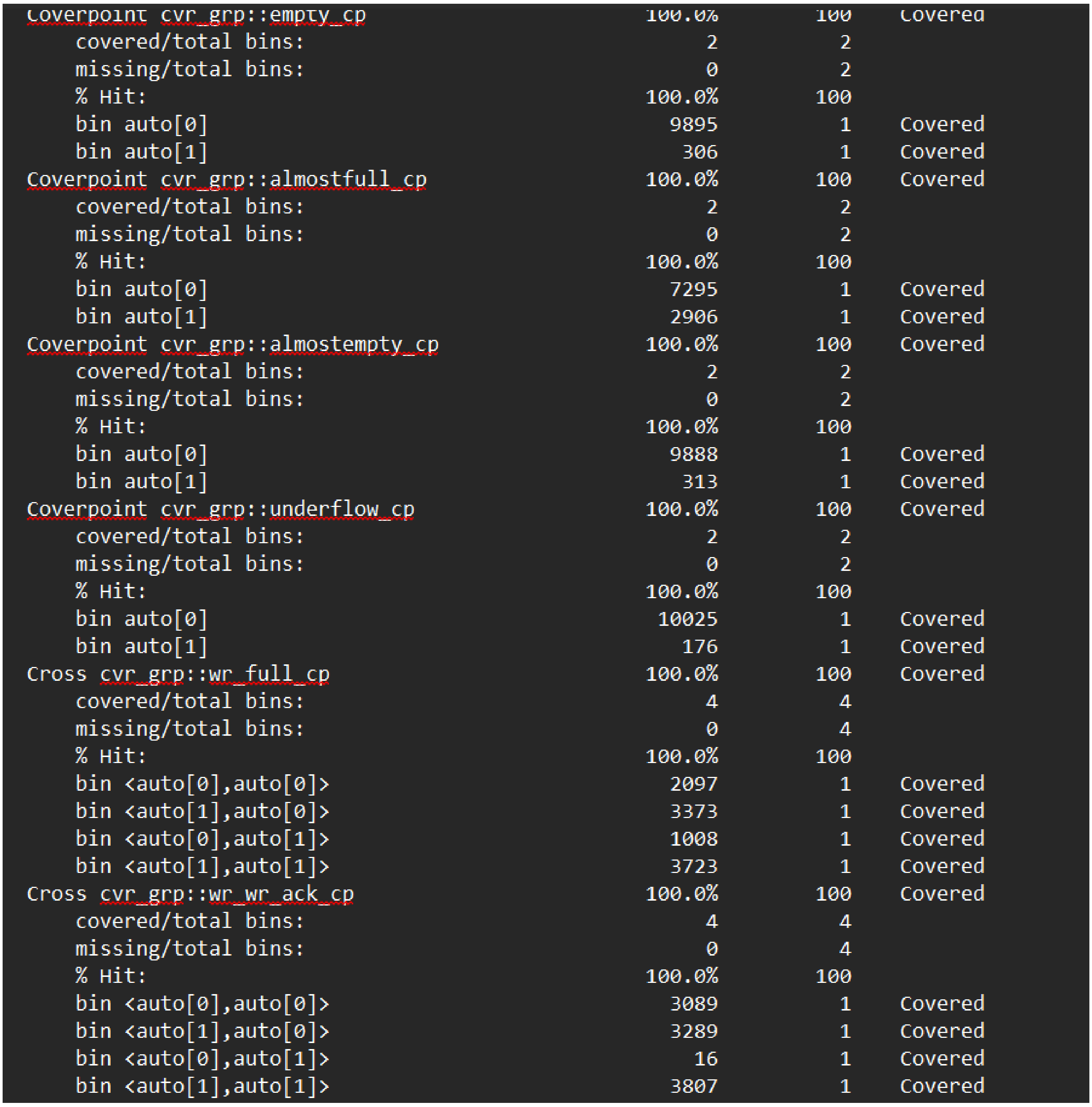


```

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

Toggle Coverage for File FIFO.sv --

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

```

15. Functional Coverage report



```
------------------------------------------------------------------------------------------------
Covergroup                                             Metric       Goal    Status

------------------------------------------------------------------------------------------------
 TYPE /FIFO_coverage_pkg/FIFO_coverage/cvr_grp         100.0%        100    Covered
    covered/total bins:                                    73         73
    missing/total bins:                                     0         73
    % Hit:                                             100.0%        100
    Coverpoint cvr_grp::wr_en_cp                       100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      3105          1    Covered
        bin auto[1]                                      7096          1    Covered
    Coverpoint cvr_grp::rd_en_cp                       100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      7089          1    Covered
        bin auto[1]                                      3112          1    Covered
    Coverpoint cvr_grp::wr_ack_cp                      100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      6378          1    Covered
        bin auto[1]                                      3823          1    Covered
    Coverpoint cvr_grp::overflow_cp                    100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      5959          1    Covered
        bin auto[1]                                      4242          1    Covered
    Coverpoint cvr_grp::full_cp                        100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      5470          1    Covered
        bin auto[1]                                      4731          1    Covered
    Coverpoint cvr_grp::empty_cp                       100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
```



```
Coverpoint cvr_grp::empty_cp                         100.0%        100     Covered
    covered/total bins:                                   2          2
    missing/total bins:                                   0          2
    % Hit:                                           100.0%        100
    bin auto[0]                                        9895          1     Covered
    bin auto[1]                                         306          1     Covered
Coverpoint cvr_grp::almostfull_cp                    100.0%        100     Covered
    covered/total bins:                                   2          2
    missing/total bins:                                   0          2
    % Hit:                                           100.0%        100
    bin auto[0]                                        7295          1     Covered
    bin auto[1]                                        2906          1     Covered
Coverpoint cvr_grp::almostempty_cp                   100.0%        100     Covered
    covered/total bins:                                   2          2
    missing/total bins:                                   0          2
    % Hit:                                           100.0%        100
    bin auto[0]                                        9888          1     Covered
    bin auto[1]                                         313          1     Covered
Coverpoint cvr_grp::underflow_cp                     100.0%        100     Covered
    covered/total bins:                                   2          2
    missing/total bins:                                   0          2
    % Hit:                                           100.0%        100
    bin auto[0]                                       10025          1     Covered
    bin auto[1]                                         176          1     Covered
Cross cvr_grp::wr_full_cp                            100.0%        100     Covered
    covered/total bins:                                   4          4
    missing/total bins:                                   0          4
    % Hit:                                           100.0%        100
    bin <auto[0],auto[0]>                              2097          1     Covered
    bin <auto[1],auto[0]>                              3373          1     Covered
    bin <auto[0],auto[1]>                              1008          1     Covered
    bin <auto[1],auto[1]>                              3723          1     Covered
Cross cvr_grp::wr_wr_ack_cp                          100.0%        100     Covered
    covered/total bins:                                   4          4
    missing/total bins:                                   0          4
    % Hit:                                           100.0%        100
    bin <auto[0],auto[0]>                              3089          1     Covered
    bin <auto[1],auto[0]>                              3289          1     Covered
    bin <auto[0],auto[1]>                                16          1     Covered
    bin <auto[1],auto[1]>                              3807          1     Covered
```