Functional Verification of a SPI Interface Using UVM and SVA: Integration of Passive RAM and Slave Agents

Using SystemVerilog

Submitted by: Mahmoud Ghamry

JUN 2025

Table of Contents

Table of Figures

Introduction

The Serial Peripheral Interface (SPI) is a widely used synchronous serial communication protocol, commonly employed in embedded systems for efficient, high-speed data exchange over short distances. As digital designs grow in complexity, ensuring reliable communication and protocol compliance becomes essential in modern hardware development.

This project focuses on the **functional verification** of an SPI interface using **Universal Verification Methodology (UVM)** and **SystemVerilog Assertions (SVA)** to validate the correctness, behavior, and protocol conformance of the design under test (DUT). The approach emphasizes modularity, reusability, and thorough coverage.

The verification process began by developing **independent UVM environments** for both the **RAM** and the **SPI slave** modules. Each environment was verified separately to ensure individual functionality and stability. After this, both components were **integrated into a top-level wrapper module**, forming a complete SPI system. This step-by-step approach enabled easier debugging, scalability, and modular validation.

To strengthen validation, **functional coverage models** were created for each module RAM, slave, and wrapper enabling measurable coverage metrics to evaluate the completeness of verification. Furthermore, **golden reference models** were built for all three components. These models served as behavioral baselines for the scoreboard, ensuring data integrity and correctness during simulation by comparing expected vs. actual outputs.

The testbench architecture employs **passive agents** for RAM and the slave, ensuring accurate signal observation without interfering with the DUT. **UVM** provides a structured and reusable verification framework, while **SVA** checks key protocol properties with temporal assertions.

Overall, this project aims to provide a comprehensive and reusable verification solution that not only detects functional bugs but also ensures high confidence through assertions, coverage collection, and comparison against golden models.

I am going to verify the design that we made before to visit it please follow this link .

Design Methodology

Signal Name	Direction	Width	Description
clk	Input	1 bit	System clock signal is used to synchronize the SPI and RAM operations.
rst_n	Input	1 bit	Active-low reset signals are used to initialize the system.
SS_n	Input	1 bit	SPI chips select signal (active low) that enables communication with the SPI slave.
MOSI	Input	1 bit	Master Out Slave In: Serial data input from the SPI master.
MISO	Output	1 bit	Master In Slave Out: Serial data output to the SPI master.
rx_data	Output (Slave → RAM)	10 bits	Parallel data received from SPI and sent to the RAM.
rx_valid	Output (Slave → RAM)	1 bit	It indicates that the rx_data is valid and ready to be written to RAM.
tx_data	Input (RAM → Slave)	8 bits	Parallel data read from RAM to be transmitted via SPI.
tx_valid	Input (RAM → Slave)	1 bit	It indicates that the tx_data is valid and ready for SPI transmission.
din	Input to RAM	10 bits	Data input to the RAM from the SPI slave (mapped from rx_data).
dout	Output from RAM	8 bits	Data output from the RAM to the SPI slave (mapped to tx_data).

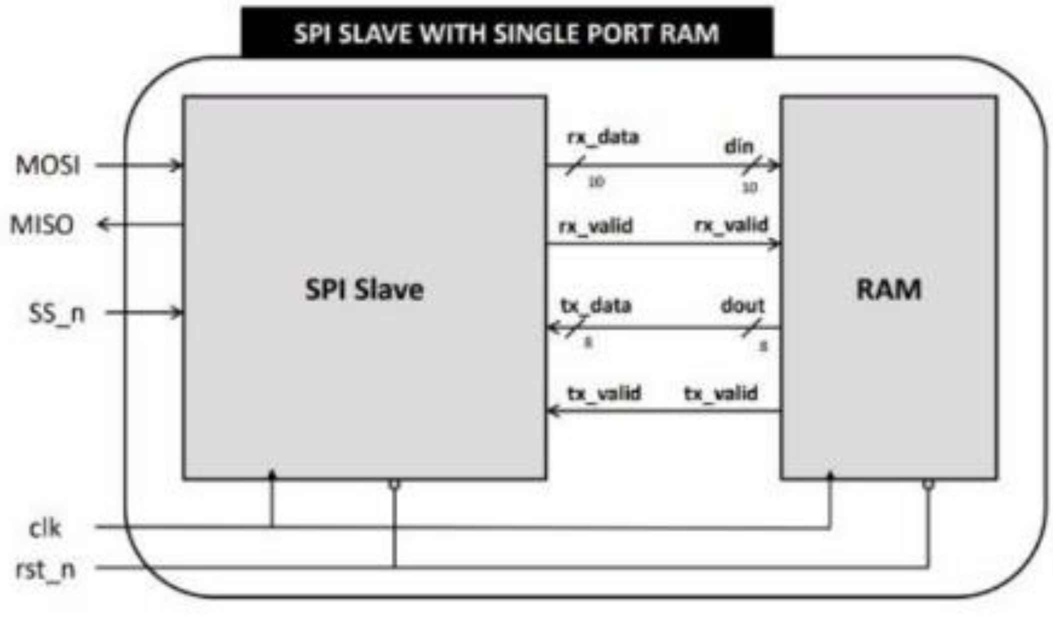


Figure 1. SPI interface design

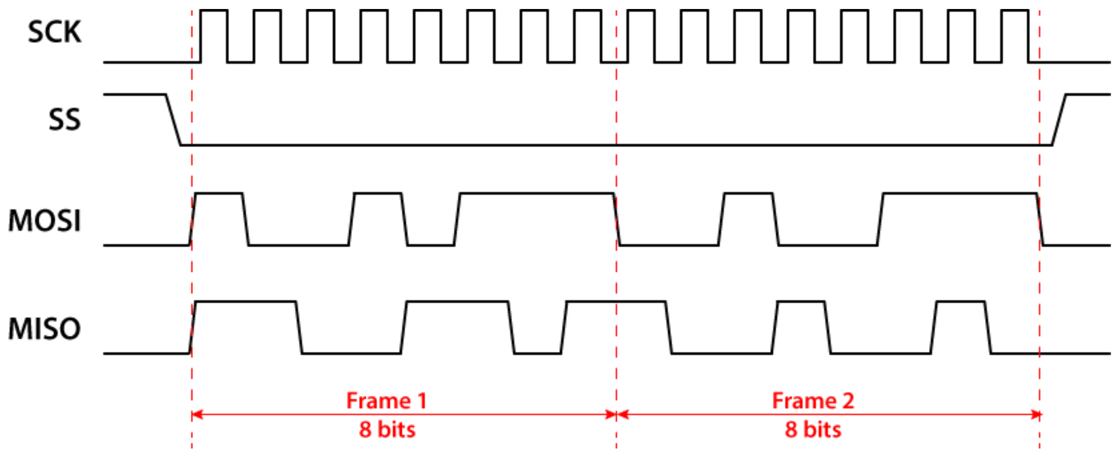


Figure 2. SPI interface frame

Verification Plan

RAM Verification Plan

Stimulus	Description	Stimulus Generation	Functional Coverage	Functionality Check	Assertion
RAM_1	When the reset is asserted, the output value & tx_vaild equal zero	Directed at the start of the simulation then randomized with constraint using sequence reset that drives the reset to off most the simulation time	Cover it with coverpoint_3	A checker in the scoreboard to make sure the output is correct	Reset_Assertion
RAM_2	When the rx_vaild is high and din[9:8] is zero so it is tx_vaild is low	It's randomized with constraint using sequence that drives using driver	Cover it with Coverpoint_0 &coverpoint_1	A checker in the scoreboard to make sure the output is correct	Tx_vaildd
RAM_3	When the rx_vaild is high and din[9:8] is 1 so it is tx_vaild is low	It's randomized with constraint using sequence that drives using driver	Cover it with Coverpoint_0 &coverpoint_1 &coverpoint_2	A checker in the scoreboard to make sure the output is correct	Tx_vaildd
RAM_4	When the rx_vaild is high and din[9:8] is 2 so it is tx_vaild is low	It's randomized with constraint using sequence that drives using driver	Cover it with Coverpoint_0 &coverpoint_1 &coverpoint_2	A checker in the scoreboard to make sure the output is correct	Tx_vaildd
RAM_5	When the rx_vaild is high and din[9:8] is 3 so it is tx_vaild is high	It's randomized with constraint using sequence that drives using driver	Cover it with Coverpoint_0 &coverpoint_1 &coverpoint_2	A checker in the scoreboard to make sure the output is correct	Tx_vaild_pp

Slave Verification Plan

Stimulus	Description	Stimulus Generation	Functional Coverage	Functionality Check	Assertion
Slave_1	When the reset is asserted, the output rx_data & rx_vaild & MISO equal zero	Directed at the start of the simulation then randomized with constraint using sequence reset that drives the reset to off most the simulation time	Cover it with Covering_rst_n	A checker in the scoreboard to make sure the output is correct	Cover_rst_n_ Assertion
Slave_2	FSM is control it	It's randomized with constraint using sequence that drives using driver	Cover it with Covering_rx_vaild &covering_miso &covering_rx_data	A checker in the scoreboard to make sure the output is correct	-

Wrapper Verification Plan

Stimulus	Description	Stimulus Generation	Functional Coverage	Functionality Check
Wrapper_1	When the reset is asserted, the output rx_data & rx_vaild & output value & tx_vaild &MISO equal zero	Directed at the start of the simulation then randomized with constraint using sequence reset that drives the reset to off most the simulation time	Cover it with coverpoint_2	A checker in the scoreboard to make sure the output is correct
Wrapper_2	FSM control slave through ram	It's randomized with constraint using sequence that drives using driver	Cover it with coverpoint_0 Coverpoint_1 Coverpoint_3	A checker in the scoreboard to make sure the output is correct

Topology

1) Firstly, I verify ram with this step
   a) I built a total project for ram with and top module that content the environment of ram that appear in figure 3 shown below.

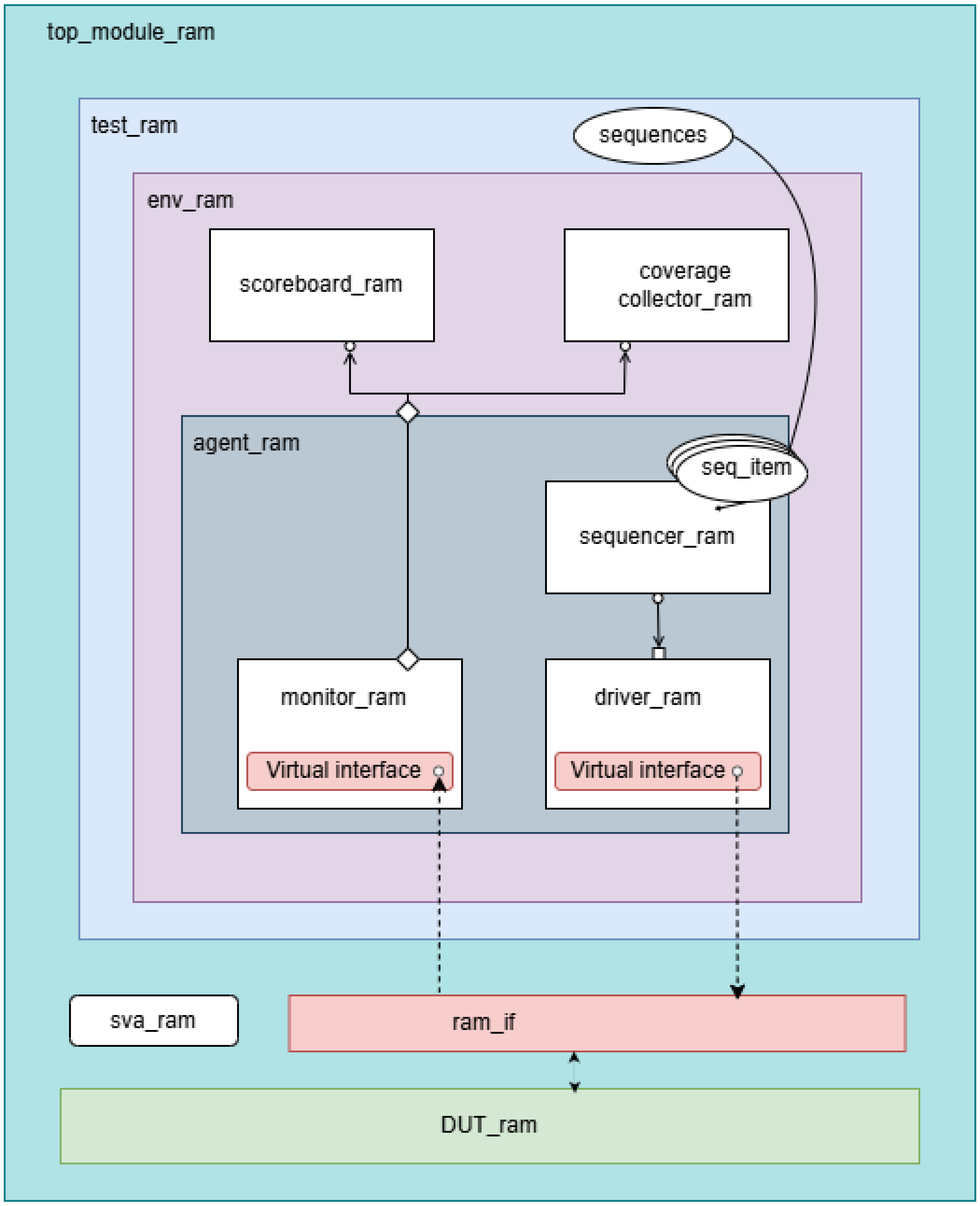


Figure 3. RAM environment

2) Secondly, I verify ram with this step

a) I built a total project for slave with and top module that content the environment of slave that appear in figure 4 shown below.

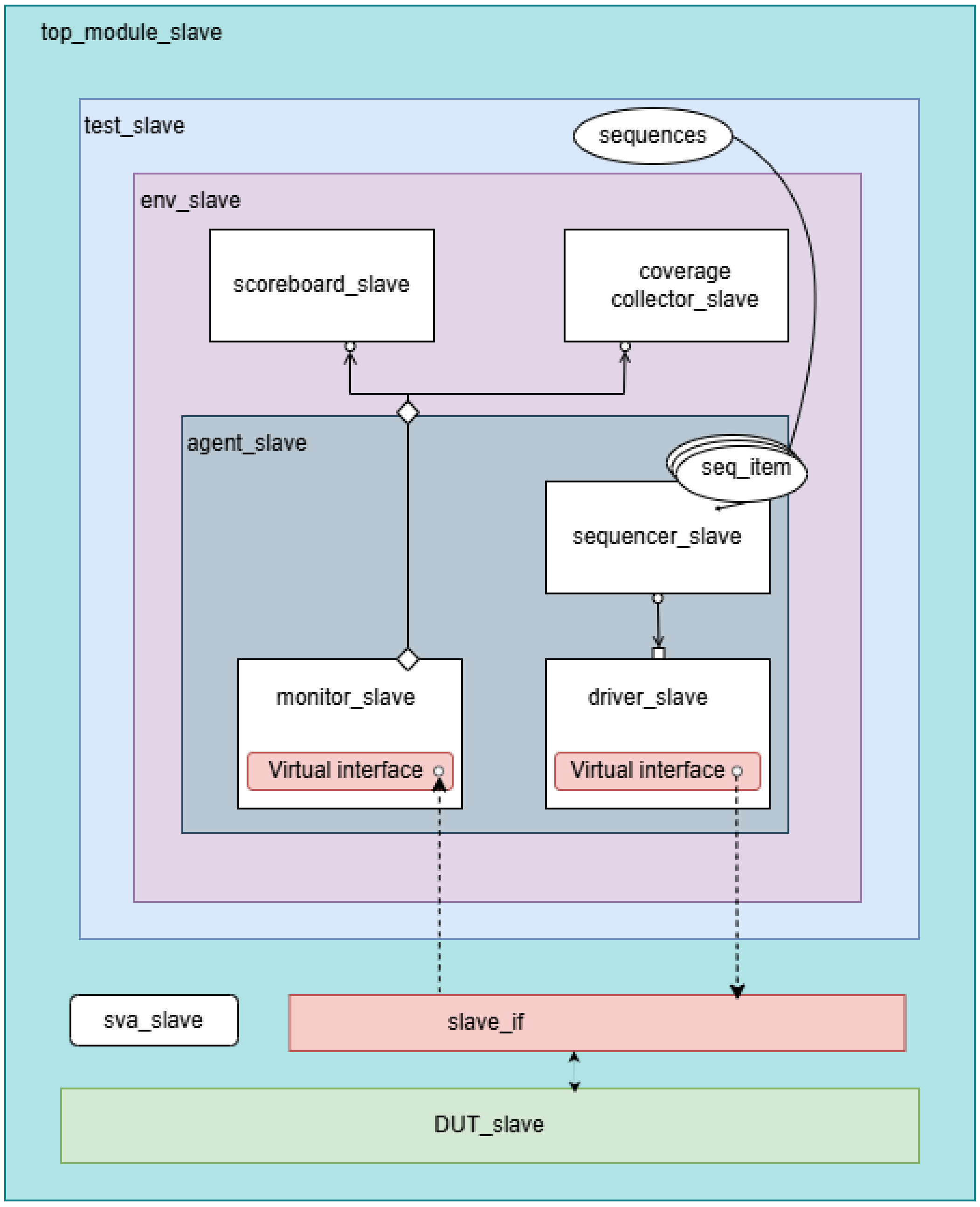


Figure 4. Slave environment

3) Finaly, I put the ram environment as a passive agent what make a driver and sequencer does not work, in this case I just make the ram env to monitor only, the same for slave environment it shown in figure 5 shown below.

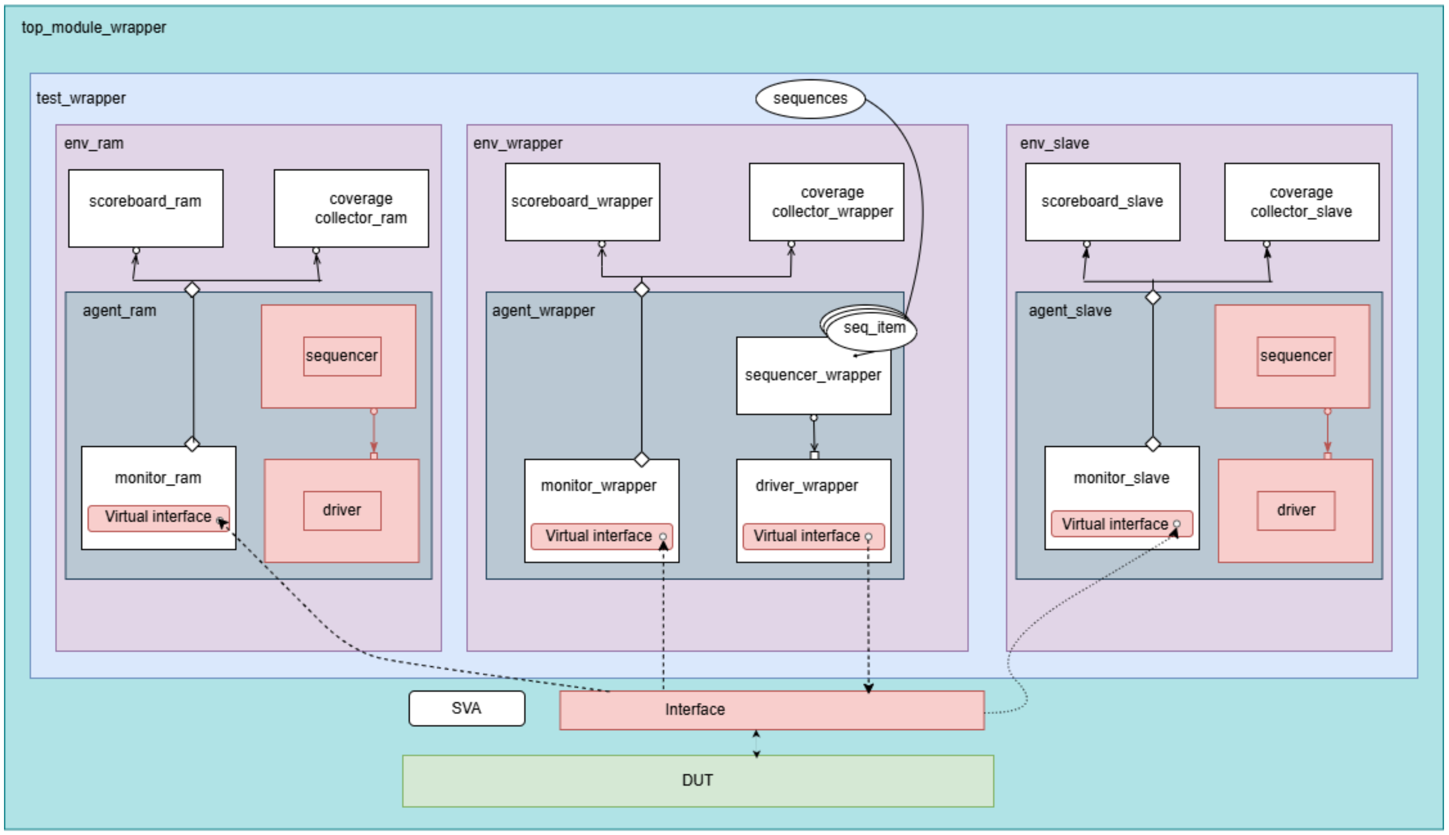


Figure 5. Wrapper environment

Results

RAM Results

Waveform

Reset waveform

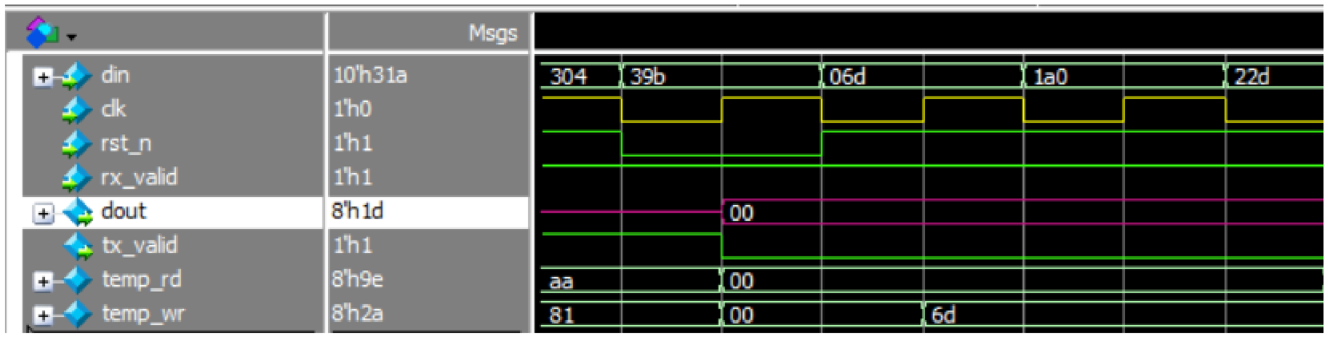
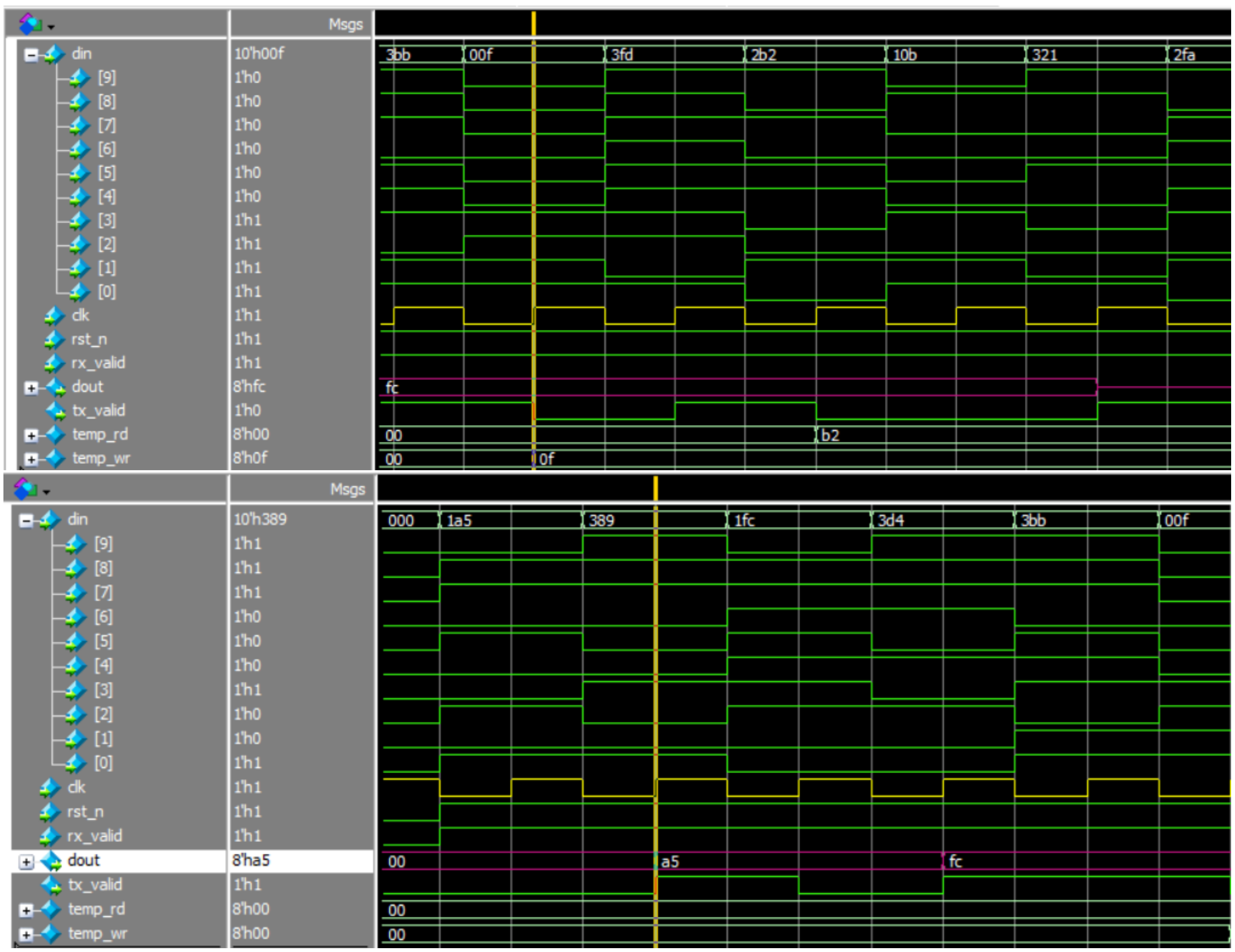


Figure 6. Reset waveform RAM

Read and Write waveform



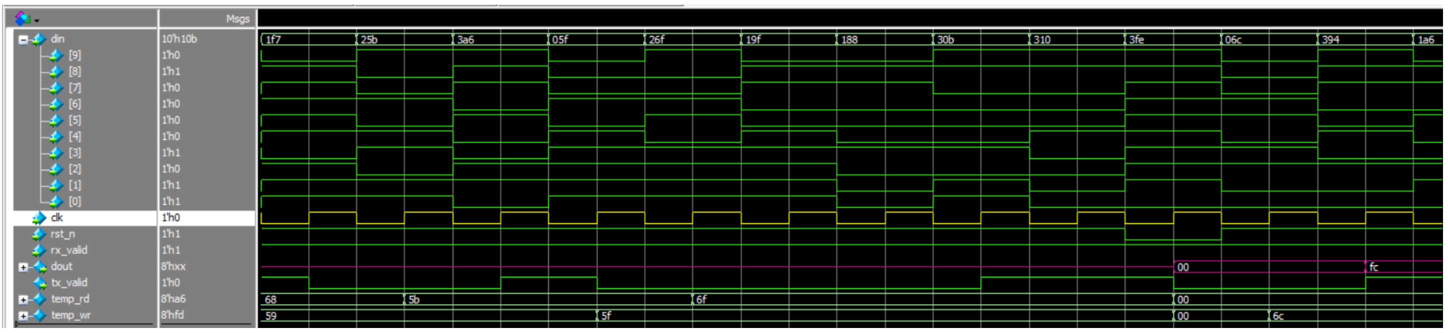


Figure 7. Read and write waveform RAM

Total view waveform

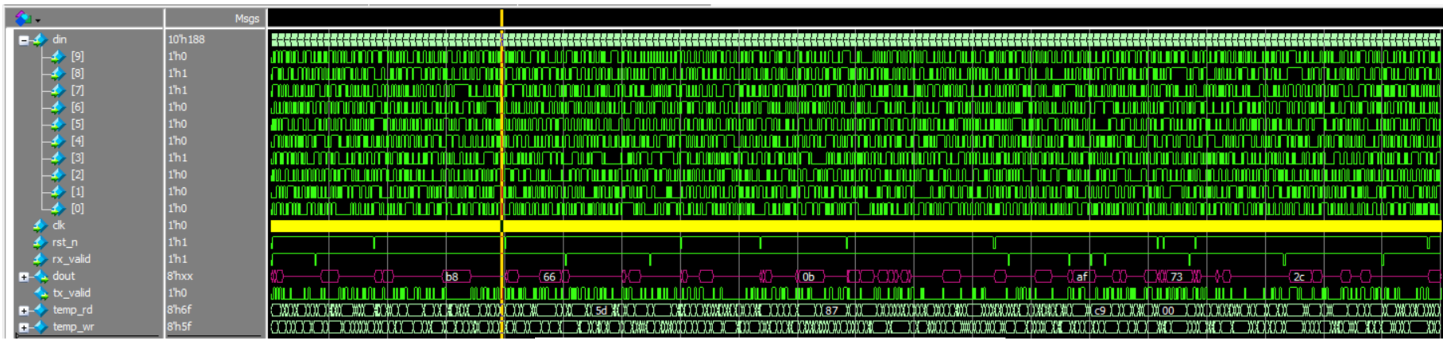
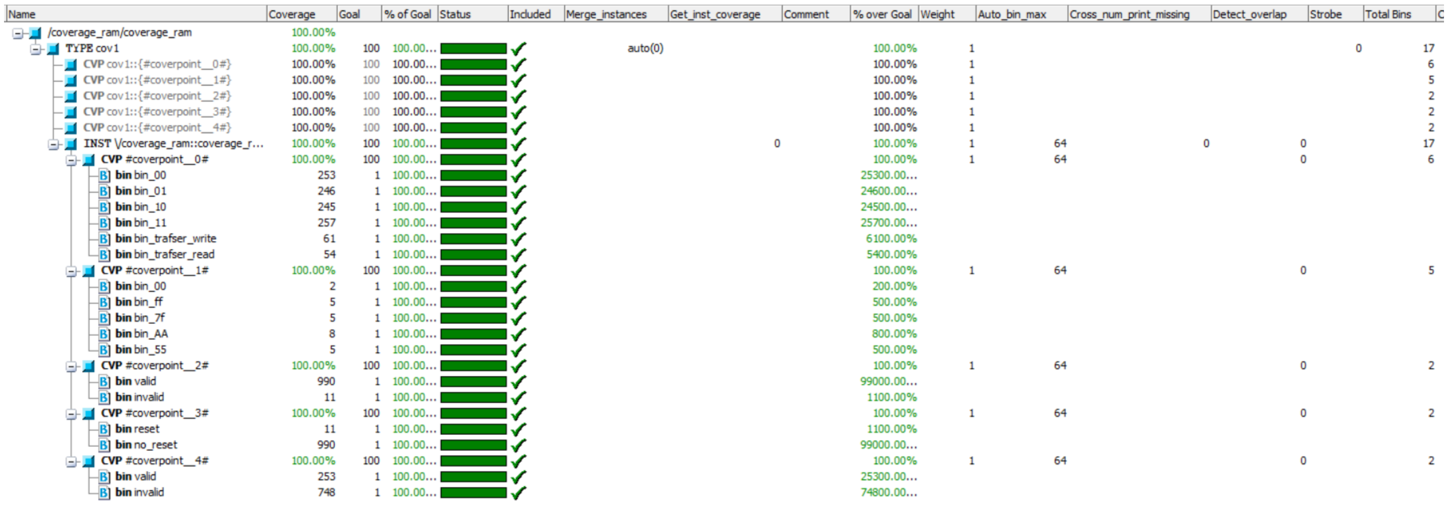


Figure 8. total view waveform RAM

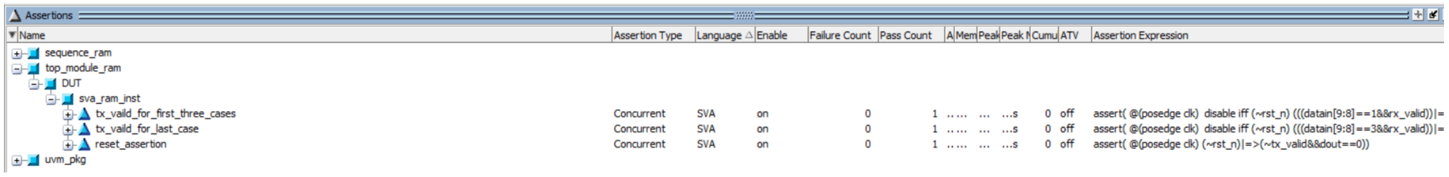
Functional Coverage



Name	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment	% over Goal	Weight	Auto_bin_max	Cross_num_print_missing	Detect_overlap	Strobe	Total Bins	C
/coverage_ram/coverage_ram	100.00%															
TYPE cov1	100.00%	100	100.00...	🟩	✓	auto(0)			100.00%	1				0	17	
CVP cov1::{#coverpoint__0#}	100.00%	100	100.00...	🟩	✓				100.00%	1					6	
CVP cov1::{#coverpoint__1#}	100.00%	100	100.00...	🟩	✓				100.00%	1					5	
CVP cov1::{#coverpoint__2#}	100.00%	100	100.00...	🟩	✓				100.00%	1					2	
CVP cov1::{#coverpoint__3#}	100.00%	100	100.00...	🟩	✓				100.00%	1					2	
CVP cov1::{#coverpoint__4#}	100.00%	100	100.00...	🟩	✓				100.00%	1					2	
INST \/coverage_ram::coverage_r...	100.00%	100	100.00...	🟩	✓		0		100.00%	1	64	0	0		17	
CVP #coverpoint__0#	100.00%	100	100.00...	🟩	✓				100.00%	1	64		0		6	
bin bin_00	253	1	100.00...	🟩	✓				25300.00...							
bin bin_01	246	1	100.00...	🟩	✓				24600.00...							
bin bin_10	245	1	100.00...	🟩	✓				24500.00...							
bin bin_11	257	1	100.00...	🟩	✓				25700.00...							
bin bin_trafser_write	61	1	100.00...	🟩	✓				6100.00%							
bin bin_trafser_read	54	1	100.00...	🟩	✓				5400.00%							
CVP #coverpoint__1#	100.00%	100	100.00...	🟩	✓				100.00%	1	64		0		5	
bin bin_00	2	1	100.00...	🟩	✓				200.00%							
bin bin_ff	5	1	100.00...	🟩	✓				500.00%							
bin bin_7f	5	1	100.00...	🟩	✓				500.00%							
bin bin_AA	8	1	100.00...	🟩	✓				800.00%							
bin bin_55	5	1	100.00...	🟩	✓				500.00%							
CVP #coverpoint__2#	100.00%	100	100.00...	🟩	✓				100.00%	1	64		0		2	
bin valid	990	1	100.00...	🟩	✓				99000.00...							
bin invalid	11	1	100.00...	🟩	✓				1100.00%							
CVP #coverpoint__3#	100.00%	100	100.00...	🟩	✓				100.00%	1	64		0		2	
bin reset	11	1	100.00...	🟩	✓				1100.00%							
bin no_reset	990	1	100.00...	🟩	✓				99000.00...							
CVP #coverpoint__4#	100.00%	100	100.00...	🟩	✓				100.00%	1	64		0		2	
bin valid	253	1	100.00...	🟩	✓				25300.00...							
bin invalid	748	1	100.00...	🟩	✓				74800.00...							

Figure 9. Functional coverage RAM

Assertion

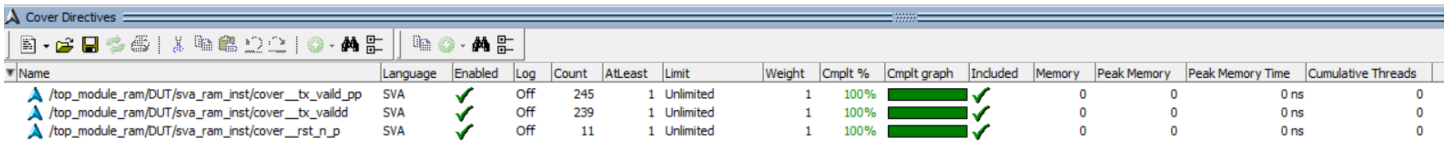


Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	A	Mem	Peak	Peak M	Cumu	ATV	Assertion Expression
sequence_ram												
top_module_ram												
DUT												
sva_ram_inst												
tx_vaild_for_first_three_cases	Concurrent	SVA	on	0	1	...	..	...	...s	0	off	assert(@(posedge clk) disable iff (~rst_n) (((datain[9:8]==1&&rx_valid))|=
tx_vaild_for_last_case	Concurrent	SVA	on	0	1	...	..	...	...s	0	off	assert(@(posedge clk) disable iff (~rst_n) (((datain[9:8]==3&&rx_valid))|=
reset_assertion	Concurrent	SVA	on	0	1	...	..	...	...s	0	off	assert(@(posedge clk) (~rst_n)|=>(~tx_valid&&dout==0))
uvm_pkg												

Figure 10. SVA for RAM

Cover Directives

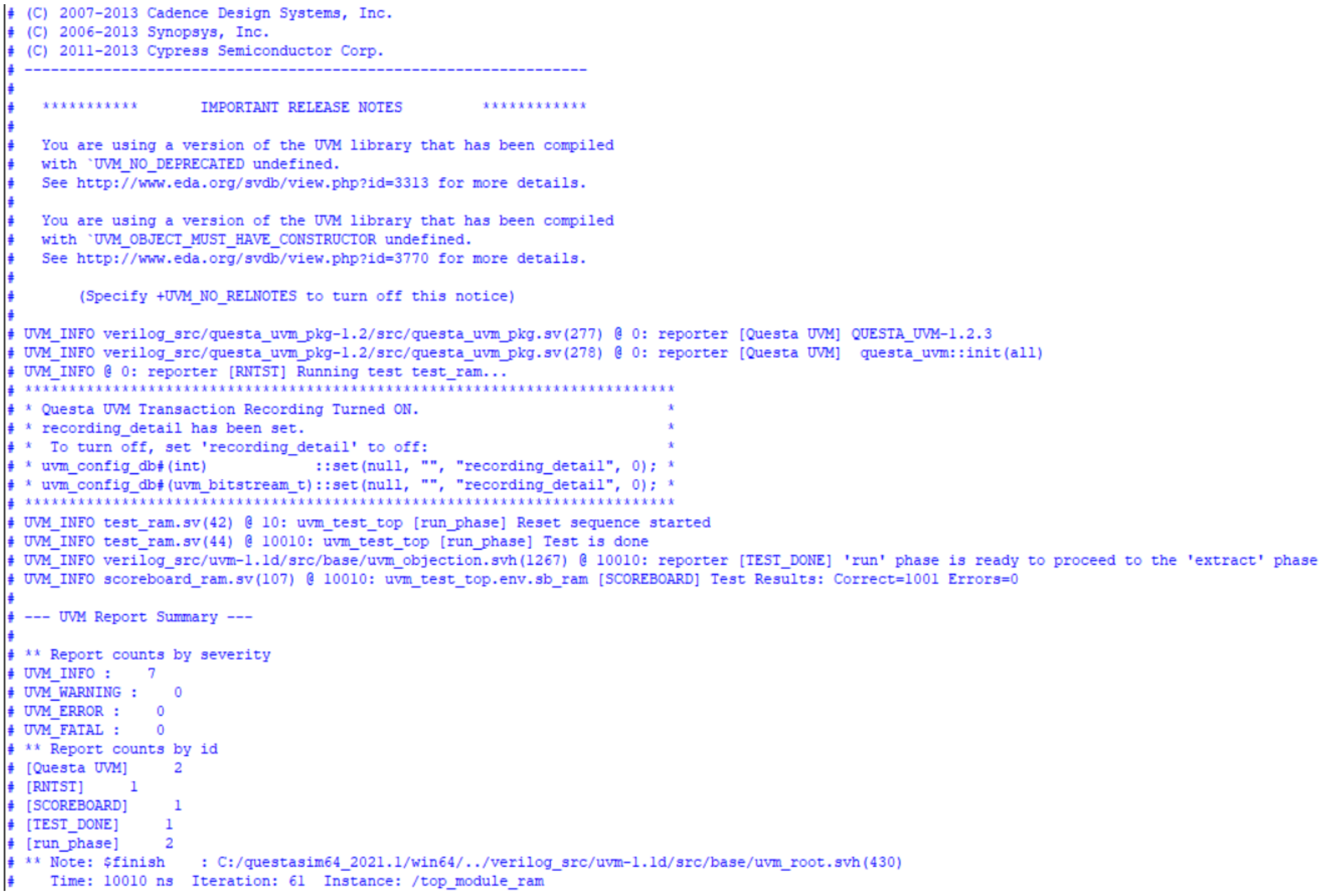


Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top_module_ram/DUT/sva_ram_inst/cover__tx_vaild_pp	SVA	✓	Off	245	1	Unlimited	1	100%	🟩	✓	0	0	0 ns	0
/top_module_ram/DUT/sva_ram_inst/cover__tx_vaildd	SVA	✓	Off	239	1	Unlimited	1	100%	🟩	✓	0	0	0 ns	0
/top_module_ram/DUT/sva_ram_inst/cover__rst_n_p	SVA	✓	Off	11	1	Unlimited	1	100%	🟩	✓	0	0	0 ns	0

Figure 11. cover directives for RAM

UVM Report



```
# (C) 2007-2013 Cadence Design Systems, Inc.
# (C) 2006-2013 Synopsys, Inc.
# (C) 2011-2013 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# 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 test_ram...
# ****************************************************************************
# * 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 test_ram.sv(42) @ 10: uvm_test_top [run_phase] Reset sequence started
# UVM_INFO test_ram.sv(44) @ 10010: uvm_test_top [run_phase] Test is done
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 10010: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO scoreboard_ram.sv(107) @ 10010: uvm_test_top.env.sb_ram [SCOREBOARD] Test Results: Correct=1001 Errors=0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    7
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]     1
# [TEST_DONE]     1
# [run_phase]     2
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 10010 ns  Iteration: 61  Instance: /top_module_ram
```

Figure 12. UVM report for RAM

Slave Results

Waveform

Reset waveform

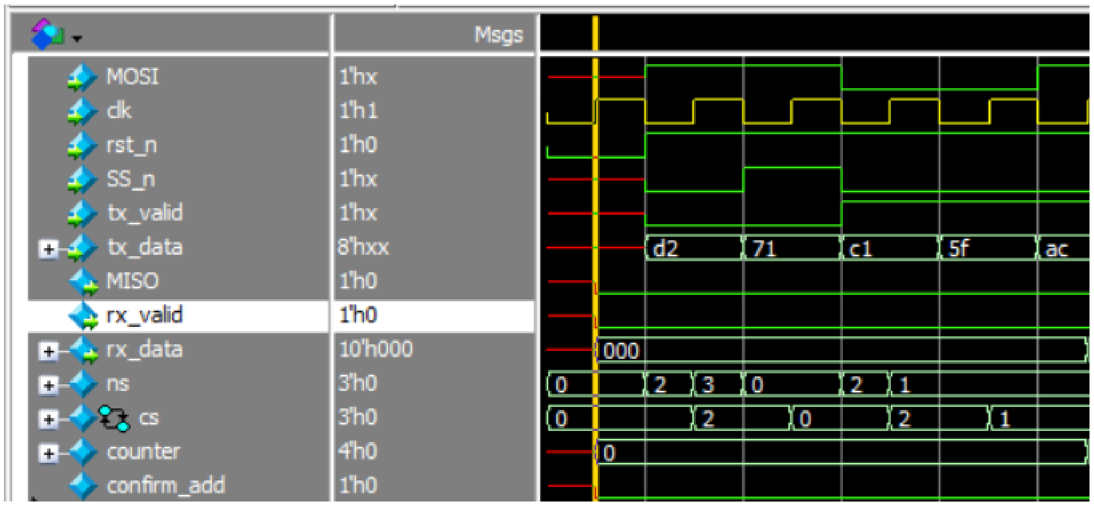
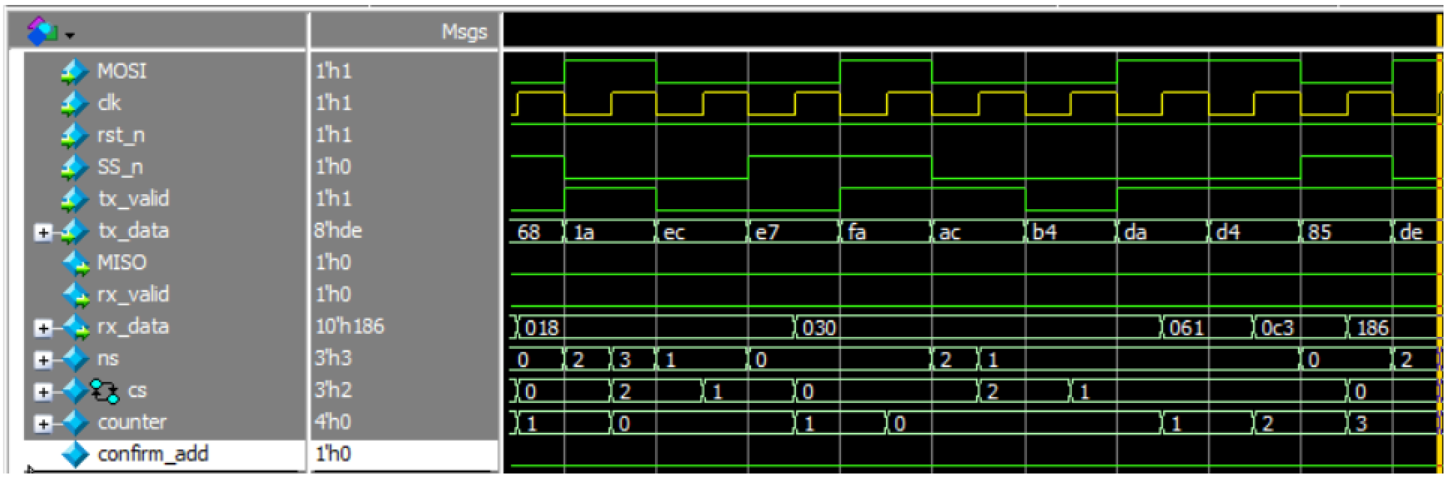
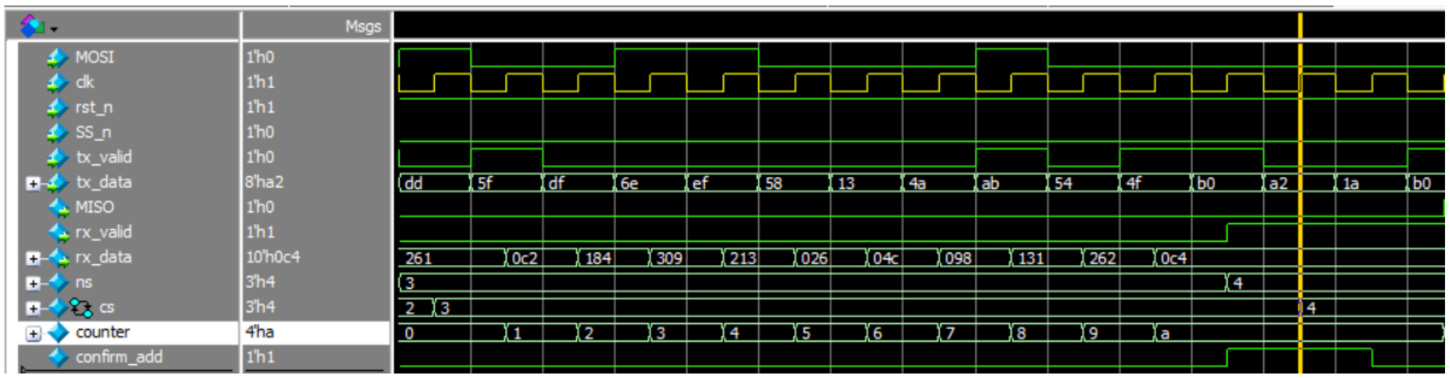


Figure 13. Reset waveform slave

Read and write waveform





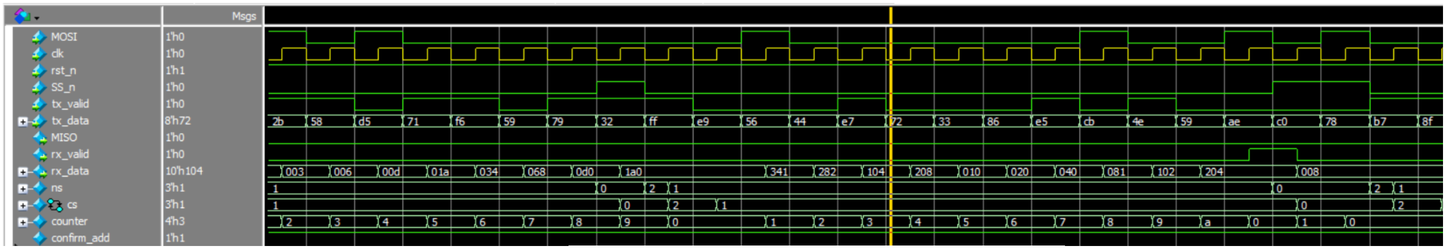


Figure 14. Read and write waveform slave

Total view waveform

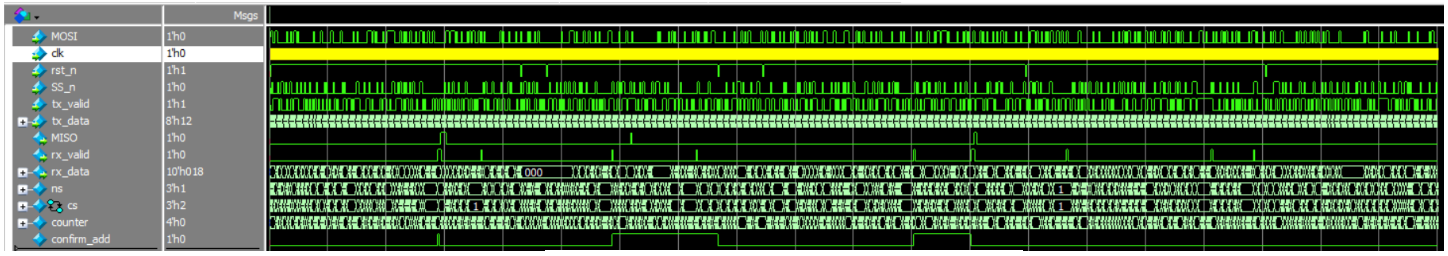
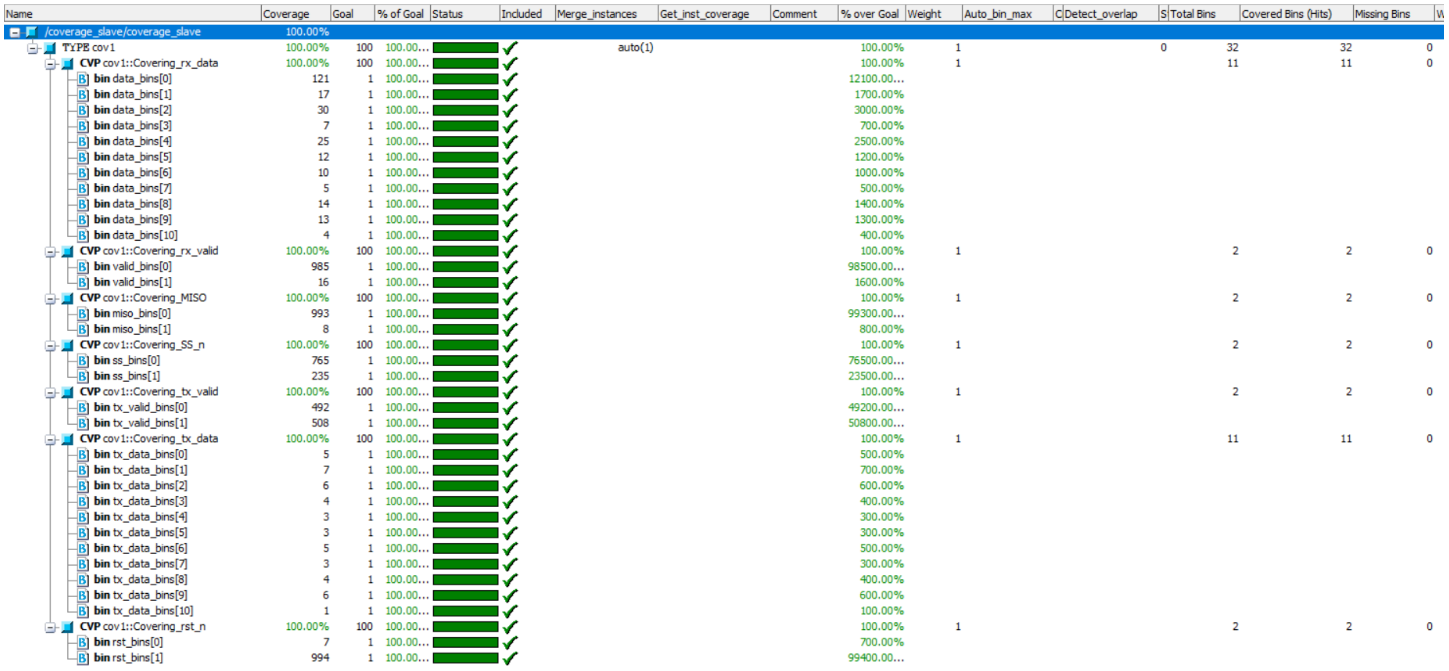


Figure 15. total view waveform for slave

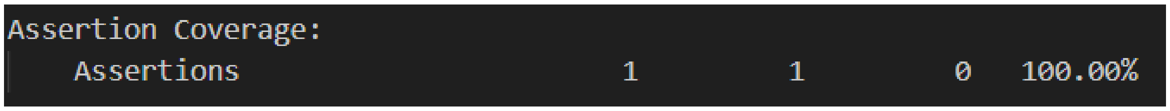
Functional Coverage



Name	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment	% over Goal	Weight	Auto_bin_max	C	Detect_overlap	S	Total Bins	Covered Bins (Hits)	Missing Bins	W
/coverage_slave/coverage_slave	100.00%																	
TYPE cov1	100.00%	100	100.00...	🟩	✓	auto(1)			100.00%	1			0		32	32	0	
CVP cov1::Covering_rx_data	100.00%	100	100.00...	🟩	✓				100.00%	1					11	11	0	
bin data_bins[0]	121	1	100.00...	🟩	✓				12100.00...									
bin data_bins[1]	17	1	100.00...	🟩	✓				1700.00%									
bin data_bins[2]	30	1	100.00...	🟩	✓				3000.00%									
bin data_bins[3]	7	1	100.00...	🟩	✓				700.00%									
bin data_bins[4]	25	1	100.00...	🟩	✓				2500.00%									
bin data_bins[5]	12	1	100.00...	🟩	✓				1200.00%									
bin data_bins[6]	10	1	100.00...	🟩	✓				1000.00%									
bin data_bins[7]	5	1	100.00...	🟩	✓				500.00%									
bin data_bins[8]	14	1	100.00...	🟩	✓				1400.00%									
bin data_bins[9]	13	1	100.00...	🟩	✓				1300.00%									
bin data_bins[10]	4	1	100.00...	🟩	✓				400.00%									
CVP cov1::Covering_rx_valid	100.00%	100	100.00...	🟩	✓				100.00%	1					2	2	0	
bin valid_bins[0]	985	1	100.00...	🟩	✓				98500.00...									
bin valid_bins[1]	16	1	100.00...	🟩	✓				1600.00%									
CVP cov1::Covering_MISO	100.00%	100	100.00...	🟩	✓				100.00%	1					2	2	0	
bin miso_bins[0]	993	1	100.00...	🟩	✓				99300.00...									
bin miso_bins[1]	8	1	100.00...	🟩	✓				800.00%									
CVP cov1::Covering_SS_n	100.00%	100	100.00...	🟩	✓				100.00%	1					2	2	0	
bin ss_bins[0]	765	1	100.00...	🟩	✓				76500.00...									
bin ss_bins[1]	235	1	100.00...	🟩	✓				23500.00...									
CVP cov1::Covering_tx_valid	100.00%	100	100.00...	🟩	✓				100.00%	1					2	2	0	
bin tx_valid_bins[0]	492	1	100.00...	🟩	✓				49200.00...									
bin tx_valid_bins[1]	508	1	100.00...	🟩	✓				50800.00...									
CVP cov1::Covering_tx_data	100.00%	100	100.00...	🟩	✓				100.00%	1					11	11	0	
bin tx_data_bins[0]	5	1	100.00...	🟩	✓				500.00%									
bin tx_data_bins[1]	7	1	100.00...	🟩	✓				700.00%									
bin tx_data_bins[2]	6	1	100.00...	🟩	✓				600.00%									
bin tx_data_bins[3]	4	1	100.00...	🟩	✓				400.00%									
bin tx_data_bins[4]	3	1	100.00...	🟩	✓				300.00%									
bin tx_data_bins[5]	3	1	100.00...	🟩	✓				300.00%									
bin tx_data_bins[6]	5	1	100.00...	🟩	✓				500.00%									
bin tx_data_bins[7]	3	1	100.00...	🟩	✓				300.00%									
bin tx_data_bins[8]	4	1	100.00...	🟩	✓				400.00%									
bin tx_data_bins[9]	6	1	100.00...	🟩	✓				600.00%									
bin tx_data_bins[10]	1	1	100.00...	🟩	✓				100.00%									
CVP cov1::Covering_rst_n	100.00%	100	100.00...	🟩	✓				100.00%	1					2	2	0	
bin rst_bins[0]	7	1	100.00...	🟩	✓				700.00%									
bin rst_bins[1]	994	1	100.00...	🟩	✓				99400.00...									

Figure 16. Functional Coverage for slave

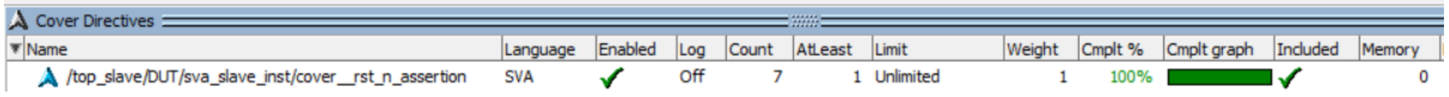
Assertion



```
Assertion Coverage:
    Assertions                    1          1          0    100.00%
```

Figure 17. SVA for slave

Code Directives

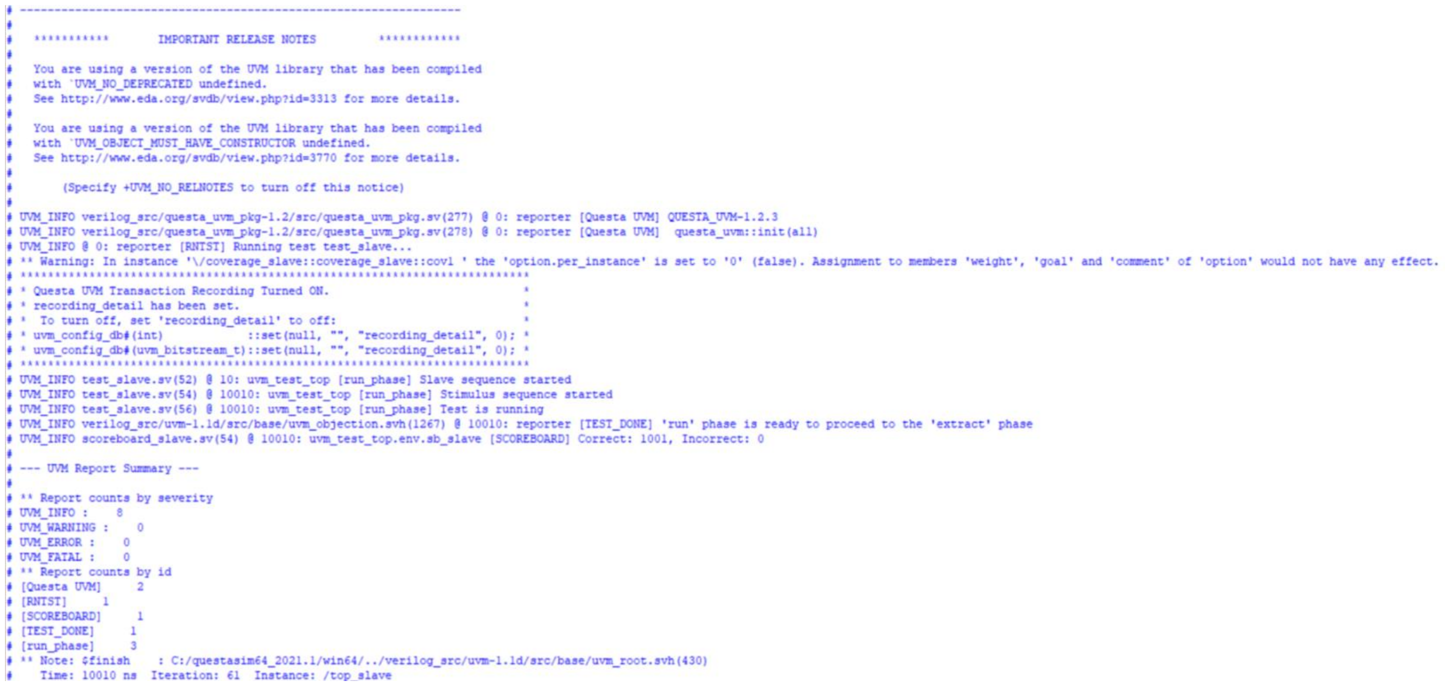


Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory
/top_slave/DUT/sva_slave_inst/cover__rst_n_assertion	SVA	✓	Off	7	1	Unlimited	1	100%	🟩	✓	0

Figure 18. Code directives for slave

UVM Report



```
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# 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 test_slave...
# ** Warning: In instance '\/coverage_slave::coverage_slave::cov1 ' the 'option.per_instance' is set to '0' (false). Assignment to members 'weight', 'goal' and 'comment' of 'option' would not have any effect.
# ****************************************************************************
# * 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 test_slave.sv(52) @ 10: uvm_test_top [run_phase] Slave sequence started
# UVM_INFO test_slave.sv(54) @ 10010: uvm_test_top [run_phase] Stimulus sequence started
# UVM_INFO test_slave.sv(56) @ 10010: uvm_test_top [run_phase] Test is running
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 10010: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO scoreboard_slave.sv(54) @ 10010: uvm_test_top.env.sb_slave [SCOREBOARD] Correct: 1001, Incorrect: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    8
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]     1
# [TEST_DONE]     1
# [run_phase]     3
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 10010 ns  Iteration: 61  Instance: /top_slave
```

Figure 19. UVM report for slave

Wrapper Results

Waveform

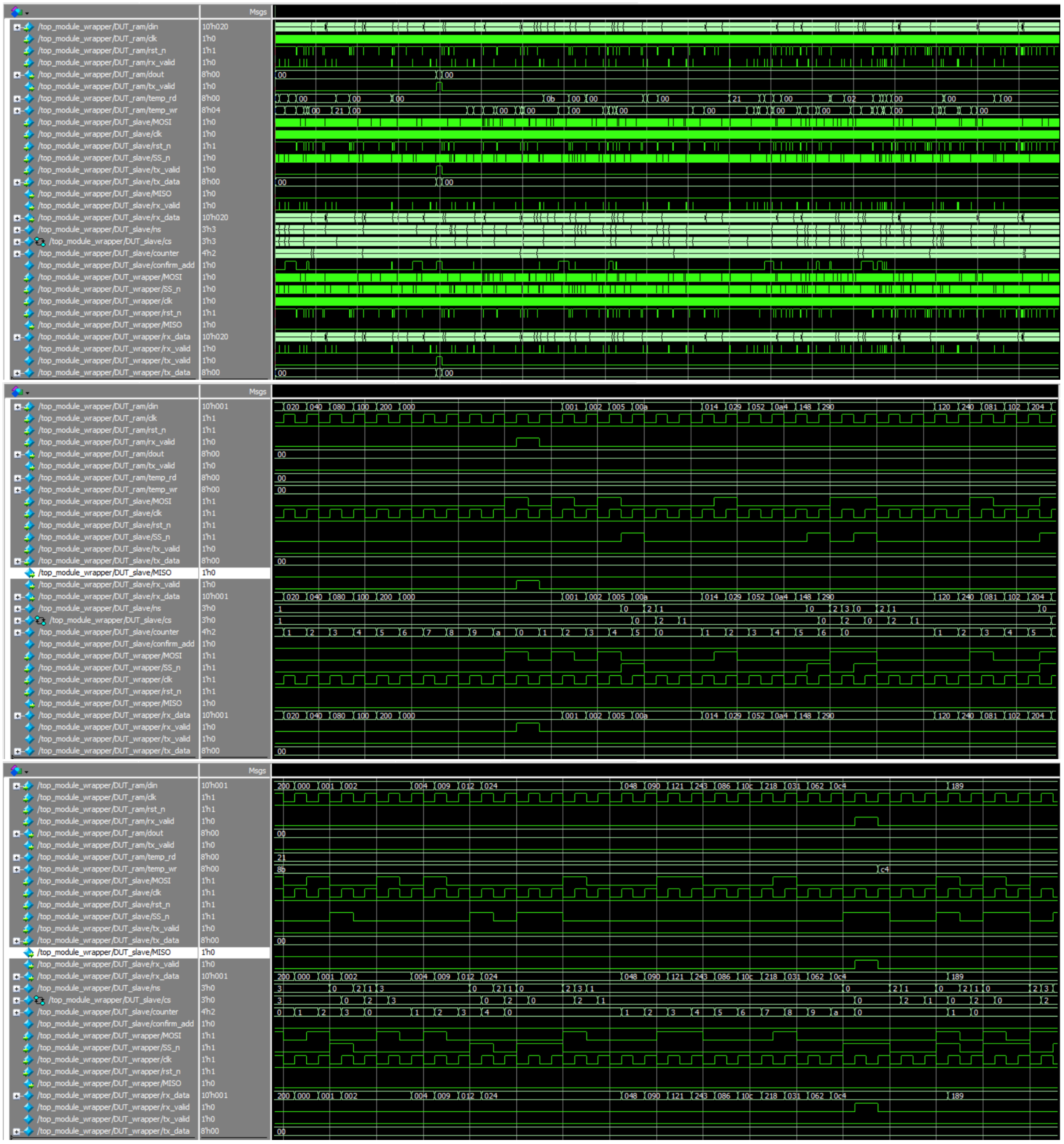
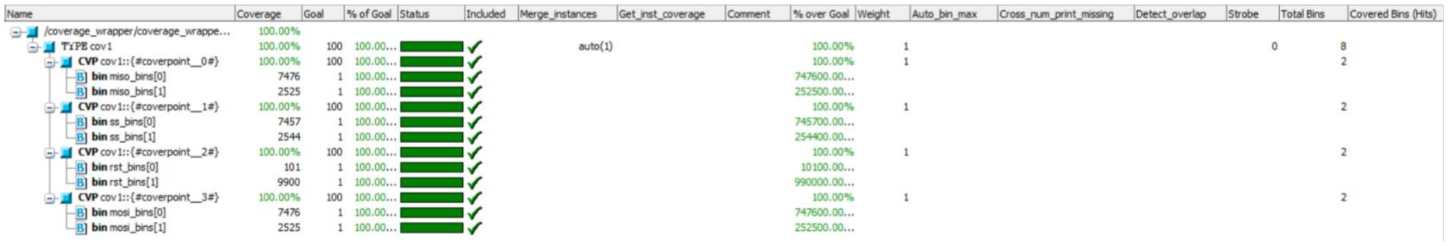

Msgs
/top_module_wrapper/DUT_ram/din
/top_module_wrapper/DUT_ram/clk
/top_module_wrapper/DUT_ram/rst_n
/top_module_wrapper/DUT_ram/rx_valid
/top_module_wrapper/DUT_ram/dout
/top_module_wrapper/DUT_ram/tx_valid
/top_module_wrapper/DUT_ram/temp_rd
/top_module_wrapper/DUT_ram/temp_wr
/top_module_wrapper/DUT_slave/MOSI
/top_module_wrapper/DUT_slave/clk
/top_module_wrapper/DUT_slave/rst_n
/top_module_wrapper/DUT_slave/SS_n
/top_module_wrapper/DUT_slave/tx_valid
/top_module_wrapper/DUT_slave/tx_data
/top_module_wrapper/DUT_slave/MISO
/top_module_wrapper/DUT_slave/rx_valid
/top_module_wrapper/DUT_slave/rx_data
/top_module_wrapper/DUT_slave/ns
/top_module_wrapper/DUT_slave/cs
/top_module_wrapper/DUT_slave/counter
/top_module_wrapper/DUT_slave/confirm_add
/top_module_wrapper/DUT_wrapper/MOSI
/top_module_wrapper/DUT_wrapper/SS_n
/top_module_wrapper/DUT_wrapper/clk
/top_module_wrapper/DUT_wrapper/rst_n
/top_module_wrapper/DUT_wrapper/MISO
/top_module_wrapper/DUT_wrapper/rx_data
/top_module_wrapper/DUT_wrapper/rx_valid
/top_module_wrapper/DUT_wrapper/tx_valid
/top_module_wrapper/DUT_wrapper/tx_data

Figure 20. waveform for wrapper

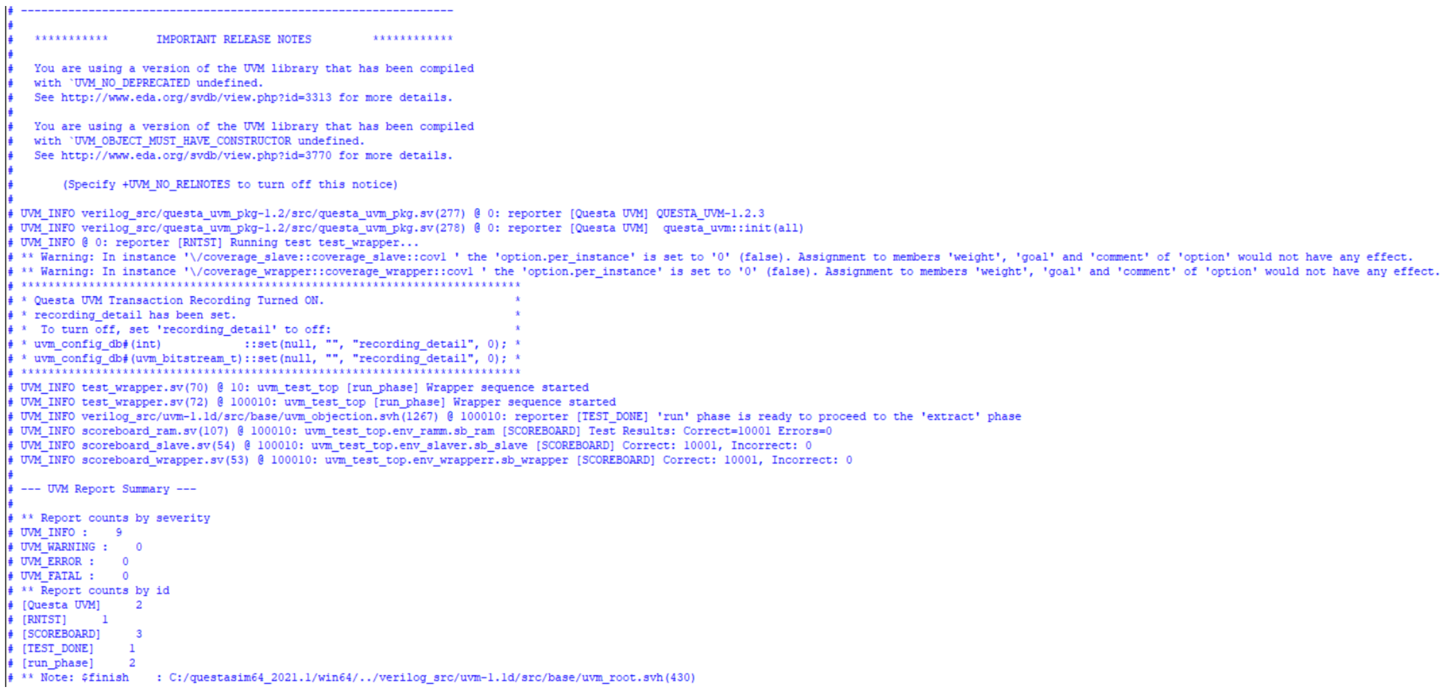
Functional Coverage



Name	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment	% over Goal	Weight	Auto_bin_max	Cross_num_print_missing	Detect_overlap	Strobe	Total Bins	Covered Bins (Hits)
/coverage_wrapper/coverage_wrappe...	100.00%															
TYPE cov1	100.00%	100	100.00...	🟩	✓	auto(1)			100.00%	1				0	8	
CVP cov1::{#coverpoint__0#}	100.00%	100	100.00...	🟩	✓				100.00%	1					2	
bin miso_bins[0]	7476	1	100.00...	🟩	✓				747600.00...							
bin miso_bins[1]	2525	1	100.00...	🟩	✓				252500.00...							
CVP cov1::{#coverpoint__1#}	100.00%	100	100.00...	🟩	✓				100.00%	1					2	
bin ss_bins[0]	7457	1	100.00...	🟩	✓				745700.00...							
bin ss_bins[1]	2544	1	100.00...	🟩	✓				254400.00...							
CVP cov1::{#coverpoint__2#}	100.00%	100	100.00...	🟩	✓				100.00%	1					2	
bin rst_bins[0]	101	1	100.00...	🟩	✓				10100.00...							
bin rst_bins[1]	9900	1	100.00...	🟩	✓				990000.00...							
CVP cov1::{#coverpoint__3#}	100.00%	100	100.00...	🟩	✓				100.00%	1					2	
bin mosi_bins[0]	7476	1	100.00...	🟩	✓				747600.00...							
bin mosi_bins[1]	2525	1	100.00...	🟩	✓				252500.00...							

Figure 21. Functional Coverage for wrapper

UVM Report



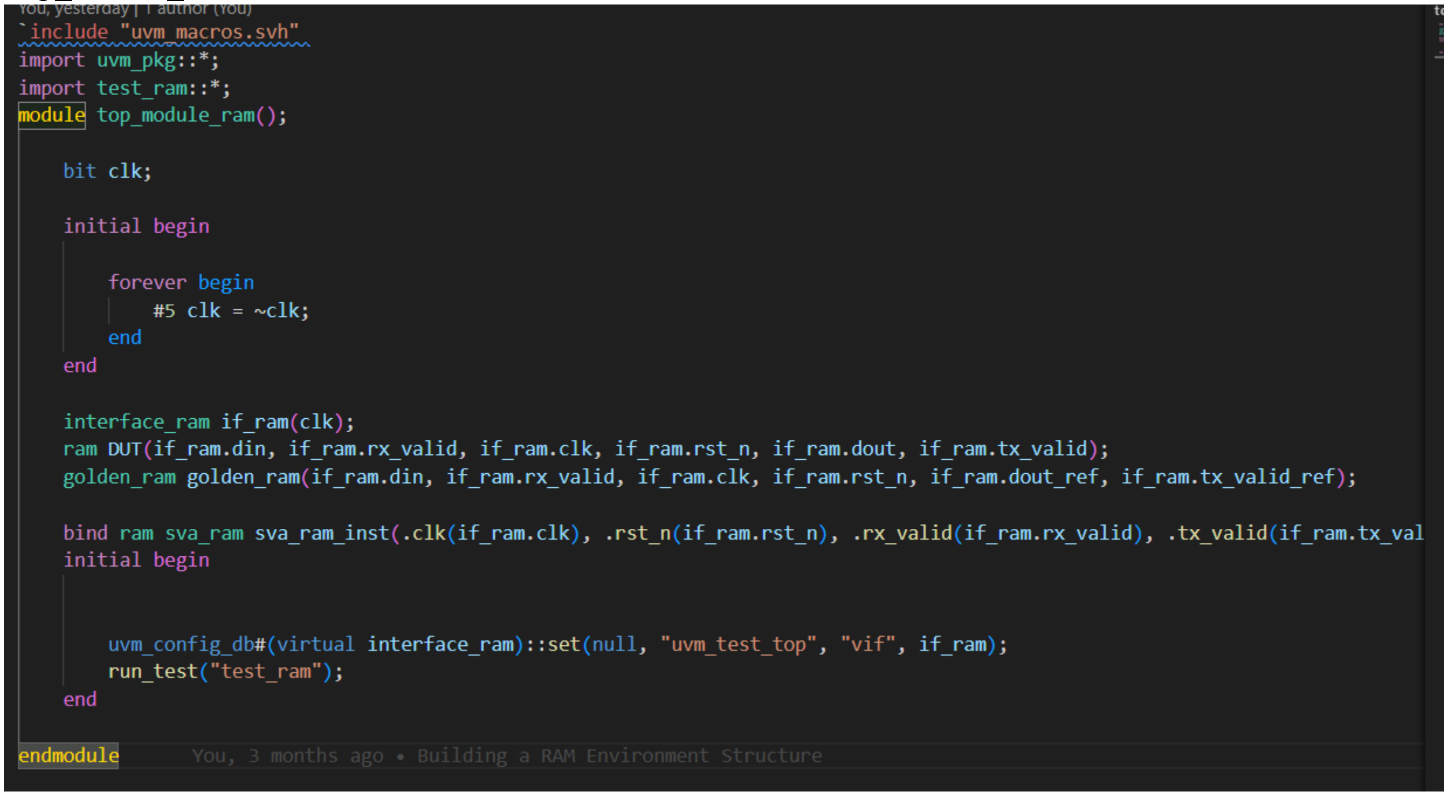
```
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# 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 test_wrapper...
# ** Warning: In instance '\/coverage_slave::coverage_slave::cov1 ' the 'option.per_instance' is set to '0' (false). Assignment to members 'weight', 'goal' and 'comment' of 'option' would not have any effect.
# ** Warning: In instance '\/coverage_wrapper::coverage_wrapper::cov1 ' the 'option.per_instance' is set to '0' (false). Assignment to members 'weight', 'goal' and 'comment' of 'option' would not have any effect.
# **********************************************************************
# * 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 test_wrapper.sv(70) @ 10: uvm_test_top [run_phase] Wrapper sequence started
# UVM_INFO test_wrapper.sv(72) @ 100010: uvm_test_top [run_phase] Wrapper sequence started
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 100010: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO scoreboard_ram.sv(107) @ 100010: uvm_test_top.env_ramm.sb_ram [SCOREBOARD] Test Results: Correct=10001 Errors=0
# UVM_INFO scoreboard_slave.sv(54) @ 100010: uvm_test_top.env_slaver.sb_slave [SCOREBOARD] Correct: 10001, Incorrect: 0
# UVM_INFO scoreboard_wrapper.sv(53) @ 100010: uvm_test_top.env_wrapperr.sb_wrapper [SCOREBOARD] Correct: 10001, Incorrect: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    9
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]     3
# [TEST_DONE]     1
# [run_phase]     2
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
```

Figure 22. UVM report for wrapper

Appendix

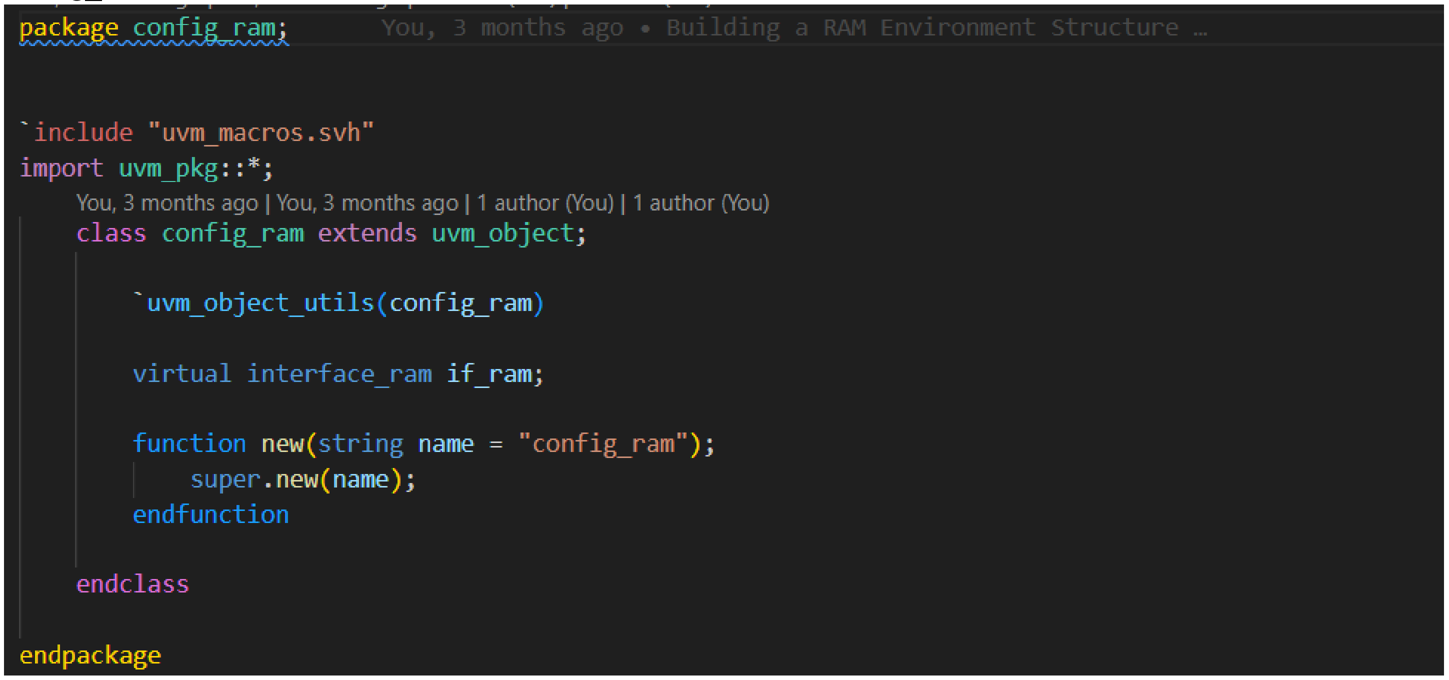
RAM Files

Top_module_ram



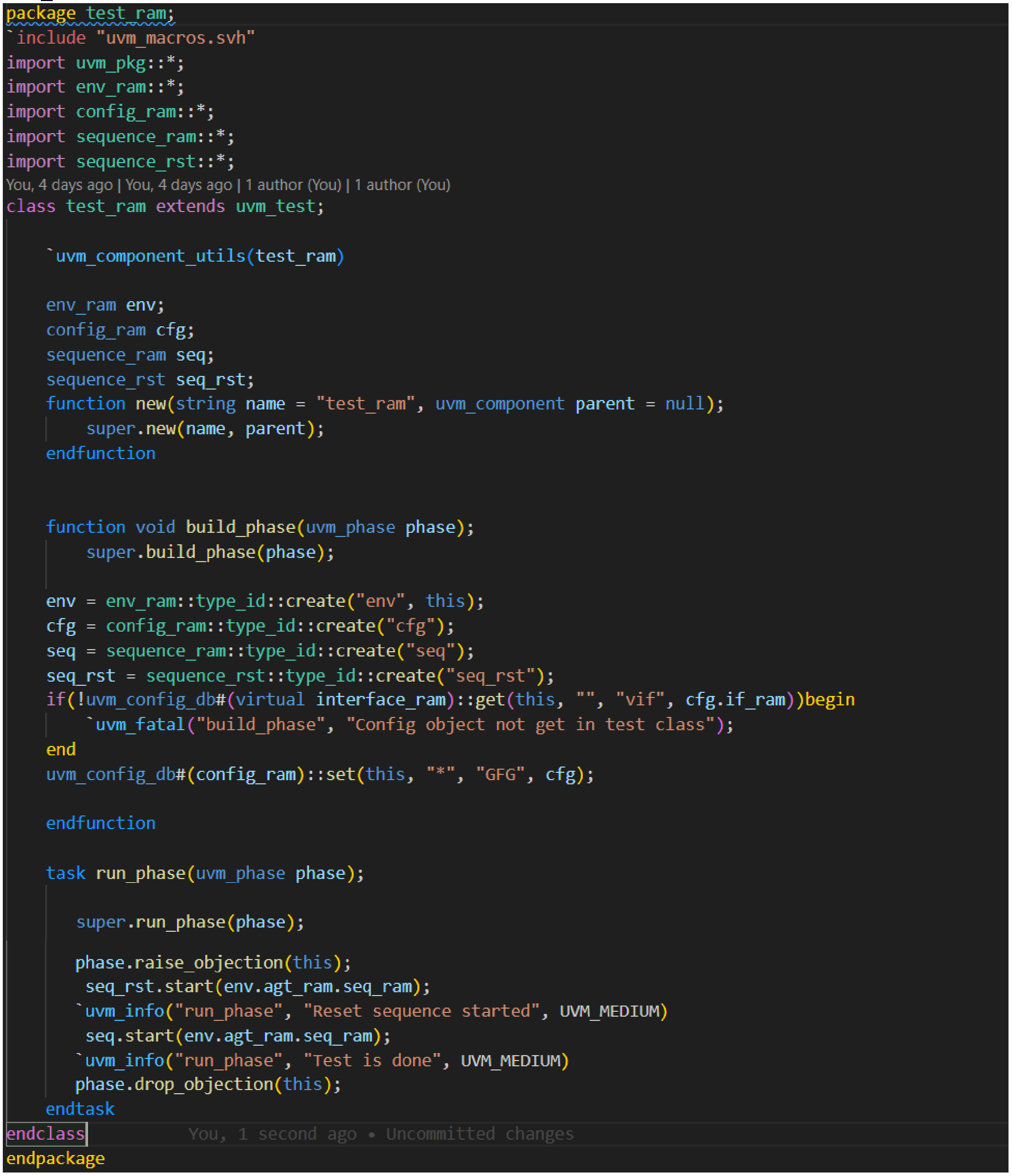
```
You, yesterday | 1 author (You)
`include "uvm_macros.svh"
import uvm_pkg::*;
import test_ram::*;
module top_module_ram();

    bit clk;

    initial begin

        forever begin
            #5 clk = ~clk;
        end
    end

    interface_ram if_ram(clk);
    ram DUT(if_ram.din, if_ram.rx_valid, if_ram.clk, if_ram.rst_n, if_ram.dout, if_ram.tx_valid);
    golden_ram golden_ram(if_ram.din, if_ram.rx_valid, if_ram.clk, if_ram.rst_n, if_ram.dout_ref, if_ram.tx_valid_ref);

    bind ram sva_ram sva_ram_inst(.clk(if_ram.clk), .rst_n(if_ram.rst_n), .rx_valid(if_ram.rx_valid), .tx_valid(if_ram.tx_val
    initial begin


        uvm_config_db#(virtual interface_ram)::set(null, "uvm_test_top", "vif", if_ram);
        run_test("test_ram");
    end

endmodule       You, 3 months ago • Building a RAM Environment Structure
```

config_ram



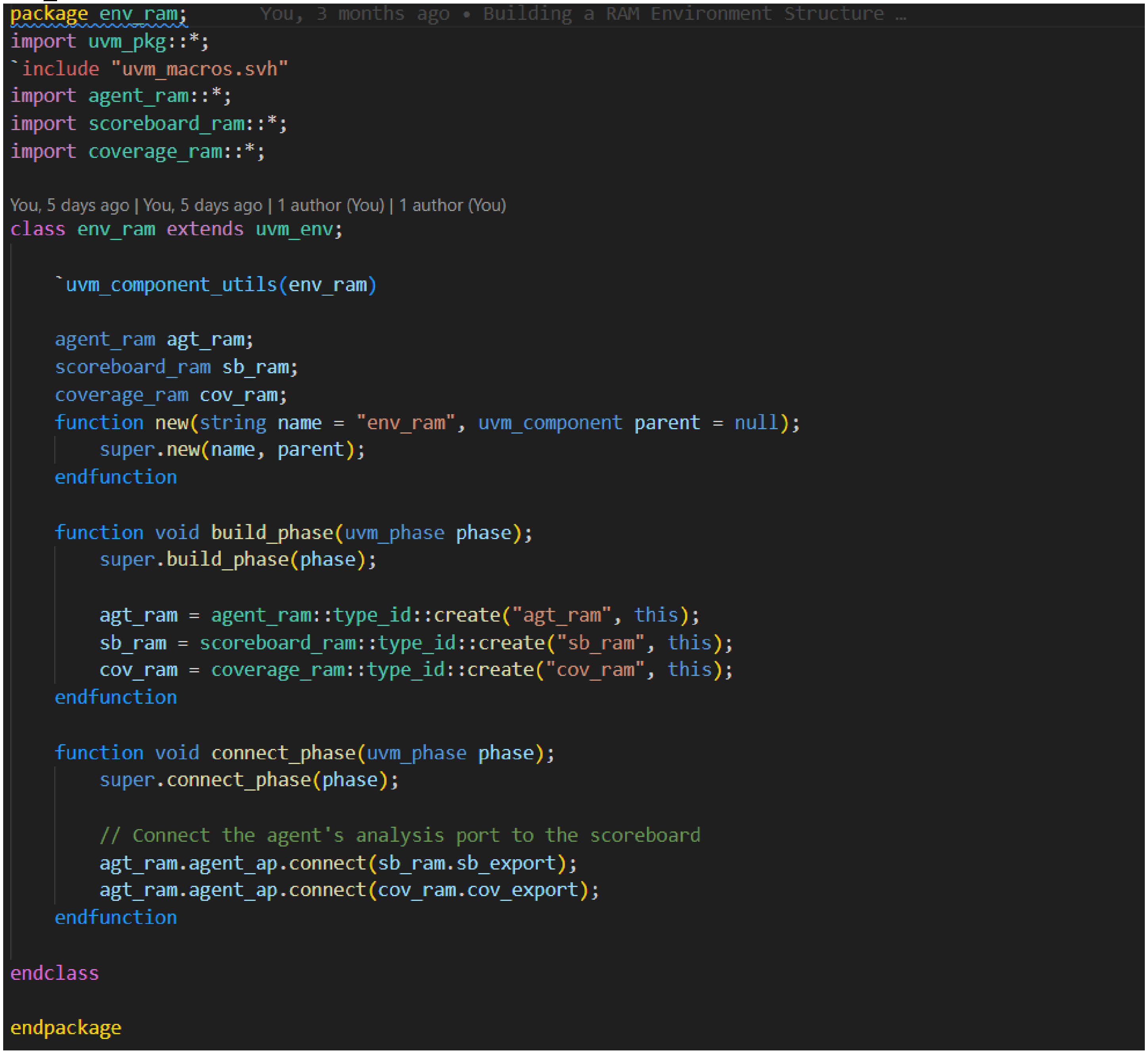
```
package config_ram;       You, 3 months ago • Building a RAM Environment Structure …


`include "uvm_macros.svh"
import uvm_pkg::*;
    You, 3 months ago | You, 3 months ago | 1 author (You) | 1 author (You)
    class config_ram extends uvm_object;

        `uvm_object_utils(config_ram)

        virtual interface_ram if_ram;

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

    endclass

endpackage
```

Test_ram



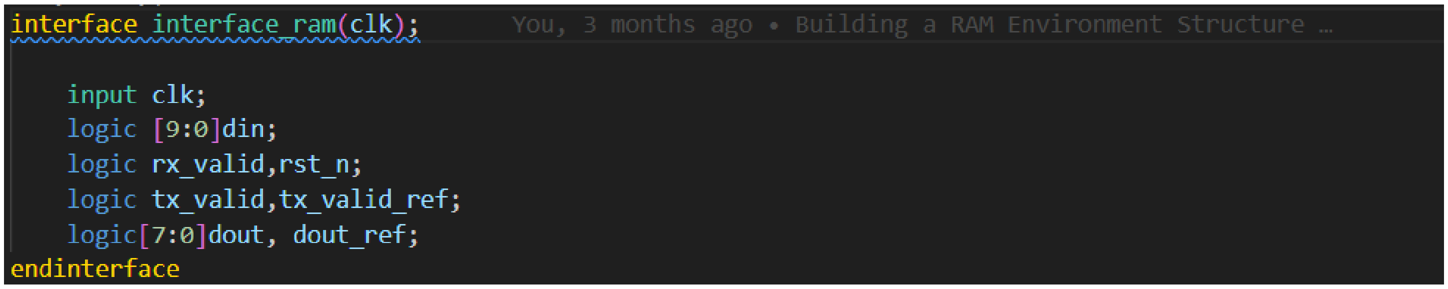
```
package test_ram;
`include "uvm_macros.svh"
import uvm_pkg::*;
import env_ram::*;
import config_ram::*;
import sequence_ram::*;
import sequence_rst::*;
You, 4 days ago | You, 4 days ago | 1 author (You) | 1 author (You)
class test_ram extends uvm_test;

    `uvm_component_utils(test_ram)

    env_ram env;
    config_ram cfg;
    sequence_ram seq;
    sequence_rst seq_rst;
    function new(string name = "test_ram", uvm_component parent = null);
        super.new(name, parent);
    endfunction


    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

    env = env_ram::type_id::create("env", this);
    cfg = config_ram::type_id::create("cfg");
    seq = sequence_ram::type_id::create("seq");
    seq_rst = sequence_rst::type_id::create("seq_rst");
    if(!uvm_config_db#(virtual interface_ram)::get(this, "", "vif", cfg.if_ram))begin
        `uvm_fatal("build_phase", "Config object not get in test class");
    end
    uvm_config_db#(config_ram)::set(this, "*", "GFG", cfg);

    endfunction

    task run_phase(uvm_phase phase);

       super.run_phase(phase);

       phase.raise_objection(this);
        seq_rst.start(env.agt_ram.seq_ram);
       `uvm_info("run_phase", "Reset sequence started", UVM_MEDIUM)
        seq.start(env.agt_ram.seq_ram);
       `uvm_info("run_phase", "Test is done", UVM_MEDIUM)
       phase.drop_objection(this);
    endtask
endclass            You, 1 second ago • Uncommitted changes
endpackage
```

Env_ram



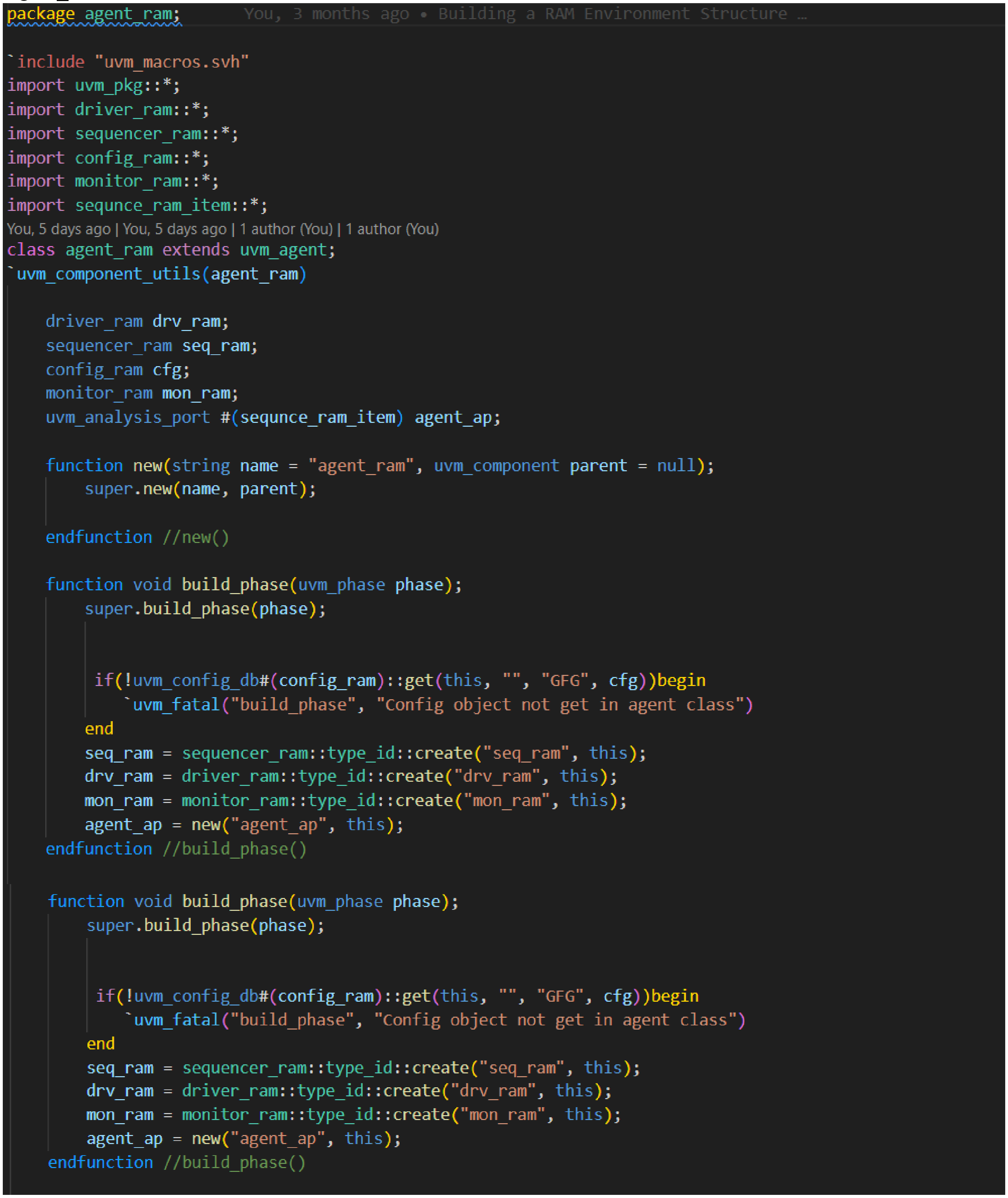
```
package env_ram;        You, 3 months ago • Building a RAM Environment Structure …
import uvm_pkg::*;
`include "uvm_macros.svh"
import agent_ram::*;
import scoreboard_ram::*;
import coverage_ram::*;

You, 5 days ago | You, 5 days ago | 1 author (You) | 1 author (You)
class env_ram extends uvm_env;

    `uvm_component_utils(env_ram)

    agent_ram agt_ram;
    scoreboard_ram sb_ram;
    coverage_ram cov_ram;
    function new(string name = "env_ram", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        agt_ram = agent_ram::type_id::create("agt_ram", this);
        sb_ram = scoreboard_ram::type_id::create("sb_ram", this);
        cov_ram = coverage_ram::type_id::create("cov_ram", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);

        // Connect the agent's analysis port to the scoreboard
        agt_ram.agent_ap.connect(sb_ram.sb_export);
        agt_ram.agent_ap.connect(cov_ram.cov_export);
    endfunction

endclass

endpackage
```

Interface_ram

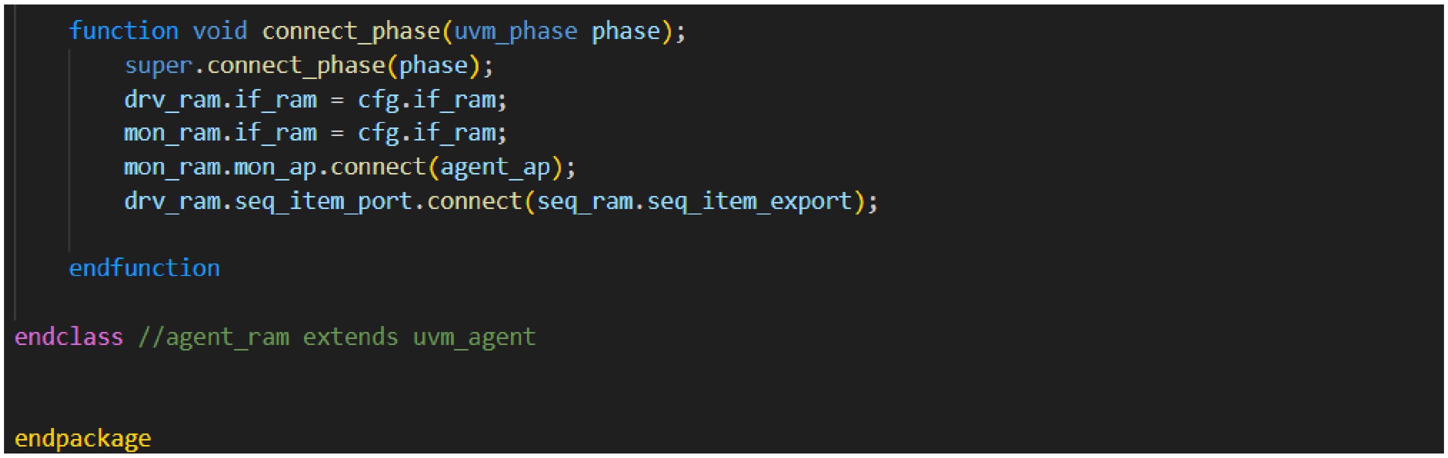


```
interface interface_ram(clk);       You, 3 months ago • Building a RAM Environment Structure …

    input clk;
    logic [9:0]din;
    logic rx_valid,rst_n;
    logic tx_valid,tx_valid_ref;
    logic[7:0]dout, dout_ref;
endinterface
```

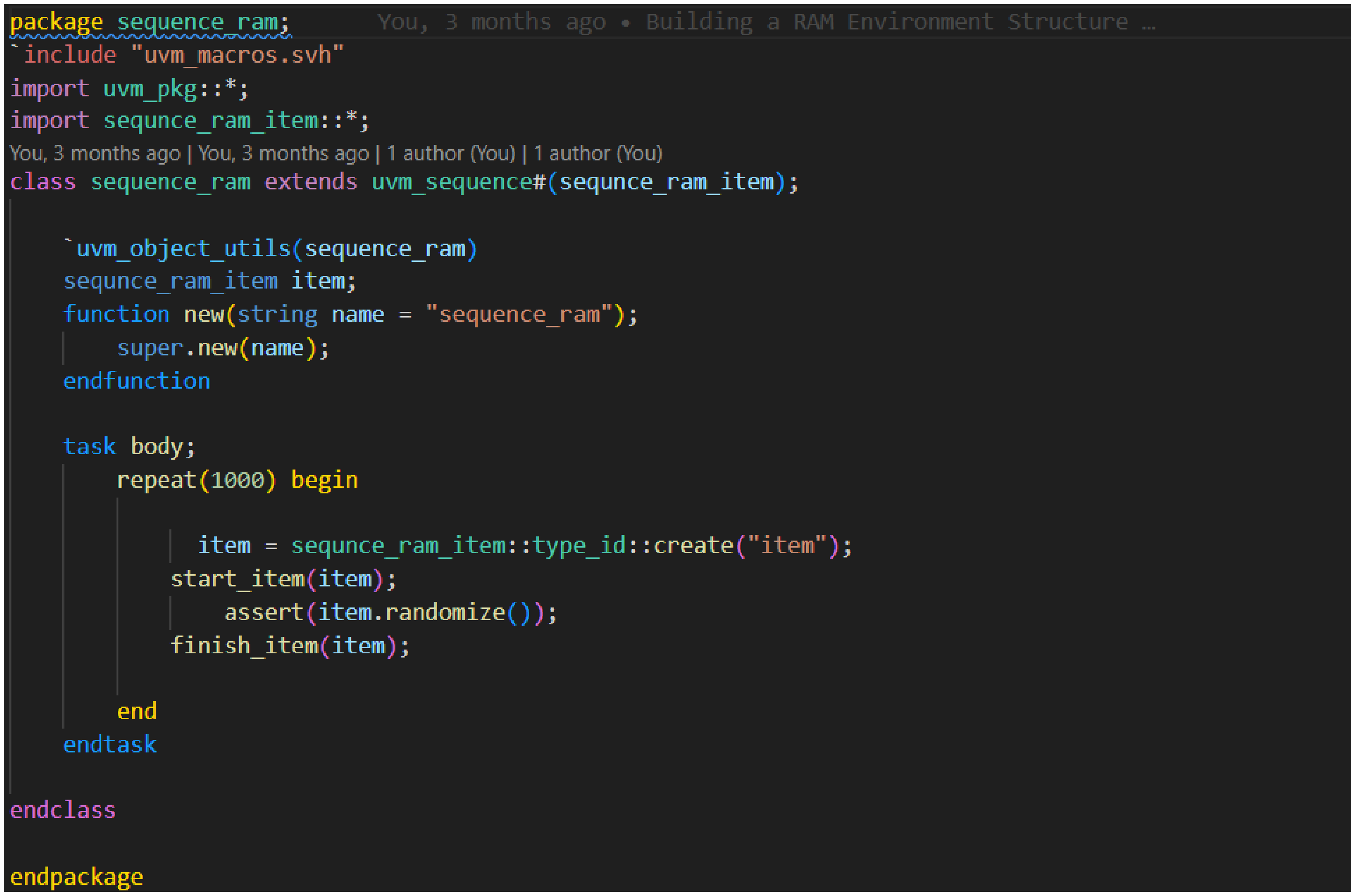
Agent_Ram



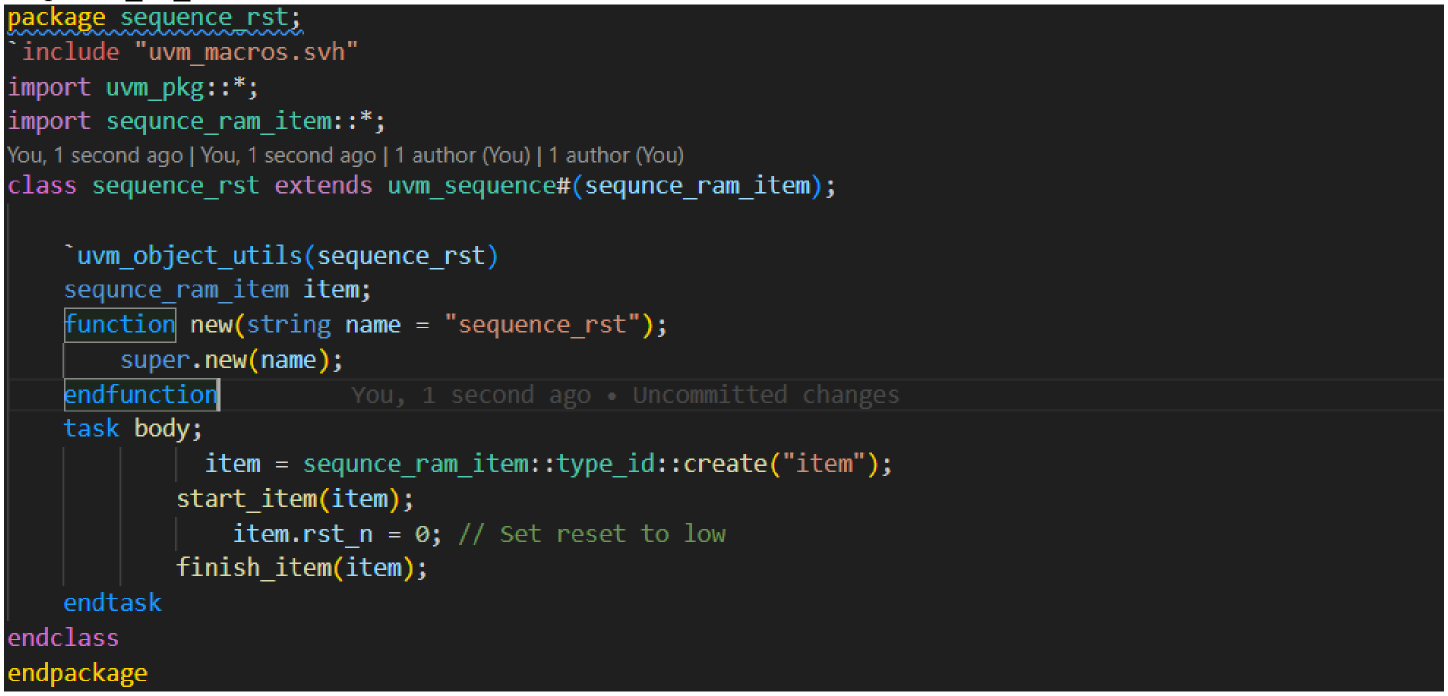
```
package agent_ram;        You, 3 months ago • Building a RAM Environment Structure …

`include "uvm_macros.svh"
import uvm_pkg::*;
import driver_ram::*;
import sequencer_ram::*;
import config_ram::*;
import monitor_ram::*;
import sequnce_ram_item::*;
You, 5 days ago | You, 5 days ago | 1 author (You) | 1 author (You)
class agent_ram extends uvm_agent;
`uvm_component_utils(agent_ram)

    driver_ram drv_ram;
    sequencer_ram seq_ram;
    config_ram cfg;
    monitor_ram mon_ram;
    uvm_analysis_port #(sequnce_ram_item) agent_ap;

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

    endfunction //new()

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);


         if(!uvm_config_db#(config_ram)::get(this, "", "GFG", cfg))begin
            `uvm_fatal("build_phase", "Config object not get in agent class")
        end
        seq_ram = sequencer_ram::type_id::create("seq_ram", this);
        drv_ram = driver_ram::type_id::create("drv_ram", this);
        mon_ram = monitor_ram::type_id::create("mon_ram", this);
        agent_ap = new("agent_ap", this);
    endfunction //build_phase()

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);


         if(!uvm_config_db#(config_ram)::get(this, "", "GFG", cfg))begin
            `uvm_fatal("build_phase", "Config object not get in agent class")
        end
        seq_ram = sequencer_ram::type_id::create("seq_ram", this);
        drv_ram = driver_ram::type_id::create("drv_ram", this);
        mon_ram = monitor_ram::type_id::create("mon_ram", this);
        agent_ap = new("agent_ap", this);
    endfunction //build_phase()
```


```
    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        drv_ram.if_ram = cfg.if_ram;
        mon_ram.if_ram = cfg.if_ram;
        mon_ram.mon_ap.connect(agent_ap);
        drv_ram.seq_item_port.connect(seq_ram.seq_item_export);

    endfunction

endclass //agent_ram extends uvm_agent


endpackage
```

Sequence_ram

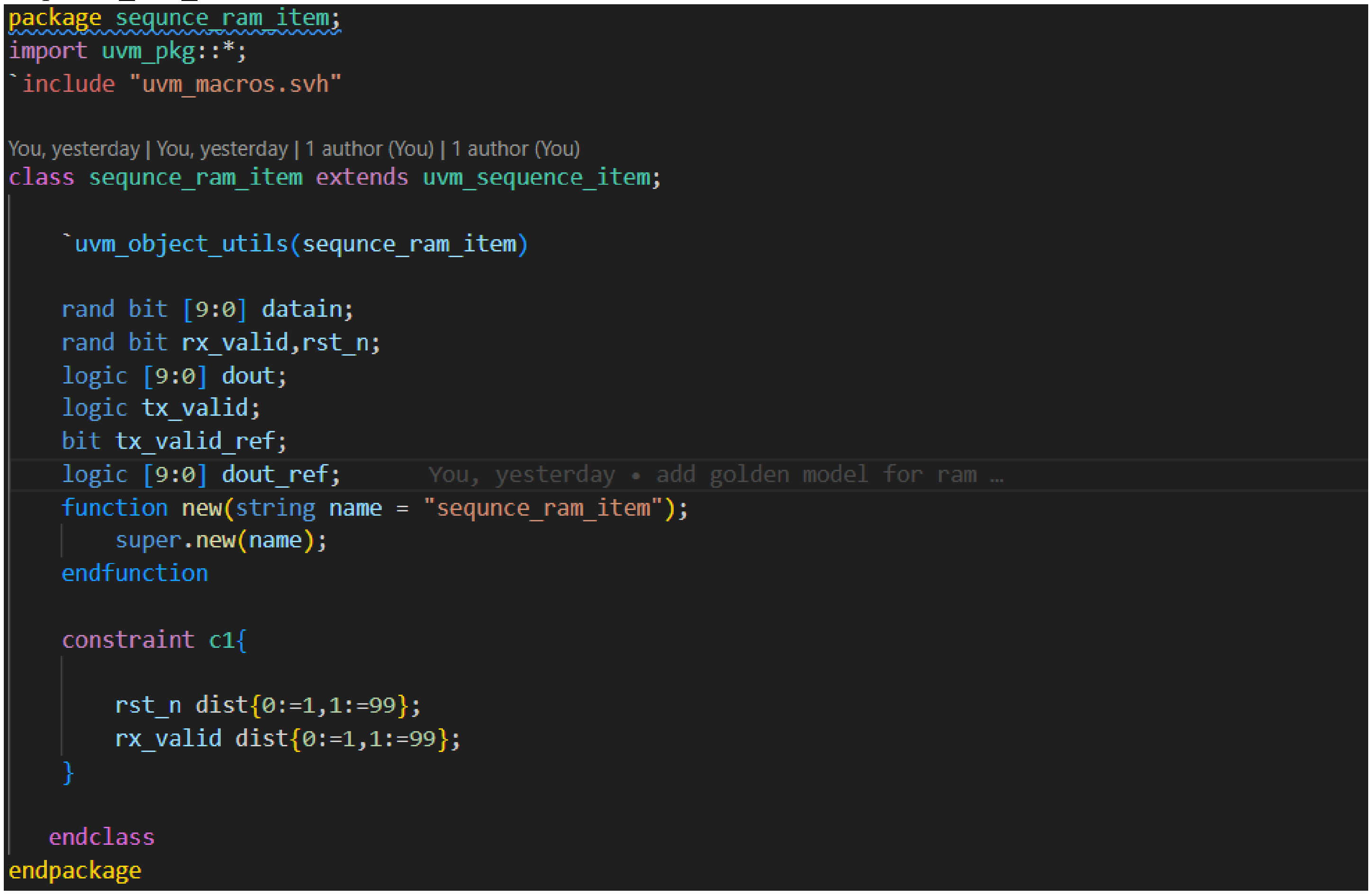

```
package sequence_ram;        You, 3 months ago • Building a RAM Environment Structure …
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequnce_ram_item::*;
You, 3 months ago | You, 3 months ago | 1 author (You) | 1 author (You)
class sequence_ram extends uvm_sequence#(sequnce_ram_item);

    `uvm_object_utils(sequence_ram)
    sequnce_ram_item item;
    function new(string name = "sequence_ram");
        super.new(name);
    endfunction

    task body;
        repeat(1000) begin

            item = sequnce_ram_item::type_id::create("item");
            start_item(item);
                assert(item.randomize());
            finish_item(item);

        end
    endtask

endclass

endpackage
```

Sequence_rst_ram



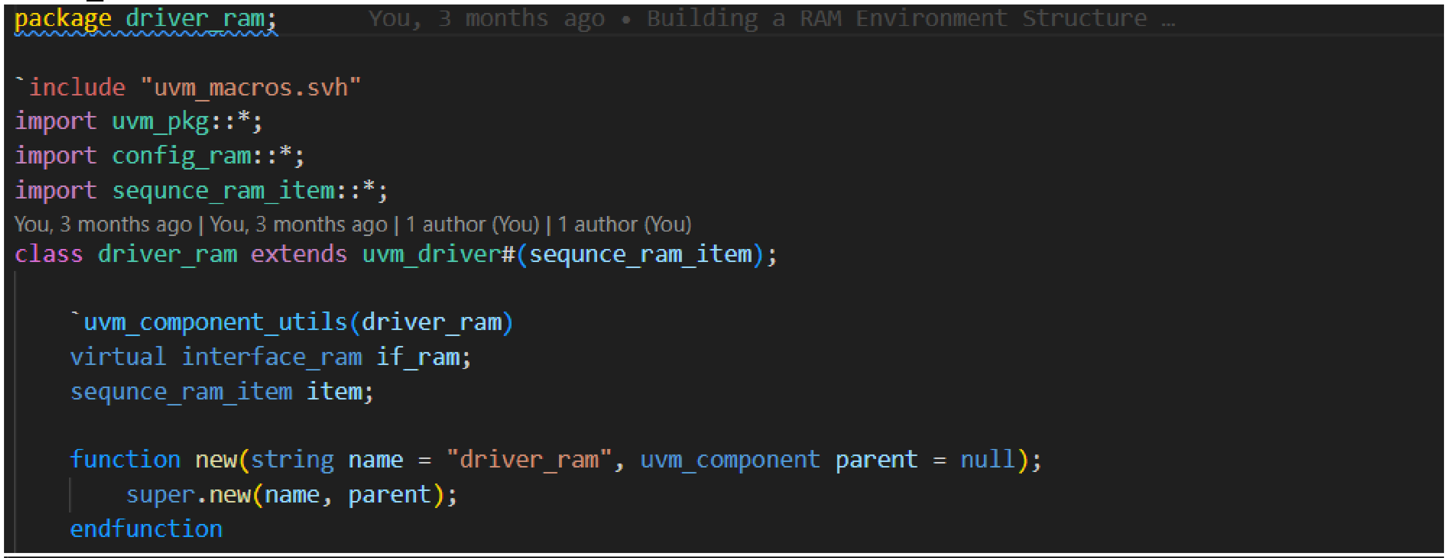
```
package sequence_rst;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequnce_ram_item::*;
You, 1 second ago | You, 1 second ago | 1 author (You) | 1 author (You)
class sequence_rst extends uvm_sequence#(sequnce_ram_item);

    `uvm_object_utils(sequence_rst)
    sequnce_ram_item item;
    function new(string name = "sequence_rst");
        super.new(name);
    endfunction          You, 1 second ago • Uncommitted changes
    task body;
            item = sequnce_ram_item::type_id::create("item");
        start_item(item);
            item.rst_n = 0; // Set reset to low
        finish_item(item);
    endtask
endclass
endpackage
```

Sequence_ram_item

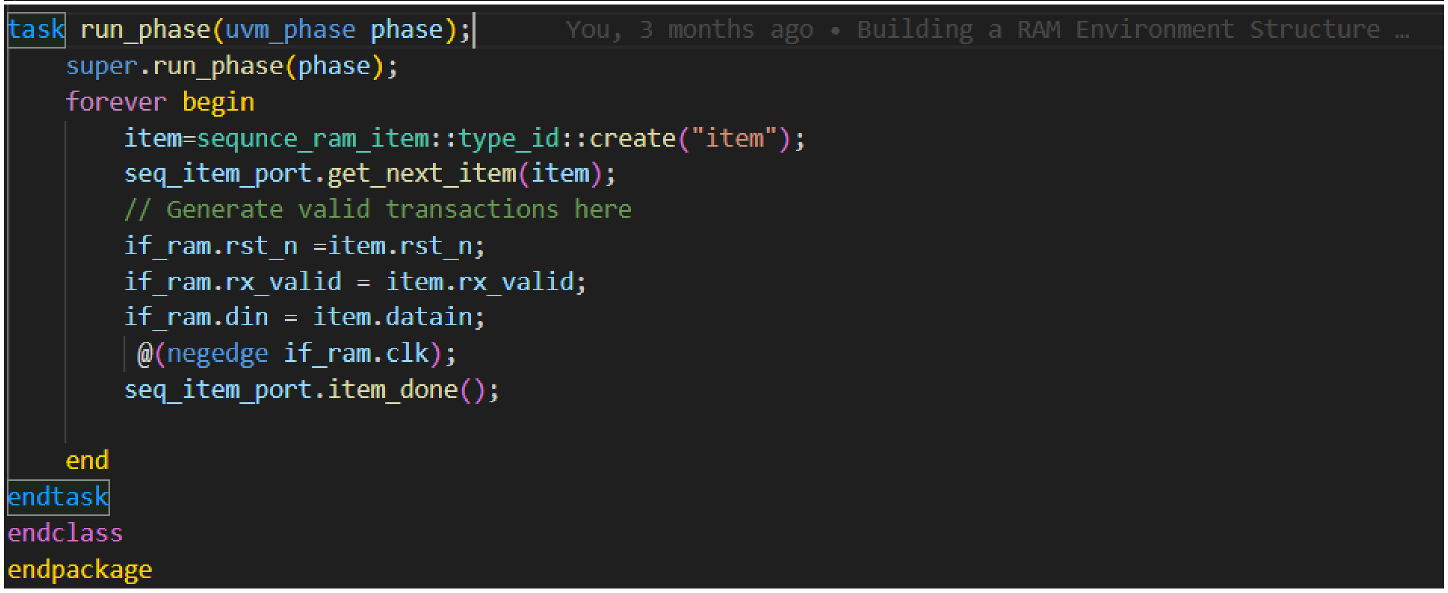


```
package sequnce_ram_item;
import uvm_pkg::*;
`include "uvm_macros.svh"

You, yesterday | You, yesterday | 1 author (You) | 1 author (You)
class sequnce_ram_item extends uvm_sequence_item;

    `uvm_object_utils(sequnce_ram_item)

    rand bit [9:0] datain;
    rand bit rx_valid,rst_n;
    logic [9:0] dout;
    logic tx_valid;
    bit tx_valid_ref;
    logic [9:0] dout_ref;        You, yesterday • add golden model for ram …
    function new(string name = "sequnce_ram_item");
        super.new(name);
    endfunction

    constraint c1{

        rst_n dist{0:=1,1:=99};
        rx_valid dist{0:=1,1:=99};
    }

   endclass
endpackage
```

Driver_ram

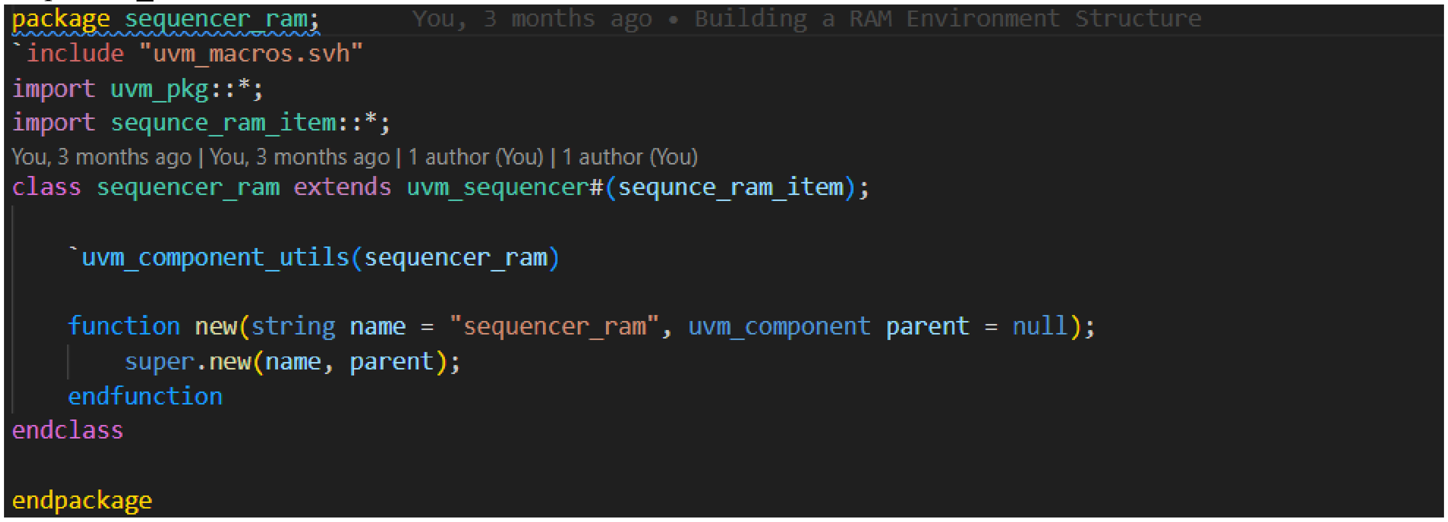


```
package driver_ram;        You, 3 months ago • Building a RAM Environment Structure …

`include "uvm_macros.svh"
import uvm_pkg::*;
import config_ram::*;
import sequnce_ram_item::*;
You, 3 months ago | You, 3 months ago | 1 author (You) | 1 author (You)
class driver_ram extends uvm_driver#(sequnce_ram_item);

    `uvm_component_utils(driver_ram)
    virtual interface_ram if_ram;
    sequnce_ram_item item;

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



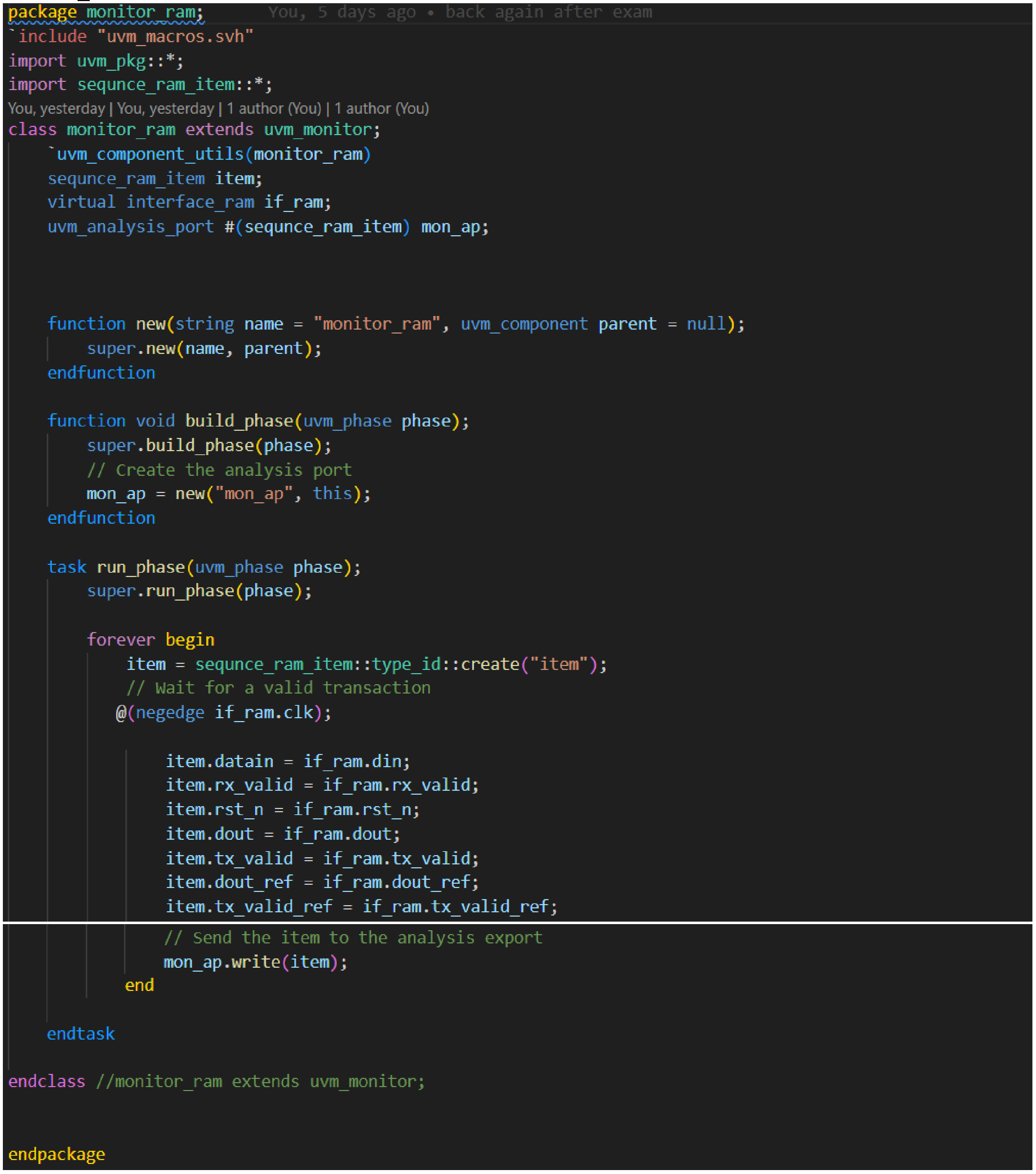
```
task run_phase(uvm_phase phase);        You, 3 months ago • Building a RAM Environment Structure …
    super.run_phase(phase);
    forever begin
        item=sequnce_ram_item::type_id::create("item");
        seq_item_port.get_next_item(item);
        // Generate valid transactions here
        if_ram.rst_n =item.rst_n;
        if_ram.rx_valid = item.rx_valid;
        if_ram.din = item.datain;
         @(negedge if_ram.clk);
        seq_item_port.item_done();

    end
endtask
endclass
endpackage
```

Sequencer_ram



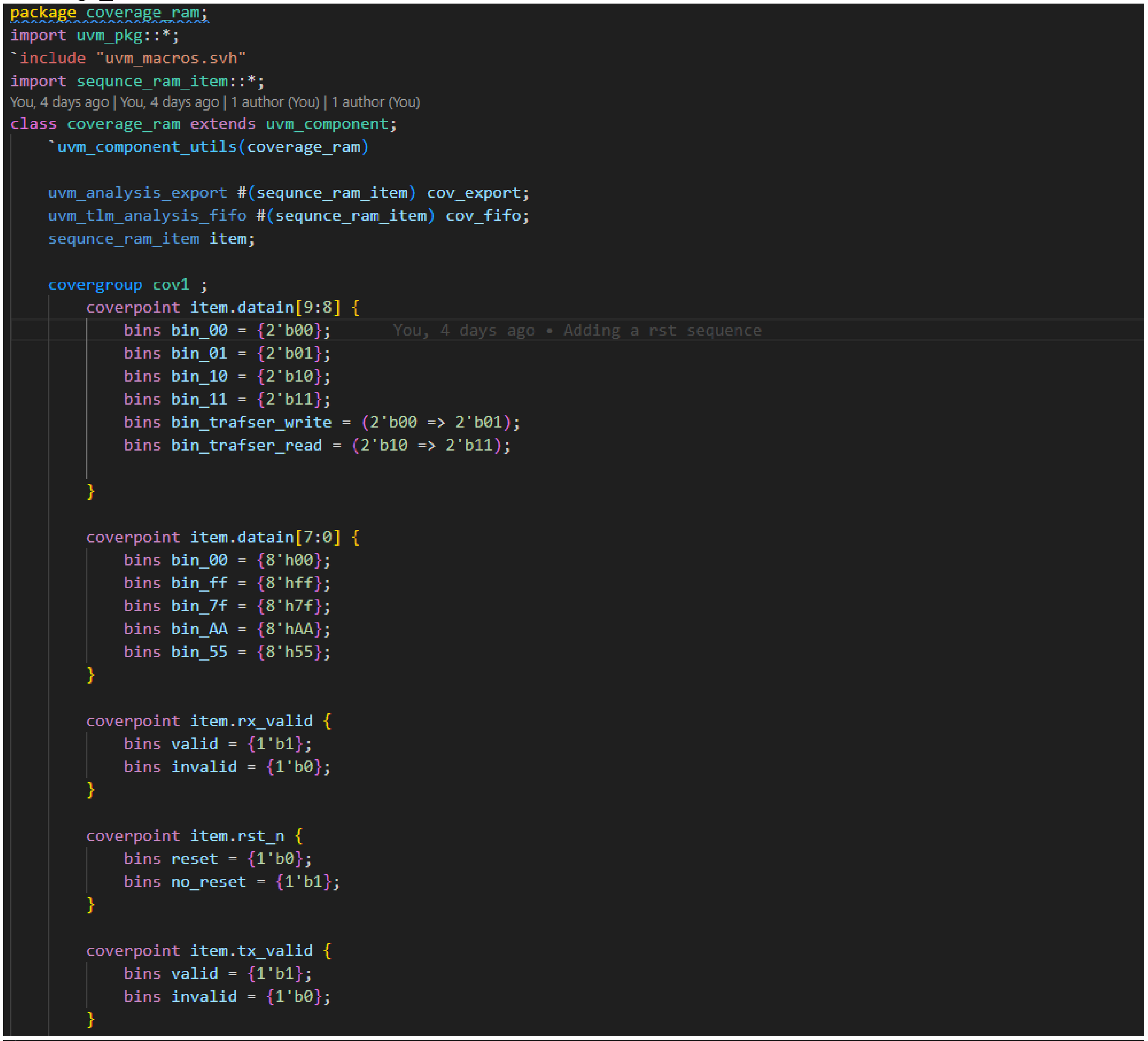
```
package sequencer_ram;        You, 3 months ago • Building a RAM Environment Structure
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequnce_ram_item::*;
You, 3 months ago | You, 3 months ago | 1 author (You) | 1 author (You)
class sequencer_ram extends uvm_sequencer#(sequnce_ram_item);

    `uvm_component_utils(sequencer_ram)

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

endpackage
```

Monitor_ram

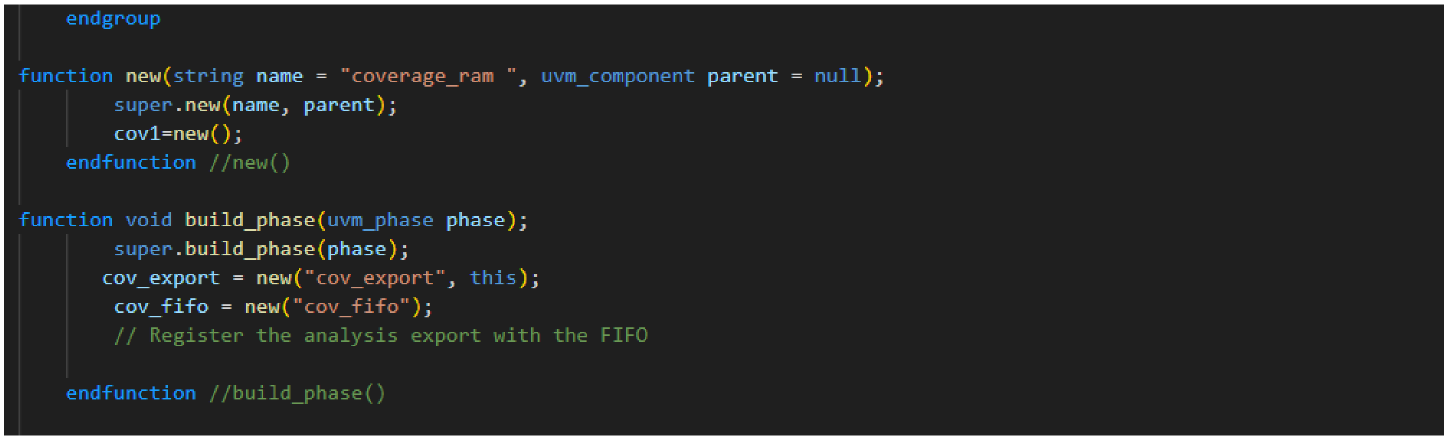


```
package monitor_ram;        You, 5 days ago • back again after exam
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequnce_ram_item::*;
You, yesterday | You, yesterday | 1 author (You) | 1 author (You)
class monitor_ram extends uvm_monitor;
    `uvm_component_utils(monitor_ram)
    sequnce_ram_item item;
    virtual interface_ram if_ram;
    uvm_analysis_port #(sequnce_ram_item) mon_ap;



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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        // Create the analysis port
        mon_ap = new("mon_ap", this);
    endfunction

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

        forever begin
            item = sequnce_ram_item::type_id::create("item");
            // Wait for a valid transaction
           @(negedge if_ram.clk);

                item.datain = if_ram.din;
                item.rx_valid = if_ram.rx_valid;
                item.rst_n = if_ram.rst_n;
                item.dout = if_ram.dout;
                item.tx_valid = if_ram.tx_valid;
                item.dout_ref = if_ram.dout_ref;
                item.tx_valid_ref = if_ram.tx_valid_ref;
                // Send the item to the analysis export
                mon_ap.write(item);
            end

    endtask

endclass //monitor_ram extends uvm_monitor;


endpackage
```

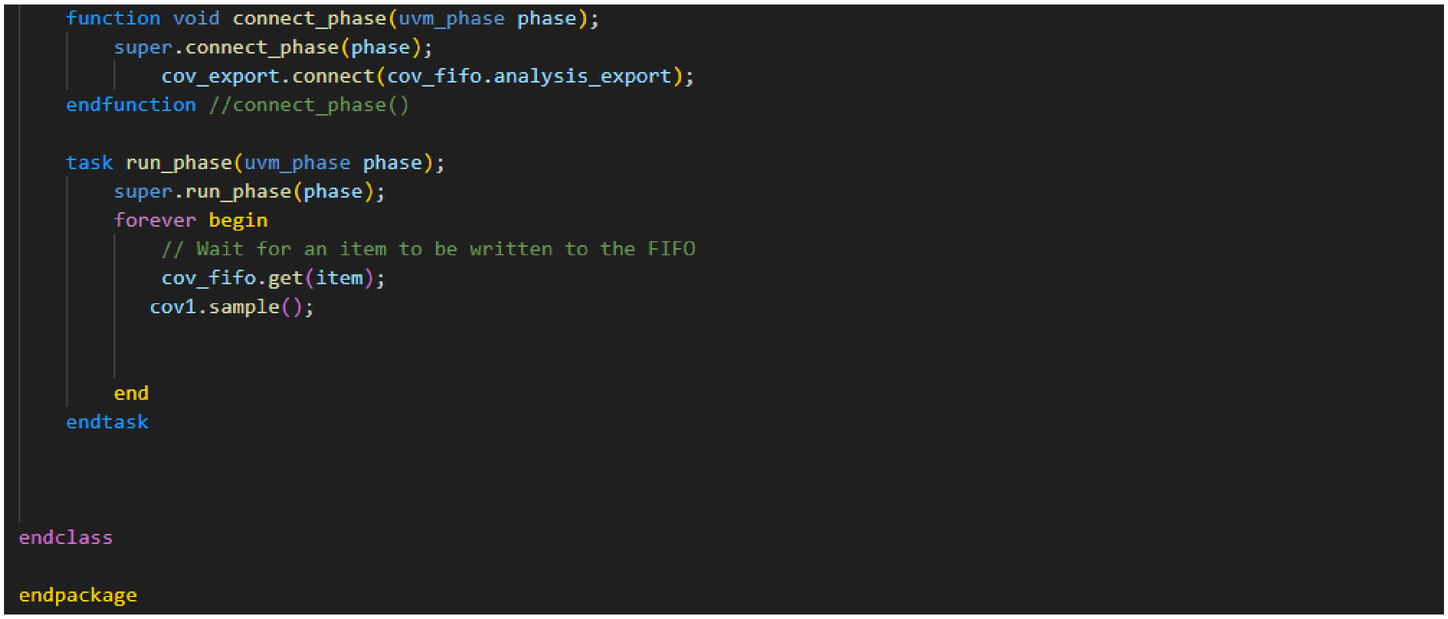
Coverage_ram



```
package coverage_ram;
import uvm_pkg::*;
`include "uvm_macros.svh"
import sequnce_ram_item::*;
You, 4 days ago | You, 4 days ago | 1 author (You) | 1 author (You)
class coverage_ram extends uvm_component;
    `uvm_component_utils(coverage_ram)

    uvm_analysis_export #(sequnce_ram_item) cov_export;
    uvm_tlm_analysis_fifo #(sequnce_ram_item) cov_fifo;
    sequnce_ram_item item;

    covergroup cov1 ;
        coverpoint item.datain[9:8] {
            bins bin_00 = {2'b00};       You, 4 days ago • Adding a rst sequence
            bins bin_01 = {2'b01};
            bins bin_10 = {2'b10};
            bins bin_11 = {2'b11};
            bins bin_trafser_write = (2'b00 => 2'b01);
            bins bin_trafser_read = (2'b10 => 2'b11);

        }

        coverpoint item.datain[7:0] {
            bins bin_00 = {8'h00};
            bins bin_ff = {8'hff};
            bins bin_7f = {8'h7f};
            bins bin_AA = {8'hAA};
            bins bin_55 = {8'h55};
        }

        coverpoint item.rx_valid {
            bins valid = {1'b1};
            bins invalid = {1'b0};
        }

        coverpoint item.rst_n {
            bins reset = {1'b0};
            bins no_reset = {1'b1};
        }

        coverpoint item.tx_valid {
            bins valid = {1'b1};
            bins invalid = {1'b0};
        }
```

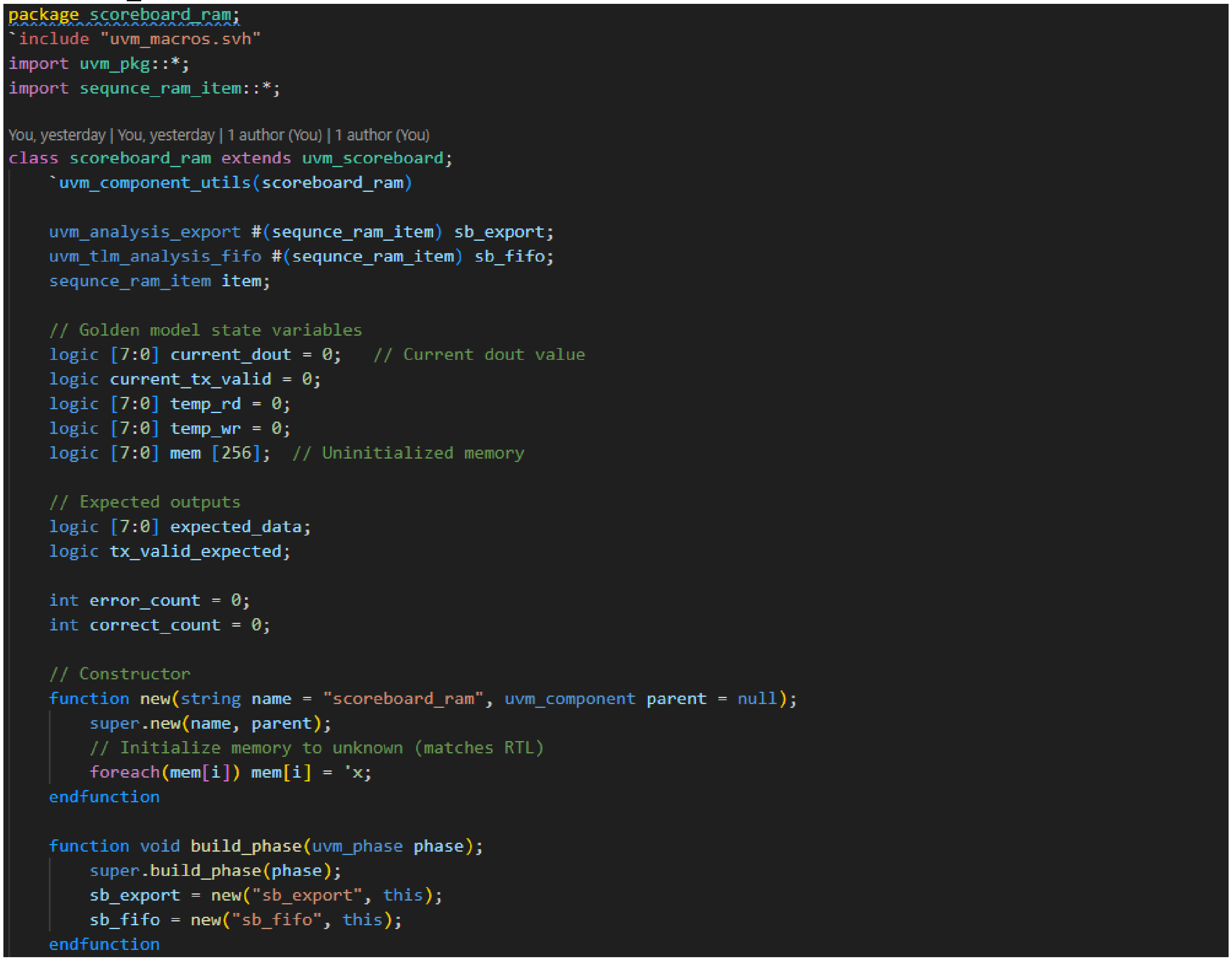


```
    endgroup

function new(string name = "coverage_ram ", uvm_component parent = null);
        super.new(name, parent);
        cov1=new();
    endfunction //new()

function void build_phase(uvm_phase phase);
        super.build_phase(phase);
       cov_export = new("cov_export", this);
        cov_fifo = new("cov_fifo");
        // Register the analysis export with the FIFO

    endfunction //build_phase()
```

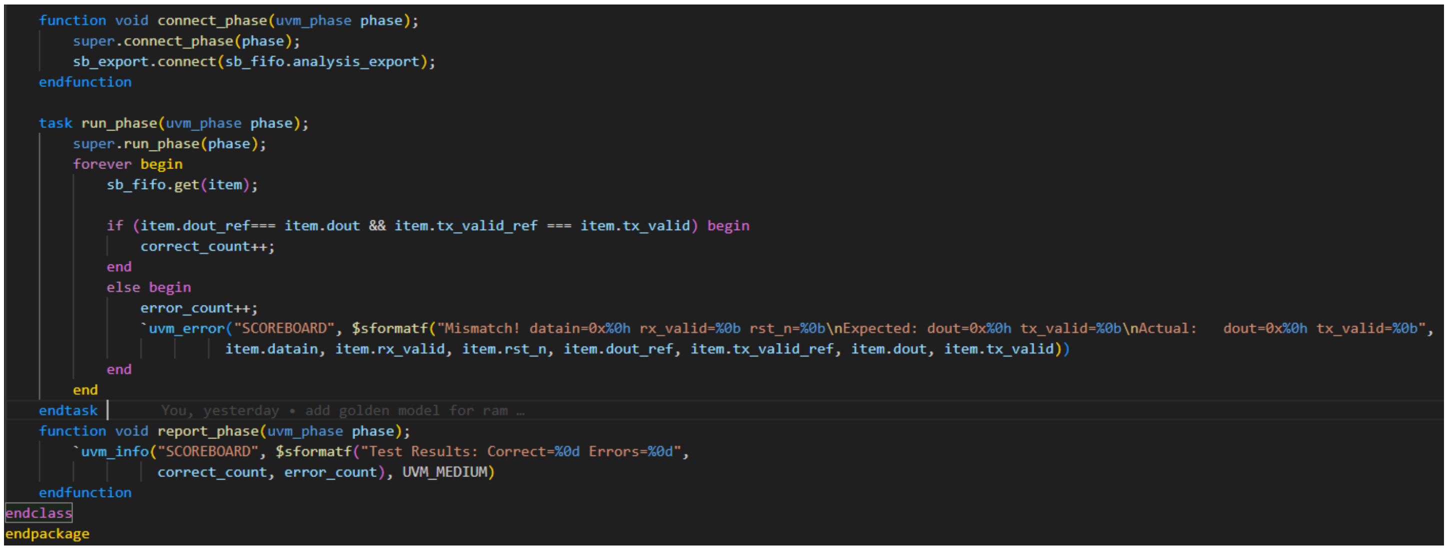


```
    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
    endfunction //connect_phase()

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            // Wait for an item to be written to the FIFO
            cov_fifo.get(item);
           cov1.sample();


        end
    endtask



endclass

endpackage
```

Scoreboard_ram

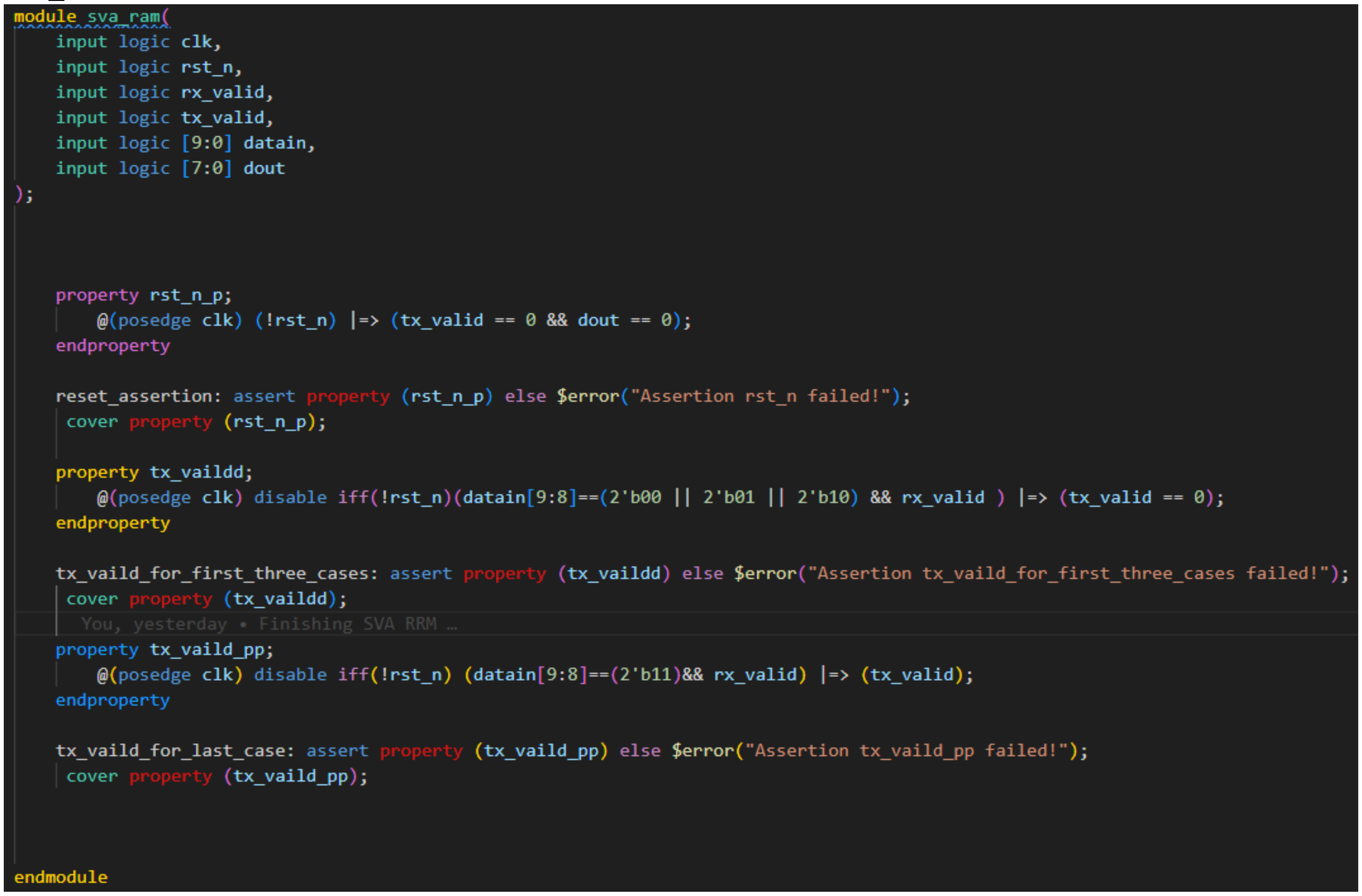


```
package scoreboard_ram;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequnce_ram_item::*;

You, yesterday | You, yesterday | 1 author (You) | 1 author (You)
class scoreboard_ram extends uvm_scoreboard;
    `uvm_component_utils(scoreboard_ram)

    uvm_analysis_export #(sequnce_ram_item) sb_export;
    uvm_tlm_analysis_fifo #(sequnce_ram_item) sb_fifo;
    sequnce_ram_item item;

    // Golden model state variables
    logic [7:0] current_dout = 0;    // Current dout value
    logic current_tx_valid = 0;
    logic [7:0] temp_rd = 0;
    logic [7:0] temp_wr = 0;
    logic [7:0] mem [256];  // Uninitialized memory

    // Expected outputs
    logic [7:0] expected_data;
    logic tx_valid_expected;

    int error_count = 0;
    int correct_count = 0;

    // Constructor
    function new(string name = "scoreboard_ram", uvm_component parent = null);
        super.new(name, parent);
        // Initialize memory to unknown (matches RTL)
        foreach(mem[i]) mem[i] = 'x;
    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
```



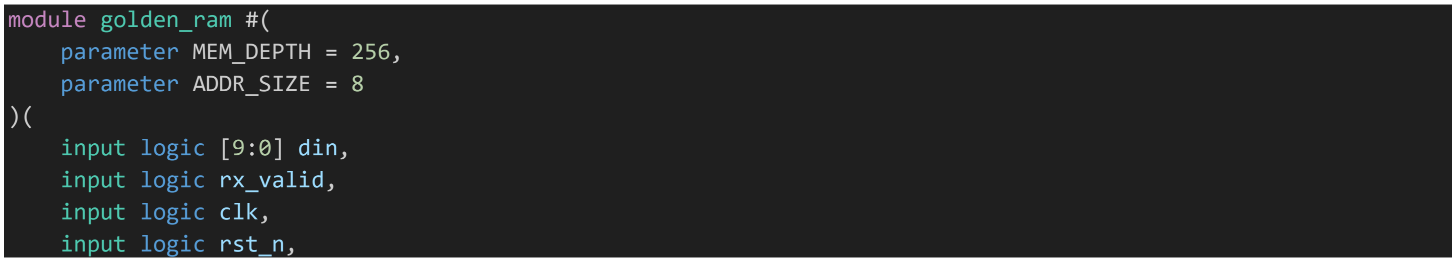
```
    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        sb_export.connect(sb_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(item);

            if (item.dout_ref=== item.dout && item.tx_valid_ref === item.tx_valid) begin
                correct_count++;
            end
            else begin
                error_count++;
                `uvm_error("SCOREBOARD", $sformatf("Mismatch! datain=0x%0h rx_valid=%0b rst_n=%0b\nExpected: dout=0x%0h tx_valid=%0b\nActual:   dout=0x%0h tx_valid=%0b",
                        item.datain, item.rx_valid, item.rst_n, item.dout_ref, item.tx_valid_ref, item.dout, item.tx_valid))
            end
        end
    endtask        You, yesterday • add golden model for ram …
    function void report_phase(uvm_phase phase);
        `uvm_info("SCOREBOARD", $sformatf("Test Results: Correct=%0d Errors=%0d",
                correct_count, error_count), UVM_MEDIUM)
    endfunction
endclass
endpackage
```

Sva_ram



```
module sva_ram(
    input logic clk,
    input logic rst_n,
    input logic rx_valid,
    input logic tx_valid,
    input logic [9:0] datain,
    input logic [7:0] dout
);



    property rst_n_p;
        @(posedge clk) (!rst_n) |=> (tx_valid == 0 && dout == 0);
    endproperty

    reset_assertion: assert property (rst_n_p) else $error("Assertion rst_n failed!");
     cover property (rst_n_p);

    property tx_vaildd;
        @(posedge clk) disable iff(!rst_n)(datain[9:8]==(2'b00 || 2'b01 || 2'b10) && rx_valid ) |=> (tx_valid == 0);
    endproperty

    tx_vaild_for_first_three_cases: assert property (tx_vaildd) else $error("Assertion tx_vaild_for_first_three_cases failed!");
     cover property (tx_vaildd);
       You, yesterday • Finishing SVA RRM …
    property tx_vaild_pp;
        @(posedge clk) disable iff(!rst_n) (datain[9:8]==(2'b11)&& rx_valid) |=> (tx_valid);
    endproperty

    tx_vaild_for_last_case: assert property (tx_vaild_pp) else $error("Assertion tx_vaild_pp failed!");
     cover property (tx_vaild_pp);


endmodule
```

Golden_model_ram



```
module golden_ram #(
    parameter MEM_DEPTH = 256,
    parameter ADDR_SIZE = 8
)(
    input logic [9:0] din,
    input logic rx_valid,
    input logic clk,
    input logic rst_n,
    output logic [7:0] dout,
    output logic tx_valid
);
    // Internal memory and registers
    logic [7:0] mem [0:MEM_DEPTH-1];
    logic [ADDR_SIZE-1:0] temp_rd;
    logic [ADDR_SIZE-1:0] temp_wr;



    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset outputs and internal registers
            dout <= '0;
            tx_valid <= 1'b0;
            temp_rd <= '0;
            temp_wr <= '0;
        end
        else begin
            // Default outputs retain previous values


            if (rx_valid) begin
                case (din[9:8])
                    2'b00: begin  // Set write address
                        temp_wr <= din[7:0];
                        tx_valid <= 1'b0;
                    end

                    2'b01: begin  // Write to memory
                        mem[temp_wr] <= din[7:0];
                        tx_valid <= 1'b0;
                    end

                    2'b10: begin  // Set read address
                        temp_rd <= din[7:0];
                        tx_valid <= 1'b0;
                    end

                    2'b11: begin  // Read from memory
                        dout <= mem[temp_rd];
                        tx_valid <= 1'b1;
                    end
                endcase
            end
        end
    end

    // Function to inspect memory (for verification)
    function logic [7:0] get_memory(input [ADDR_SIZE-1:0] addr);
        return mem[addr];
    endfunction

    // Function to get current read address
    function logic [ADDR_SIZE-1:0] get_read_addr();
        return temp_rd;
    endfunction
    // Function to get current write address
    function logic [ADDR_SIZE-1:0] get_write_addr();
        return temp_wr;
    endfunction
endmodule
```

RAM

```
module ram (
    din,rx_valid,clk,rst_n,dout,tx_valid

);

    parameter MEM_DEPTH = 256 ;
    parameter ADDR_SIZE = 8 ;
    input [9:0]din;
    input clk,rst_n,rx_valid;
    output reg[7:0]dout;
    output reg tx_valid;
    reg [ADDR_SIZE-1:0]mem[MEM_DEPTH-1:0];
    reg [ADDR_SIZE-1:0]temp_rd,temp_wr;

    always @(posedge clk ) begin
        if (~rst_n) begin
            dout <= 0 ;
            tx_valid <= 0 ;
            temp_rd <= 0 ;
            temp_wr <= 0 ;

        end
        else begin
            if (rx_valid) begin
                case(din[9:8])

                2'b00:begin
                    temp_wr <= din[7:0];
                    tx_valid <= 0 ;
                end

                2'b01:begin
                    mem[temp_wr] <= din[7:0];
                    tx_valid <= 0;
                end
                2'b10:begin
                    temp_rd<=din[7:0];
                    tx_valid <= 0;
                end
                2'b11:begin

                    dout <= mem[temp_rd] ;
                    tx_valid <= 1;
                end
                endcase
            end
        end
    end
endmodule : ram
```

Slave Files

Top_module_slave

```
module top_slave();
`include "uvm_macros.svh"
import test_slave::*;
import uvm_pkg::*;
    bit clk;

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

    interface_slave if_slave(clk);
    slave DUT(if_slave.MOSI,if_slave.MISO, if_slave.SS_n,if_slave.clk, if_slave.rst_n, if_slave.rx_data,if_slave.rx_valid,if_slave.tx_data,
if_slave.tx_valid);
    golden_slave golden(if_slave.MOSI,
    if_slave.clk, if_slave.rst_n,if_slave.SS_n,if_slave.tx_valid,if_slave.tx_data,if_slave.MISO_ref,if_slave.rx_valid_ref,
if_slave.rx_data_ref);

    bind slave sva_slave sva_slave_inst(
        if_slave.clk,
        if_slave.rst_n,
        if_slave.MOSI,
        if_slave.SS_n,
        if_slave.tx_valid,
        if_slave.tx_data,
        if_slave.MISO,
        if_slave.rx_valid,
        if_slave.rx_data
    );
    initial begin

        uvm_config_db#(virtual interface_slave)::set(null, "uvm_test_top", "vif", if_slave);
        run_test("test_slave");

    end
endmodule
```

test_slave

```
package test_slave;
`include "uvm_macros.svh"
import uvm_pkg::*;
import env_slave::*;
import sequence_rst_slave::*;
import sequence_stim_slave::*;
import config_slave::*;

class test_slave extends uvm_test;
    `uvm_component_utils(test_slave)
    env_slave env; // Environment for the slave
    config_slave cfg; // Configuration object for the slave
    sequence_rst_slave seq_rst_slave; // Sequence for reset operation
    sequence_stim_slave seq_stim_slave; // Sequence for stimulus operation
    // Constructor
    function new(string name = "test_slave", uvm_component parent = null);
        super.new(name, parent);

    endfunction //new()

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        // Create the environment
       env = env_slave::type_id::create("env", this);

        // Create the configuration object

       cfg = config_slave::type_id::create("cfg");
        seq_rst_slave = sequence_rst_slave::type_id::create("seq_rst_slave");
        seq_stim_slave = sequence_stim_slave::type_id::create("seq_stim_slave");
        // Get the virtual interface from the configuration database
        if (!uvm_config_db#(virtual interface_slave)::get(this, "", "vif", cfg.if_slave)) begin
            `uvm_fatal("build_phase", "Config object not get in test class");
        end

        // Set the configuration object in the database
        uvm_config_db#(config_slave)::set(this, "*", "GFG", cfg);

    endfunction //build_phase

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

        // Raise an objection to keep the test running
        phase.raise_objection(this);

        // Start the sequences
        seq_rst_slave.start(env.agt_slave.seq_slave);
        `uvm_info("run_phase", "Slave sequence started", UVM_MEDIUM)
        seq_stim_slave.start(env.agt_slave.seq_slave);
        `uvm_info("run_phase", "Stimulus sequence started", UVM_MEDIUM)

        `uvm_info("run_phase", "Test is running", UVM_MEDIUM)

        // Drop the objection when done
        phase.drop_objection(this);
    endtask //run_phase
endclass //test_slave extends uvm_test

endpackage
```

env_slave

```
package env_slave;
`include "uvm_macros.svh"
import uvm_pkg::*;
import agent_slave::*;
import config_slave::*;
import coverage_slave::*;
import scoreboard_slave::*;
class env_slave extends uvm_env;
    `uvm_component_utils(env_slave)


    agent_slave agt_slave;
    coverage_slave cov_slave;
    scoreboard_slave sb_slave;
    // Constructor
    function new(string name = "env_slave", uvm_component parent = null);
        super.new(name, parent);
    endfunction // new

    // Build phase
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);


        agt_slave = agent_slave::type_id::create("agt_slave", this);
        cov_slave = coverage_slave::type_id::create("cov_slave", this);
        sb_slave = scoreboard_slave::type_id::create("sb_slave", this);



    endfunction // build_phase
    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);

        // Connect the agent's analysis port to the coverage
        agt_slave.agent_cov_ap.connect(cov_slave.cov_export);
        agt_slave.agent_cov_ap.connect(sb_slave.sb_export);
    endfunction
endclass // env_slave extends uvm_env

endpackage
```

agent_slave

```
package agent_slave;
`include "uvm_macros.svh"
import uvm_pkg::*;
import config_slave::*;
import sequencer_slave::*;
import driver_slave::*;
import monitor_slave::*;
import sequence_slave_item::*;
class agent_slave extends uvm_agent;
`uvm_component_utils(agent_slave)

    config_slave cfg;
    sequencer_slave seq_slave;
    driver_slave drv_slave;
    monitor_slave mon_slave;

    uvm_analysis_port#(sequence_slave_item) agent_cov_ap;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        // Create the configuration object
        if(!uvm_config_db#(config_slave)::get(this, "", "GFG", cfg))begin
            `uvm_fatal("build_phase", "Config object not get in agent class")        end

             // Create the sequencer
        seq_slave = sequencer_slave::type_id::create("seq_slave", this);
        drv_slave = driver_slave::type_id::create("drv_slave", this);
        mon_slave = monitor_slave::type_id::create("mon_slave", this);
        agent_cov_ap = new("agent_cov_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        // Connect the sequencer to the driver

        drv_slave.if_slave = cfg.if_slave;
        mon_slave.if_slave = cfg.if_slave;
        mon_slave.mon_ap.connect(agent_cov_ap);
        drv_slave.seq_item_port.connect(seq_slave.seq_item_export);
    endfunction
endclass

endpackage
```

sequence_slave

```
package sequence_stim_slave; //Stimulus
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_slave_item::*;
class sequence_stim_slave extends uvm_sequence#(sequence_slave_item);

    `uvm_object_utils(sequence_stim_slave)
    sequence_slave_item item;
    // Constructor
    function new(string name = "sequence_stim_slave");
        super.new(name);
    endfunction
    // Body phase
    virtual task body();
        sequence_slave_item item;
        // Create a new sequence item
        item = sequence_slave_item::type_id::create("item");
        repeat(1000) begin
             // Start the item
            start_item(item);
            // Randomize the item
            assert(item.randomize());
            // Finish the item
            finish_item(item);
        end
    endtask
endclass
endpackage
```

sequence_rst_slave

```
package sequence_rst_slave; //Stimulus
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_slave_item::*;
class sequence_rst_slave extends uvm_sequence#(sequence_slave_item);

    `uvm_object_utils(sequence_rst_slave)
    sequence_slave_item item;
    // Constructor
    function new(string name = "sequence_rst_slave");
        super.new(name);
    endfunction

    // Body phase
    virtual task body();

        // Create a new sequence item
        item = sequence_slave_item::type_id::create("item");

            start_item(item);
            // rst
            item.rst_n = 0; // Set reset to low
            // Finish the item
            finish_item(item);
    endtask
endclass
endpackage
```

sequence_slave_item

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

class sequence_slave_item extends uvm_sequence_item;

    `uvm_object_utils(sequence_slave_item)

   rand logic rst_n; // Reset signal for the slave
   rand logic MOSI,SS_n,tx_valid;
   rand logic [7:0]tx_data;
   logic MISO,rx_valid,MISO_ref,rx_valid_ref;
   logic [9:0]rx_data,rx_data_ref; // Data received from the slave
    // Constructor
    function new(string name = "sequence_slave_item");
        super.new(name);
    endfunction

    // Randomization constraints

    constraint c1 {
        rst_n dist {0 := 1, 1 := 99}; // Reset signal is low for 1% of the time
        MOSI dist {0 := 75, 1 := 25}; // MOSI signal is low for 1% of the time
        SS_n dist {0 := 75, 1 := 25}; // Slave Select signal is low for 1% of the time
        tx_valid dist {0 := 50, 1 := 50}; // Transmission valid signal is low for 1% of the time

    }


endclass

endpackage
```

Sequencer_slave

```
package sequencer_slave;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_slave_item::*;
class sequencer_slave extends uvm_sequencer#(sequence_slave_item);

    `uvm_component_utils(sequencer_slave)

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

endpackage
```

sva_slave

```
module sva_slave(
    input clk,
    input logic rst_n,
    input logic MOSI,
    input logic SS_n,
    input logic tx_valid,
    input logic [7:0] tx_data,
    input logic MISO,
    input logic rx_valid,
    input logic [9:0] rx_data



);
    property rst_n_assertion;
        @(posedge clk) (!rst_n) |=> (rx_valid == 0 && rx_data == 0 && MISO == 0)
    endproperty
   assert property (rst_n_assertion)
        else $error("rst_n assertion failed: rst_n is low while rx_valid, rx_data, and MISO are not zero");

    cover property (rst_n_assertion);
endmodule
```

scoreboard_slave

```
package scoreboard_slave;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_slave_item::*;
class scoreboard_slave extends uvm_scoreboard;
    `uvm_component_utils(scoreboard_slave)

    sequence_slave_item item;
    int correct = 0;
    int incorrect = 0;
    ;//logic MISO_ref, rx_valid_ref;
   // logic [9:0] rx_data_ref;

    // sberage export
    uvm_analysis_export#(sequence_slave_item) sb_export;
    uvm_tlm_analysis_fifo#(sequence_slave_item) sb_fifo;

    // Constructor
    function new(string name = "scoreboard_slave", 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);
        // Connect the export to the FIFO
        sb_export.connect(sb_fifo.analysis_export);
    endfunction
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            // Wait for an item to be written to the FIFO
        sb_fifo.get(item);

          if (item.rx_data == item.rx_data_ref && item.rx_valid == item.rx_valid_ref && item.MISO == item.MISO_ref) begin
            correct++;
          end
          else begin
            incorrect++;
            `uvm_error("SCOREBOARD", $sformatf("Incorrect data: Expected %0d, Got %0d", item.rx_data_ref, item.rx_data))
          end

        end
    endtask
    function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("SCOREBOARD", $sformatf("Correct: %0d, Incorrect: %0d", correct, incorrect), UVM_LOW)
    endfunction
endclass

endpackage
```

monitor_slave

```
package monitor_slave;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_slave_item::*;
class monitor_slave extends uvm_monitor;
    `uvm_component_utils(monitor_slave)

    virtual interface_slave if_slave;
    sequence_slave_item item;
    uvm_analysis_port#(sequence_slave_item) mon_ap;
    // Constructor
    function new(string name = "monitor_slave", uvm_component parent = null);
        super.new(name, parent);
    endfunction // new

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        // Create the analysis port
        mon_ap = new("mon_ap", this);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        // Monitor the virtual interface
        forever begin
        item = sequence_slave_item::type_id::create("item");

            @(negedge if_slave.clk); // Wait for clock edge
          //  @(negedge if_slave.clk); // Wait for clock edge

            item.rst_n = if_slave.rst_n;
            item.MOSI = if_slave.MOSI;
            item.SS_n = if_slave.SS_n;
            item.tx_valid = if_slave.tx_valid;
            item.tx_data = if_slave.tx_data;
            item.MISO = if_slave.MISO;
            item.rx_valid = if_slave.rx_valid;
            item.rx_data = if_slave.rx_data;
            item.MISO_ref = if_slave.MISO_ref;
            item.rx_valid_ref = if_slave.rx_valid_ref;
            item.rx_data_ref = if_slave.rx_data_ref;


        mon_ap.write(item); // Send the item to the analysis export
        end
    endtask // run_phase

endclass // monitor_slave extends uvm_monitor

endpackage
```

interface_slave

```
interface interface_slave(clk);
    input clk;
    logic MOSI,rst_n,SS_n,tx_valid;
    logic [7:0]tx_data;
    logic MISO,rx_valid,MISO_ref,rx_valid_ref;
    logic [9:0]rx_data,rx_data_ref;
endinterface
```

config_slave

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

class config_slave extends uvm_object;
    `uvm_object_utils(config_slave)

    // Virtual interface for the slave
    virtual interface_slave if_slave;

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


endclass

endpackage
```

coverage_slave

```
package coverage_slave;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_slave_item::*;
class coverage_slave extends uvm_component;

    `uvm_component_utils(coverage_slave)
    sequence_slave_item item;
    // Coverage export
    uvm_analysis_export#(sequence_slave_item) cov_export;
    uvm_tlm_analysis_fifo#(sequence_slave_item) cov_fifo;

    // functional coverage
    // uvm_covergroup cov1;

    covergroup cov1;

    Covering_rx_data:coverpoint item.rx_data {
        option.comment = "Covering rx_data";
        bins data_bins[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    }
   Covering_rx_valid: coverpoint item.rx_valid {
        option.comment = "Covering rx_valid";
        bins valid_bins[] = {0, 1};
    }
    Covering_MISO:coverpoint item.MISO {
        option.comment = "Covering MISO";
        bins miso_bins[] = {0, 1};
    }
    Covering_SS_n:coverpoint item.SS_n {
        option.comment = "Covering SS_n";
        bins ss_bins[] = {0, 1};
    }

    Covering_tx_valid:coverpoint item.tx_valid {
        option.comment = "Covering tx_valid";
        bins tx_valid_bins[] = {0, 1};
    }

    Covering_tx_data:coverpoint item.tx_data {
        option.comment = "Covering tx_data";
        bins tx_data_bins[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    }
    Covering_rst_n:coverpoint item.rst_n {
        option.comment = "Covering rst_n";
        bins rst_bins[] = {0, 1};
    }



    endgroup

    // Constructor
    function new(string name = "coverage_slave", uvm_component parent = null);
        super.new(name, parent);
        cov1 = 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);
        // Connect the export to the FIFO
        cov_export.connect(cov_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            // Wait for an item to be written to the FIFO
        cov_fifo.get(item);
        cov1.sample();


        end
    endtask
endclass


endpackage
```

driver_slave

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

class driver_slave extends uvm_driver#(sequence_slave_item);
    `uvm_component_utils(driver_slave)
    sequence_slave_item item;
    virtual interface_slave if_slave;
    // Constructor
    function new(string name = "driver_slave", uvm_component parent = null);
        super.new(name, parent);
    endfunction
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        item = sequence_slave_item::type_id::create("item");

    endfunction // build_phase
    // Main task to drive the sequence item
    task run_phase(uvm_phase phase);
        super.run_phase(phase);


        forever begin
            // Get the next item from the sequencer
            seq_item_port.get_next_item(item);

            // Drive the item to the virtual interface
            if (item != null) begin
                // Add your driving logic here
                if_slave.rst_n = item.rst_n;
                if_slave.MOSI = item.MOSI;
                if_slave.SS_n = item.SS_n;
                if_slave.tx_valid = item.tx_valid;
                if_slave.tx_data = item.tx_data;
                @(negedge if_slave.clk); // Wait for the clock edge
                seq_item_port.item_done();
            end
        end
    endtask
 endclass // driver_slave extends uvm_driver

endpackage
```

golden_model_slave

```
module golden_slave (
    input MOSI, clk, rst_n, SS_n, tx_valid,
    input [7:0] tx_data,
    output reg MISO, rx_valid,
    output reg [9:0] rx_data
);
    // State definitions - must match RTL exactly
    parameter IDLE      = 3'b000;
    parameter WRITE     = 3'b001;
    parameter CHK_CMD   = 3'b010;
    parameter READ_ADD  = 3'b011;
    parameter READ_DATA = 3'b100;

    reg [2:0] cs, ns;
    reg [3:0] counter;
    reg confirm_add;
    reg [9:0] rx_data_internal;

   (*fsm_encoding="one_hot"*)
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            cs <= IDLE;
            counter <= 0;
            confirm_add <= 0;
            rx_data_internal <= 0;
            MISO <= 0;
            rx_valid <= 0;
        end
        else begin
           cs <= ns;
            // State behavior
            case (cs)
                IDLE: begin
                    counter <= 0;
                    rx_valid <= 0;
                    MISO <= 0;
                end
                WRITE: begin
                    counter <= 0 ;
                    if (counter < 10) begin
                        rx_data_internal <= {rx_data_internal[8:0], MOSI};
                        counter <= counter + 1;
                        rx_valid <= 0;
                    end
                    else if (counter == 10) begin
                        rx_valid <= 1;
                    end
                end
                READ_ADD: begin
                    if (counter < 10) begin
                        rx_data_internal <= {rx_data_internal[8:0], MOSI};
                        counter <= counter + 1;
                        rx_valid <= 0;
                    end
                    else if (counter == 10) begin
                        rx_valid <= 1;
                        confirm_add <= 1;
                    end
                end
                READ_DATA: begin
                    if (!tx_valid) begin
                        if (counter < 10) begin
                            rx_data_internal <= {rx_data_internal[8:0], MOSI};
                            counter <= counter + 1;
                            rx_valid <= 0;
                        end
                        else if (counter == 10) begin
                            rx_valid <= 1;
                            confirm_add <= 0;
                        end
                    end
                    else begin
                        counter <= 10;
                        if (counter >= 3 && counter <= 10) begin
                            MISO <= tx_data[counter-3];
                            counter <= counter - 1;
                        end
                    end
                end
                default: begin
                    // Safe default
                    counter <= 0;
                    rx_valid <= 0;
                    MISO <= 0;
                end
            endcase
        end
    end

    // Next state logic (combinational)
    always @(*) begin
        case (cs)
            IDLE: begin
                ns = (~SS_n) ? CHK_CMD : IDLE;
            end

            CHK_CMD: begin
                if (SS_n) begin
                    ns = IDLE;
                end
                else if (~SS_n && ~MOSI) begin
                    ns = WRITE;
                end
                else if (~SS_n && MOSI && ~confirm_add) begin
                    ns = READ_ADD;
                end
                else if (~SS_n && MOSI && confirm_add) begin
                    ns = READ_DATA;
                end
                else begin
                    ns = CHK_CMD;
                end
            end

            WRITE: begin
                ns = SS_n ? IDLE : WRITE;
            end

            READ_ADD: begin
                if (SS_n) begin
                    ns = IDLE;
                end
                else if (confirm_add) begin
                    ns = READ_DATA;
                end
                else begin
                    ns = READ_ADD;
                end
            end

            READ_DATA: begin
                ns = SS_n ? IDLE : READ_DATA;
            end

            default: ns = IDLE;
        endcase
    end

    // Output assignment
    assign rx_data = rx_data_internal;
    endmodule
```

slave

```verilog
module slave (

    MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid

);
    parameter IDLE = 3'b000 ;
    parameter WRITE = 3'b001 ;
    parameter CHK_CMD = 3'b010 ;
    parameter READ_ADD = 3'b011 ;
    parameter READ_DATA = 3'b100 ;



    input MOSI,clk,rst_n,SS_n,tx_valid;
    input [7:0]tx_data;
    output reg MISO,rx_valid;
    output reg[9:0]rx_data;

    reg [2:0]ns,cs;
    reg [3:0]counter;
    reg confirm_add;

    (*fsm_encoding="one_hot"*)
    always @(posedge clk or negedge rst_n) begin
        if (~rst_n) begin
            cs <= IDLE ;
        end
        else begin
            cs <= ns ;
        end
    end

    always @(*) begin
        case(cs)
        IDLE:begin
            if (~SS_n) begin
                ns = CHK_CMD;
            end
            else begin
                ns = IDLE ;
            end
        end

        CHK_CMD:begin
            if (SS_n) begin
                ns = IDLE;
            end
            else if (~SS_n && ~MOSI) begin
                ns = WRITE;
            end
            else if (~SS_n && MOSI && ~confirm_add) begin
                ns = READ_ADD;

            end
            else if (~SS_n && MOSI && confirm_add) begin
                ns = READ_DATA;

            end
        end
        WRITE:begin
            if (SS_n) begin
                ns = IDLE;
            end
            else begin
                ns = WRITE;
            end
        end
        READ_ADD:begin
            if (SS_n) begin
                ns = IDLE;
            end
            else if(confirm_add)begin
                ns = READ_DATA;
            end
            else begin
                ns = READ_ADD;
            end
        end
        READ_DATA:begin
                if (SS_n) begin
                    ns = IDLE;
                end
                else begin
                    ns = READ_DATA;
                end
            end
        default : ns = IDLE ;
        endcase
    end
    always @(posedge clk ) begin
        if (~rst_n) begin
            counter <= 0 ;
            confirm_add <= 0 ;
            rx_data <= 0;
            rx_valid <= 0 ;
            MISO <= 0 ;
        end
        else begin
            case(cs)
            IDLE:begin
                counter <= 0 ;
                rx_valid <= 0 ;
                MISO <= 0 ;
            end
            WRITE:begin
                counter <= 0 ;
                if( counter < 10 ) begin
                    rx_data <= {rx_data[8:0],MOSI};
                    rx_valid <= 0 ;
                    counter <= counter + 1 ;
                end
                 if (counter == 10) begin
                    rx_valid <= 1 ;
                end
            end
            READ_ADD:begin
                 if( counter < 10 ) begin
                    rx_data <= {rx_data[8:0],MOSI};
                    rx_valid <= 0 ;
                    counter <= counter + 1 ;
                end
                if (counter == 10) begin
                    rx_valid <= 1 ;
                    confirm_add <= 1 ;
                end
            end
            READ_DATA:begin
            if(~tx_valid)begin
                if( counter < 10 ) begin
                    rx_data <= {rx_data[8:0],MOSI};
                    rx_valid <= 0 ;
                    counter <= counter + 1 ;
                end
             if (counter == 10) begin
                    rx_valid <= 1 ;
                    confirm_add <= 0 ;

                end
            end
            else begin
                counter <= 10 ;
                if (3 <= counter) begin
                    MISO <= tx_data[counter-3];
                    counter <= counter - 1 ;
                end
            end
            end
            endcase
        end
    end
endmodule : slave
```

Wrapper_files

The ram and slave I use without modify it, so I use the files above with small change in config and agents

Top_module_wrapper

```
module top_module_wrapper();
`include "uvm_macros.svh"
import uvm_pkg::*;
import test_wrapper::*;
    bit clk;
    initial begin
        clk = 0;
        forever begin
            #5 clk = ~clk;
        end
    end

interface_wrapper if_wrapper(clk);
interface_slave if_slave(clk);
interface_ram if_ram(clk);

ram DUT_ram(if_ram.din, if_ram.rx_valid, if_ram.clk, if_ram.rst_n, if_ram.dout, if_ram.tx_valid);
golden_ram golden_ram(if_ram.din, if_ram.rx_valid, if_ram.clk, if_ram.rst_n, if_ram.dout_ref, if_ram.tx_valid_ref);
slave DUT_slave(if_slave.MOSI,if_slave.MISO, if_slave.SS_n,if_slave.clk, if_slave.rst_n,
if_slave.rx_data,if_slave.rx_valid,if_slave.tx_data, if_slave.tx_valid);
 golden_slave golden(if_slave.MOSI,
    if_slave.clk, if_slave.rst_n,if_slave.SS_n,if_slave.tx_valid,if_slave.tx_data,if_slave.MISO_ref,if_slave.rx_valid_ref,
if_slave.rx_data_ref);

wrapper DUT_wrapper(if_wrapper.MOSI, if_wrapper.MISO, if_wrapper.SS_n, if_wrapper.clk, if_wrapper.rst_n);
golden_wrapper golden_wrapper(if_wrapper.MOSI,if_wrapper.SS_n ,if_wrapper.clk, if_wrapper.rst_n,if_wrapper.MISO_ref);

assign if_slave.MOSI=DUT_wrapper.MOSI;
assign if_slave.SS_n=DUT_wrapper.SS_n;
assign if_slave.rst_n=DUT_wrapper.rst_n;
assign if_ram.din=if_slave.rx_data;
assign if_ram.rx_valid=if_slave.rx_valid;
assign if_slave.tx_data=if_ram.dout;
assign if_slave.tx_valid=if_ram.tx_valid;
assign if_ram.rst_n=DUT_wrapper.rst_n;
initial begin

    uvm_config_db#(virtual interface_wrapper)::set(null, "uvm_test_top", "vif", if_wrapper);
    uvm_config_db#(virtual interface_slave)::set(null, "uvm_test_top", "vif_slave", if_slave);
    uvm_config_db#(virtual interface_ram)::set(null, "uvm_test_top", "vif_ram", if_ram);
        run_test("test_wrapper");

end
endmodule
```

test_wrapper

```
package test_wrapper;
`include "uvm_macros.svh"
import uvm_pkg::*;
import env_wrapper::*;
import env_slave::*;
import env_ram::*;
import config_slave::*;
import config_wrapper::*;
import config_ram::*;
import sequence_wrapper_item::*;
import sequence_rst_wrapper::*;
import sequence_wrapper::*;
class test_wrapper extends uvm_test;
    `uvm_component_utils(test_wrapper)
    env_wrapper env_wrapperr;
    env_slave env_slaver;
    env_ram env_ramm;
    config_slave cfg_slave;
    config_wrapper cfg_wrapper;
    config_ram cfg_ram;
    sequence_rst_wrapper seq_rst_wrapper;
    sequence_wrapper seq_wrapper;
    // Constructor
    function new(string name = "test_wrapper", uvm_component parent = null);
        super.new(name, parent);
    endfunction // new

    // Build phase
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        // Create the environment
        env_wrapperr = env_wrapper::type_id::create("env_wrapperr", this);
        env_slaver = env_slave::type_id::create("env_slaver", this);
        env_ramm = env_ram::type_id::create("env_ramm", this);

        // Create the configuration objects
        cfg_slave = config_slave::type_id::create("cfg_slave", this);
        cfg_wrapper = config_wrapper::type_id::create("cfg_wrapper", this);
        cfg_ram = config_ram::type_id::create("cfg_ram", this);

        seq_rst_wrapper = sequence_rst_wrapper::type_id::create("seq_rst_wrapper", this);
        seq_wrapper = sequence_wrapper::type_id::create("seq_wrapper", this);

        // get the configuration objects in the UVM config database
        if (!uvm_config_db#(virtual interface_slave)::get(this, "", "vif_slave", cfg_slave.if_slave)) begin
            `uvm_fatal("build_phase", "Config object not get in test class");
        end
        if (!uvm_config_db#(virtual interface_wrapper)::get(this, "", "vif", cfg_wrapper.if_wrapper)) begin
            `uvm_fatal("build_phase", "Config object not get in test class");
        end
        if (!uvm_config_db#(virtual interface_ram)::get(this, "", "vif_ram", cfg_ram.if_ram)) begin
            `uvm_fatal("build_phase", "Config object not get in test class");
        end
        // Set the configuration objects in the database
        uvm_config_db#(config_slave)::set(this, "*", "GFG_slave", cfg_slave);
        uvm_config_db#(config_wrapper)::set(this, "*", "GFG", cfg_wrapper);
        uvm_config_db#(config_ram)::set(this, "*", "GFG_ram", cfg_ram);


    endfunction // build_phase

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

        // Raise an objection to keep the test running
        phase.raise_objection(this);

        seq_rst_wrapper.start(env_wrapperr.agt_wrapper.seq_wrapper);
        `uvm_info("run_phase", "Wrapper sequence started", UVM_MEDIUM)
        seq_wrapper.start(env_wrapperr.agt_wrapper.seq_wrapper);
        `uvm_info("run_phase", "Wrapper sequence started", UVM_MEDIUM)

        // Drop the objection when done
        phase.drop_objection(this);
    endtask // run_phase

endclass // test_wrapper extends uvm_test

endpackage
```

env_wrapper

```
package env_wrapper;
import uvm_pkg::*;
`include "uvm_macros.svh"
import agent_wrapper::*;
import scoreboard_wrapper::*;
import coverage_wrapper::*;

class env_wrapper extends uvm_env;

    `uvm_component_utils(env_wrapper)

    agent_wrapper agt_wrapper;
    scoreboard_wrapper sb_wrapper;
    coverage_wrapper cov_wrapper;
    function new(string name = "env_wrapper", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
```

```
        super.build_phase(phase);

        agt_wrapper = agent_wrapper::type_id::create("agt_wrapper", this);
        sb_wrapper = scoreboard_wrapper::type_id::create("sb_wrapper", this);
        cov_wrapper = coverage_wrapper::type_id::create("cov_wrapper", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);

        // Connect the agent's analysis port to the scoreboard
        agt_wrapper.agent_ap.connect(sb_wrapper.sb_export);
        agt_wrapper.agent_ap.connect(cov_wrapper.cov_export);
    endfunction

endclass

endpackage
```

agent_wrapper

```
package agent_wrapper;

`include "uvm_macros.svh"
import uvm_pkg::*;
import driver_wrapper::*;
import sequencer_wrapper::*;
import config_wrapper::*;
import monitor_wrapper::*;
import sequence_wrapper_item::*;
class agent_wrapper extends uvm_agent;
`uvm_component_utils(agent_wrapper)

    driver_wrapper drv_wrapper;
    sequencer_wrapper seq_wrapper;
    config_wrapper cfg;
    monitor_wrapper mon_wrapper;
    uvm_analysis_port #(sequence_wrapper_item) agent_ap;

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

    endfunction //new()

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);


         if(!uvm_config_db#(config_wrapper)::get(this, "", "GFG", cfg))begin
            `uvm_fatal("build_phase", "Config object not get in agent class")
        end
        seq_wrapper = sequencer_wrapper::type_id::create("seq_wrapper", this);
        drv_wrapper = driver_wrapper::type_id::create("drv_wrapper", this);
        mon_wrapper = monitor_wrapper::type_id::create("mon_wrapper", this);
        agent_ap = new("agent_ap", this);
    endfunction //build_phase()

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        drv_wrapper.if_wrapper = cfg.if_wrapper;
        mon_wrapper.if_wrapper = cfg.if_wrapper;
        mon_wrapper.mon_ap.connect(agent_ap);
        drv_wrapper.seq_item_port.connect(seq_wrapper.seq_item_export);

    endfunction

endclass //agent_wrapper extends uvm_agent


endpackage
```

config_ram

```
package config_ram;


`include "uvm_macros.svh"
import uvm_pkg::*;
    class config_ram extends uvm_object;

        `uvm_object_utils(config_ram)

        virtual interface_ram if_ram;
        uvm_active_passive_enum is_passive = UVM_PASSIVE; // Default to passive agent
        function new(string name = "config_ram");
            super.new(name);
        endfunction

    endclass

endpackage
```

config_slave

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

class config_slave extends uvm_object;
    `uvm_object_utils(config_slave)

    // Virtual interface for the slave
    virtual interface_slave if_slave;
    uvm_active_passive_enum is_passive = UVM_PASSIVE; // Default to passive agent
    // Constructor
    function new(string name = "config_slave");
        super.new(name);
    endfunction


endclass

endpackage
```

config_wrapper

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

class config_wrapper extends uvm_object;
    `uvm_object_utils(config_wrapper)

    // Virtual interface for the wrapper
    virtual interface_wrapper if_wrapper;

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


endclass

endpackage
```

interface_wrapper

```
interface interface_wrapper(clk);
    input clk;
    logic MOSI,MISO,SS_n,rst_n,MISO_ref;
endinterface
```

coverage_wrapper

```
package coverage_wrapper;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_wrapper_item::*;
class coverage_wrapper extends uvm_component;

    `uvm_component_utils(coverage_wrapper)
    sequence_wrapper_item item;
    // Coverage export
    uvm_analysis_export#(sequence_wrapper_item) cov_export;
    uvm_tlm_analysis_fifo#(sequence_wrapper_item) cov_fifo;

    // functional coverage
    // uvm_covergroup cov1;

    covergroup cov1;

    coverpoint item.MOSI {
        option.comment = "Covering MISO";
        bins miso_bins[] = {0, 1};
    }
    coverpoint item.SS_n {
        option.comment = "Covering SS_n";
        bins ss_bins[] = {0, 1};
    }
    coverpoint item.rst_n {
        option.comment = "Covering rst_n";
        bins rst_bins[] = {0, 1};
    }
    coverpoint item.MOSI {
        option.comment = "Covering MOSI";
        bins mosi_bins[] = {0, 1};
    }
    endgroup

    // Constructor
    function new(string name = "coverage_wrapper", uvm_component parent = null);
        super.new(name, parent);
        cov1 = 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);
        // Connect the export to the FIFO
        cov_export.connect(cov_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            // Wait for an item to be written to the FIFO
        cov_fifo.get(item);
        cov1.sample();


        end
    endtask
endclass


endpackage
```

driver_wrapper

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

class driver_wrapper extends uvm_driver#(sequence_wrapper_item);
    `uvm_component_utils(driver_wrapper)
    sequence_wrapper_item item;
    virtual interface_wrapper if_wrapper;
    // Constructor
    function new(string name = "driver_wrapper", uvm_component parent = null);
        super.new(name, parent);
    endfunction
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        item = sequence_wrapper_item::type_id::create("item");

    endfunction // build_phase
    // Main task to drive the sequence item
    task run_phase(uvm_phase phase);
        super.run_phase(phase);


        forever begin
            // Get the next item from the sequencer
            seq_item_port.get_next_item(item);

            // Drive the item to the virtual interface
            if (item != null) begin
                // Add your driving logic here
                if_wrapper.rst_n = item.rst_n;
                if_wrapper.MOSI = item.MOSI;
                if_wrapper.SS_n = item.SS_n;

                @(negedge if_wrapper.clk); // Wait for the clock edge
                seq_item_port.item_done();
            end
        end
    endtask
 endclass // driver_wrapper extends uvm_driver

endpackage
```

golden_model_wrapper

```
module golden_wrapper #(
    parameter MEM_DEPTH = 256,
    parameter ADDR_SIZE = 8
)(
    input logic MOSI,
    input logic SS_n,
    input logic clk,
    input logic rst_n,
    output logic MISO
);

    // FSM states
    typedef enum {
        IDLE,
        CHK_CMD,
        WRITE,
        READ_ADD,
        READ_DATA
    } state_t;

    // Internal signals
    state_t current_state, next_state;
    logic [9:0] rx_shift;
    logic [9:0] ram_din;
    logic rx_valid;
    logic [7:0] ram_dout;
    logic ram_tx_valid;
    logic [3:0] bit_counter;
    logic confirm_add;
    logic last_ssn;
    // RAM model
    logic [7:0] mem [0:MEM_DEPTH-1];
    logic [ADDR_SIZE-1:0] temp_rd, temp_wr;
    // Slave output register
    logic miso_reg;
    // FSM state transition
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            current_state <= IDLE;
            last_ssn <= 1'b1;
        end else begin
            current_state <= next_state;
            last_ssn <= SS_n;
        end
    end
    // FSM next state logic
    always_comb begin
        next_state = current_state;
        case (current_state)
            IDLE: begin
                if (!SS_n) next_state = CHK_CMD;
            end
            CHK_CMD: begin
                if (SS_n) begin
                    next_state = IDLE;
                end else if (!MOSI) begin
                    next_state = WRITE;
                end else if (MOSI && !confirm_add) begin
                    next_state = READ_ADD;
                end else if (MOSI && confirm_add) begin
                    next_state = READ_DATA;
                end
            end
            WRITE: begin
                if (SS_n) next_state = IDLE;
            end
            READ_ADD: begin
                if (SS_n) next_state = IDLE;
                else if (confirm_add) next_state = READ_DATA;
            end
            READ_DATA: begin
                if (SS_n) next_state = IDLE;
            end
            default: next_state = IDLE;
        endcase
    end
    // Bit counter and shift register
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            bit_counter <= 0;
            rx_shift <= 0;
            confirm_add <= 0;
            rx_valid <= 0;
            ram_din <= 0;
            miso_reg <= 0;
        end else begin
            rx_valid <= 0;  // Default no valid data
            case (current_state)
                IDLE: begin
                    bit_counter <= 0;
                    miso_reg <= 0;
                end
                CHK_CMD: begin
                    bit_counter <= 0;
                end
                WRITE: begin
                    if (bit_counter < 10) begin
                        rx_shift <= {rx_shift[8:0], MOSI};
                        bit_counter <= bit_counter + 1;
                    end
                    if (bit_counter == 9) begin
                        rx_valid <= 1;
                        ram_din <= {rx_shift[8:0], MOSI};  // Capture 10th bit
                    end
                end
                READ_ADD: begin
```

```
                    if (bit_counter < 10) begin
                        rx_shift <= {rx_shift[8:0], MOSI};
                        bit_counter <= bit_counter + 1;
                    end

                    if (bit_counter == 9) begin
                        rx_valid <= 1;
                        ram_din <= {rx_shift[8:0], MOSI};  // Capture 10th bit
                        confirm_add <= 1;
                    end
                end

                READ_DATA: begin
                    if (!ram_tx_valid) begin
                        if (bit_counter < 10) begin
                            rx_shift <= {rx_shift[8:0], MOSI};
                            bit_counter <= bit_counter + 1;
                        end

                        if (bit_counter == 9) begin
                            rx_valid <= 1;
                            ram_din <= {rx_shift[8:0], MOSI};
                            confirm_add <= 0;
                        end
                    end else begin
                        // Transmit data
                        if (bit_counter >= 3 && bit_counter <= 10) begin
                            miso_reg <= ram_dout[10 - bit_counter];
                        end

                        if (bit_counter > 3) begin
                            bit_counter <= bit_counter - 1;
                        end
                    end
                end
            endcase
            // Reset bit counter on SS_n edge
            if (last_ssn && !SS_n) begin
                bit_counter <= 0;
            end
        end
    end
    // RAM model
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            ram_dout <= 0;
            ram_tx_valid <= 0;
            temp_rd <= 0;
            temp_wr <= 0;
        end else begin
            ram_tx_valid <= 0;  // Default no valid output
            if (rx_valid) begin
                case (ram_din[9:8])
                    2'b00: begin  // Set write address
                        temp_wr <= ram_din[7:0];
                    end
                    2'b01: begin  // Write data
                        mem[temp_wr] <= ram_din[7:0];
                    end
                    2'b10: begin  // Set read address
                        temp_rd <= ram_din[7:0];
                    end
                    2'b11: begin  // Read data
                        ram_dout <= mem[temp_rd];
                        ram_tx_valid <= 1;
                    end
                endcase
            end
        end
    end
    // MISO output
    assign MISO = (current_state == READ_DATA && ram_tx_valid) ? miso_reg : 1'b0;
    // Debug signals (can be removed for synthesis)
    logic [7:0] debug_mem_read;
    assign debug_mem_read = mem[temp_rd];

endmodule
```

monitor_wrapper

```systemverilog
package monitor_wrapper;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_wrapper_item::*;
class monitor_wrapper extends uvm_monitor;
    `uvm_component_utils(monitor_wrapper)
    sequence_wrapper_item item;
    virtual interface_wrapper if_wrapper;
    uvm_analysis_port #(sequence_wrapper_item) mon_ap;



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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        // Create the analysis port
        mon_ap = new("mon_ap", this);
    endfunction

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

        forever begin
            item = sequence_wrapper_item::type_id::create("item");
            // Wait for a valid transaction
           @(negedge if_wrapper.clk);

                item.MOSI = if_wrapper.MOSI;
                item.MISO = if_wrapper.MISO;
                item.rst_n = if_wrapper.rst_n;
                item.SS_n = if_wrapper.SS_n;
                item.MISO_ref = if_wrapper.MISO_ref;


                // Send the item to the analysis export
                mon_ap.write(item);
            end

    endtask

endclass //monitor_wrapper extends uvm_monitor;


endpackage
```

scoreboard_wrapper

```systemverilog
package scoreboard_wrapper;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_wrapper_item::*;
class scoreboard_wrapper extends uvm_scoreboard;
    `uvm_component_utils(scoreboard_wrapper)

    sequence_wrapper_item item;
    int correct = 0;
    int incorrect = 0;


    // sberage export
    uvm_analysis_export#(sequence_wrapper_item) sb_export;
    uvm_tlm_analysis_fifo#(sequence_wrapper_item) sb_fifo;

    // Constructor
    function new(string name = "scoreboard_wrapper", 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);
        // Connect the export to the FIFO
        sb_export.connect(sb_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            // Wait for an item to be written to the FIFO
        sb_fifo.get(item);

          if ( item.MISO == item.MISO_ref) begin
            correct++;
          end
          else begin
            incorrect++;
           `uvm_error("SCOREBOARD", $sformatf("Incorrect data: Expected %0d, Got %0d", item.MISO, item.MISO_ref))
         end

        end
    endtask
    function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("SCOREBOARD", $sformatf("Correct: %0d, Incorrect: %0d", correct, incorrect), UVM_LOW)
    endfunction
endclass

endpackage
```

sequence_rst_wrapper

```
package sequence_rst_wrapper; //Stimulus
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_wrapper_item::*;
class sequence_rst_wrapper extends uvm_sequence#(sequence_wrapper_item);

    `uvm_object_utils(sequence_rst_wrapper)
    sequence_wrapper_item item;
    // Constructor
    function new(string name = "sequence_rst_wrapper");
        super.new(name);
    endfunction
    // Body phase
    virtual task body();
        // Create a new sequence item
        item = sequence_wrapper_item::type_id::create("item");
            start_item(item);
            // rst
            item.rst_n = 0; // Set reset to low
            // Finish the item
            finish_item(item);
    endtask
endclass
endpackage
```

sequence_wrapper_item

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

class sequence_wrapper_item extends uvm_sequence_item;

    `uvm_object_utils(sequence_wrapper_item)

    rand bit rst_n; // Reset signal, active low
    rand bit MOSI;
    rand bit SS_n; // Slave Select, active low
    bit MISO,MISO_ref; // Master In Slave Out
    // Constructor
    function new(string name = "sequence_wrapper_item");
        super.new(name);
    endfunction
    constraint c1 {
        rst_n dist {0 := 1, 1 := 99}; // Reset signal is low for 1% of the time
        MOSI dist {0 := 75, 1 := 25}; // MOSI signal is low for 1% of the time
        SS_n dist {0 := 75, 1 := 25}; // Slave Select signal is low for 1% of the time

    }
endclass
endpackage
```

sequence_wrapper

```
package sequence_wrapper; //Stimulus
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_wrapper_item::*;
class sequence_wrapper extends uvm_sequence#(sequence_wrapper_item);

    `uvm_object_utils(sequence_wrapper)
    sequence_wrapper_item item;
    // Constructor
    function new(string name = "sequence_wrapper");
        super.new(name);
    endfunction
    // Body phase
    virtual task body();
        sequence_wrapper_item item;

        // Create a new sequence item
        item = sequence_wrapper_item::type_id::create("item");

        repeat(10000) begin
             // Start the item
            start_item(item);
            // Randomize the item
            assert(item.randomize());
            // Finish the item
            finish_item(item);
        end
    endtask
endclass

endpackage
```

sequencer_wrapper

```
package sequencer_wrapper;
`include "uvm_macros.svh"
import uvm_pkg::*;
import sequence_wrapper_item::*;
class sequencer_wrapper extends uvm_sequencer#(sequence_wrapper_item);

    `uvm_component_utils(sequencer_wrapper)

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

endpackage
```

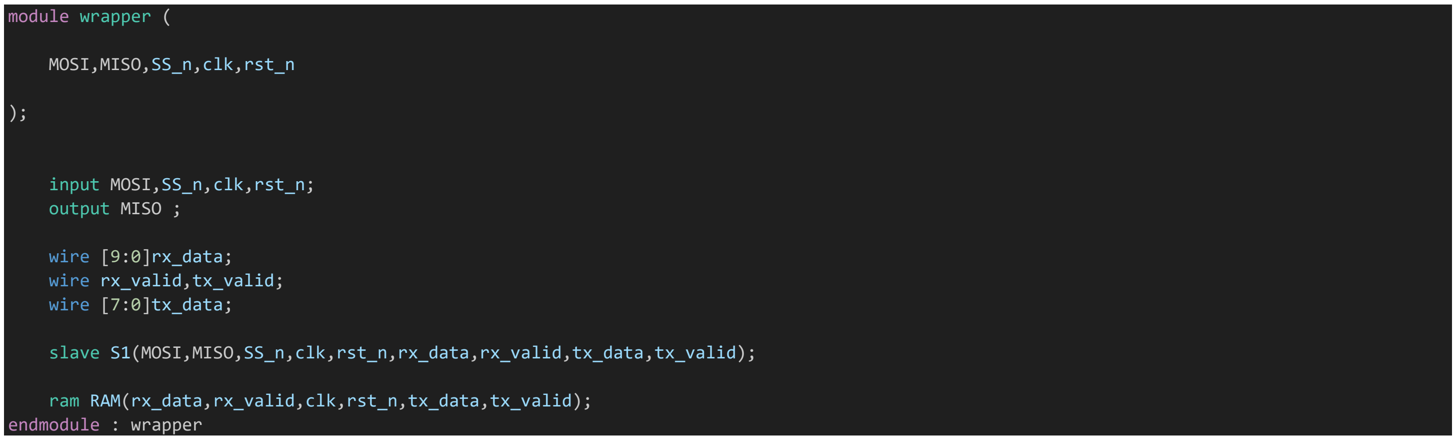
agent_slave_change_only

```
if(cfg.is_passive == UVM_PASSIVE )begin
        seq_slave = sequencer_slave::type_id::create("seq_slave", this);
        drv_slave = driver_slave::type_id::create("drv_slave", this);
        end
```

agent_ram_change_only

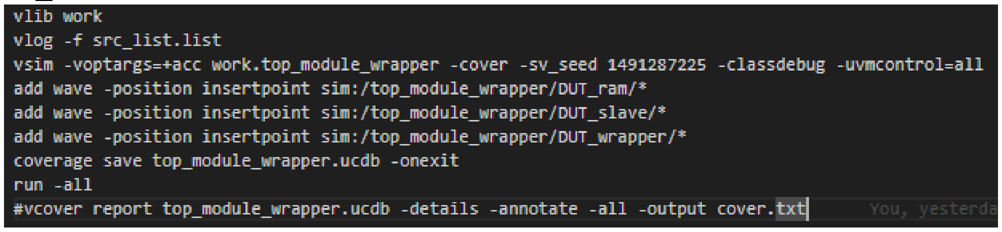
```
if(cfg.is_passive == UVM_PASSIVE )begin

        seq_ram = sequencer_ram::type_id::create("seq_ram", this);
        drv_ram = driver_ram::type_id::create("drv_ram", this);
        end
```

wrapper



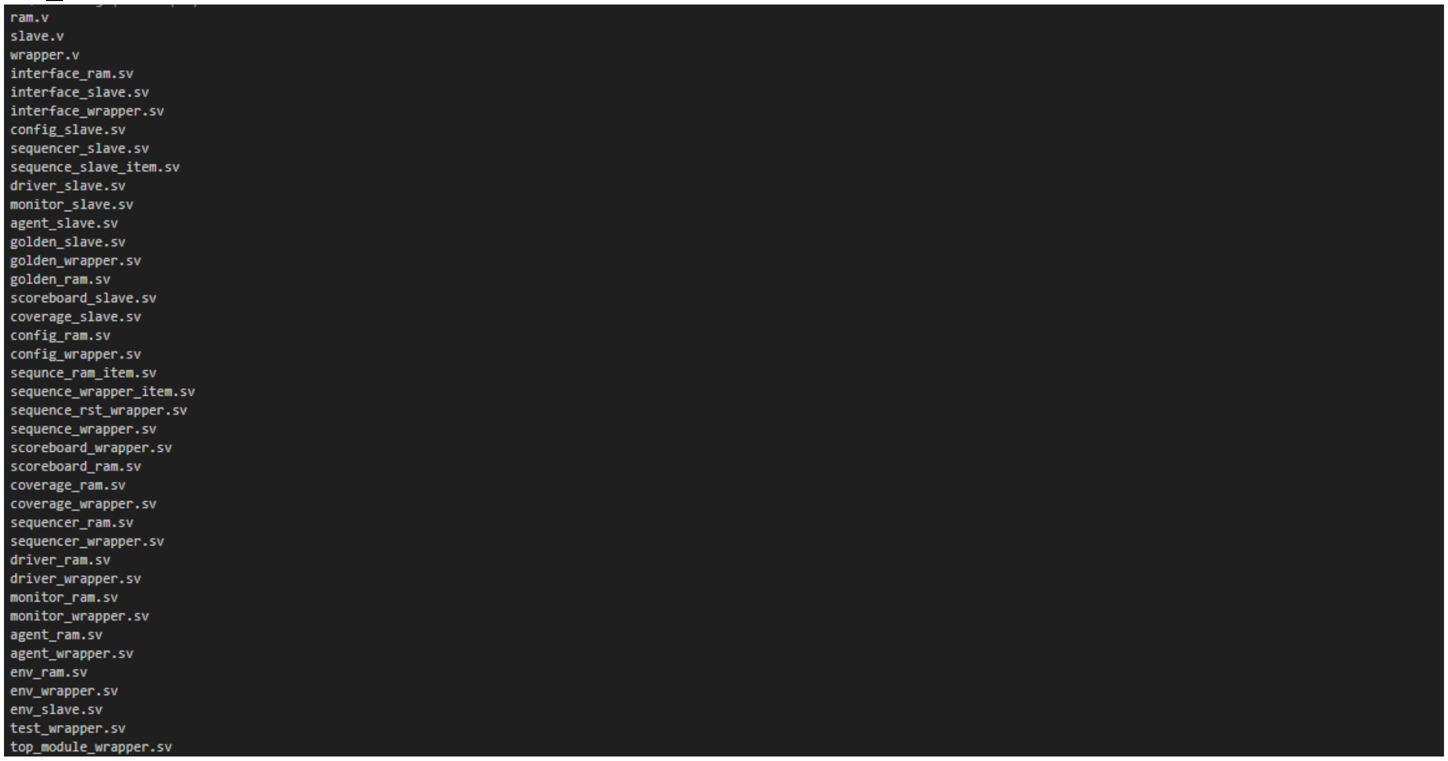
```
module wrapper (

    MOSI,MISO,SS_n,clk,rst_n

);


    input MOSI,SS_n,clk,rst_n;
    output MISO ;

    wire [9:0]rx_data;
    wire rx_valid,tx_valid;
    wire [7:0]tx_data;

    slave S1(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);

    ram RAM(rx_data,rx_valid,clk,rst_n,tx_data,tx_valid);
endmodule : wrapper
```

do_file



```
vlib work
vlog -f src_list.list
vsim -voptargs=+acc work.top_module_wrapper -cover -sv_seed 1491287225 -classdebug -uvmcontrol=all
add wave -position insertpoint sim:/top_module_wrapper/DUT_ram/*
add wave -position insertpoint sim:/top_module_wrapper/DUT_slave/*
add wave -position insertpoint sim:/top_module_wrapper/DUT_wrapper/*
coverage save top_module_wrapper.ucdb -onexit
run -all
#vcover report top_module_wrapper.ucdb -details -annotate -all -output cover.txt      You, yesterda
```

Src_list



```
ram.v
slave.v
wrapper.v
interface_ram.sv
interface_slave.sv
interface_wrapper.sv
config_slave.sv
sequencer_slave.sv
sequence_slave_item.sv
driver_slave.sv
monitor_slave.sv
agent_slave.sv
golden_slave.sv
golden_wrapper.sv
golden_ram.sv
scoreboard_slave.sv
coverage_slave.sv
config_ram.sv
config_wrapper.sv
sequnce_ram_item.sv
sequence_wrapper_item.sv
sequence_rst_wrapper.sv
sequence_wrapper.sv
scoreboard_wrapper.sv
scoreboard_ram.sv
coverage_ram.sv
coverage_wrapper.sv
sequencer_ram.sv
sequencer_wrapper.sv
driver_ram.sv
driver_wrapper.sv
monitor_ram.sv
monitor_wrapper.sv
agent_ram.sv
agent_wrapper.sv
env_ram.sv
env_wrapper.sv
env_slave.sv
test_wrapper.sv
top_module_wrapper.sv
```

For more information, please visit my repo: