FIFO Verification

Using SV and UVM

Mahmoud Ghamry

OCT 2024

Table of Contents

1.Introduction

In modern digital systems, the First-In-First-Out (FIFO) buffer plays a crucial role in managing data flow between components operating at different speeds. Verifying the functionality, reliability, and performance of a FIFO design is essential to ensure data integrity and seamless communication within these systems. To achieve a robust verification process, the Universal Verification Methodology (UVM) is employed due to its modular and reusable testbench structure.

This report focuses on the verification of a FIFO design using UVM. The primary objective of this project was to create a scalable and reusable test environment to validate the FIFO's functionality under various conditions, including boundary cases and stress scenarios. The verification environment was built with components such as UVM agents, sequences, scoreboards, and coverage analysis to ensure comprehensive functional verification.

This project aims to achieve the following objectives:

- Comprehensive Functional Testing: Validate the FIFO's correct behavior, including handling of read and write operations, data storage, and retrieval, as well as its response to full and empty conditions.
- Coverage-Driven Verification: Measure and analyze the functional coverage to identify gaps in the test scenarios and to ensure that all critical cases are addressed.
- Scalability and Reusability: Design the testbench components to be modular and reusable for other projects or variations of the FIFO design, adhering to best practices in UVM-based verification.

2. Architecture

The architecture of the FIFO UVM verification environment is designed to ensure maximum test coverage and modularity. It consists of the following key components:

Testbench Hierarchy

- Environment (env): The UVM environment serves as the top-level component that integrates the UVM agents, scoreboard, and coverage monitors. It orchestrates the communication between different components to ensure proper data flow and verification checks.
- UVM Agent: The UVM agent encapsulates the driver, sequencer, and monitor. It acts as an interface between the testbench and the Device Under Test (DUT), generating stimuli and collecting responses.
    - Driver: The driver converts the transaction-level sequences into pin-level signal activities, driving the input signals to the FIFO.
    - Sequencer: The sequencer controls the sequence of transactions that are sent to the driver. It plays a vital role in defining the order of data sent to the FIFO, ensuring the proper stimulus for each test scenario.
    - Monitor: The monitor observes the signal activities and extracts transaction-level data from the DUT's outputs, which are then sent to the scoreboard for comparison.

Scoreboard

The scoreboard is used to verify the functional correctness of the FIFO design by comparing the actual output data from the DUT with the expected data. It leverages a reference model to generate the expected values and highlights any discrepancies between the actual and predicted behavior. This ensures that the FIFO design operates as intended under all test conditions.

Coverage Collection

Functional coverage is a critical component in the verification process to measure the extent to which the FIFO's functionalities have been exercised by the test cases. Covergroups are used to track different scenarios such as:

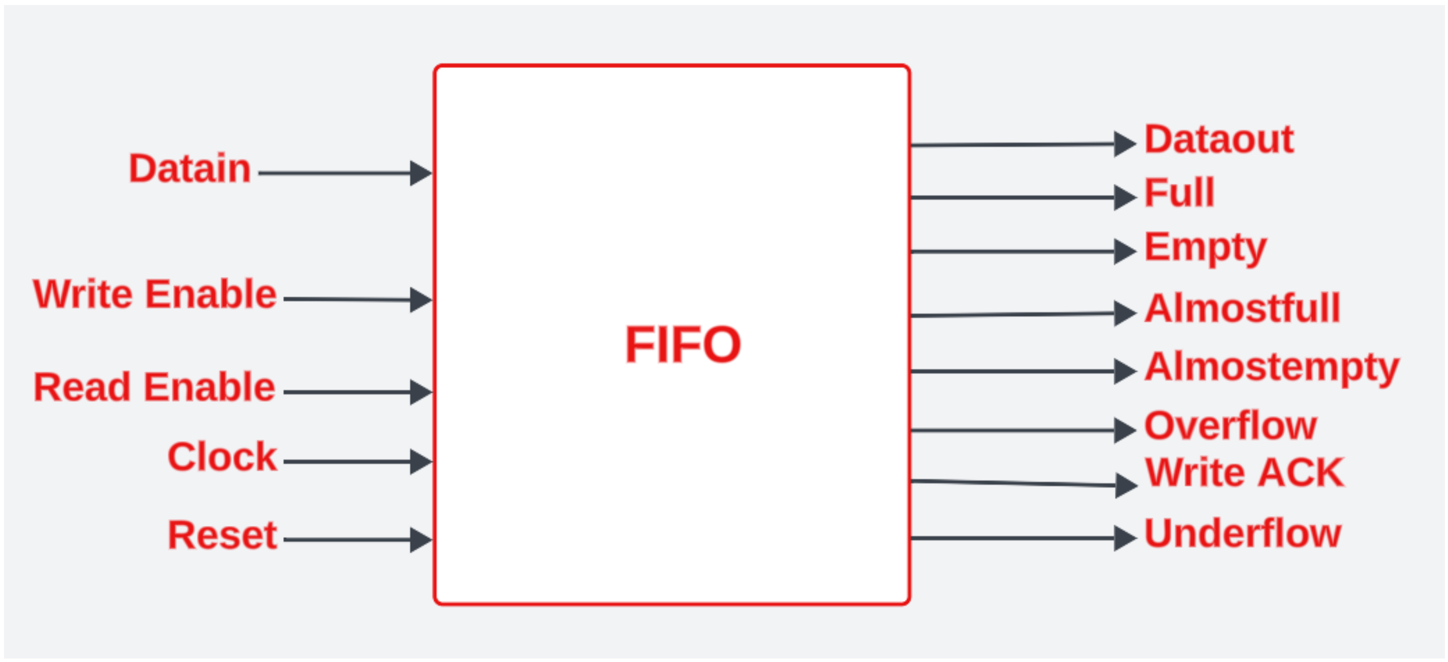
- FIFO Full: Verifies that the FIFO correctly indicates when it reaches its maximum capacity.
- FIFO Empty: Checks that the FIFO accurately signals when all data has been read out.

- Almost Full: Tests the FIFO's behavior when it is nearly full, ensuring that control logic is triggered before overflow occurs.
- Almost Empty: Assesses the FIFO's state when it is almost empty, verifying proper signals before it reaches an empty condition.
- Write Acknowledge (wr_Ack): Confirms that the FIFO acknowledges write operations appropriately, even under high data input rates.

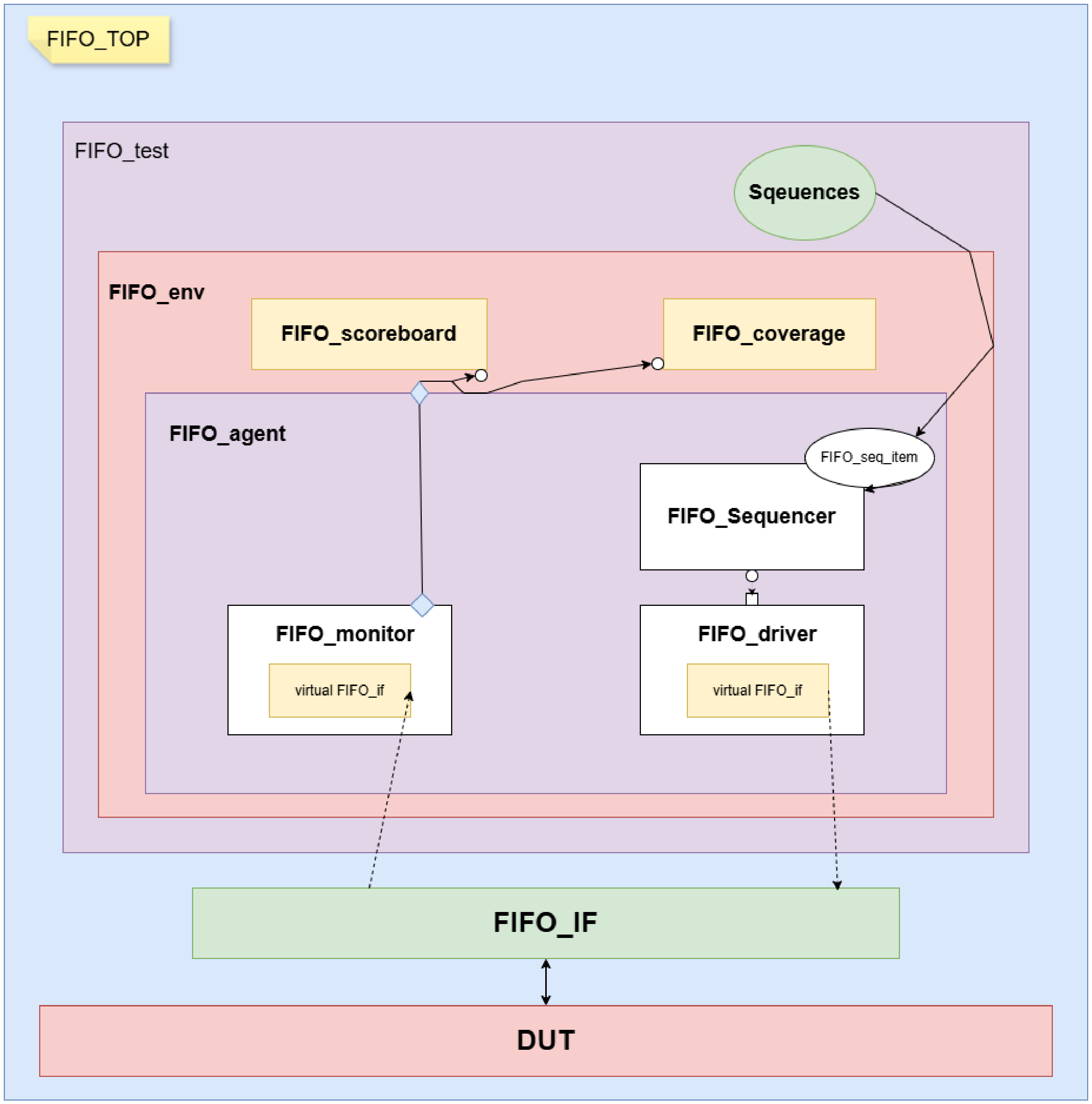
Test Scenarios

The verification environment includes a variety of directed and random test scenarios designed to validate the FIFO's behavior under normal and edge cases. Some of the specific test sequences include:

- Reset Sequence: Verifies that the FIFO properly initializes and resets all internal states upon receiving a reset signal.
- Write-Only Sequence: Tests the FIFO by performing multiple write operations without reading, observing how the FIFO handles data overflow conditions.
- Read-Only Sequence: Evaluates the FIFO’s behavior when only read operations are performed, ensuring correct data output and proper handling of underflow conditions.
- Write and Read Sequence: Simulates simultaneous read and write operations to test the FIFO's capability to handle continuous data flow in both directions.
- Random Sequence: Uses randomization to generate unpredictable read and write .



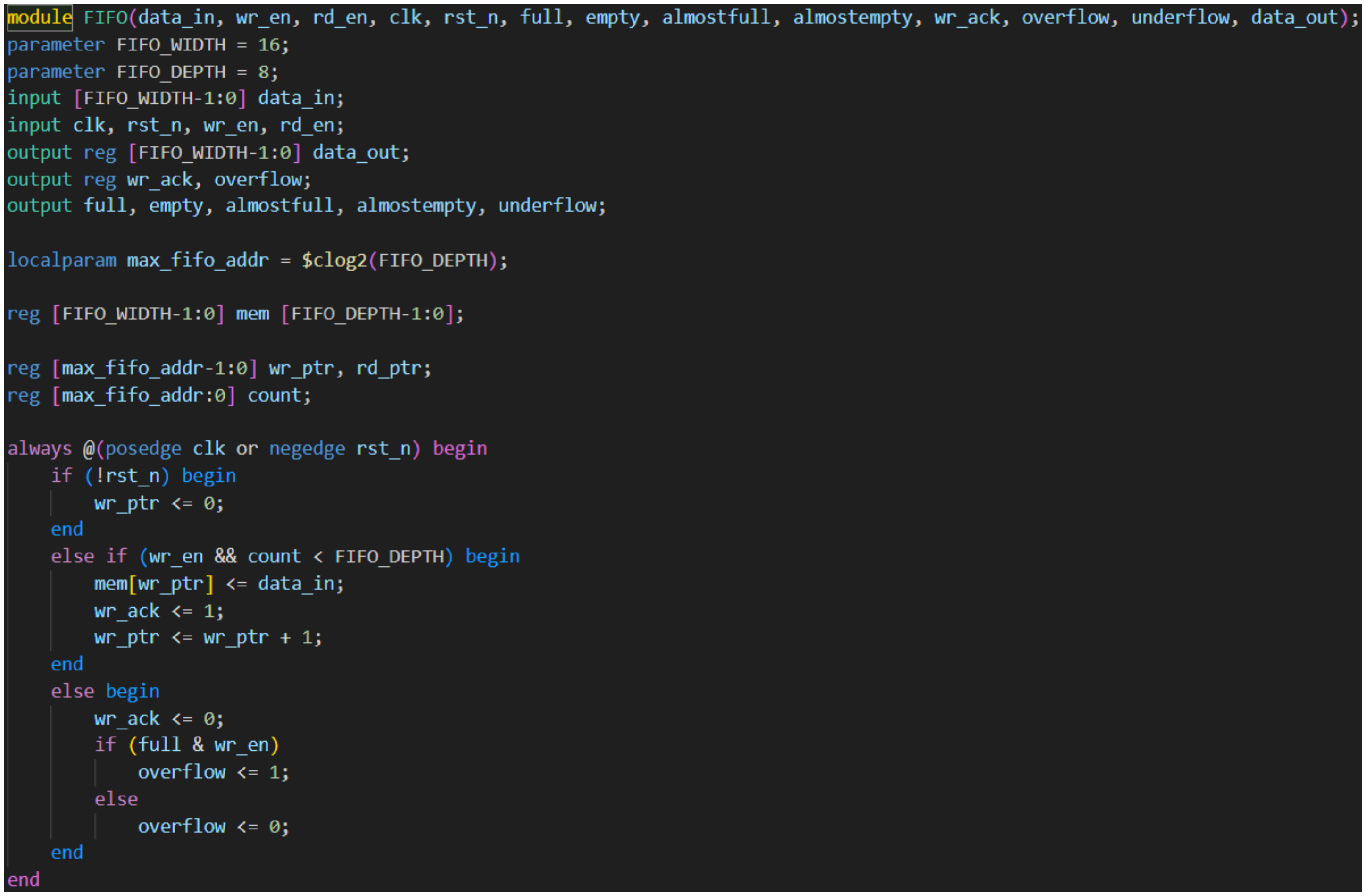
3. Verification Flow



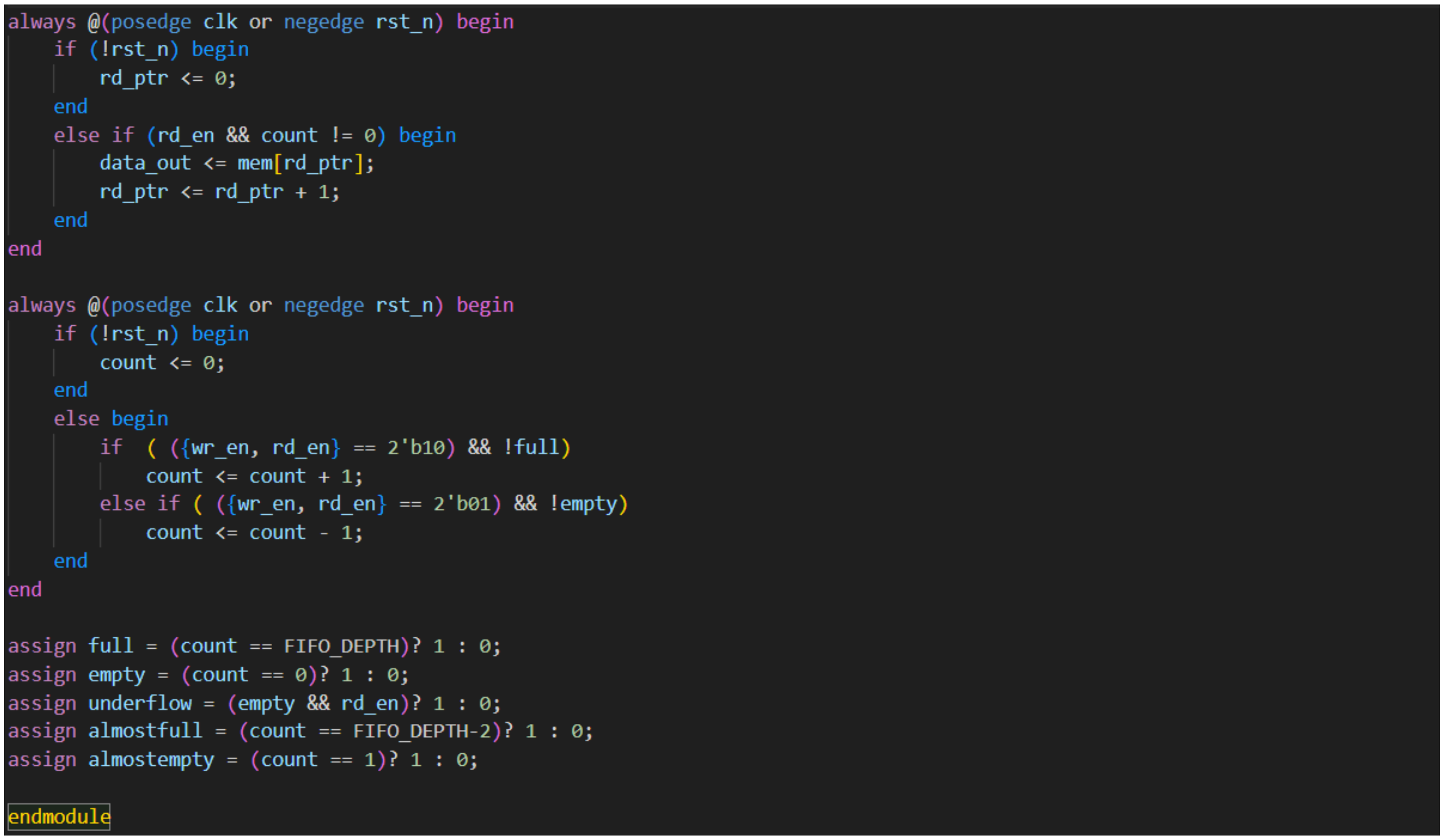
4. Verification Plan

	Description	Stimulus Generation	Functional Coverage	Functionality Check	Assertion
FIFO_1	When the reset is asserted, the output value not change and empty flag is high	Directed at the start of the simulation then randomzied with constraint uisng sequence reset that drive the reset to off most the simulation time	-	A checker in the scoreboard to make sure the output is correct and concurrent assertion to check the empty	(@(posedge clk) (!count) |-> (empty==1))
FIFO_2	when the write and read asserted and the fifo is empty ,the write is higher priority so the output value for almostempty is high	Randomization under constraint on the write enable to be high randomized using sequence write from the simulation time	cover all values of flags	A checker in the scoreboard to make sure the output is correct and concurrent assertion to check the almostempty	(@(posedge clk) (count==1) |-> (almostempty==1))
FIFO_3	when the write and read asserted and the fifo is not empty nor full so the output value make the two order in the same time	Randomization under constraint on the write enable to be high randomized using sequence write and read enable from the simulation time	cover all values of flags	A checker in the scoreboard to make sure the output is correct	
FIFO_4	when the write and read asserted and the fifo is full ,the read is higher priority so the output value for almostfull is high	Randomization under constraint on the write enable to be high randomized using sequence write and read enable from the simulation time	cover all values of flags	A checker in the scoreboard to make sure the output is correct and concurrent assertion to check the almostfull	(@(posedge clk) (count == FIFO_DEPTH-1) |-> (almostfull==1))
FIFO_5	when the write is high and read is low and the fifo is full , so the output value for overflow is high	Randomization under constraint on the write enable to be high randomized from using sequence read only the simulation time	cover all values of flags	A checker in the scoreboard to make sure the output is correct and concurrent assertion to check the overflow	(@(posedge clk) disable iff(!rst_n)(full && wr_en && !rd_en) |=> overflow == 1)
FIFO_6	when the write is low and read is high and the fifo is empty so the output value for underflow is high	Randomization under constraint on the write enable to be high randomized using random sequence from the simulation time	cover all values of flags	A checker in the scoreboard to make sure the output is correct and concurrent assertion to check the underflow	(@(posedge clk)disable iff(!rst_n) (rd_en&&!wr_en&&empty) |=>underflow==1)
FIFO_7	Random	Using radom sequence	cover all values of flags	A checker in the scoreboard	(@(posedge clk) (count == FIFO_DEPTH) |-> (full==1))
FIFO_8	Random and the wr_Ack is low when it is full	Using radom sequence	cover all values of flags	A checker in the scoreboard	(@(posedge clk)disable iff(!rst_n ||(!wr_en&&!rd_en)) (full |=> !wr_ack))

5.Design



```
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg wr_ack, overflow;
output full, empty, almostfull, almostempty, underflow;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

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

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

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
```



```
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

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

assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign underflow = (empty && rd_en)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;

endmodule
```

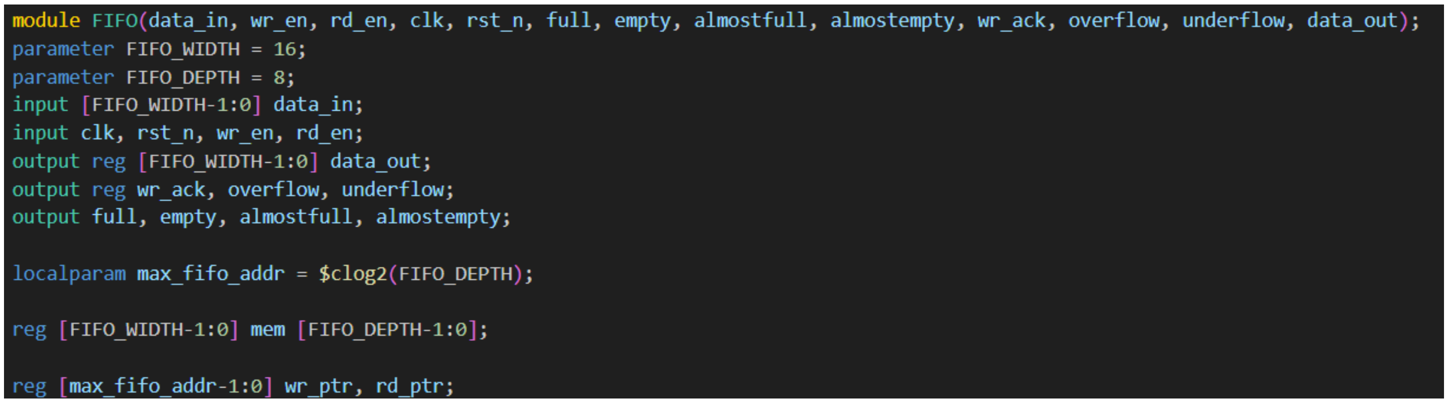
6.Bugs

- The underflow flag should be registered.
- Reset the flags (empty – almostempty – wr_ack – full – alomstfull).
- Add cases.
    - If the wr_en , rd_en and empty is high so the write is the higher priority and counter decreases .
    - If the wr_en , rd_en and full is high so the Read is the higher priority and counter increases .
    - If the wr_en and rd_en is high and not full nor empty so the two cases run together but the counter not changed.
    - If the wr_en and rd_en is low so everything not changed.

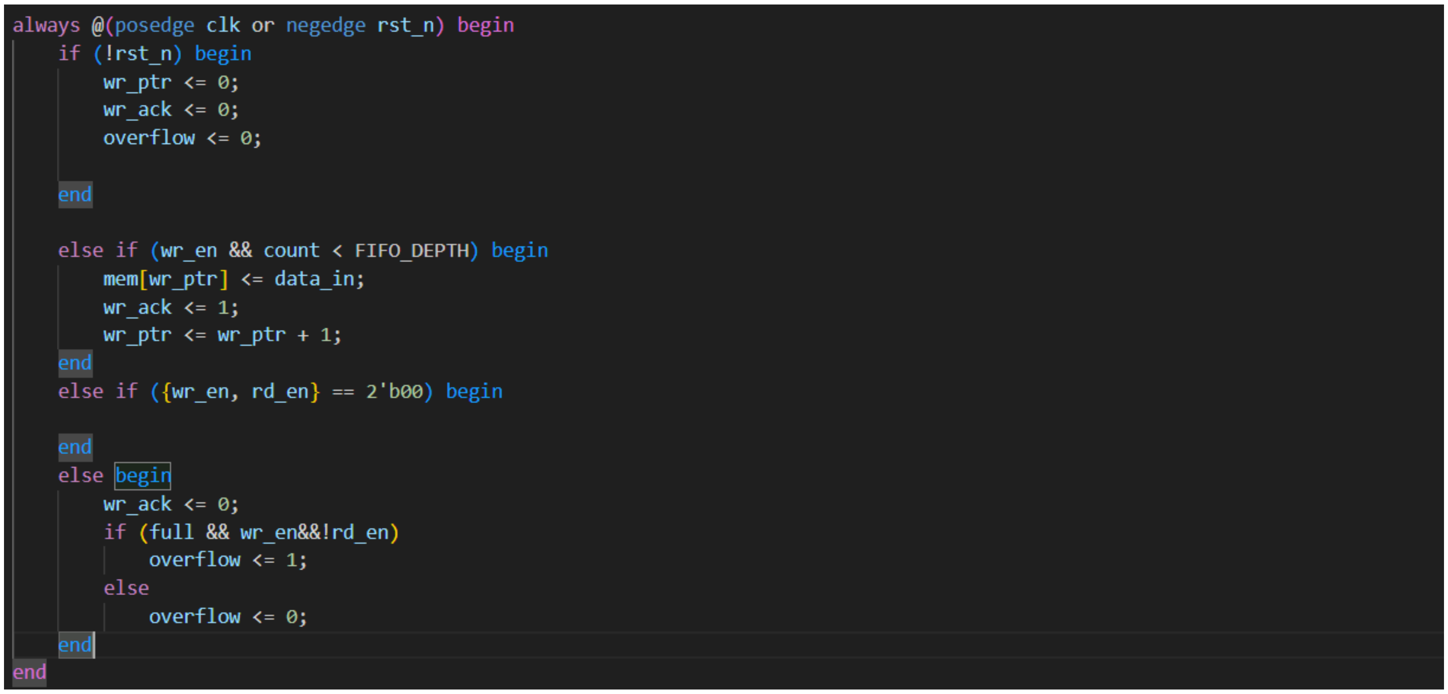
- The almostfull must be high if the FIFO need one input to be high So I edit it
- Edit underflow, overflow and count

7. Verification Files

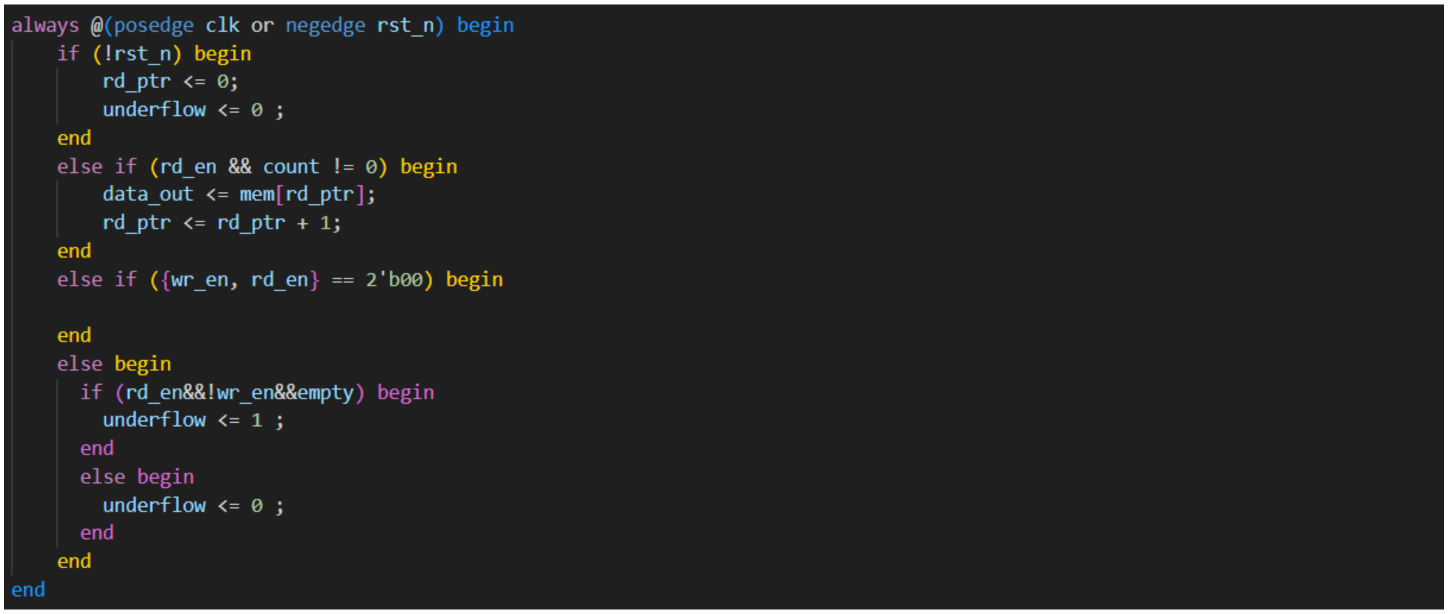
7.1 Design After Edit



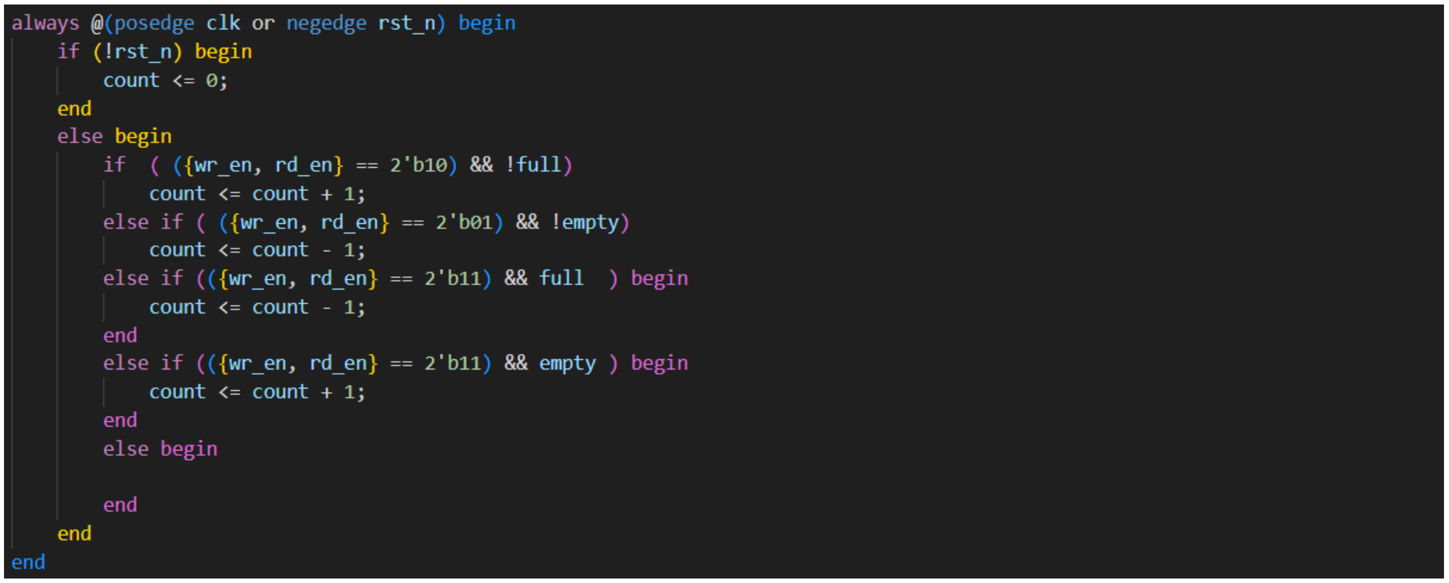
```verilog
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg wr_ack, overflow, underflow;
output full, empty, almostfull, almostempty;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

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

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



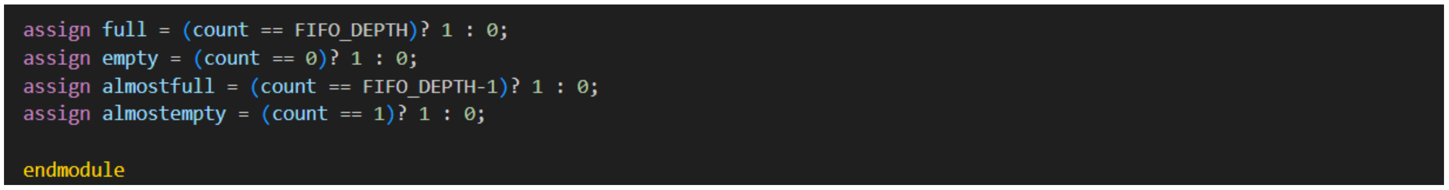
```verilog
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
        wr_ack <= 0;
        overflow <= 0;

    end

    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else if ({wr_en, rd_en} == 2'b00) begin

    end
    else begin
        wr_ack <= 0;
        if (full && wr_en&&!rd_en)
            overflow <= 1;
        else
            overflow <= 0;
    end
end
```



```verilog
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
        underflow <= 0 ;
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else if ({wr_en, rd_en} == 2'b00) begin

    end
    else begin
      if (rd_en&&!wr_en&&empty) begin
        underflow <= 1 ;
      end
      else begin
        underflow <= 0 ;
      end
    end
end
```

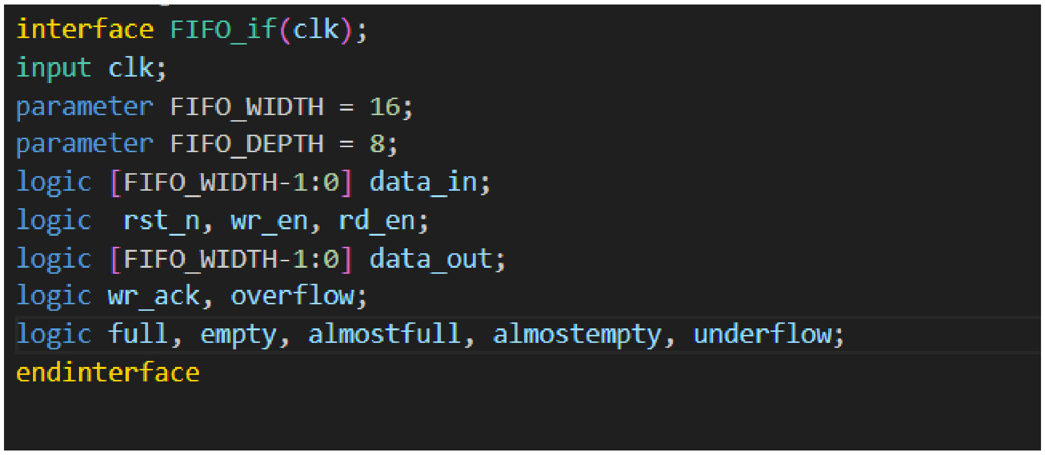


```verilog
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
        else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

        end
    end
end
```



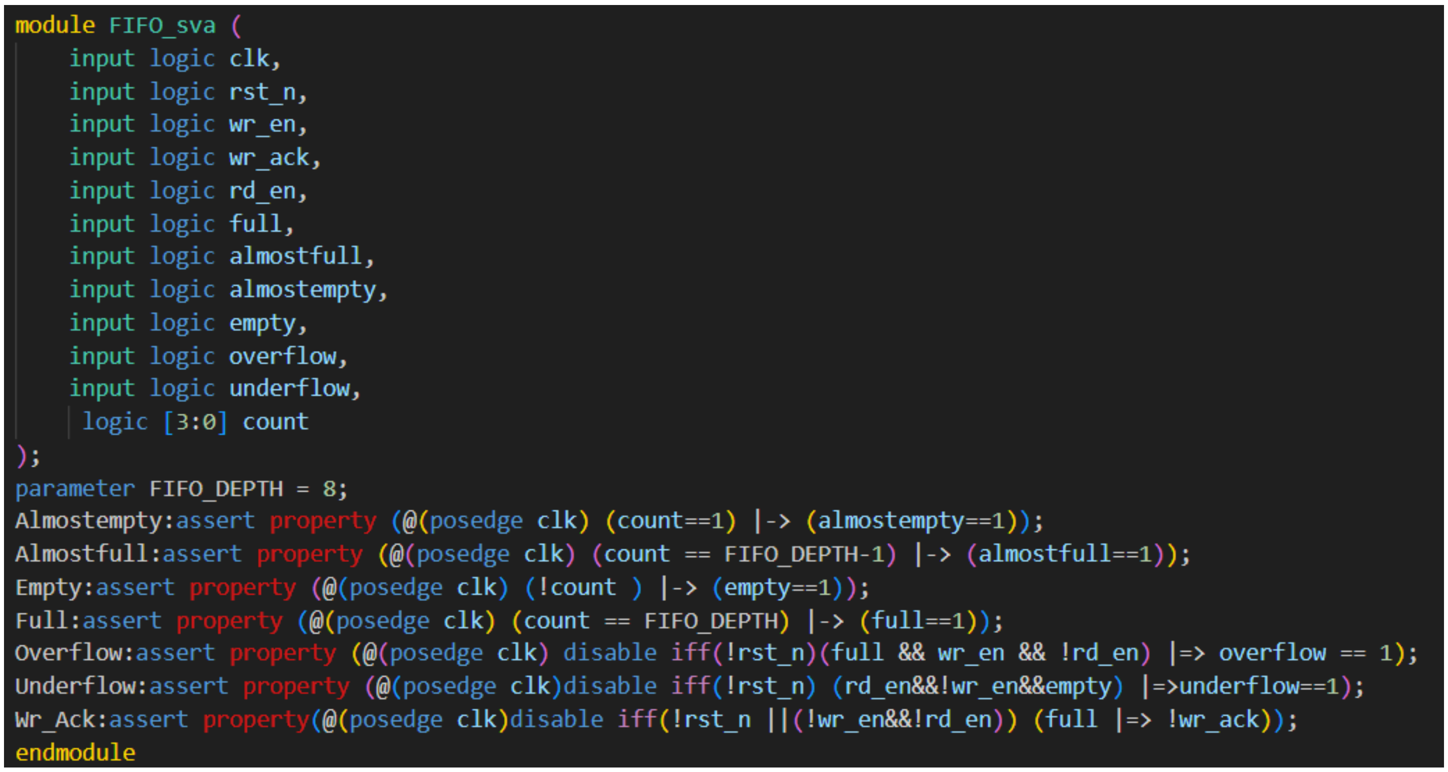
```verilog
assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;

endmodule
```

7.2 FIFO Interface



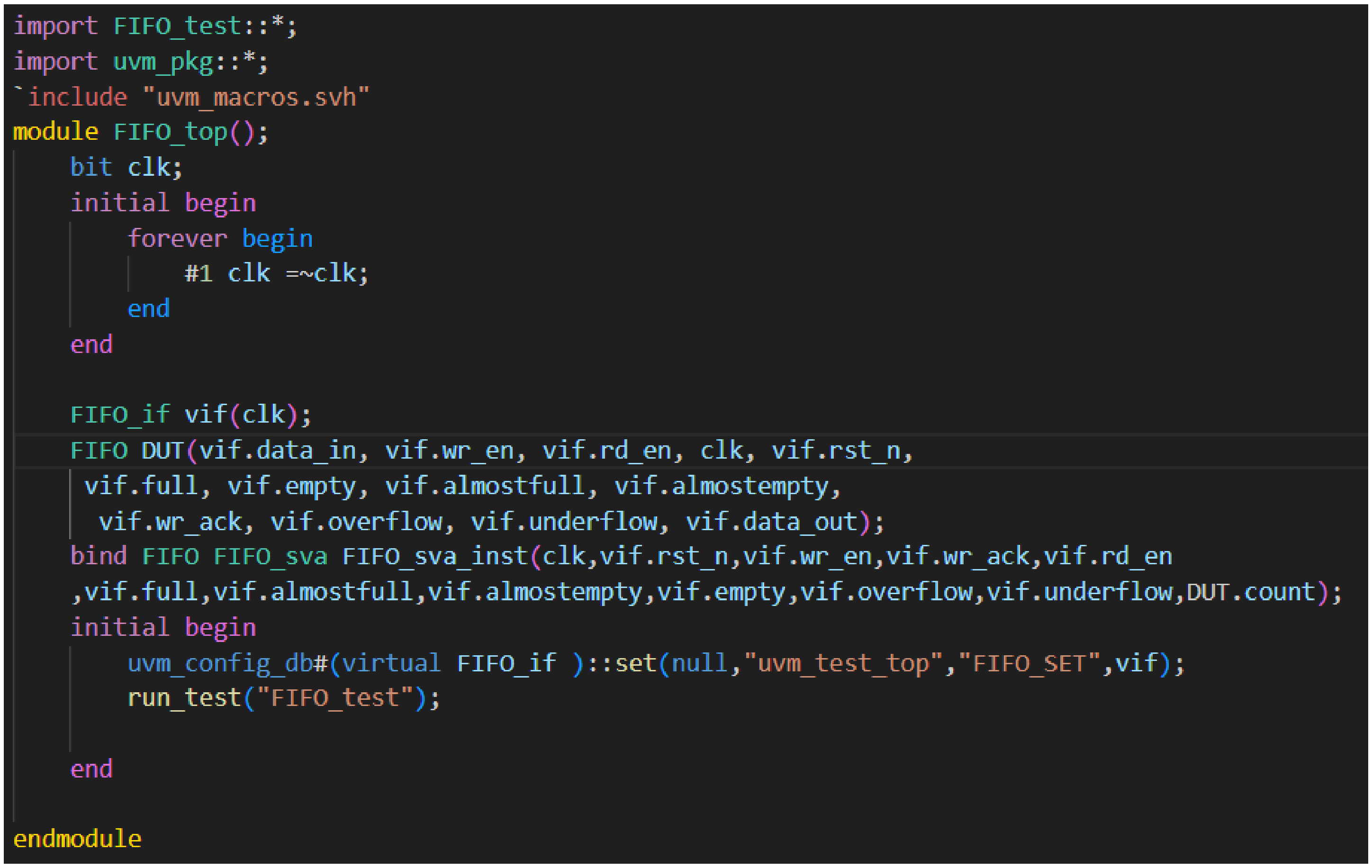
```
interface FIFO_if(clk);
input clk;
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
logic [FIFO_WIDTH-1:0] data_in;
logic  rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
endinterface
```

7.3 FIFO Assertion



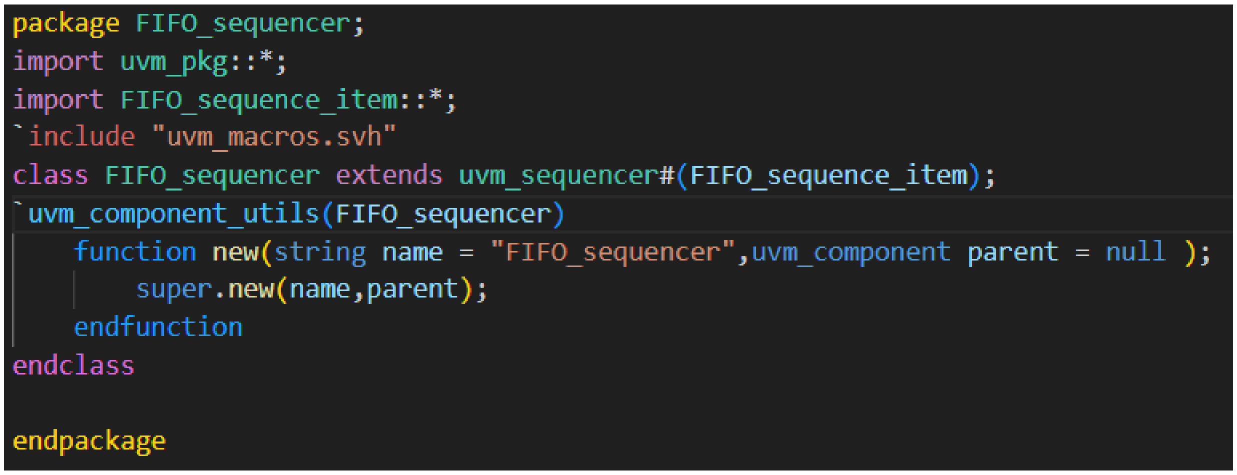
```
module FIFO_sva (
    input logic clk,
    input logic rst_n,
    input logic wr_en,
    input logic wr_ack,
    input logic rd_en,
    input logic full,
    input logic almostfull,
    input logic almostempty,
    input logic empty,
    input logic overflow,
    input logic underflow,
     logic [3:0] count
);
parameter FIFO_DEPTH = 8;
Almostempty:assert property (@(posedge clk) (count==1) |-> (almostempty==1));
Almostfull:assert property (@(posedge clk) (count == FIFO_DEPTH-1) |-> (almostfull==1));
Empty:assert property (@(posedge clk) (!count ) |-> (empty==1));
Full:assert property (@(posedge clk) (count == FIFO_DEPTH) |-> (full==1));
Overflow:assert property (@(posedge clk) disable iff(!rst_n)(full && wr_en && !rd_en) |=> overflow == 1);
Underflow:assert property (@(posedge clk)disable iff(!rst_n) (rd_en&&!wr_en&&empty) |=>underflow==1);
Wr_Ack:assert property(@(posedge clk)disable iff(!rst_n ||(!wr_en&&!rd_en)) (full |=> !wr_ack));
endmodule
```

7.4 FIFO TOP



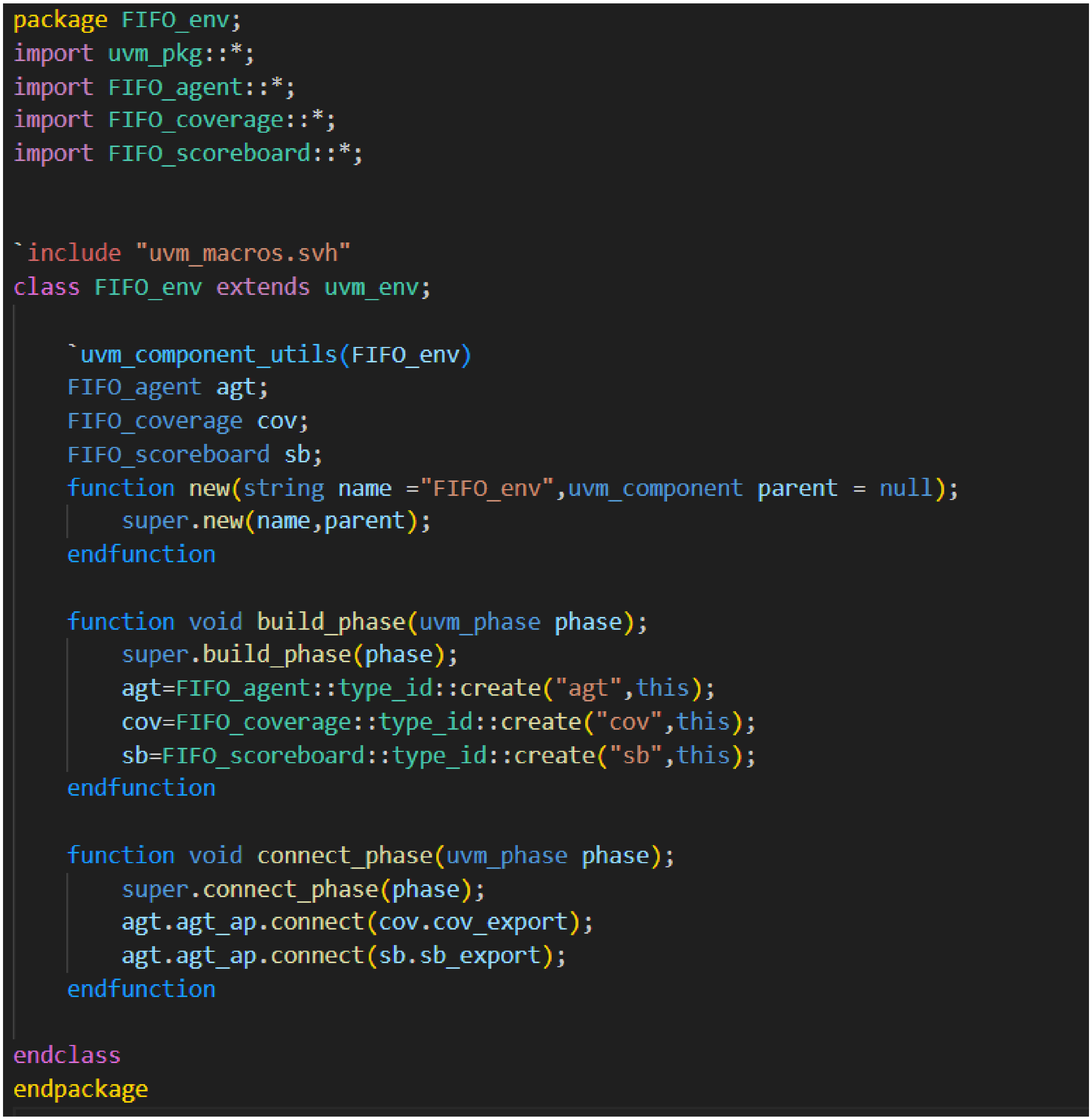
```
import FIFO_test::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
module FIFO_top();
    bit clk;
    initial begin
        forever begin
            #1 clk =~clk;
        end
    end

    FIFO_if vif(clk);
    FIFO DUT(vif.data_in, vif.wr_en, vif.rd_en, clk, vif.rst_n,
     vif.full, vif.empty, vif.almostfull, vif.almostempty,
      vif.wr_ack, vif.overflow, vif.underflow, vif.data_out);
    bind FIFO FIFO_sva FIFO_sva_inst(clk,vif.rst_n,vif.wr_en,vif.wr_ack,vif.rd_en
,vif.full,vif.almostfull,vif.almostempty,vif.empty,vif.overflow,vif.underflow,DUT.count);
    initial begin
        uvm_config_db#(virtual FIFO_if )::set(null,"uvm_test_top","FIFO_SET",vif);
        run_test("FIFO_test");

    end

endmodule
```

7.5 FIFO Sequencer



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

endpackage
```

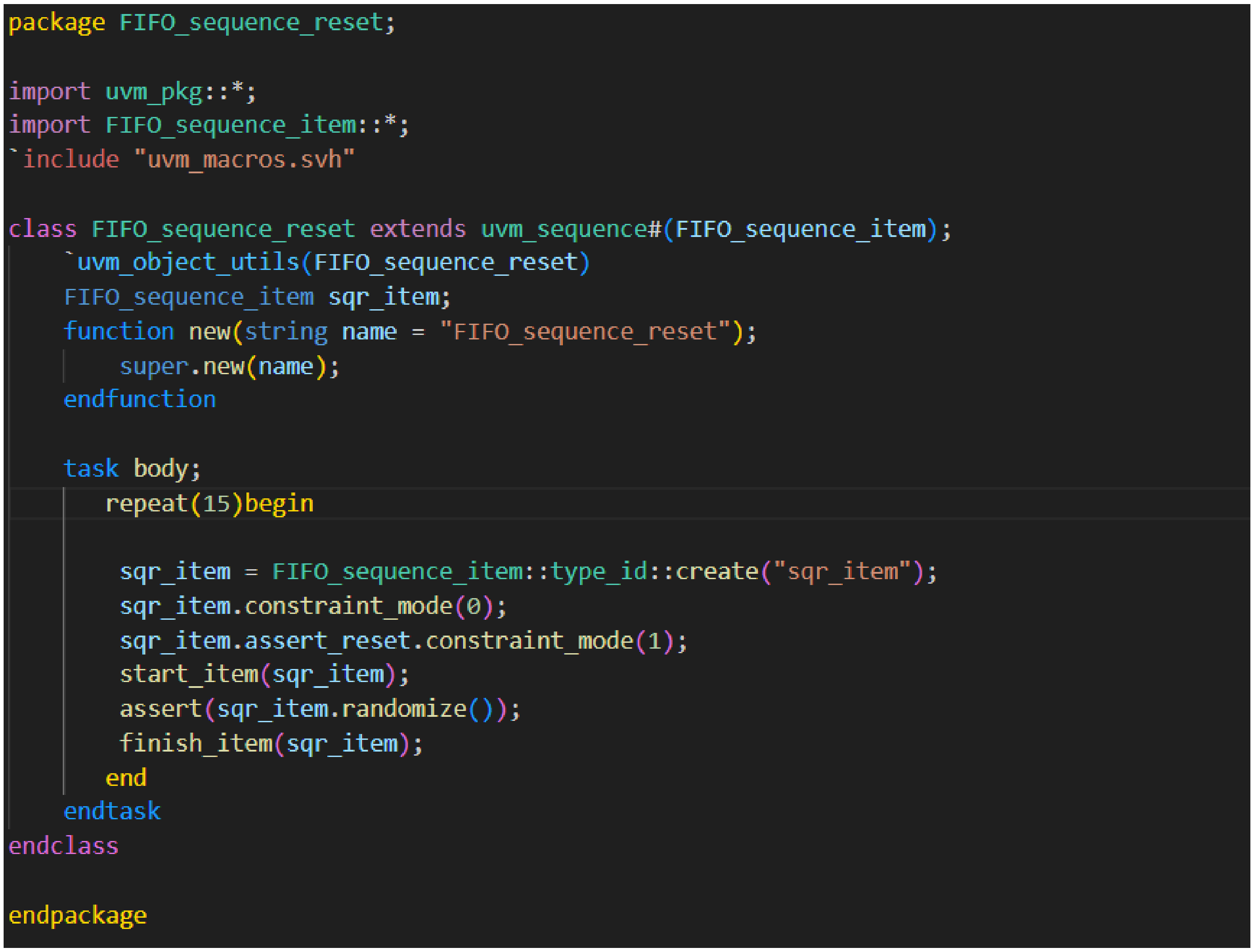
7.6 FIFO Environment



```
package FIFO_env;
import uvm_pkg::*;
import FIFO_agent::*;
import FIFO_coverage::*;
import FIFO_scoreboard::*;


`include "uvm_macros.svh"
class FIFO_env extends uvm_env;

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

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

endclass
endpackage
```

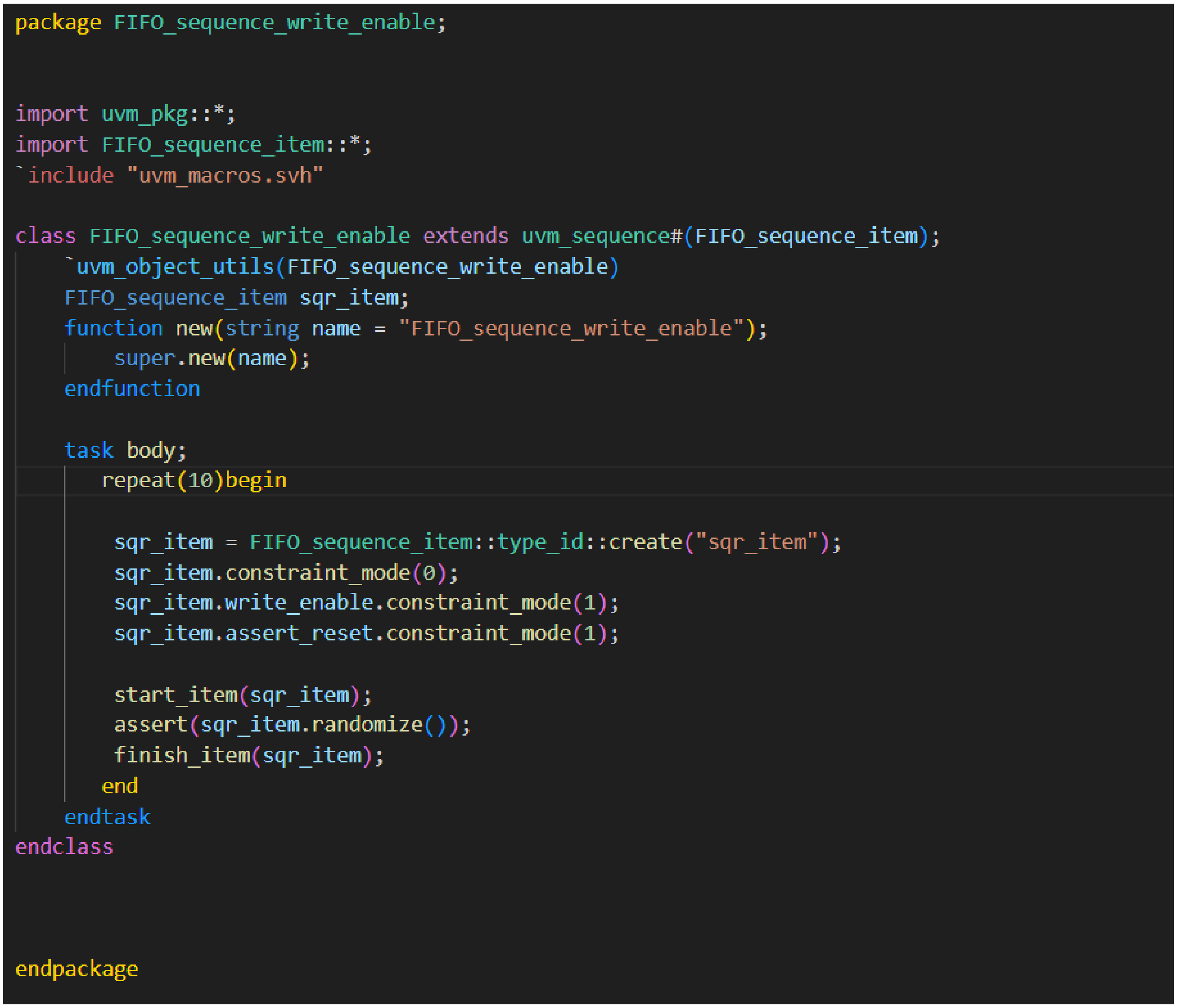
7.7 FIFO Sequences

7.7.1 RESET



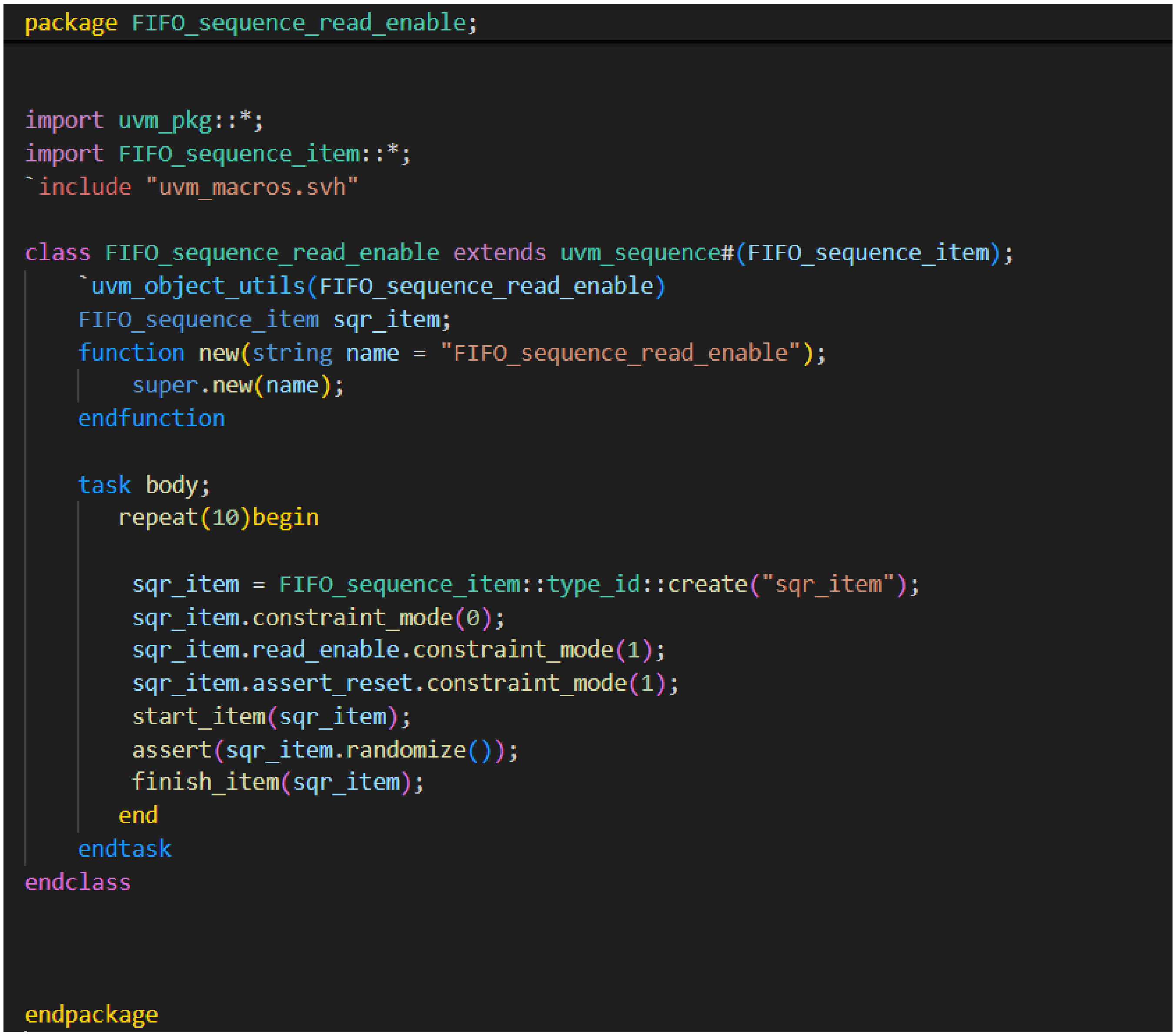
```
package FIFO_sequence_reset;

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

class FIFO_sequence_reset extends uvm_sequence#(FIFO_sequence_item);
    `uvm_object_utils(FIFO_sequence_reset)
    FIFO_sequence_item sqr_item;
    function new(string name = "FIFO_sequence_reset");
        super.new(name);
    endfunction

    task body;
       repeat(15)begin

        sqr_item = FIFO_sequence_item::type_id::create("sqr_item");
        sqr_item.constraint_mode(0);
        sqr_item.assert_reset.constraint_mode(1);
        start_item(sqr_item);
        assert(sqr_item.randomize());
        finish_item(sqr_item);
       end
    endtask
endclass

endpackage
```

7.7.2 Write Only



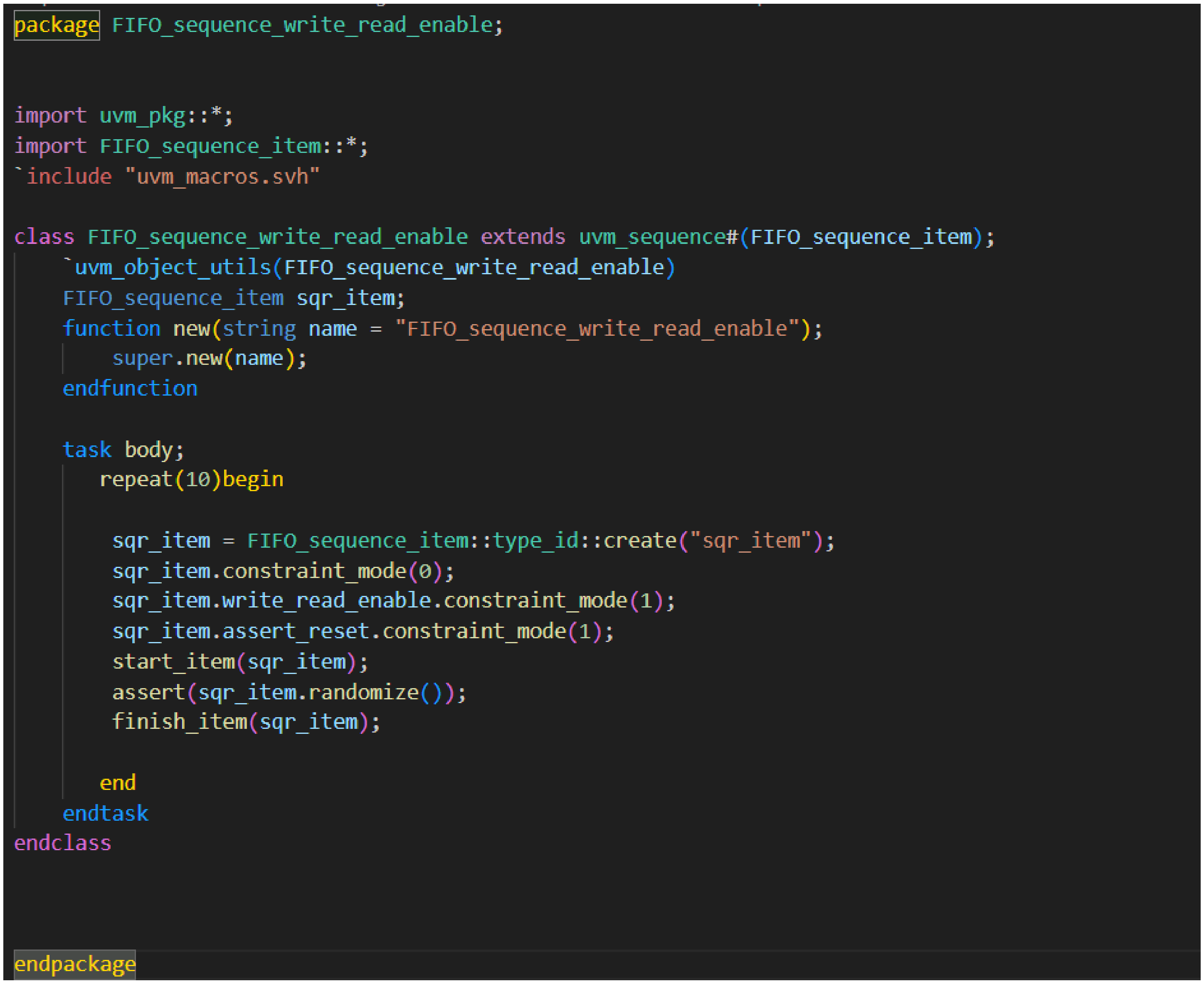
```
package FIFO_sequence_write_enable;


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

class FIFO_sequence_write_enable extends uvm_sequence#(FIFO_sequence_item);
    `uvm_object_utils(FIFO_sequence_write_enable)
    FIFO_sequence_item sqr_item;
    function new(string name = "FIFO_sequence_write_enable");
        super.new(name);
    endfunction

    task body;
      repeat(10)begin

        sqr_item = FIFO_sequence_item::type_id::create("sqr_item");
        sqr_item.constraint_mode(0);
        sqr_item.write_enable.constraint_mode(1);
        sqr_item.assert_reset.constraint_mode(1);

        start_item(sqr_item);
        assert(sqr_item.randomize());
        finish_item(sqr_item);
      end
    endtask
endclass



endpackage
```

7.7.3 Read Only



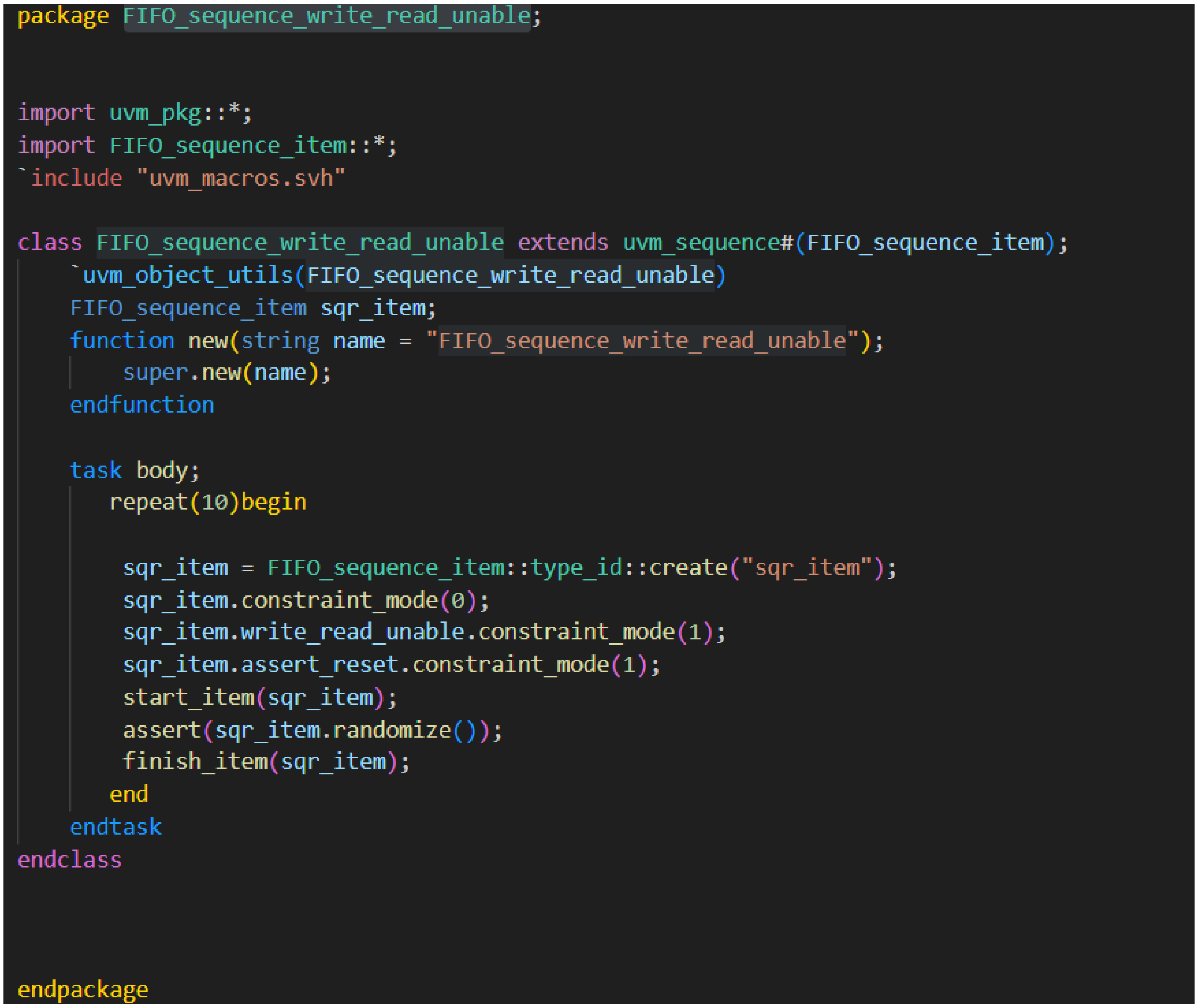
```
package FIFO_sequence_read_enable;


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

class FIFO_sequence_read_enable extends uvm_sequence#(FIFO_sequence_item);
    `uvm_object_utils(FIFO_sequence_read_enable)
    FIFO_sequence_item sqr_item;
    function new(string name = "FIFO_sequence_read_enable");
        super.new(name);
    endfunction

    task body;
       repeat(10)begin

        sqr_item = FIFO_sequence_item::type_id::create("sqr_item");
        sqr_item.constraint_mode(0);
        sqr_item.read_enable.constraint_mode(1);
        sqr_item.assert_reset.constraint_mode(1);
        start_item(sqr_item);
        assert(sqr_item.randomize());
        finish_item(sqr_item);
       end
    endtask
endclass



endpackage
```

7.7.4 Read and Write Enable



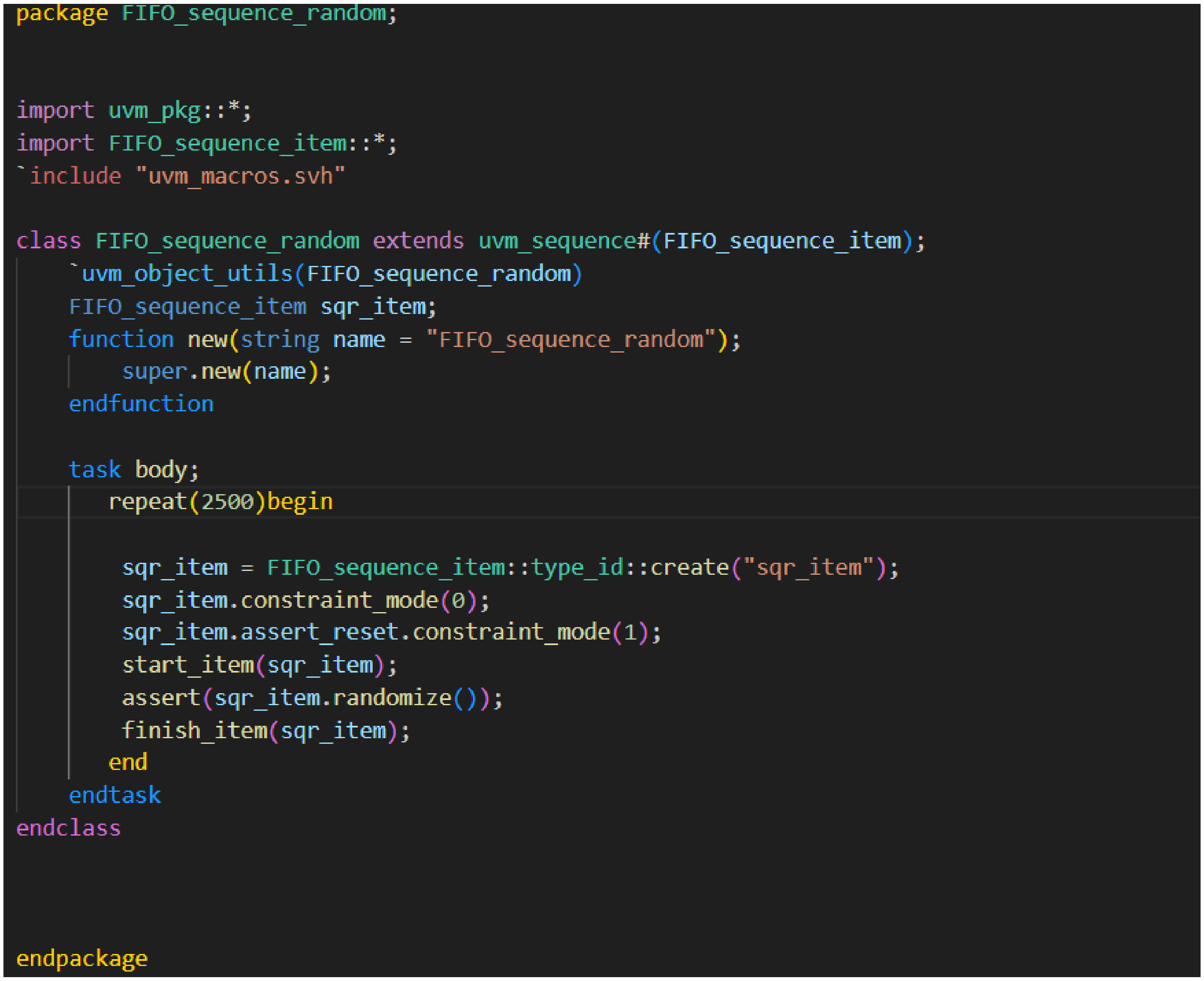
```
package FIFO_sequence_write_read_enable;


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

class FIFO_sequence_write_read_enable extends uvm_sequence#(FIFO_sequence_item);
    `uvm_object_utils(FIFO_sequence_write_read_enable)
    FIFO_sequence_item sqr_item;
    function new(string name = "FIFO_sequence_write_read_enable");
        super.new(name);
    endfunction

    task body;
       repeat(10)begin

        sqr_item = FIFO_sequence_item::type_id::create("sqr_item");
        sqr_item.constraint_mode(0);
        sqr_item.write_read_enable.constraint_mode(1);
        sqr_item.assert_reset.constraint_mode(1);
        start_item(sqr_item);
        assert(sqr_item.randomize());
        finish_item(sqr_item);

       end
    endtask
endclass



endpackage
```

7.7.5 Write and Read Disable



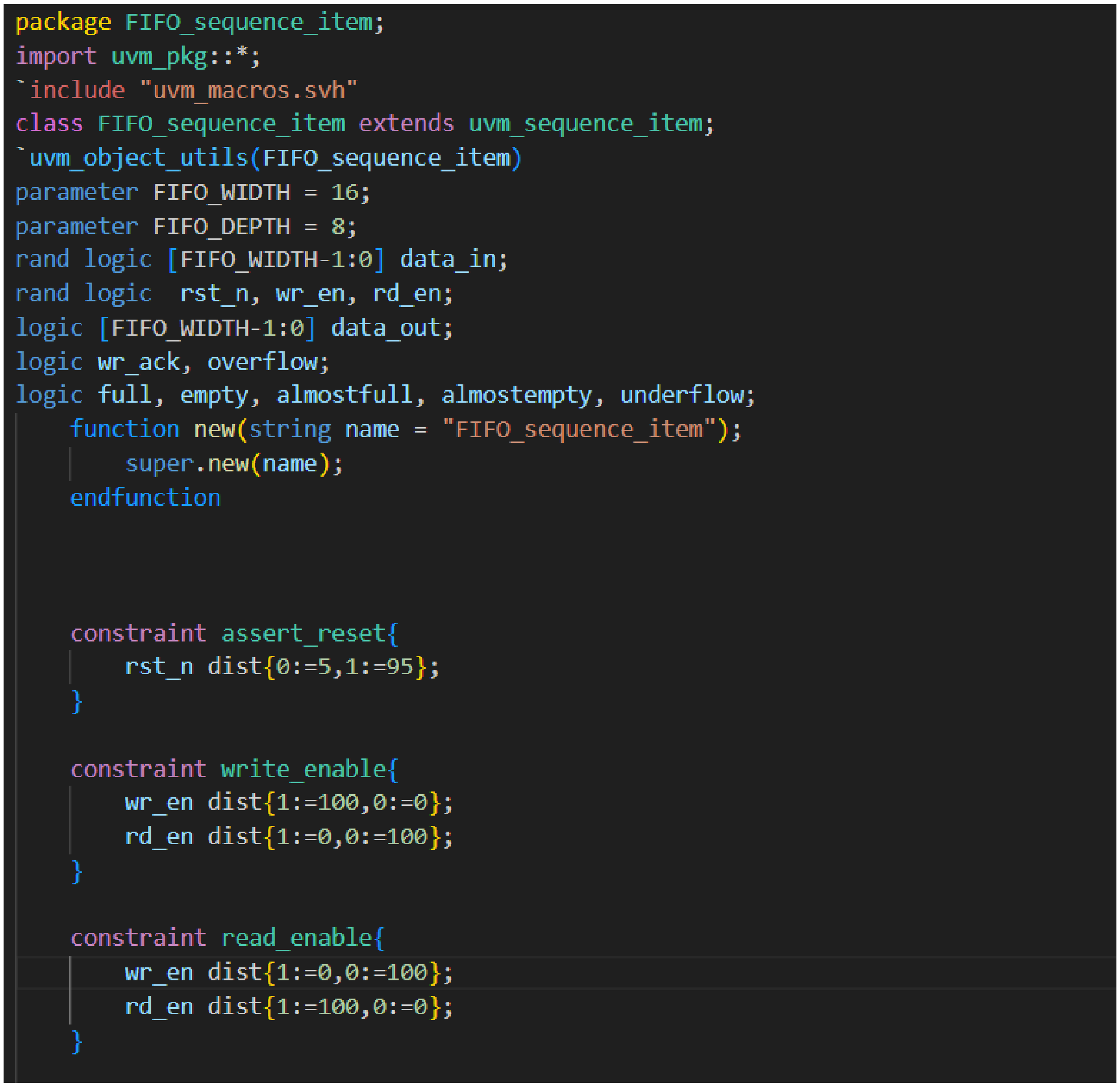
```
package FIFO_sequence_write_read_unable;


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

class FIFO_sequence_write_read_unable extends uvm_sequence#(FIFO_sequence_item);
    `uvm_object_utils(FIFO_sequence_write_read_unable)
    FIFO_sequence_item sqr_item;
    function new(string name = "FIFO_sequence_write_read_unable");
        super.new(name);
    endfunction

    task body;
       repeat(10)begin

        sqr_item = FIFO_sequence_item::type_id::create("sqr_item");
        sqr_item.constraint_mode(0);
        sqr_item.write_read_unable.constraint_mode(1);
        sqr_item.assert_reset.constraint_mode(1);
        start_item(sqr_item);
        assert(sqr_item.randomize());
        finish_item(sqr_item);
       end
    endtask
endclass



endpackage
```

7.7.6 Random

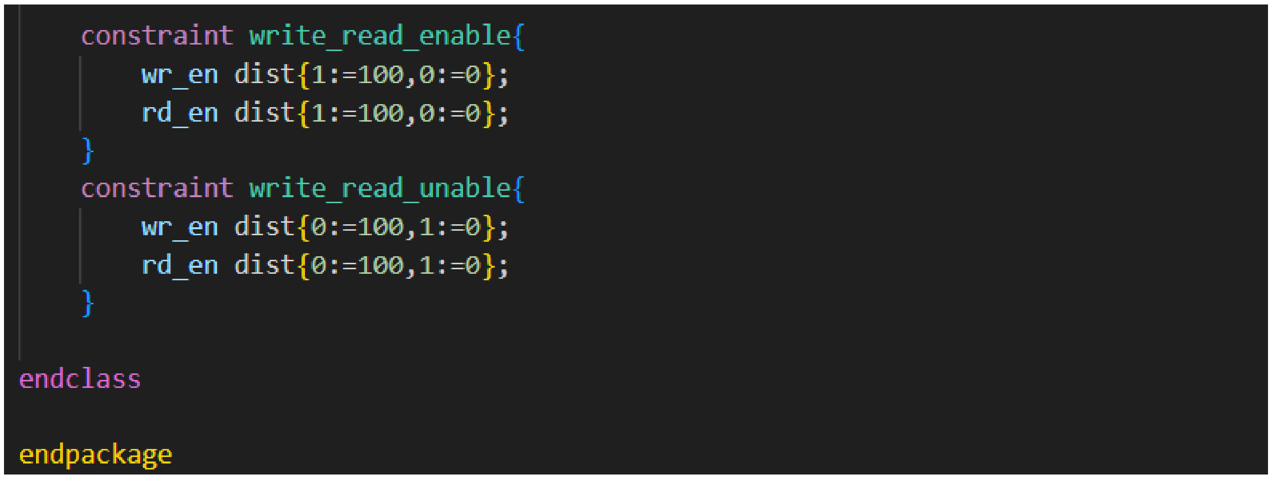


```
package FIFO_sequence_random;


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

class FIFO_sequence_random extends uvm_sequence#(FIFO_sequence_item);
    `uvm_object_utils(FIFO_sequence_random)
    FIFO_sequence_item sqr_item;
    function new(string name = "FIFO_sequence_random");
        super.new(name);
    endfunction

    task body;
       repeat(2500)begin

        sqr_item = FIFO_sequence_item::type_id::create("sqr_item");
        sqr_item.constraint_mode(0);
        sqr_item.assert_reset.constraint_mode(1);
        start_item(sqr_item);
        assert(sqr_item.randomize());
        finish_item(sqr_item);
       end
    endtask
endclass



endpackage
```

7.8 FIFO Sequence item

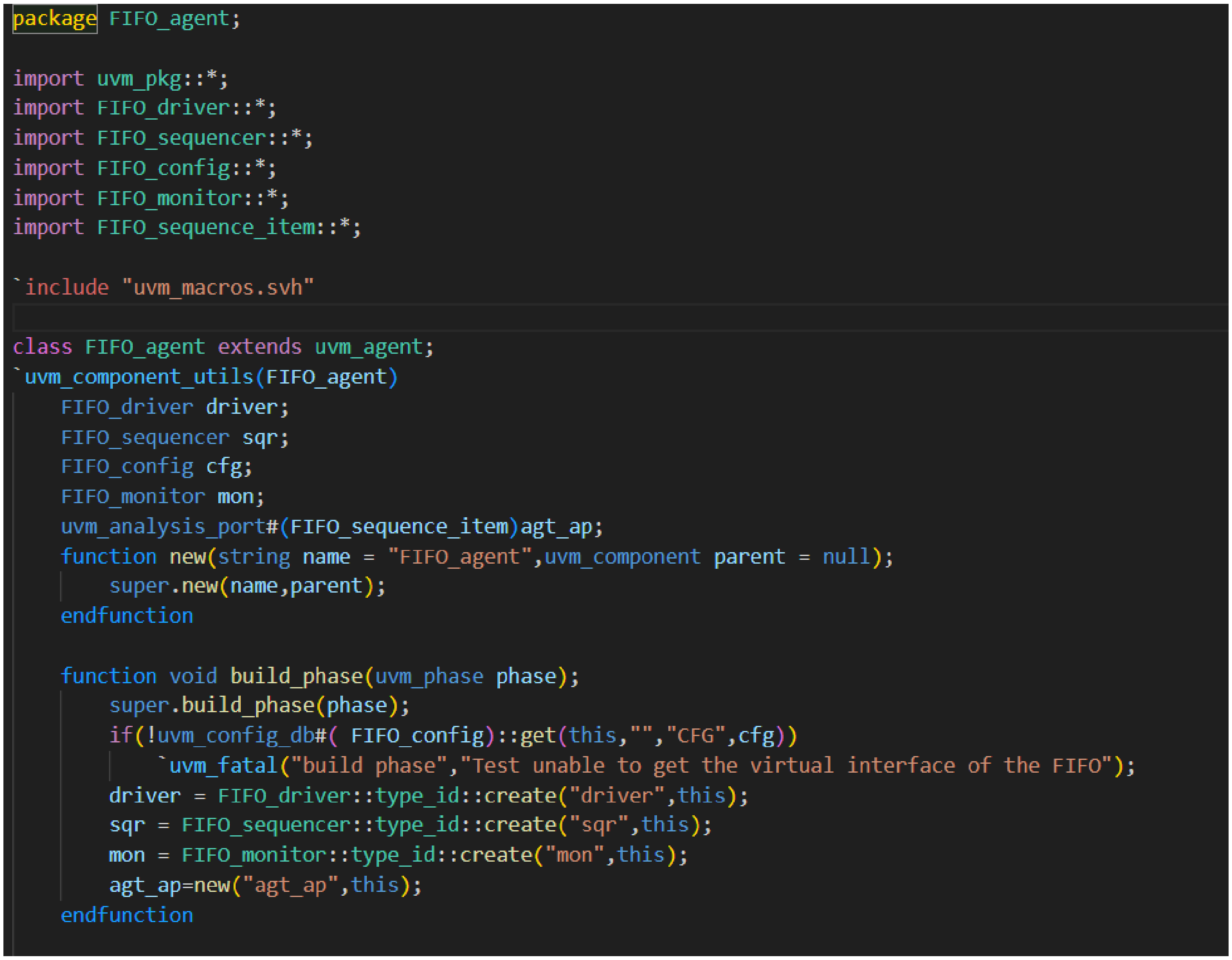


```
package FIFO_sequence_item;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_sequence_item extends uvm_sequence_item;
`uvm_object_utils(FIFO_sequence_item)
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
rand logic [FIFO_WIDTH-1:0] data_in;
rand logic  rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
    function new(string name = "FIFO_sequence_item");
        super.new(name);
    endfunction



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

    constraint write_enable{
        wr_en dist{1:=100,0:=0};
        rd_en dist{1:=0,0:=100};
    }

    constraint read_enable{
        wr_en dist{1:=0,0:=100};
        rd_en dist{1:=100,0:=0};
    }
```

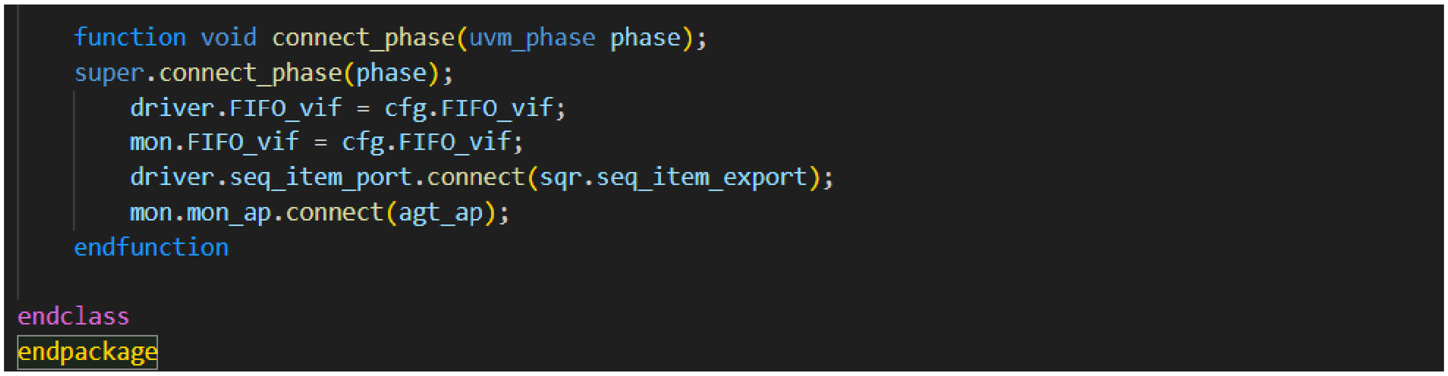


```
    constraint write_read_enable{
        wr_en dist{1:=100,0:=0};
        rd_en dist{1:=100,0:=0};
    }
    constraint write_read_unable{
        wr_en dist{0:=100,1:=0};
        rd_en dist{0:=100,1:=0};
    }

endclass

endpackage
```

7.9 FIFO Agent

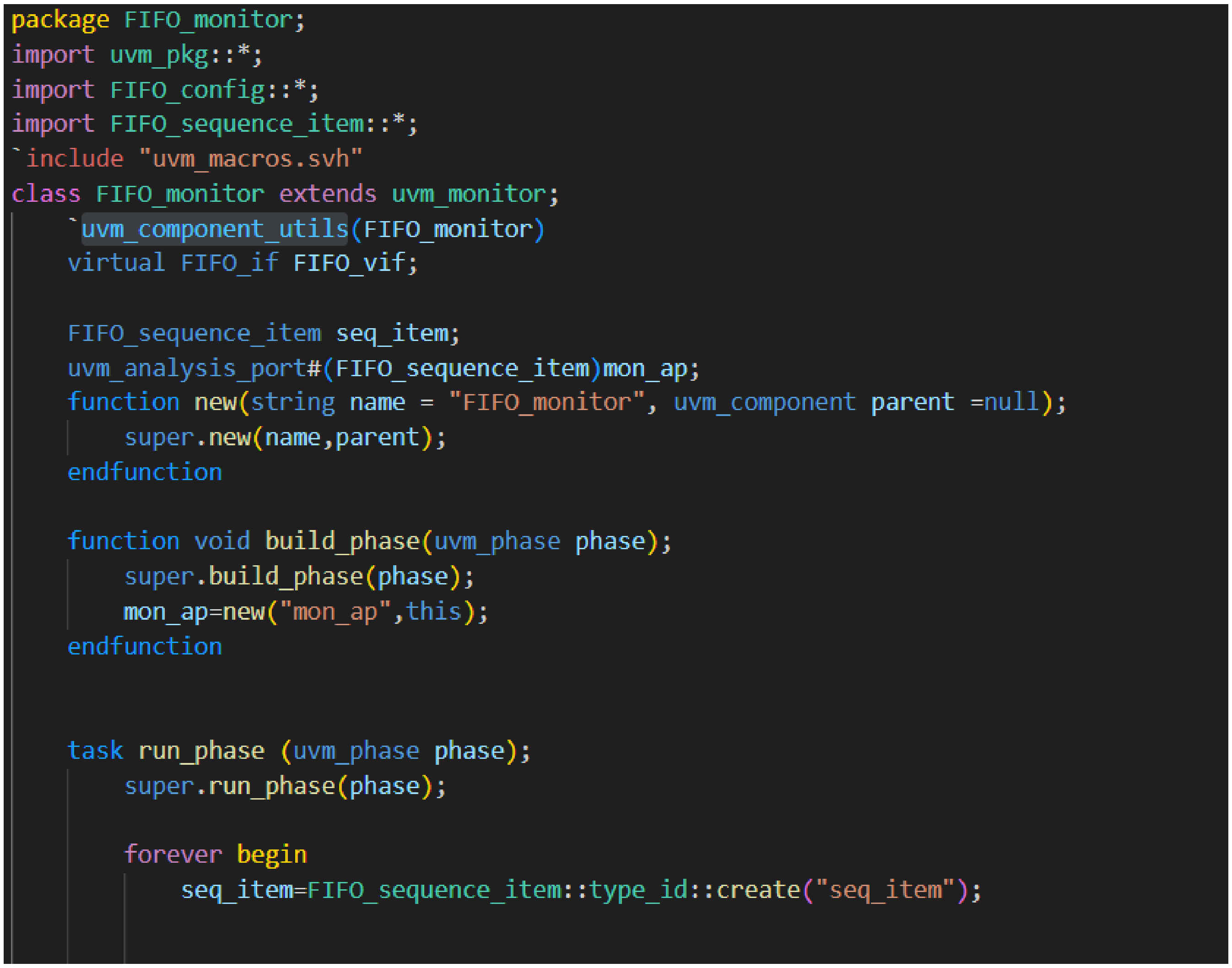
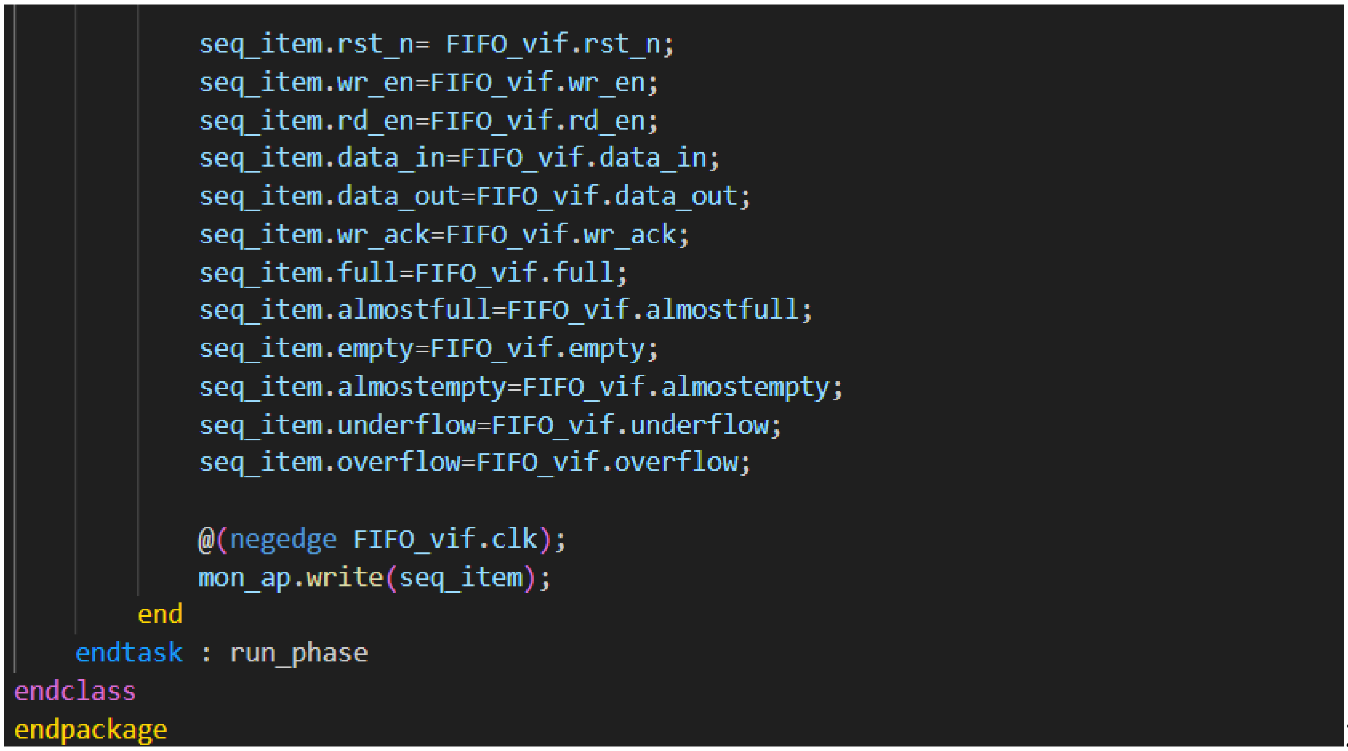


```
package FIFO_agent;

import uvm_pkg::*;
import FIFO_driver::*;
import FIFO_sequencer::*;
import FIFO_config::*;
import FIFO_monitor::*;
import FIFO_sequence_item::*;

`include "uvm_macros.svh"

class FIFO_agent extends uvm_agent;
`uvm_component_utils(FIFO_agent)
    FIFO_driver driver;
    FIFO_sequencer sqr;
    FIFO_config cfg;
    FIFO_monitor mon;
    uvm_analysis_port#(FIFO_sequence_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",cfg))
            `uvm_fatal("build phase","Test unable to get the virtual interface of the FIFO");
        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
```



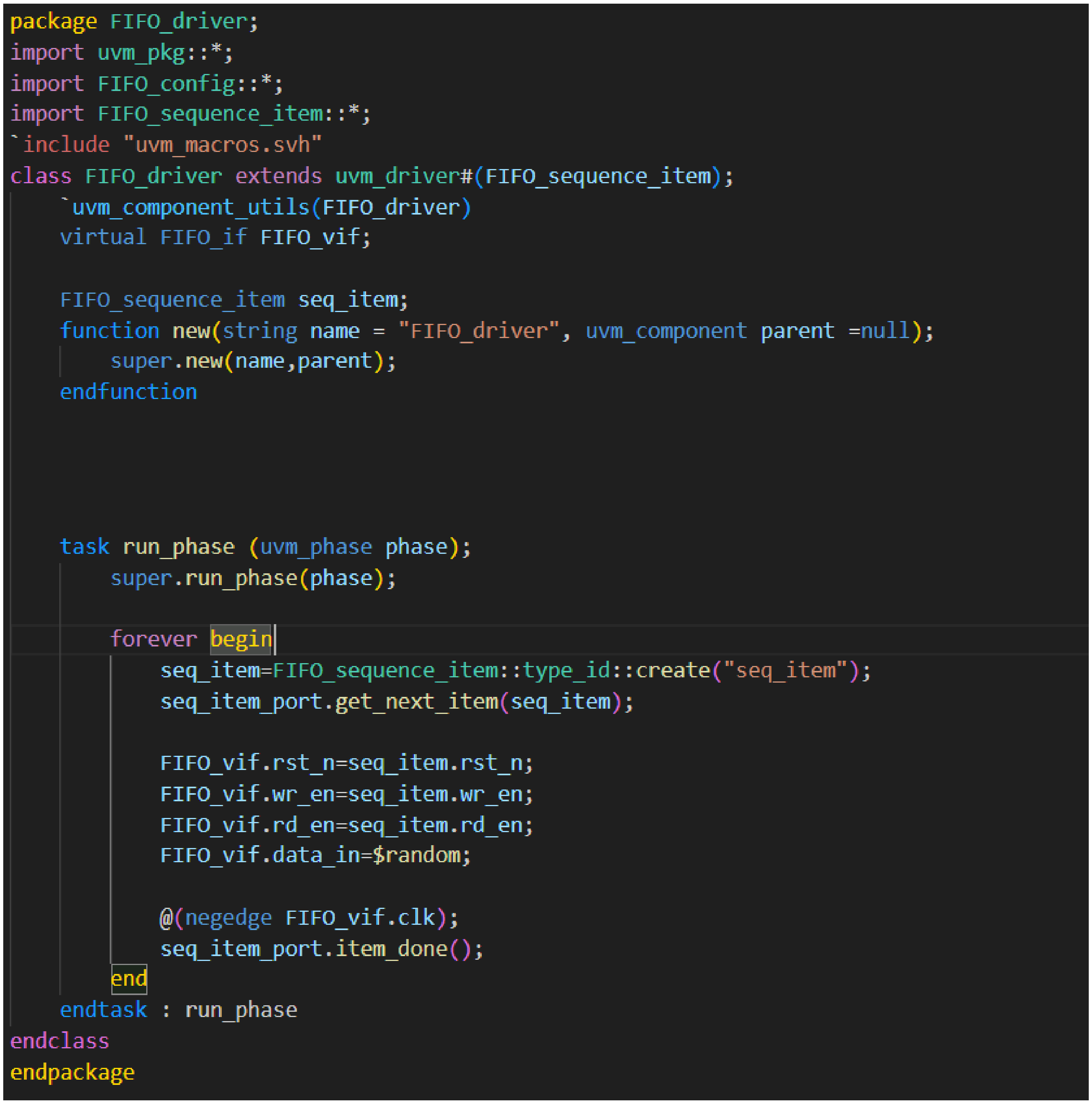
```
    function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
        driver.FIFO_vif = cfg.FIFO_vif;
        mon.FIFO_vif = cfg.FIFO_vif;
        driver.seq_item_port.connect(sqr.seq_item_export);
        mon.mon_ap.connect(agt_ap);
    endfunction

endclass
endpackage
```

7.10 FIFO Monitor

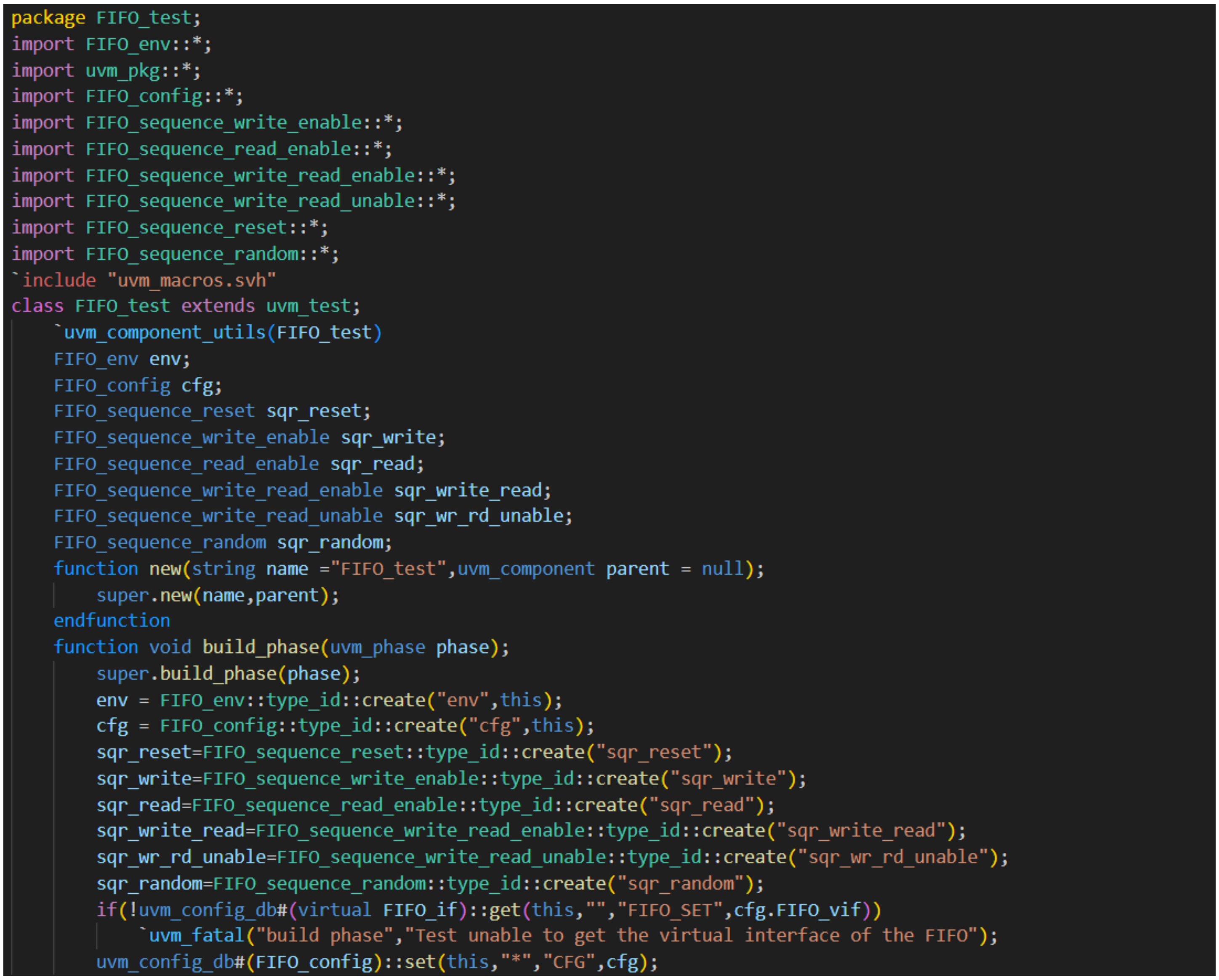
```
package FIFO_monitor;
import uvm_pkg::*;
import FIFO_config::*;
import FIFO_sequence_item::*;
`include "uvm_macros.svh"
class FIFO_monitor extends uvm_monitor;
    `uvm_component_utils(FIFO_monitor)
    virtual FIFO_if FIFO_vif;

    FIFO_sequence_item seq_item;
    uvm_analysis_port#(FIFO_sequence_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
            seq_item=FIFO_sequence_item::type_id::create("seq_item");

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

            @(negedge FIFO_vif.clk);
            mon_ap.write(seq_item);
        end
    endtask : run_phase
endclass
endpackage
```

7.11 FIFO Driver

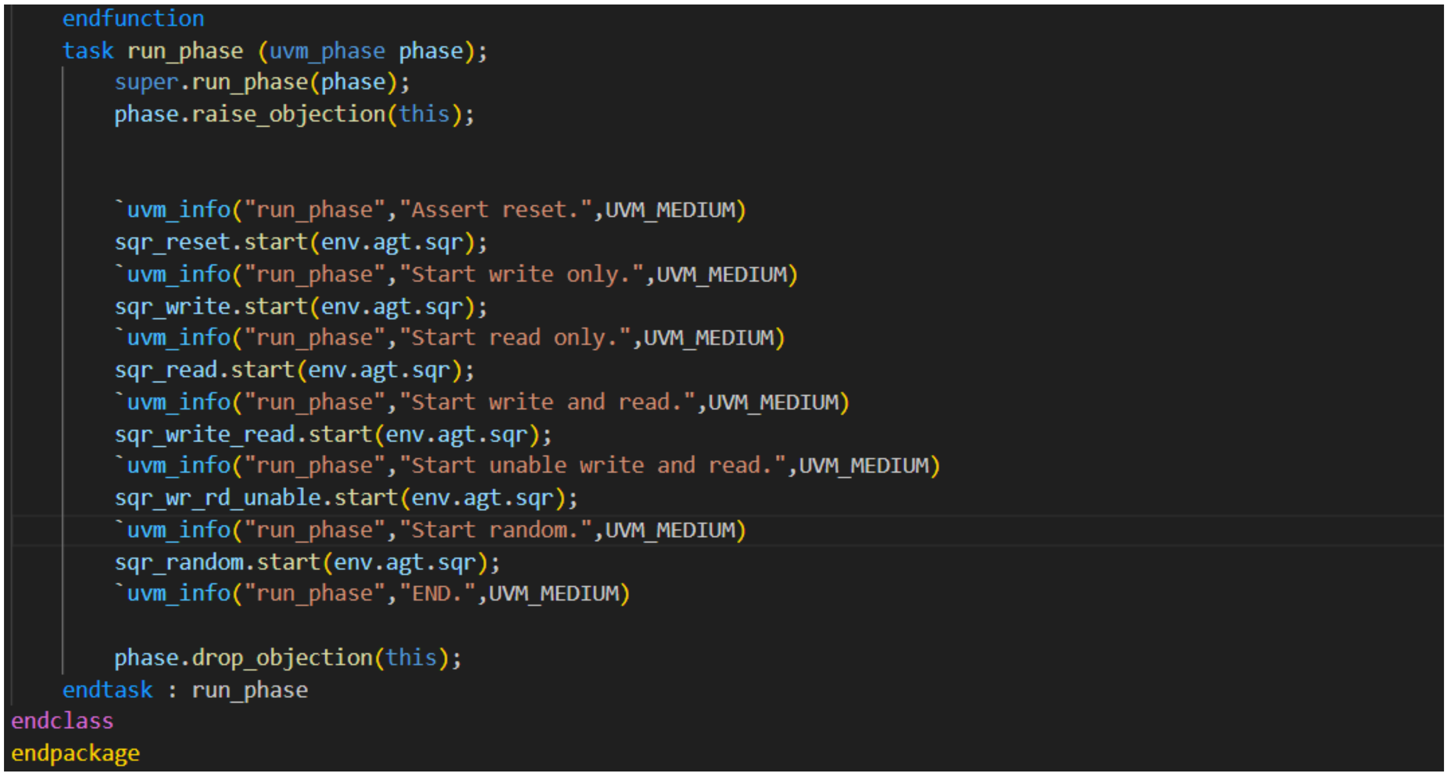


```
package FIFO_driver;
import uvm_pkg::*;
import FIFO_config::*;
import FIFO_sequence_item::*;
`include "uvm_macros.svh"
class FIFO_driver extends uvm_driver#(FIFO_sequence_item);
    `uvm_component_utils(FIFO_driver)
    virtual FIFO_if FIFO_vif;

    FIFO_sequence_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_sequence_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=$random;

            @(negedge FIFO_vif.clk);
            seq_item_port.item_done();
        end
    endtask : run_phase
endclass
endpackage
```

7.12 FIFO Test

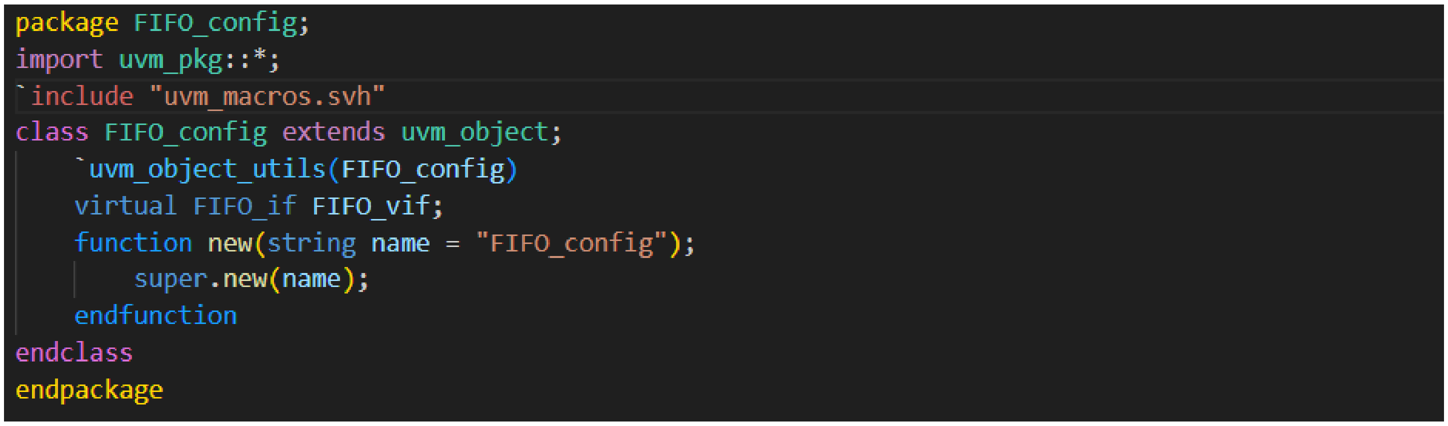


```
package FIFO_test;
import FIFO_env::*;
import uvm_pkg::*;
import FIFO_config::*;
import FIFO_sequence_write_enable::*;
import FIFO_sequence_read_enable::*;
import FIFO_sequence_write_read_enable::*;
import FIFO_sequence_write_read_unable::*;
import FIFO_sequence_reset::*;
import FIFO_sequence_random::*;
`include "uvm_macros.svh"
class FIFO_test extends uvm_test;
    `uvm_component_utils(FIFO_test)
    FIFO_env env;
    FIFO_config cfg;
    FIFO_sequence_reset sqr_reset;
    FIFO_sequence_write_enable sqr_write;
    FIFO_sequence_read_enable sqr_read;
    FIFO_sequence_write_read_enable sqr_write_read;
    FIFO_sequence_write_read_unable sqr_wr_rd_unable;
    FIFO_sequence_random sqr_random;
    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);
        cfg = FIFO_config::type_id::create("cfg",this);
        sqr_reset=FIFO_sequence_reset::type_id::create("sqr_reset");
        sqr_write=FIFO_sequence_write_enable::type_id::create("sqr_write");
        sqr_read=FIFO_sequence_read_enable::type_id::create("sqr_read");
        sqr_write_read=FIFO_sequence_write_read_enable::type_id::create("sqr_write_read");
        sqr_wr_rd_unable=FIFO_sequence_write_read_unable::type_id::create("sqr_wr_rd_unable");
        sqr_random=FIFO_sequence_random::type_id::create("sqr_random");
        if(!uvm_config_db#(virtual FIFO_if)::get(this,"","FIFO_SET",cfg.FIFO_vif))
            `uvm_fatal("build phase","Test unable to get the virtual interface of the FIFO");
        uvm_config_db#(FIFO_config)::set(this,"*","CFG",cfg);
```



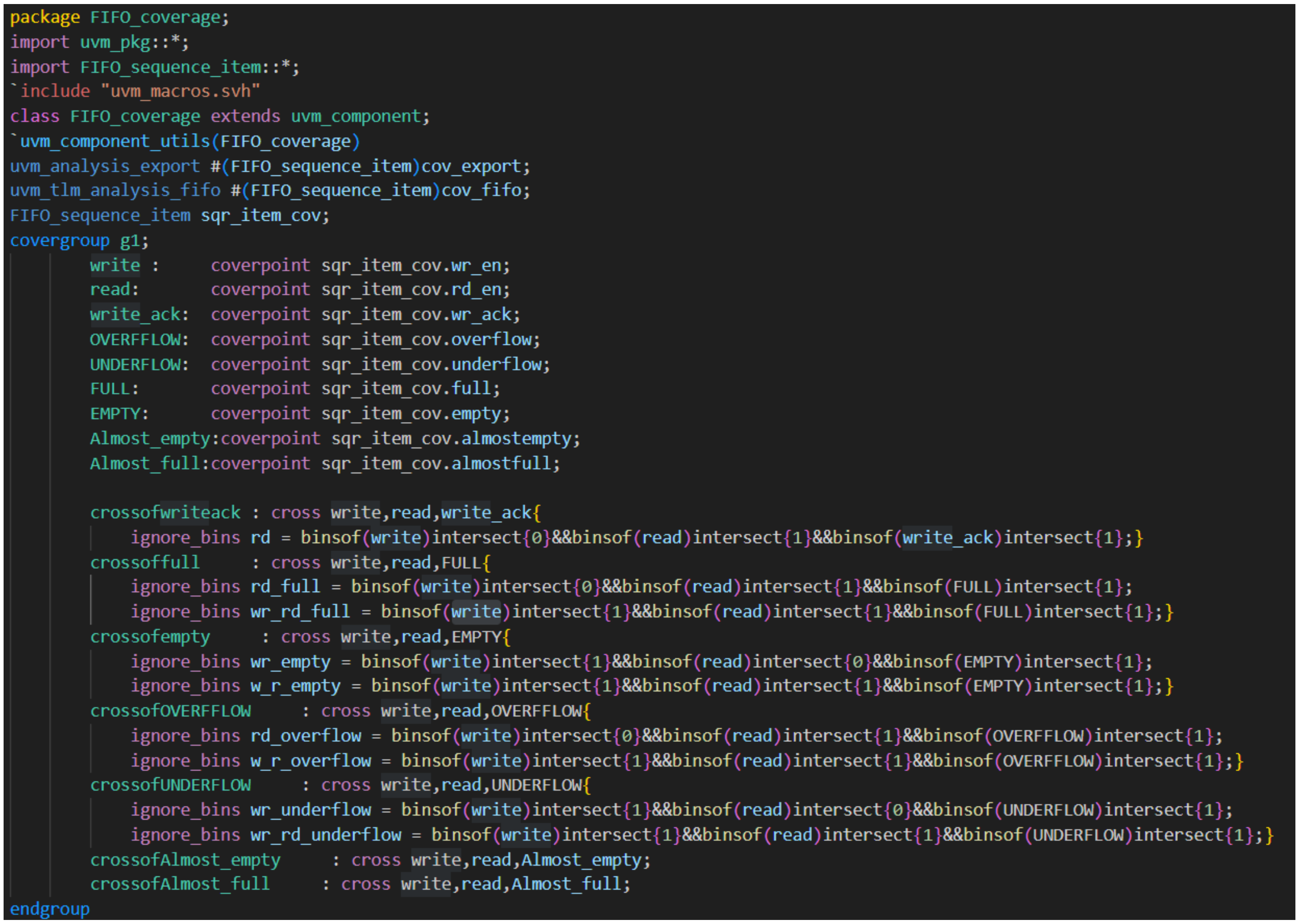
```
    endfunction
    task run_phase (uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);


        `uvm_info("run_phase","Assert reset.",UVM_MEDIUM)
        sqr_reset.start(env.agt.sqr);
        `uvm_info("run_phase","Start write only.",UVM_MEDIUM)
        sqr_write.start(env.agt.sqr);
        `uvm_info("run_phase","Start read only.",UVM_MEDIUM)
        sqr_read.start(env.agt.sqr);
        `uvm_info("run_phase","Start write and read.",UVM_MEDIUM)
        sqr_write_read.start(env.agt.sqr);
        `uvm_info("run_phase","Start unable write and read.",UVM_MEDIUM)
        sqr_wr_rd_unable.start(env.agt.sqr);
        `uvm_info("run_phase","Start random.",UVM_MEDIUM)
        sqr_random.start(env.agt.sqr);
        `uvm_info("run_phase","END.",UVM_MEDIUM)

        phase.drop_objection(this);
    endtask : run_phase
endclass
endpackage
```

7.13 FIFO Configuration

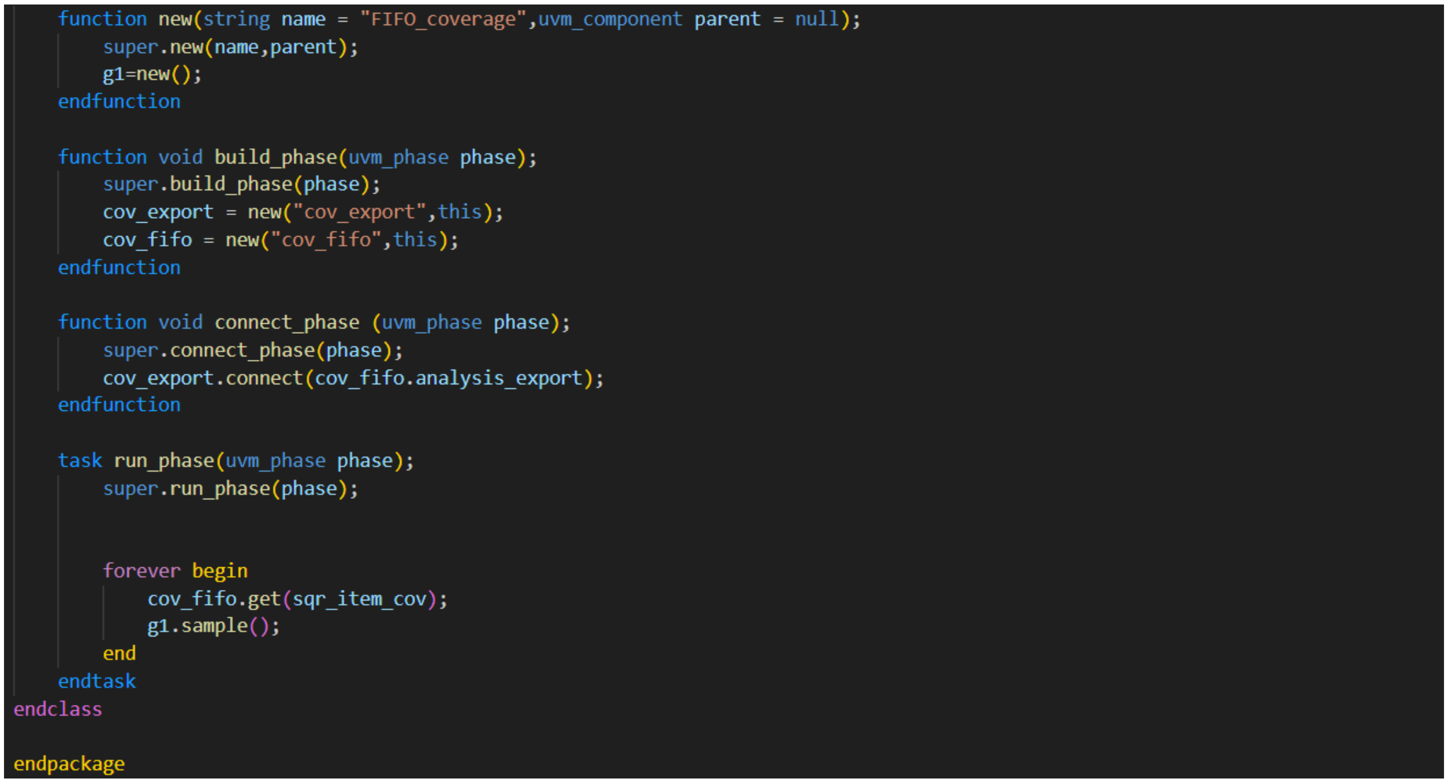


```
package FIFO_config;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_config extends uvm_object;
    `uvm_object_utils(FIFO_config)
    virtual FIFO_if FIFO_vif;
    function new(string name = "FIFO_config");
        super.new(name);
    endfunction
endclass
endpackage
```

7.14 FIFO Coverage

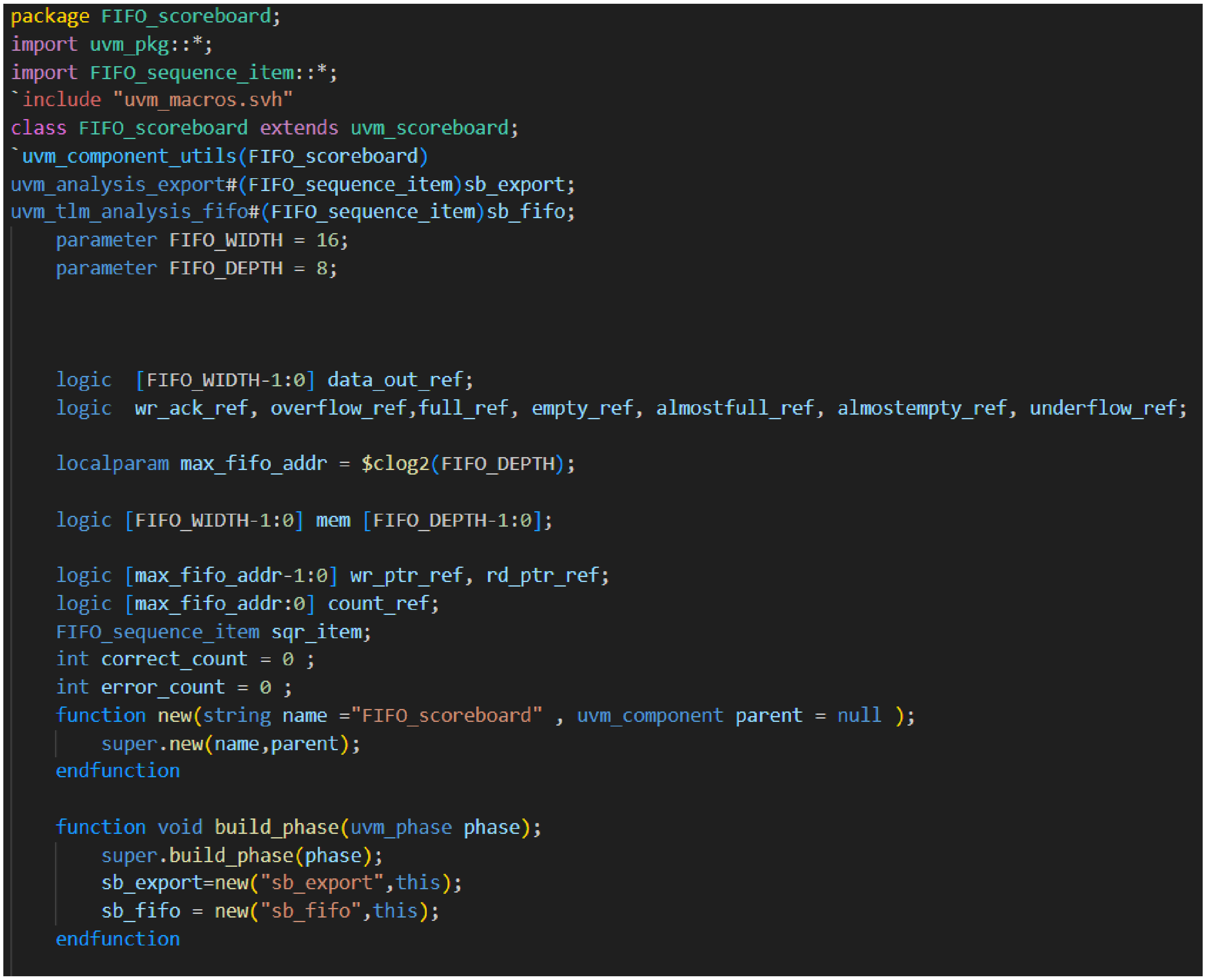


```
package FIFO_coverage;
import uvm_pkg::*;
import FIFO_sequence_item::*;
`include "uvm_macros.svh"
class FIFO_coverage extends uvm_component;
`uvm_component_utils(FIFO_coverage)
uvm_analysis_export #(FIFO_sequence_item)cov_export;
uvm_tlm_analysis_fifo #(FIFO_sequence_item)cov_fifo;
FIFO_sequence_item sqr_item_cov;
covergroup g1;
        write :     coverpoint sqr_item_cov.wr_en;
        read:       coverpoint sqr_item_cov.rd_en;
        write_ack:  coverpoint sqr_item_cov.wr_ack;
        OVERFFLOW:  coverpoint sqr_item_cov.overflow;
        UNDERFLOW:  coverpoint sqr_item_cov.underflow;
        FULL:       coverpoint sqr_item_cov.full;
        EMPTY:      coverpoint sqr_item_cov.empty;
        Almost_empty:coverpoint sqr_item_cov.almostempty;
        Almost_full:coverpoint sqr_item_cov.almostfull;

        crossofwriteack : cross write,read,write_ack{
            ignore_bins rd = binsof(write)intersect{0}&&binsof(read)intersect{1}&&binsof(write_ack)intersect{1};}
        crossoffull     : cross write,read,FULL{
            ignore_bins rd_full = binsof(write)intersect{0}&&binsof(read)intersect{1}&&binsof(FULL)intersect{1};
            ignore_bins wr_rd_full = binsof(write)intersect{1}&&binsof(read)intersect{1}&&binsof(FULL)intersect{1};}
        crossofempty     : cross write,read,EMPTY{
            ignore_bins wr_empty = binsof(write)intersect{1}&&binsof(read)intersect{0}&&binsof(EMPTY)intersect{1};
            ignore_bins w_r_empty = binsof(write)intersect{1}&&binsof(read)intersect{1}&&binsof(EMPTY)intersect{1};}
        crossofOVERFFLOW     : cross write,read,OVERFFLOW{
            ignore_bins rd_overflow = binsof(write)intersect{0}&&binsof(read)intersect{1}&&binsof(OVERFFLOW)intersect{1};
            ignore_bins w_r_overflow = binsof(write)intersect{1}&&binsof(read)intersect{1}&&binsof(OVERFFLOW)intersect{1};}
        crossofUNDERFLOW     : cross write,read,UNDERFLOW{
            ignore_bins wr_underflow = binsof(write)intersect{1}&&binsof(read)intersect{0}&&binsof(UNDERFLOW)intersect{1};
            ignore_bins wr_rd_underflow = binsof(write)intersect{1}&&binsof(read)intersect{1}&&binsof(UNDERFLOW)intersect{1};}
        crossofAlmost_empty     : cross write,read,Almost_empty;
        crossofAlmost_full     : cross write,read,Almost_full;
endgroup
```

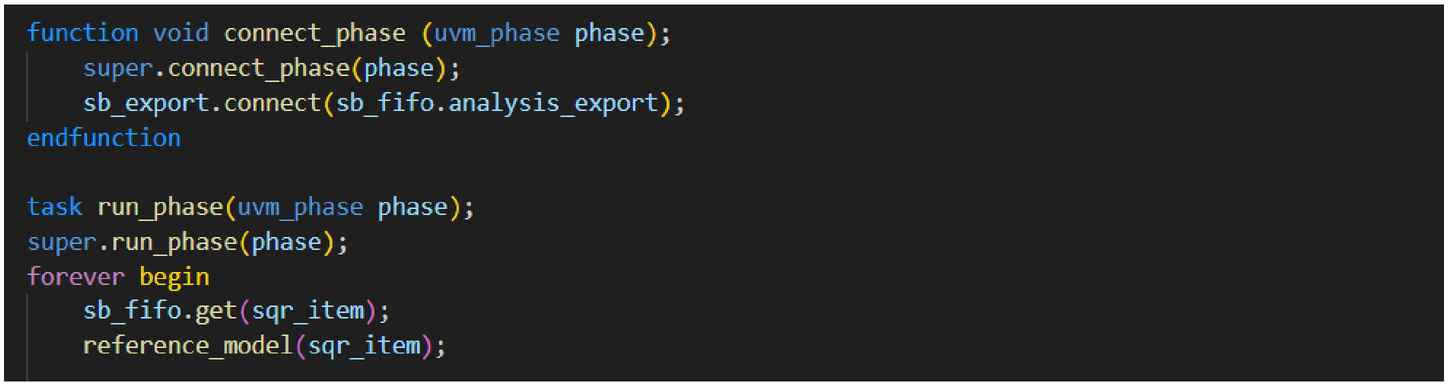


```
    function new(string name = "FIFO_coverage",uvm_component parent = null);
        super.new(name,parent);
        g1=new();
    endfunction

    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(sqr_item_cov);
            g1.sample();
        end
    endtask
endclass

endpackage
```

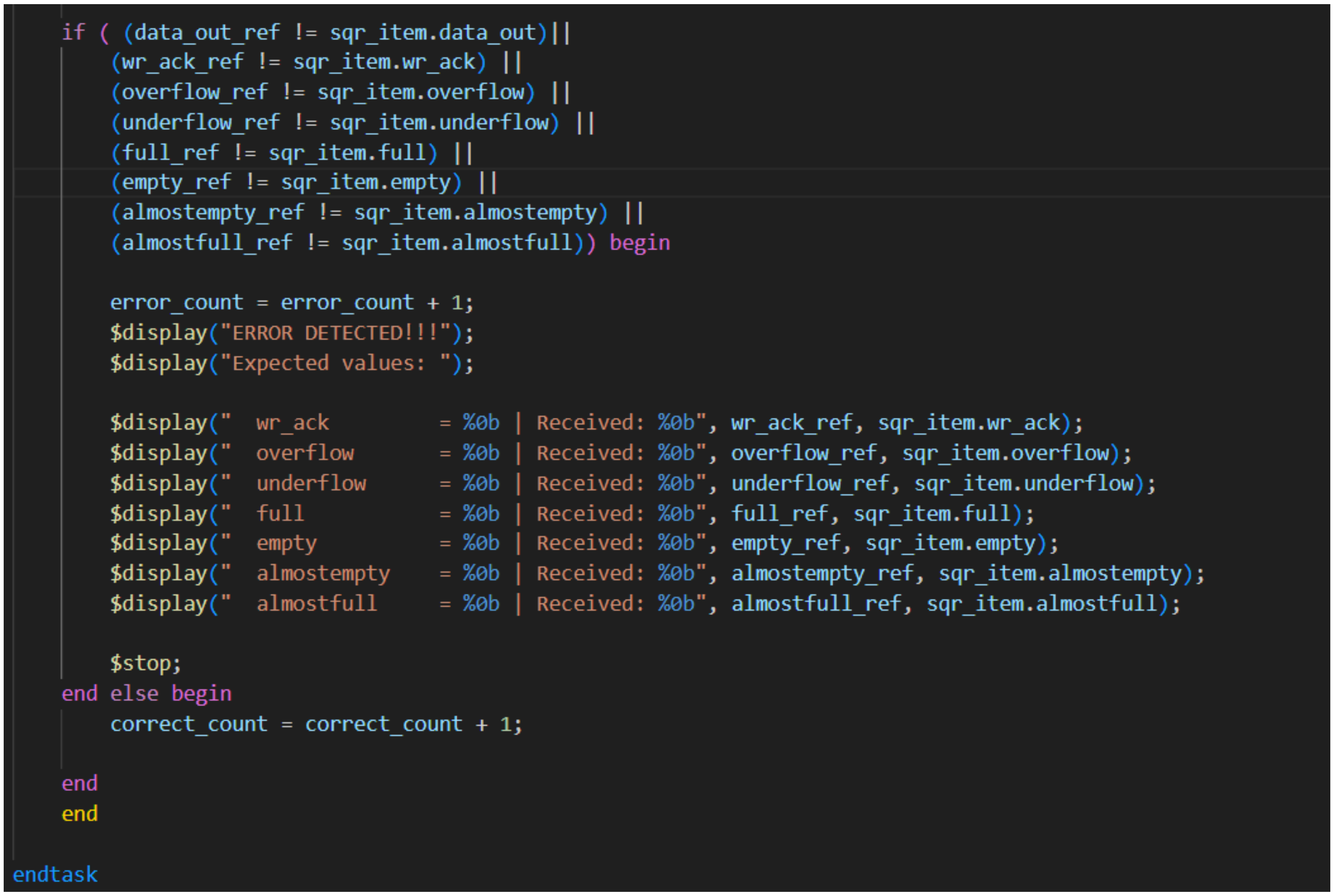
7.15 FIFO Scoreboard



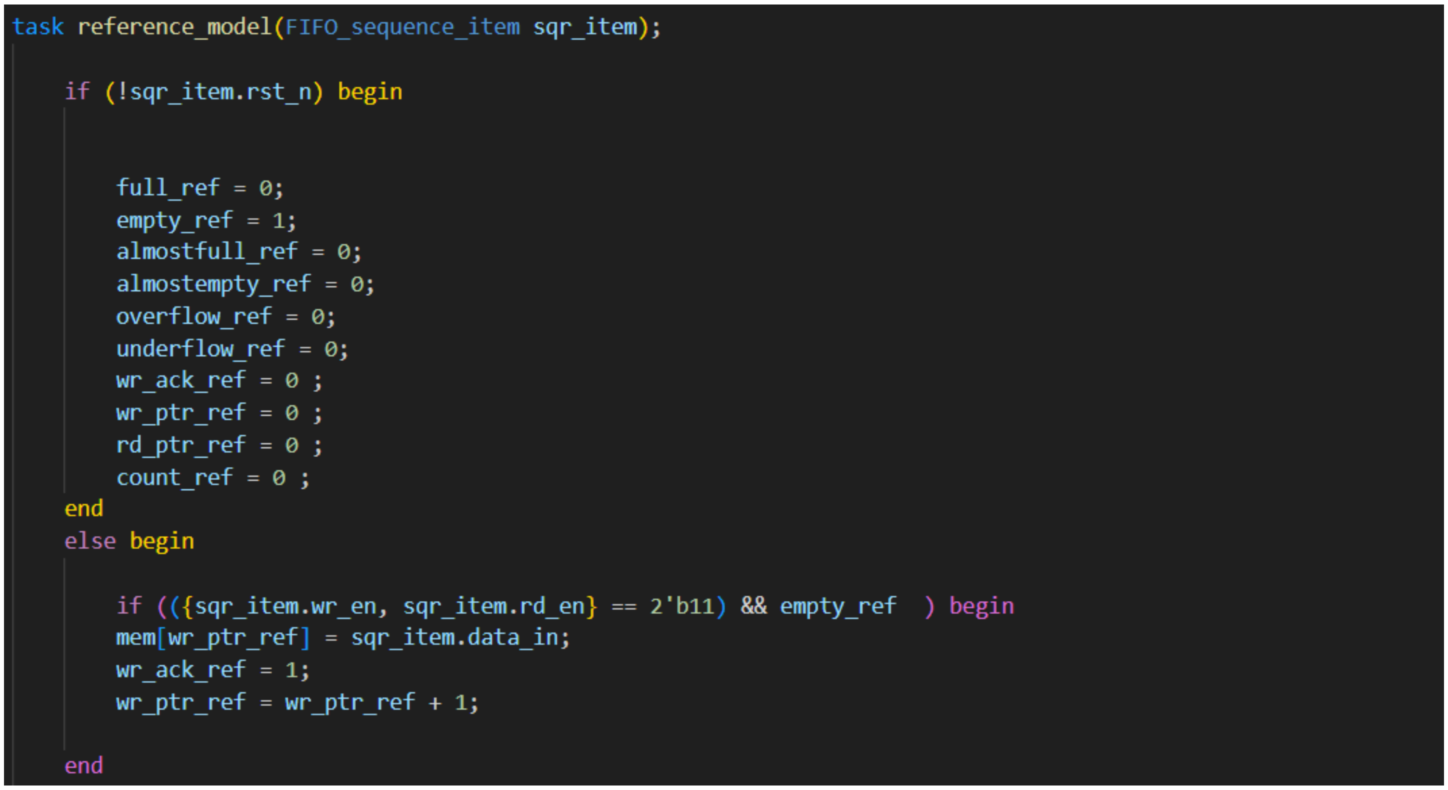
```
package FIFO_scoreboard;
import uvm_pkg::*;
import FIFO_sequence_item::*;
`include "uvm_macros.svh"
class FIFO_scoreboard extends uvm_scoreboard;
`uvm_component_utils(FIFO_scoreboard)
uvm_analysis_export#(FIFO_sequence_item)sb_export;
uvm_tlm_analysis_fifo#(FIFO_sequence_item)sb_fifo;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;



    logic  [FIFO_WIDTH-1:0] data_out_ref;
    logic  wr_ack_ref, overflow_ref,full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

    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_ref, rd_ptr_ref;
    logic [max_fifo_addr:0] count_ref;
    FIFO_sequence_item sqr_item;
    int correct_count = 0 ;
    int error_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
```



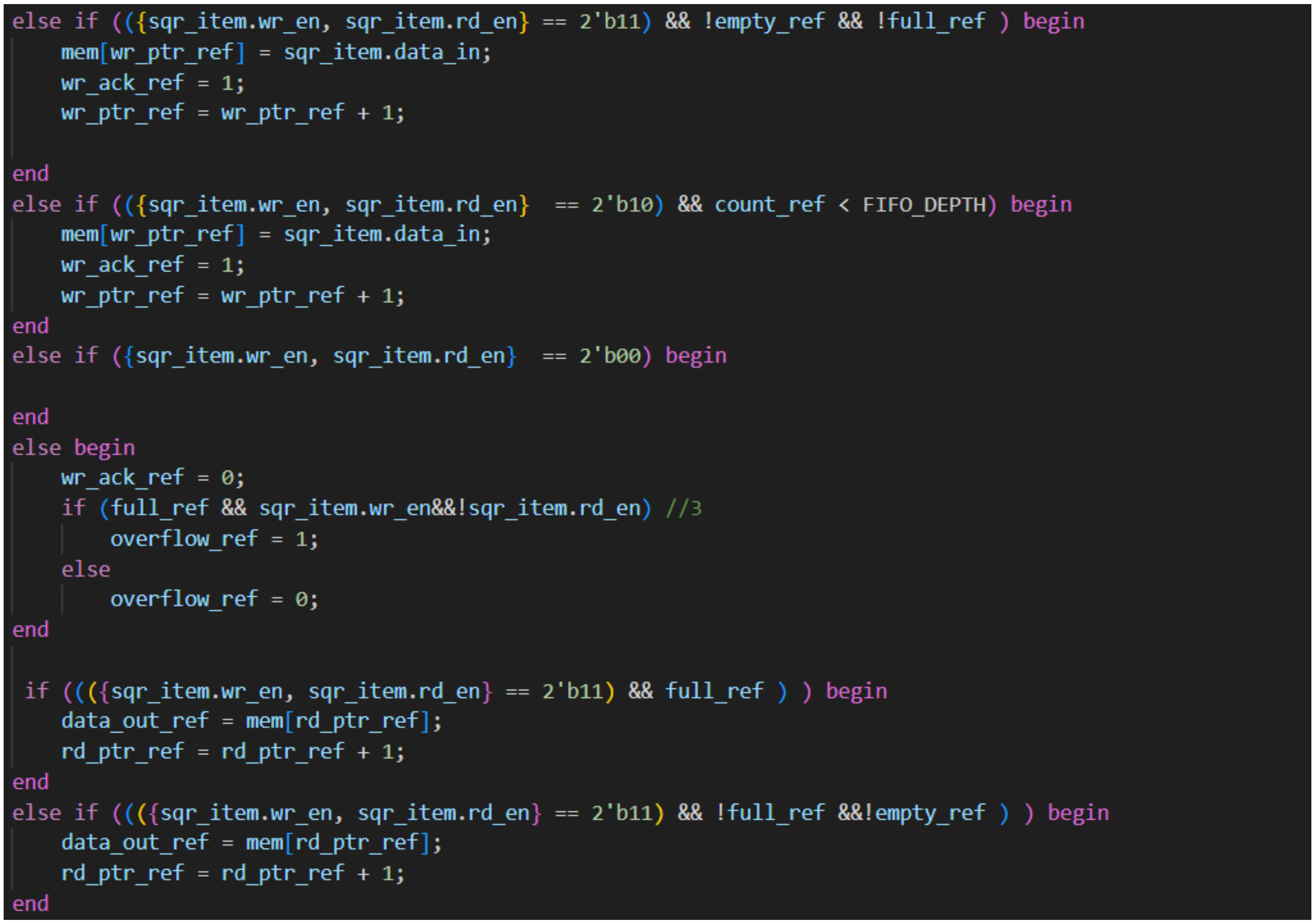
```
function void connect_phase (uvm_phase phase);
    super.connect_phase(phase);
    sb_export.connect(sb_fifo.analysis_export);
endfunction

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
    sb_fifo.get(sqr_item);
    reference_model(sqr_item);
```



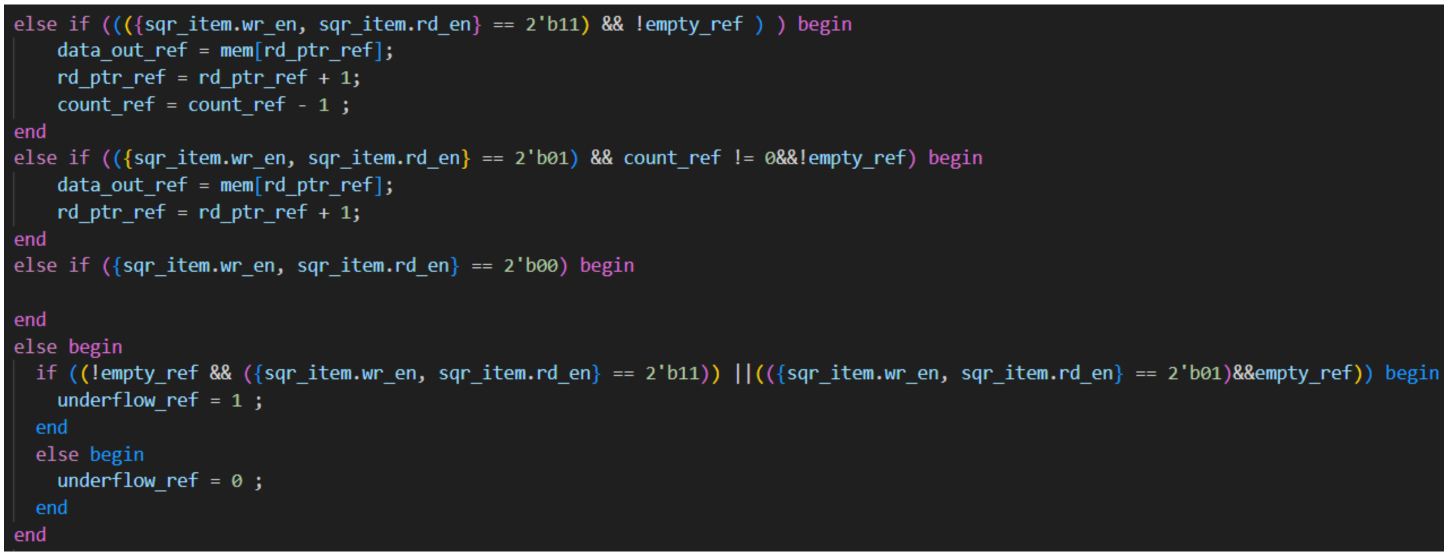
```
    if ( (data_out_ref != sqr_item.data_out)||
        (wr_ack_ref != sqr_item.wr_ack) ||
        (overflow_ref != sqr_item.overflow) ||
        (underflow_ref != sqr_item.underflow) ||
        (full_ref != sqr_item.full) ||
        (empty_ref != sqr_item.empty) ||
        (almostempty_ref != sqr_item.almostempty) ||
        (almostfull_ref != sqr_item.almostfull)) begin

        error_count = error_count + 1;
        $display("ERROR DETECTED!!!");
        $display("Expected values: ");

        $display("  wr_ack          = %0b | Received: %0b", wr_ack_ref, sqr_item.wr_ack);
        $display("  overflow        = %0b | Received: %0b", overflow_ref, sqr_item.overflow);
        $display("  underflow       = %0b | Received: %0b", underflow_ref, sqr_item.underflow);
        $display("  full            = %0b | Received: %0b", full_ref, sqr_item.full);
        $display("  empty           = %0b | Received: %0b", empty_ref, sqr_item.empty);
        $display("  almostempty     = %0b | Received: %0b", almostempty_ref, sqr_item.almostempty);
        $display("  almostfull      = %0b | Received: %0b", almostfull_ref, sqr_item.almostfull);

        $stop;
    end else begin
        correct_count = correct_count + 1;

    end
    end

endtask
```



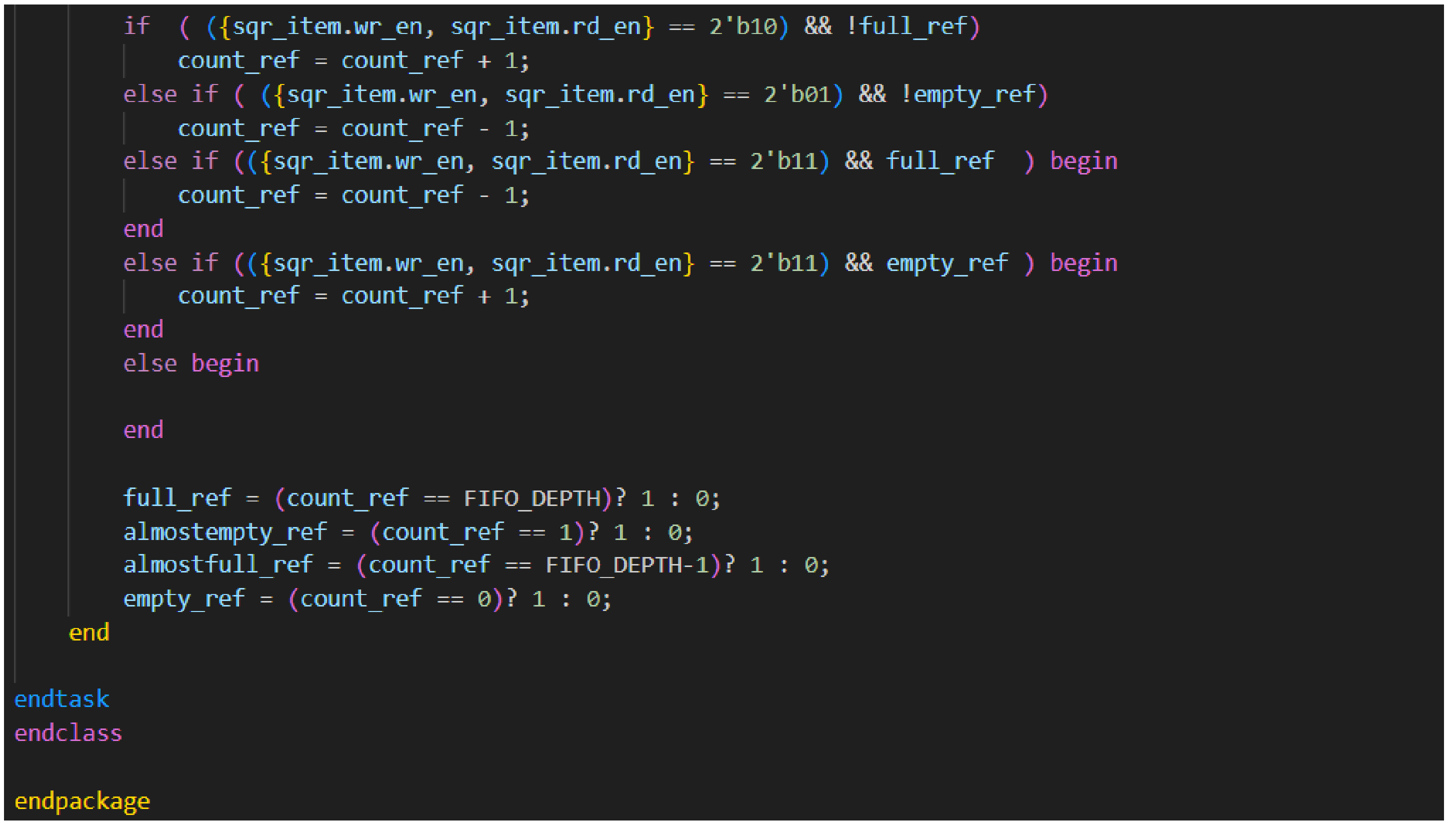
```
task reference_model(FIFO_sequence_item sqr_item);

    if (!sqr_item.rst_n) begin


        full_ref = 0;
        empty_ref = 1;
        almostfull_ref = 0;
        almostempty_ref = 0;
        overflow_ref = 0;
        underflow_ref = 0;
        wr_ack_ref = 0 ;
        wr_ptr_ref = 0 ;
        rd_ptr_ref = 0 ;
        count_ref = 0 ;
    end
    else begin

        if (({sqr_item.wr_en, sqr_item.rd_en} == 2'b11) && empty_ref  ) begin
        mem[wr_ptr_ref] = sqr_item.data_in;
        wr_ack_ref = 1;
        wr_ptr_ref = wr_ptr_ref + 1;

    end
```



```
else if (({sqr_item.wr_en, sqr_item.rd_en} == 2'b11) && !empty_ref && !full_ref ) begin
    mem[wr_ptr_ref] = sqr_item.data_in;
    wr_ack_ref = 1;
    wr_ptr_ref = wr_ptr_ref + 1;

end
else if (({sqr_item.wr_en, sqr_item.rd_en}  == 2'b10) && count_ref < FIFO_DEPTH) begin
    mem[wr_ptr_ref] = sqr_item.data_in;
    wr_ack_ref = 1;
    wr_ptr_ref = wr_ptr_ref + 1;
end
else if ({sqr_item.wr_en, sqr_item.rd_en}  == 2'b00) begin

end
else begin
    wr_ack_ref = 0;
    if (full_ref && sqr_item.wr_en&&!sqr_item.rd_en) //3
        overflow_ref = 1;
    else
        overflow_ref = 0;
end

 if ((({sqr_item.wr_en, sqr_item.rd_en} == 2'b11) && full_ref ) ) begin
    data_out_ref = mem[rd_ptr_ref];
    rd_ptr_ref = rd_ptr_ref + 1;
end
else if ((({sqr_item.wr_en, sqr_item.rd_en} == 2'b11) && !full_ref &&!empty_ref ) ) begin
    data_out_ref = mem[rd_ptr_ref];
    rd_ptr_ref = rd_ptr_ref + 1;
end
```

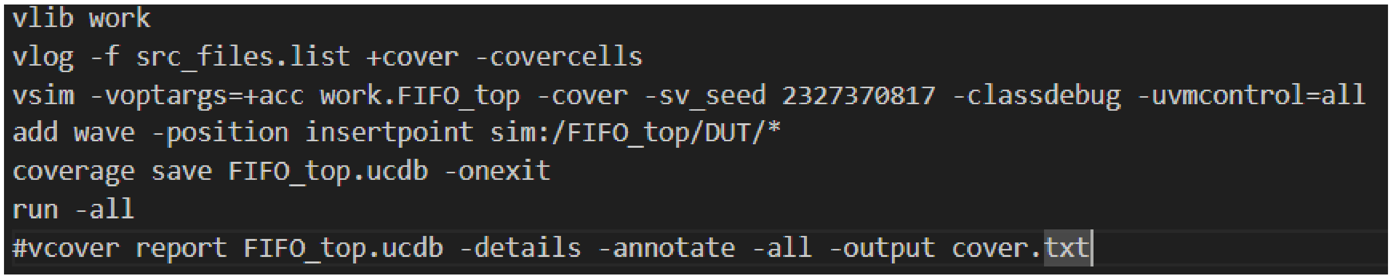


```
else if ((({sqr_item.wr_en, sqr_item.rd_en} == 2'b11) && !empty_ref ) ) begin
    data_out_ref = mem[rd_ptr_ref];
    rd_ptr_ref = rd_ptr_ref + 1;
    count_ref = count_ref - 1 ;
end
else if (({sqr_item.wr_en, sqr_item.rd_en} == 2'b01) && count_ref != 0&&!empty_ref) begin
    data_out_ref = mem[rd_ptr_ref];
    rd_ptr_ref = rd_ptr_ref + 1;
end
else if ({sqr_item.wr_en, sqr_item.rd_en} == 2'b00) begin

end
else begin
  if ((!empty_ref && ({sqr_item.wr_en, sqr_item.rd_en} == 2'b11)) ||(({sqr_item.wr_en, sqr_item.rd_en} == 2'b01)&&empty_ref)) begin
    underflow_ref = 1 ;
  end
  else begin
    underflow_ref = 0 ;
  end
end
```



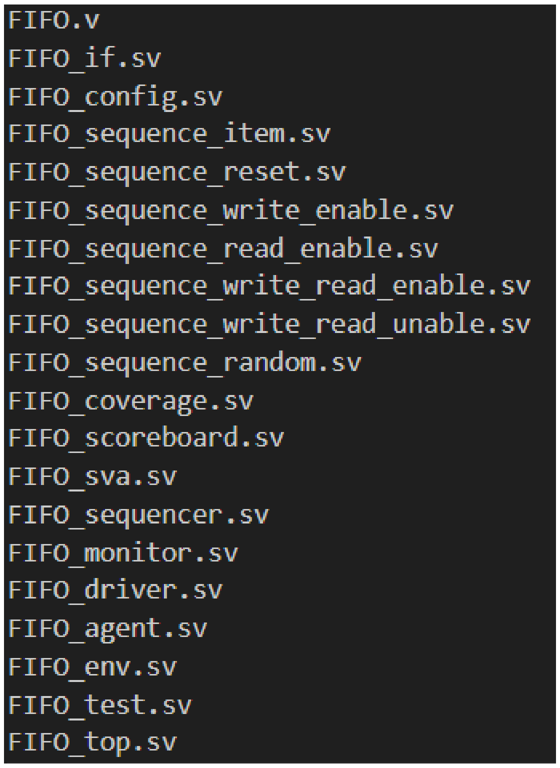
```
        if  ( ({sqr_item.wr_en, sqr_item.rd_en} == 2'b10) && !full_ref)
            count_ref = count_ref + 1;
        else if ( ({sqr_item.wr_en, sqr_item.rd_en} == 2'b01) && !empty_ref)
            count_ref = count_ref - 1;
        else if (({sqr_item.wr_en, sqr_item.rd_en} == 2'b11) && full_ref   ) begin
            count_ref = count_ref - 1;
        end
        else if (({sqr_item.wr_en, sqr_item.rd_en} == 2'b11) && empty_ref ) begin
            count_ref = count_ref + 1;
        end
        else begin

        end

        full_ref = (count_ref == FIFO_DEPTH)? 1 : 0;
        almostempty_ref = (count_ref == 1)? 1 : 0;
        almostfull_ref = (count_ref == FIFO_DEPTH-1)? 1 : 0;
        empty_ref = (count_ref == 0)? 1 : 0;
    end

endtask
endclass

endpackage
```

7.16 DO File



```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.FIFO_top -cover -sv_seed 2327370817 -classdebug -uvmcontrol=all
add wave -position insertpoint sim:/FIFO_top/DUT/*
coverage save FIFO_top.ucdb -onexit
run -all
#vcover report FIFO_top.ucdb -details -annotate -all -output cover.txt
```

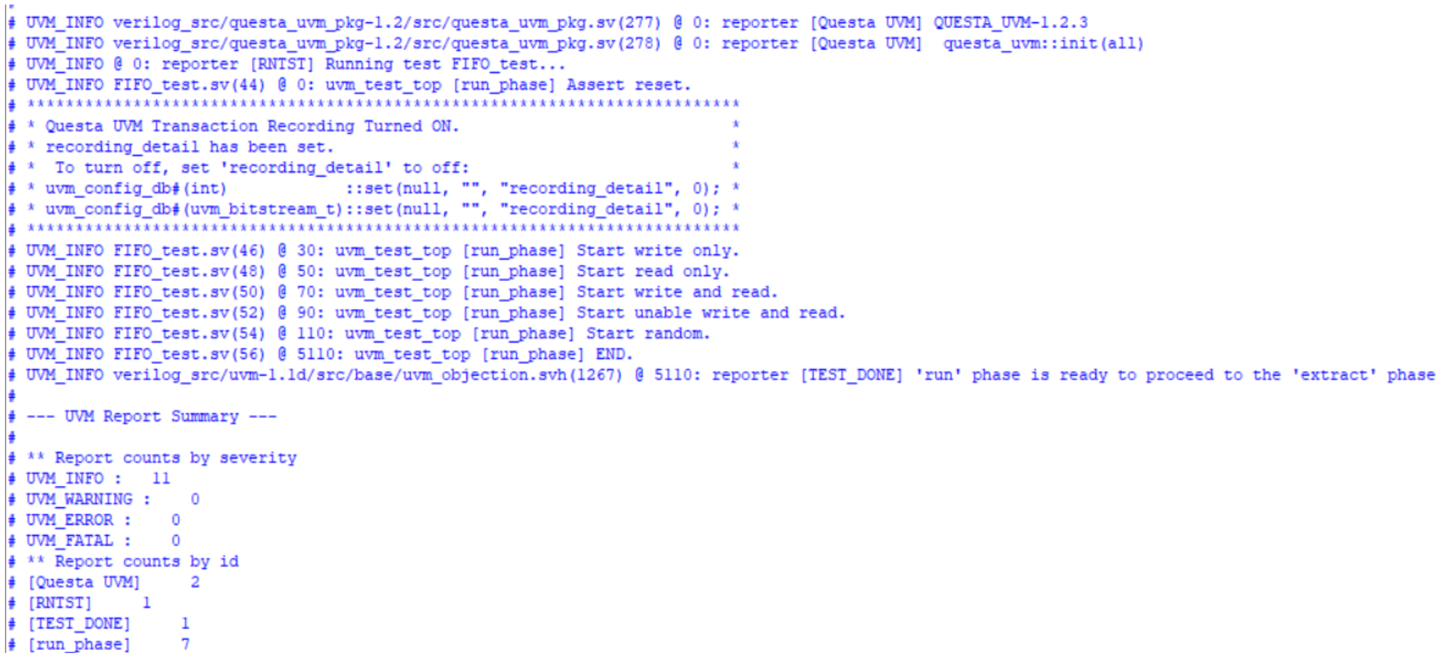
7.17 Source file



```
FIFO.v
FIFO_if.sv
FIFO_config.sv
FIFO_sequence_item.sv
FIFO_sequence_reset.sv
FIFO_sequence_write_enable.sv
FIFO_sequence_read_enable.sv
FIFO_sequence_write_read_enable.sv
FIFO_sequence_write_read_unable.sv
FIFO_sequence_random.sv
FIFO_coverage.sv
FIFO_scoreboard.sv
FIFO_sva.sv
FIFO_sequencer.sv
FIFO_monitor.sv
FIFO_driver.sv
FIFO_agent.sv
FIFO_env.sv
FIFO_test.sv
FIFO_top.sv
```

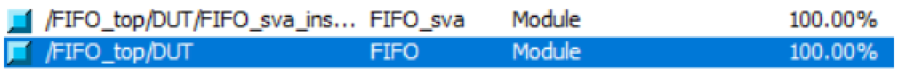
8.RESULT

8.1 UVM REPORT

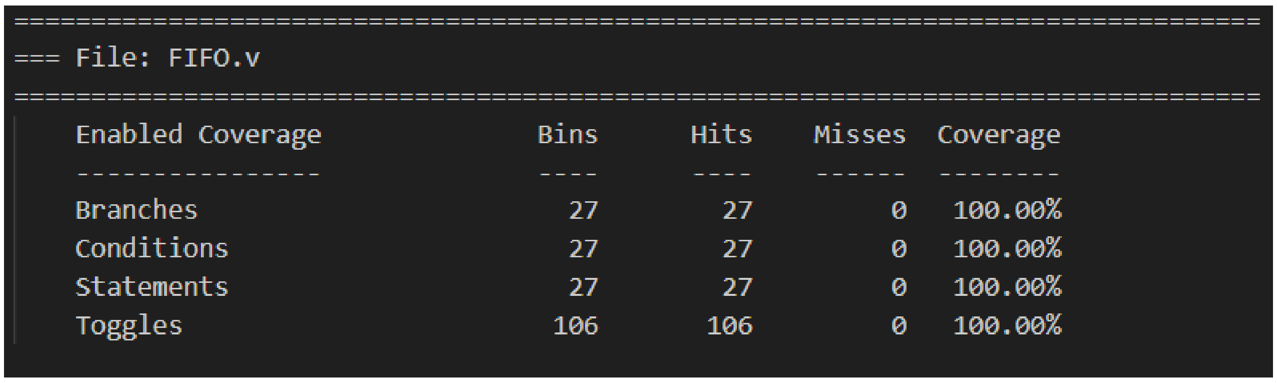


```
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(all)
# UVM_INFO @ 0: reporter [RNTST] Running test FIFO_test...
# UVM_INFO FIFO_test.sv(44) @ 0: uvm_test_top [run_phase] Assert reset.
# ***************************************************************************
# * Questa UVM Transaction Recording Turned ON.                              *
# * recording_detail has been set.                                           *
# *  To turn off, set 'recording_detail' to off:                             *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ***************************************************************************
# UVM_INFO FIFO_test.sv(46) @ 30: uvm_test_top [run_phase] Start write only.
# UVM_INFO FIFO_test.sv(48) @ 50: uvm_test_top [run_phase] Start read only.
# UVM_INFO FIFO_test.sv(50) @ 70: uvm_test_top [run_phase] Start write and read.
# UVM_INFO FIFO_test.sv(52) @ 90: uvm_test_top [run_phase] Start unable write and read.
# UVM_INFO FIFO_test.sv(54) @ 110: uvm_test_top [run_phase] Start random.
# UVM_INFO FIFO_test.sv(56) @ 5110: uvm_test_top [run_phase] END.
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 5110: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   11
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [run_phase]     7
```

8.2 Code Coverage

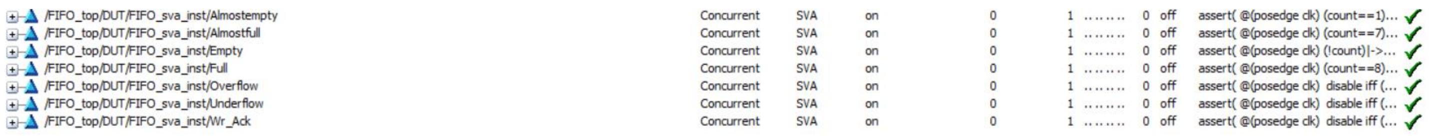


```
/FIFO_top/DUT/FIFO_sva_ins...  FIFO_sva   Module         100.00%
/FIFO_top/DUT                  FIFO       Module         100.00%
```

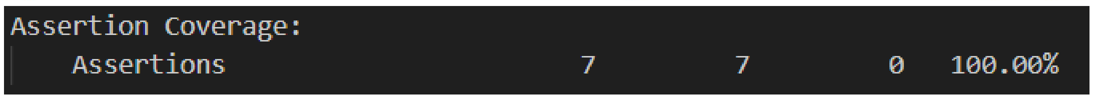


```
=====================================================================
=== File: FIFO.v
=====================================================================
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        27        27         0   100.00%
    Conditions                      27        27         0   100.00%
    Statements                      27        27         0   100.00%
    Toggles                        106       106         0   100.00%
```

8.3 Assertion Coverage

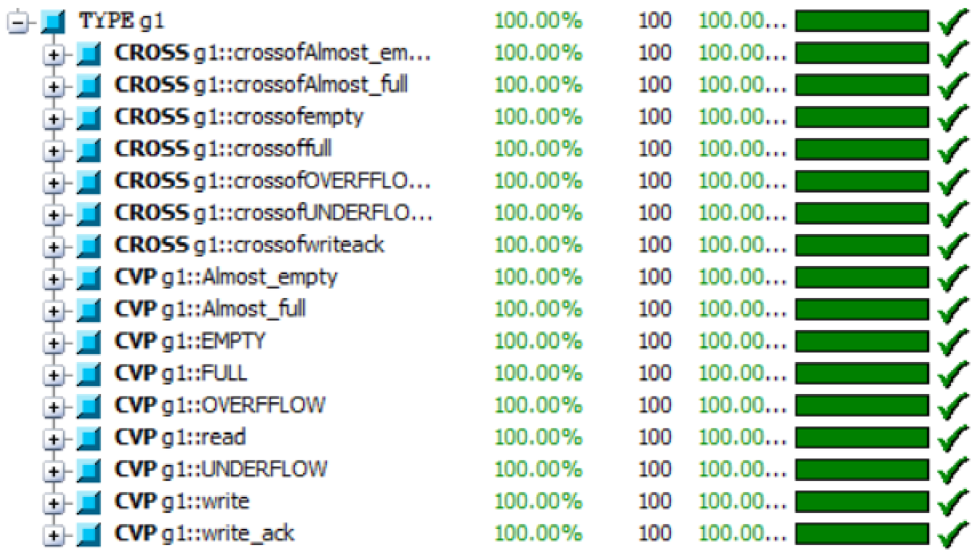


```
/FIFO_top/DUT/FIFO_sva_inst/Almostempty   Concurrent  SVA  on   0   1 .......  0  off  assert( @(posedge clk) (count==1)... ✓
/FIFO_top/DUT/FIFO_sva_inst/Almostfull    Concurrent  SVA  on   0   1 .......  0  off  assert( @(posedge clk) (count==7)... ✓
/FIFO_top/DUT/FIFO_sva_inst/Empty         Concurrent  SVA  on   0   1 .......  0  off  assert( @(posedge clk) (!count)|->... ✓
/FIFO_top/DUT/FIFO_sva_inst/Full          Concurrent  SVA  on   0   1 .......  0  off  assert( @(posedge clk) (count==8)... ✓
/FIFO_top/DUT/FIFO_sva_inst/Overflow      Concurrent  SVA  on   0   1 .......  0  off  assert( @(posedge clk)  disable iff (... ✓
/FIFO_top/DUT/FIFO_sva_inst/Underflow     Concurrent  SVA  on   0   1 .......  0  off  assert( @(posedge clk)  disable iff (... ✓
/FIFO_top/DUT/FIFO_sva_inst/Wr_Ack        Concurrent  SVA  on   0   1 .......  0  off  assert( @(posedge clk)  disable iff (... ✓
```

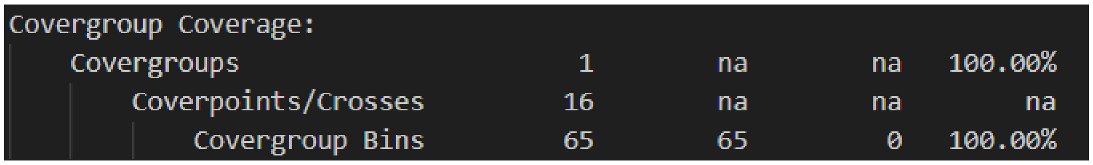


```
Assertion Coverage:
    Assertions                       7          7          0   100.00%
```

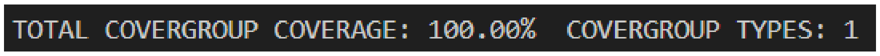
8.4 Function Coverage



TYPE g1	100.00%	100	100.00...	🟩	✓
CROSS g1::crossofAlmost_em...	100.00%	100	100.00...	🟩	✓
CROSS g1::crossofAlmost_full	100.00%	100	100.00...	🟩	✓
CROSS g1::crossofempty	100.00%	100	100.00...	🟩	✓
CROSS g1::crossoffull	100.00%	100	100.00...	🟩	✓
CROSS g1::crossofOVERFFLO...	100.00%	100	100.00...	🟩	✓
CROSS g1::crossofUNDERFLO...	100.00%	100	100.00...	🟩	✓
CROSS g1::crossofwriteack	100.00%	100	100.00...	🟩	✓
CVP g1::Almost_empty	100.00%	100	100.00...	🟩	✓
CVP g1::Almost_full	100.00%	100	100.00...	🟩	✓
CVP g1::EMPTY	100.00%	100	100.00...	🟩	✓
CVP g1::FULL	100.00%	100	100.00...	🟩	✓
CVP g1::OVERFFLOW	100.00%	100	100.00...	🟩	✓
CVP g1::read	100.00%	100	100.00...	🟩	✓
CVP g1::UNDERFLOW	100.00%	100	100.00...	🟩	✓
CVP g1::write	100.00%	100	100.00...	🟩	✓
CVP g1::write_ack	100.00%	100	100.00...	🟩	✓



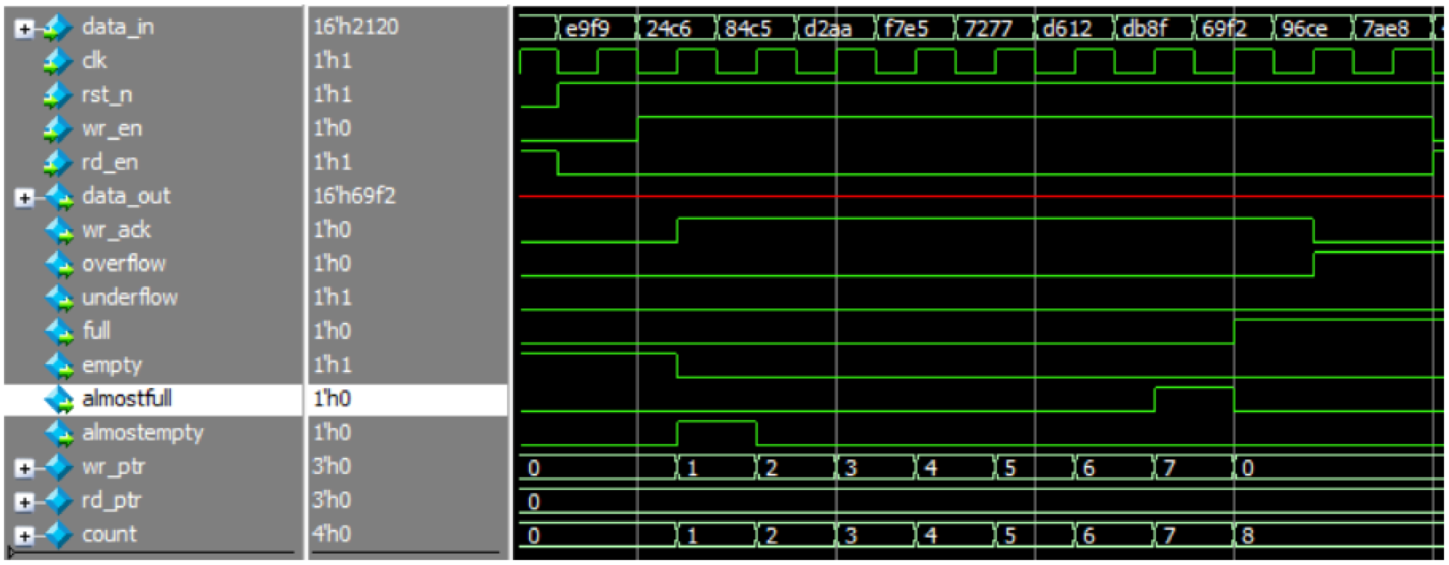
```
Covergroup Coverage:
    Covergroups                      1         na         na    100.00%
        Coverpoints/Crosses         16         na         na         na
            Covergroup Bins         65         65          0    100.00%
```



```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1
```

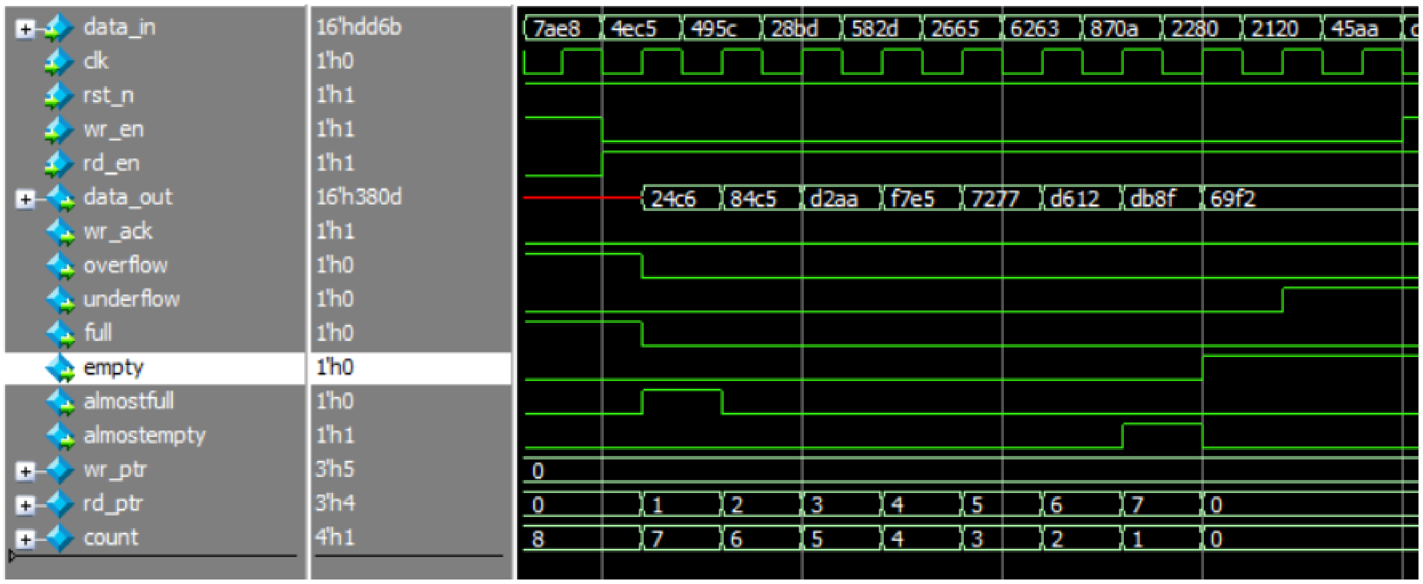
8.5 WaveForm

Write Sequence

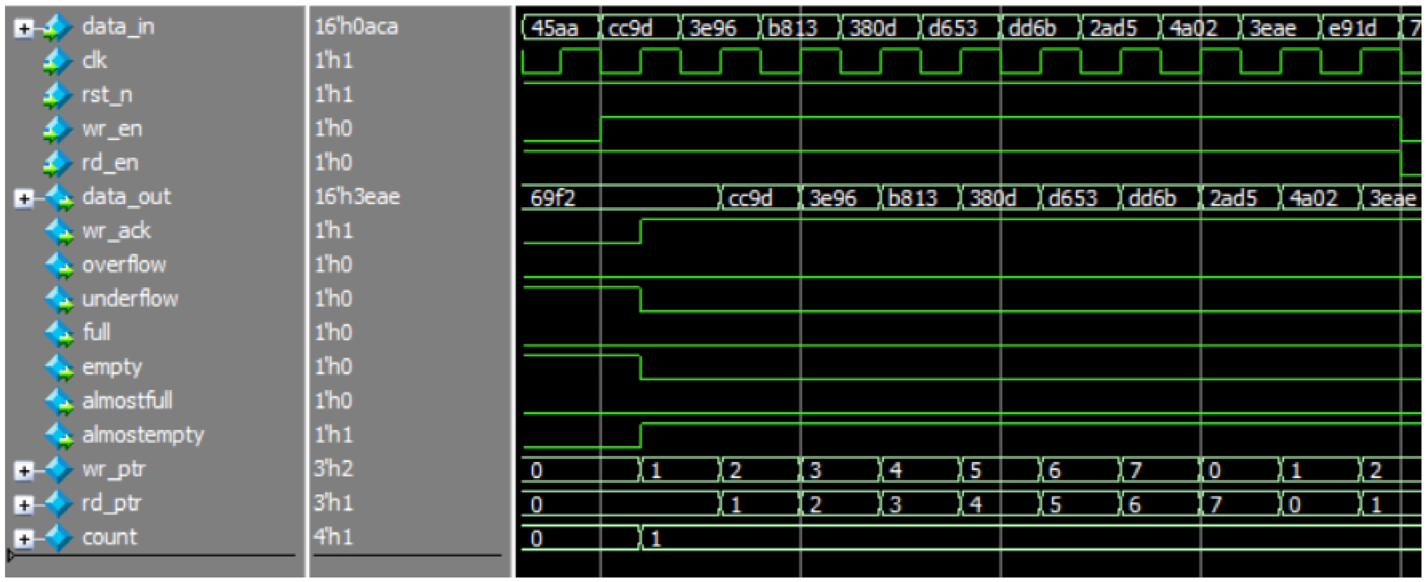


data_in	16'h2120
clk	1'h1
rst_n	1'h1
wr_en	1'h0
rd_en	1'h1
data_out	16'h69f2
wr_ack	1'h0
overflow	1'h0
underflow	1'h1
full	1'h0
empty	1'h1
almostfull	1'h0
almostempty	1'h0
wr_ptr	3'h0
rd_ptr	3'h0
count	4'h0

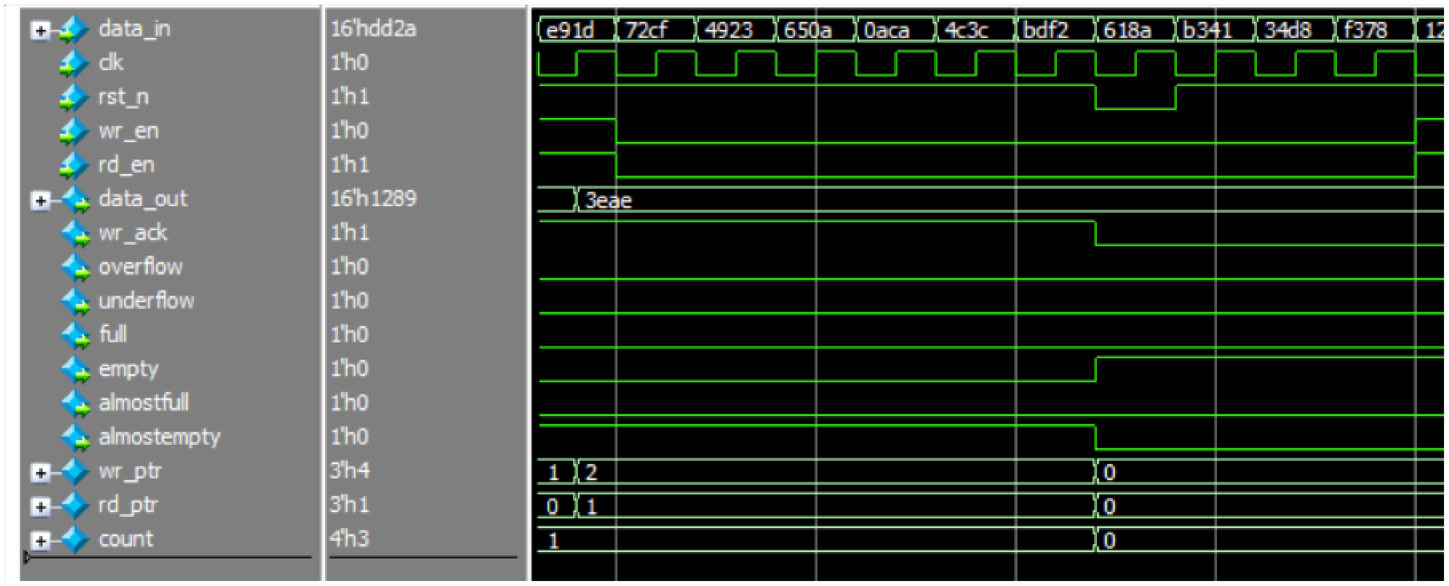
Read Sequence



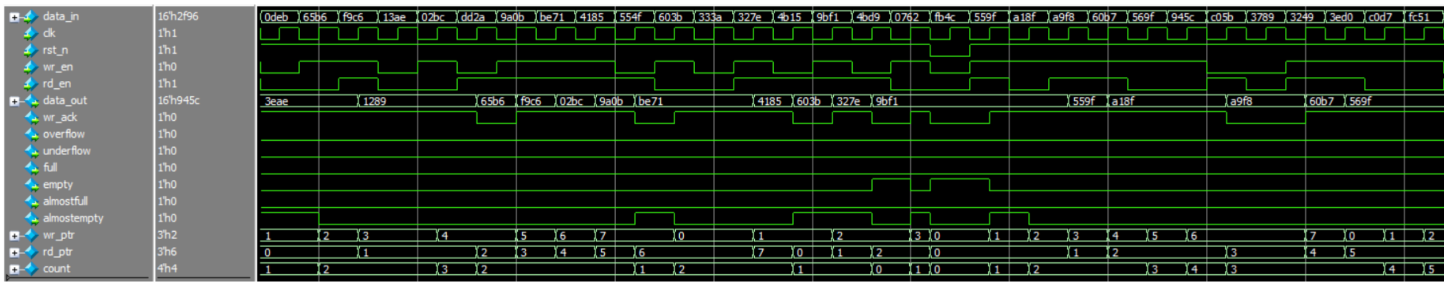
Read and Write Sequence

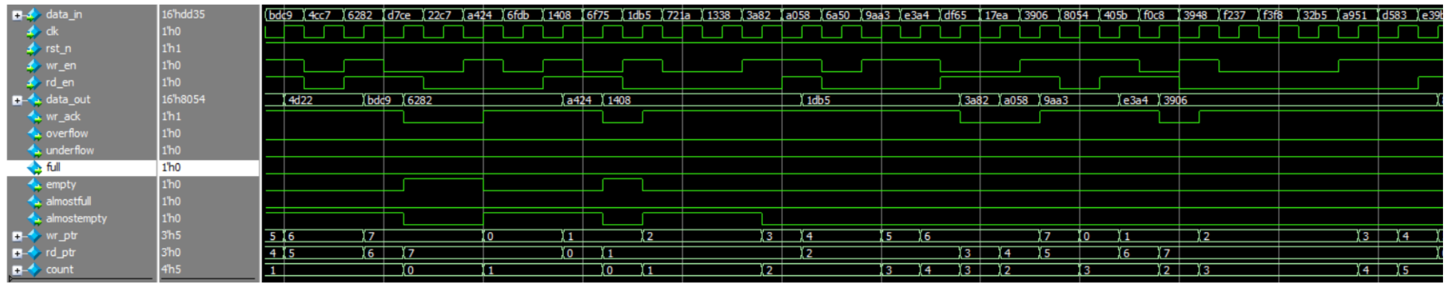


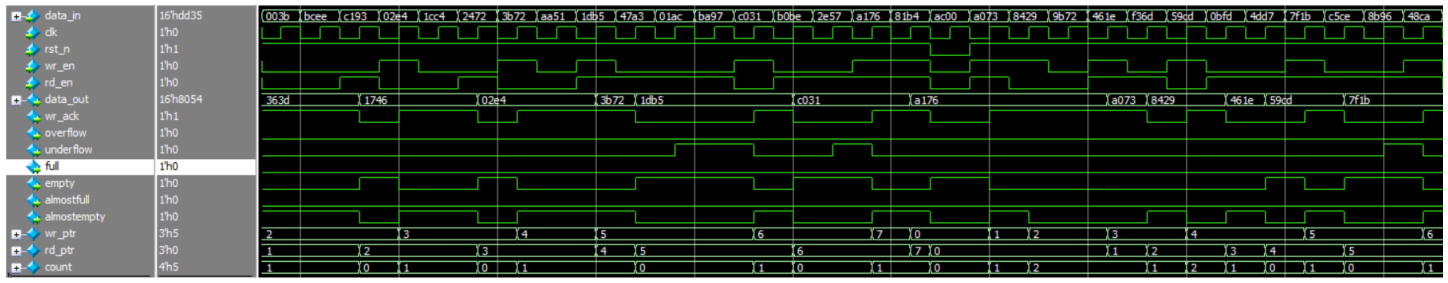
Read and Write Disable Sequence



Random Sequence







If you want to simulate my UVM Project please visit my repo