Manipal Center of Excellence
Single_Port_RAM
mirafra
TECHNOLOGIES
BATCH-9

Date	Document Version	Remarks	Drafted by
25-06-2024	Version 1.0	Design Overview	K Vijay Kumar
26-06-2024	Version 2.0	Updated Verification Architecture and Implementation	K Vijay Kumar
04-07-2024	Version 3.0	Updated test cases to simulation results	K Vijay Kumar

VERIFICATION DOCUMENT- Single Port RAM

Contents

CHAPTER 1 - DESIGN OVERVIEW

1.1 Single port RAM:

A single port memory is a type of memory that can be accessed by only one device or process at a time. In this type of memory, data can be written and read from the same port. It is generally used in applications where only one processor is used, and the memory is not required to be accessed by multiple processors simultaneously.

1.2 Advantages of single port RAM:

1. Simplicity: Single port memories are simple to design and implement compared to multi-port memories.
2. Lower power consumption: Single port memories consume less power compared to multi-port memories because they do not require complex circuitry.
3. Reduced complexity: Since single port memories can be accessed by only one device at a time, there is no need for synchronization between multiple devices, thus reducing complexity.

1.3 Disadvantages of single port RAM:

1. Limited access: Single port memories can only be accessed by one device at a time, which limits their use in multi-processor systems.
2. Lower throughput: Since only one device can access the memory at a time, the throughput is lower compared to multi-port memories.
3. Higher latency: Due to the limited access, single port memories may have higher latency when multiple devices are waiting to access the memory

1.4 Use cases of single port RAM:

1. **Microcontroller systems:** Single port memories are widely used in microcontroller systems, where only one processor is used.
2. **Embedded systems:** Single port memories are also used in embedded systems, where low power consumption and simplicity are critical factors.
3. **Image and video processing:** Single port memories can be used in image and video processing applications to store and retrieve data.
4. **Signal processing:** Single port memories can also be used in signal processing applications to store and retrieve data.

1.5 Project Overview of single port RAM:

- This is a single port RAM
- The operating frequency for the RAM is 25MHz. All changes in the RAM happen at the positive edge of the clock.
- This RAM is 8 bits wide and 32 locations deep.
- When the synchronous active high reset is asserted, the RAM goes into an idle state, i.e., the data output becomes Z.
- RAM supports both Read and write operations. If an invalid address is specified for a write operation, the RAM does nothing. However, if an invalid address is specified for a read operation,
  i.e., address value > that supported by the address width, the invalid address is truncated to fit in the address width and the data read from the RAM will be done using that truncated location.
- It does not support concurrent read and write operations.

1.6 Single Port Design Features

- The design is a single port RAM
- The RAM clock frequency is 25 MHz
- The depth of the RAM is 32 locations
- The data width is 8-bits
- Changes in the RAM happen at positive edge of clock
- Synchronous active high reset. On assertion, the data output becomes ‘z’
- The write operation into the RAM is supported. For invalid address, the RAM does nothing
- The read operation from the RAM is supported.
  For invalid address, i.e., address value > that supported by the address width, the invalid address is truncated to fit in the address width and the data read from the RAM will be done using that truncated location

1.7 Design Limitation

- The RAM does not support simultaneous (concurrent) read and write operations

1.8 Design diagram with interface signals

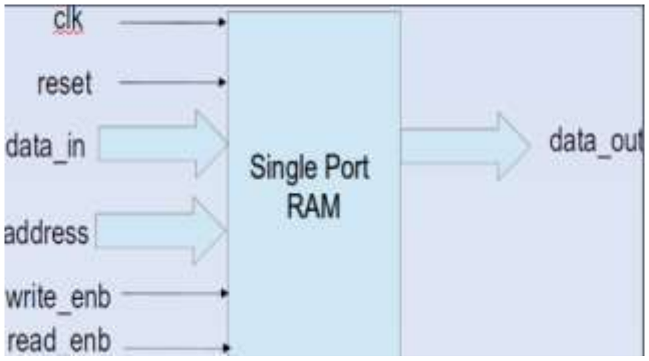


Fig 1.1: SinglePort RAM Design block

Pin name	*Direction*	*Width in bits*	*Functionality*
clk	Input	1	Clock signal to the RAM
reset	Input	1	Active high reset signal, to reset the RAM operations
write_enb	Input	1	Active high signal to enable write operation
read_enb	Input	1	Active high signal to enable the read operation
data_in	Input	8	Data input to the RAM
address	Input	5	Address to identify the memory location from 0 to 31
data_out	Output	8	Data output from the RAM

Fig 1.2: Ports information about Single Port RAM

CHAPTER 2 – TEST BENCH ARCHITECTURE

2.1 Verification Architecture:

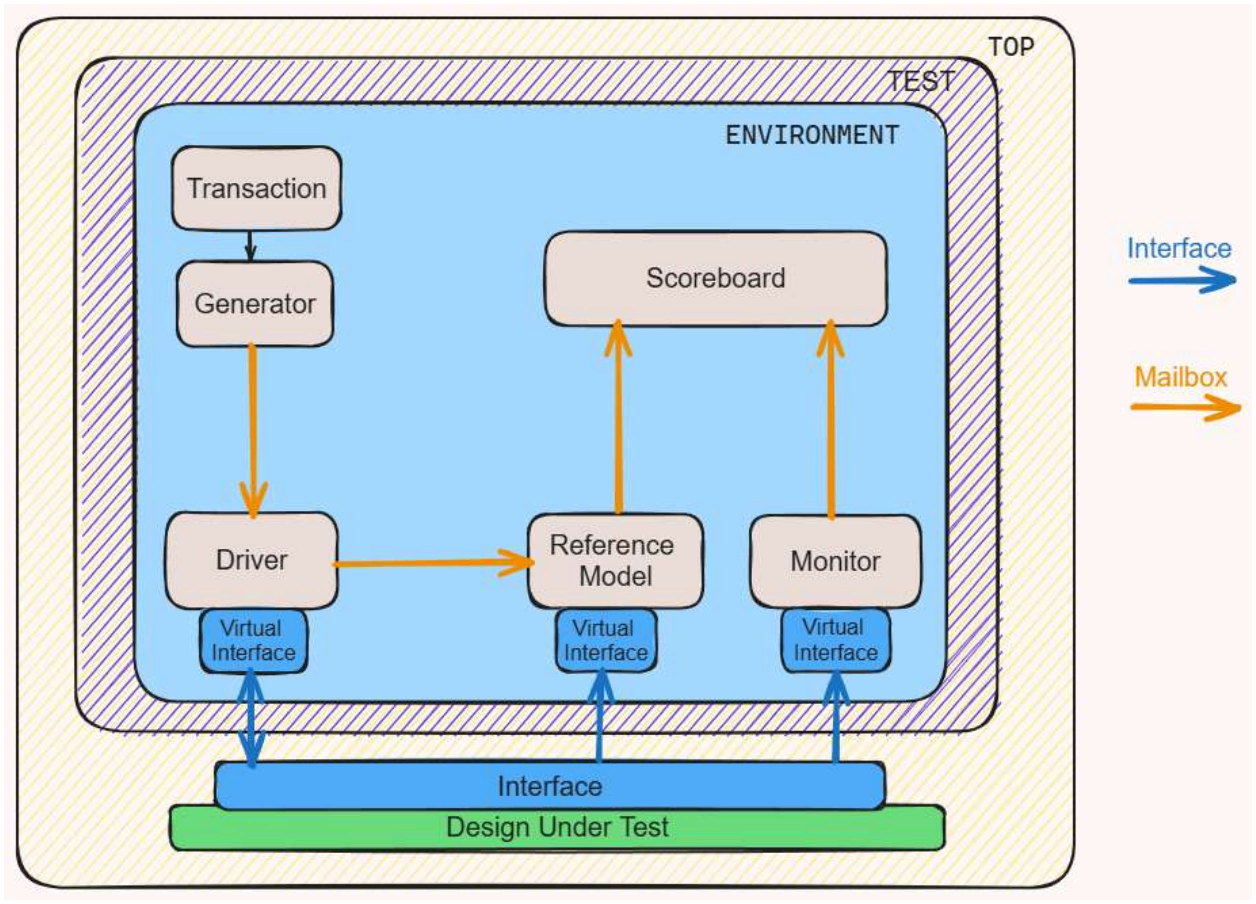


Fig 2.1 System Verilog Verification Architecture

A Testbench architecture typically refers to the structure and organization of a testbench environment used for verifying the design under test. It defines the components of the SV testbench, such as the top-level testbench module, driver, monitor, scoreboard, generator, transactions, reference module. It also specifies how these components interact with each other and with the design under test through interface.

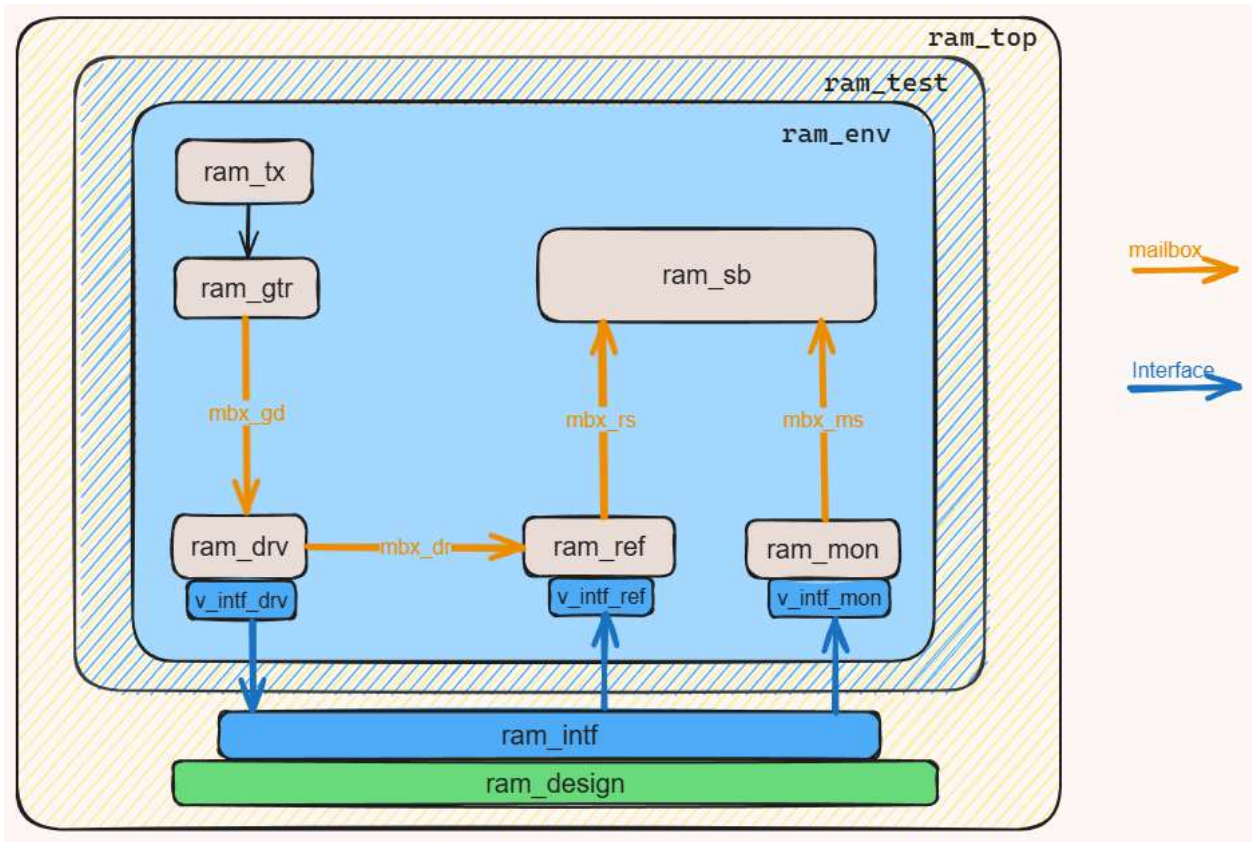
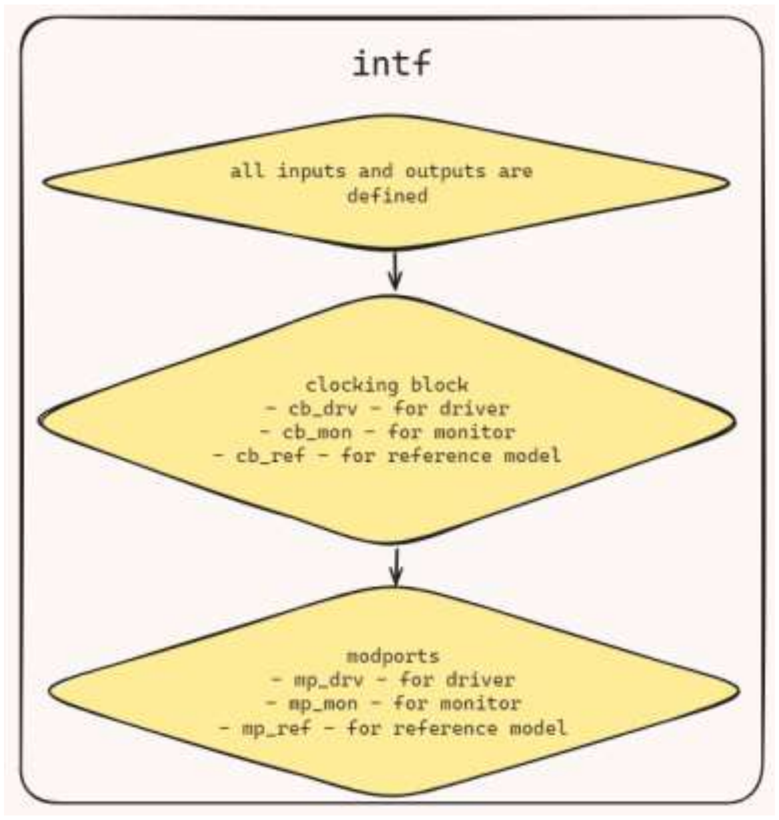


Fig 2.2 Verification Architecture for Single Port RAM

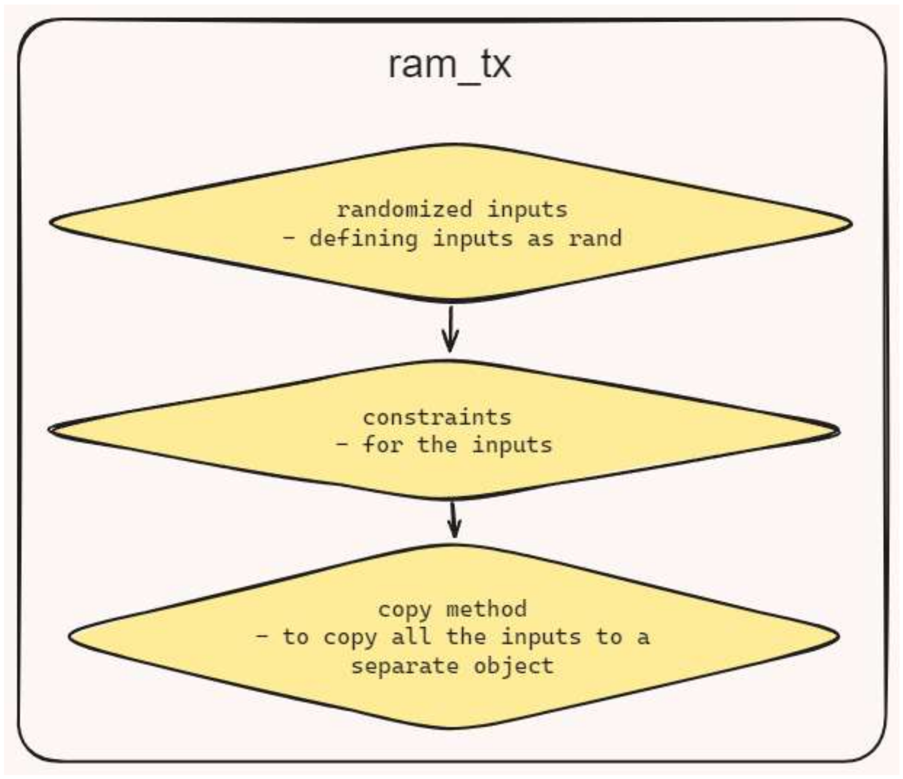
CHAPTER 3 – TEST BENCH IMPLEMENTATION

3.1 Interface: (intf)



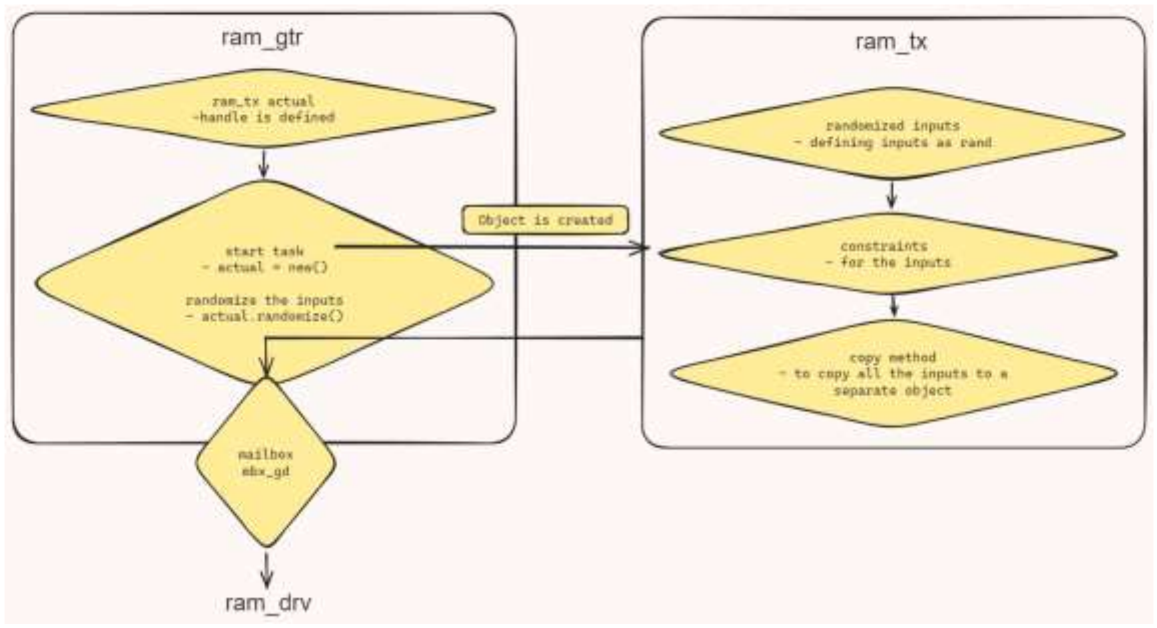
- All the inputs and outputs of the RAM are defined
- Clocking blocks are defined separately for driver, monitor and reference model (cb_drv, cb_mon, cb_ref)
- Modports are defined separately for driver, monitor and reference model (mp_drv, mp_mon, mp_ref)
- The clocking block provides input and output skew

3.2 Transaction: (ram_tx)


ram_tx
randomized inputs
- defining inputs as rand
constraints
- for the inputs
copy method
- to copy all the inputs to a separate object

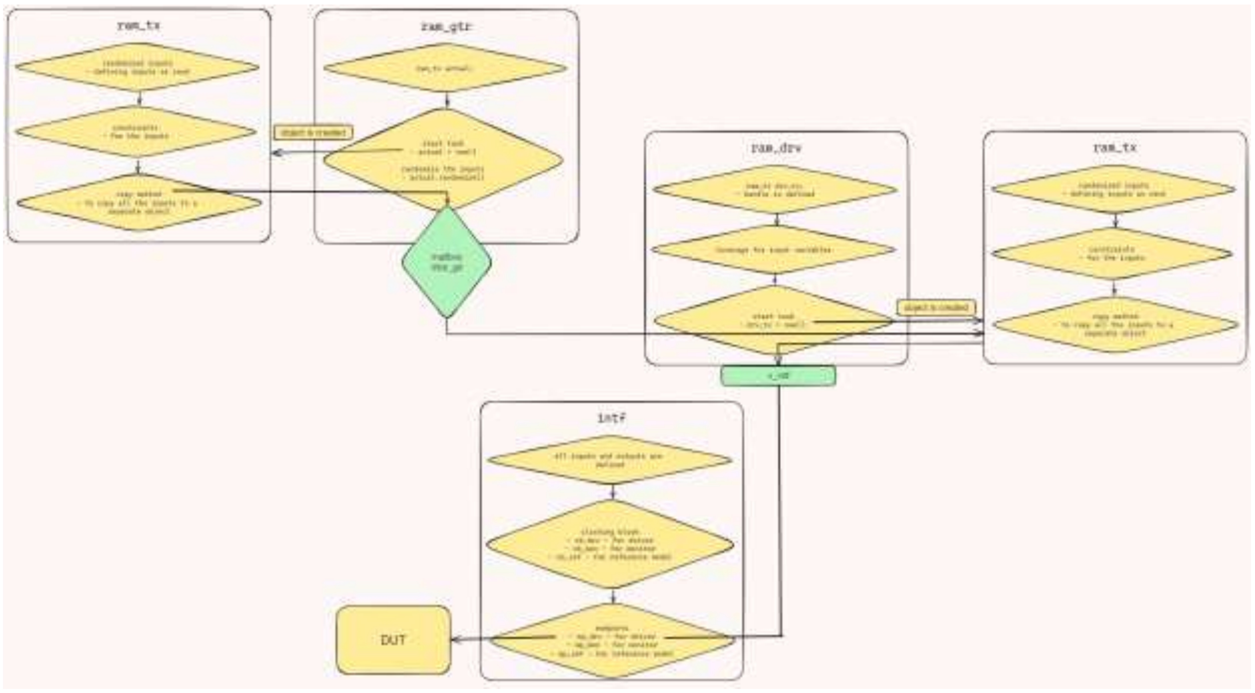
- All the inputs are declared as random variables
- Constraints are defined for the required inputs
- A copy function is defined to copy all the inputs to a separate object

3.3 Generator: (ram_gtr)



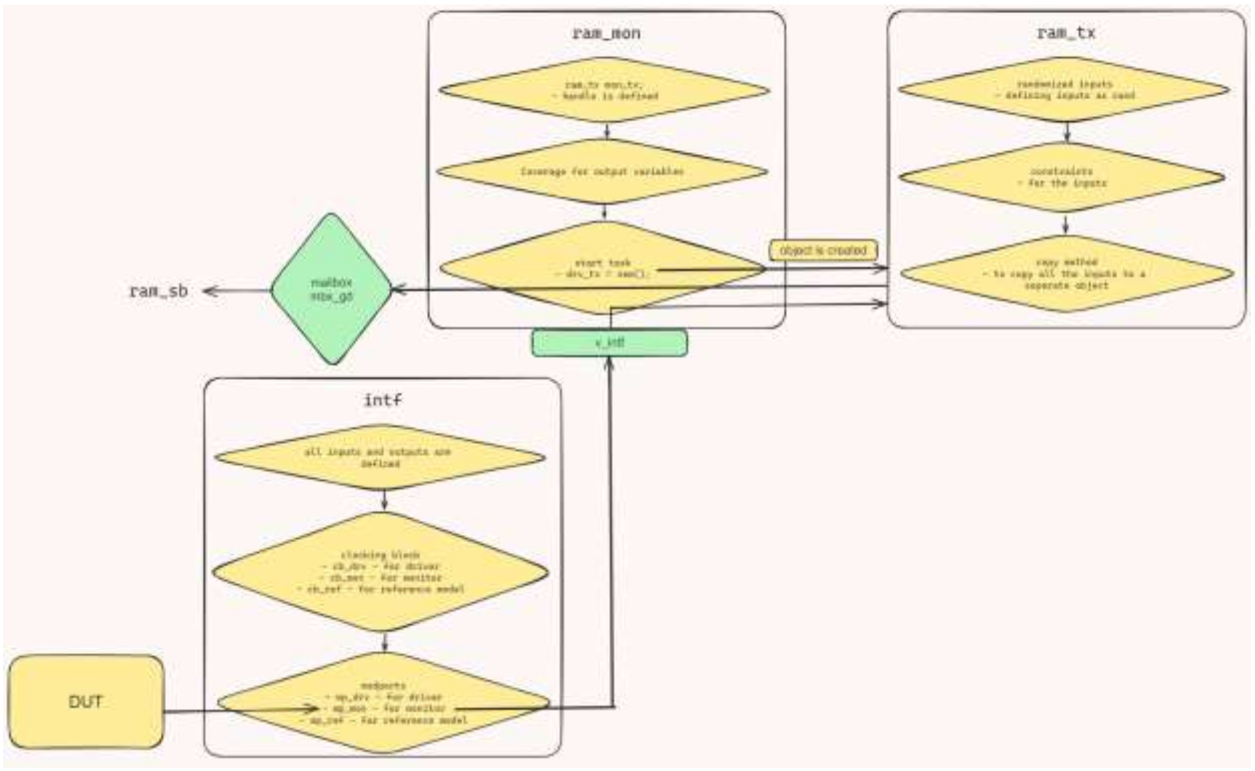
- Handle (original_tx) is defined for the class ram_tx
- Mailbox (mbx_gd) is defined which is used to transfer data between generator and driver
- A “start” task is used to randomize the input values using the object, actual and the randomized values are put into mailbox (mbx_gd) by calling copy method of class ram_tx using object, actual

3.4 Driver: (ram_drv)



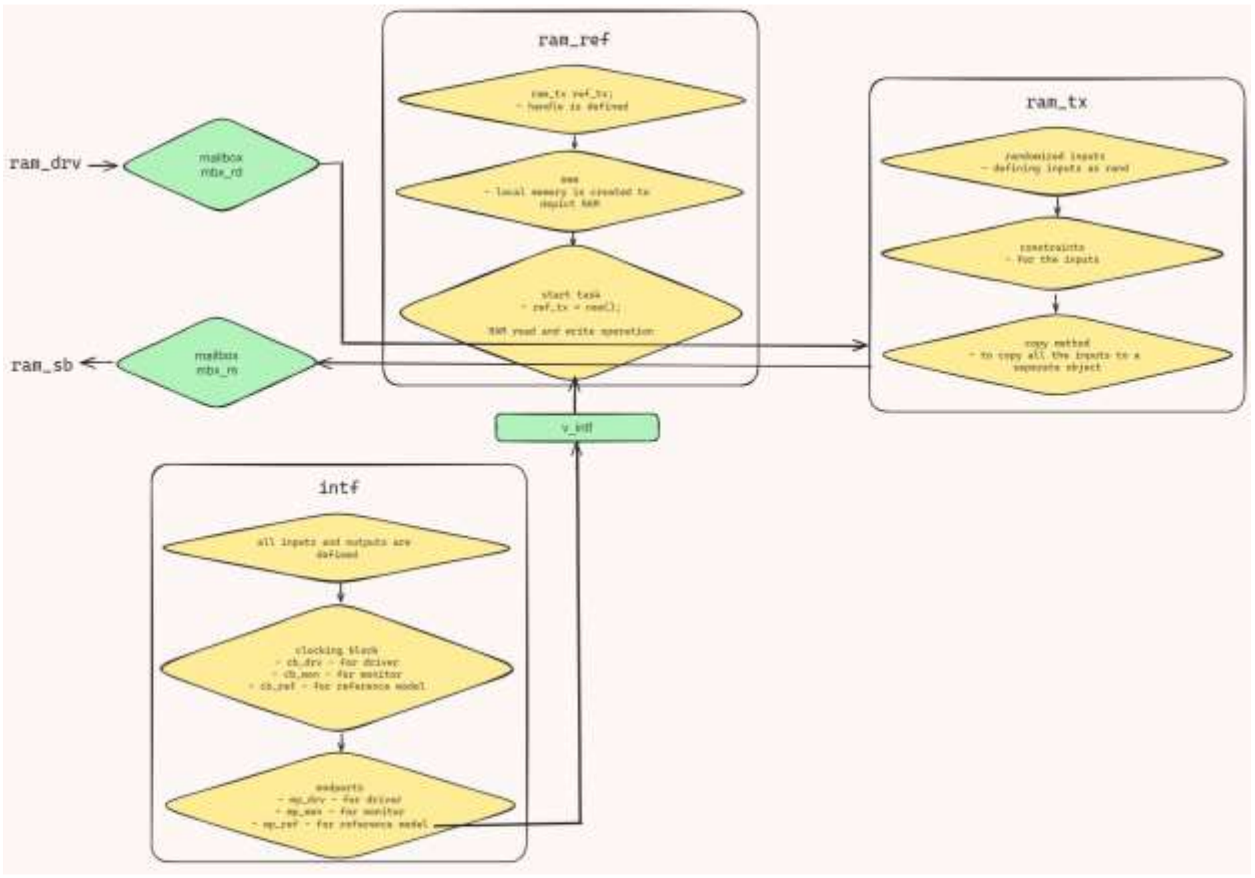
- Handle (drv_tx) is created for the class ram_tx
- Mailbox (mbx_dg) is defined which is used to transfer data between driver and generator
- Mailbox (mbx_dr) is defined which is used to transfer data between driver and reference model
- Virtual Interface v_intf_drv is defined to drive the DUT
- Coverage of the input variables are defined
- A start task is defined to get randomized input values from the generator using mailbox (mbx_dg) and assign it to variables of object, drv_tx. Then drives the DUT through virtual interface (v_intf_drv)

3.5 Monitor: (ram_mon)


ram_mon
ram_tx mon_tx; - handle is defined
Coverage for output variables
start task - drv_tx = new();
ram_tx
randomized inputs - defining inputs as rand
constraints - for the inputs
copy method - to copy all the inputs to a separate object
object is created
ram_sb
mailbox mbx_gd
v_intf
intf
all inputs and outputs are defined
clocking block - cb_drv - for driver - cb_mon - for monitor - cb_ref - for reference model
modports - mp_drv - for driver - mp_mon - for monitor - mp_ref - for reference model
DUT

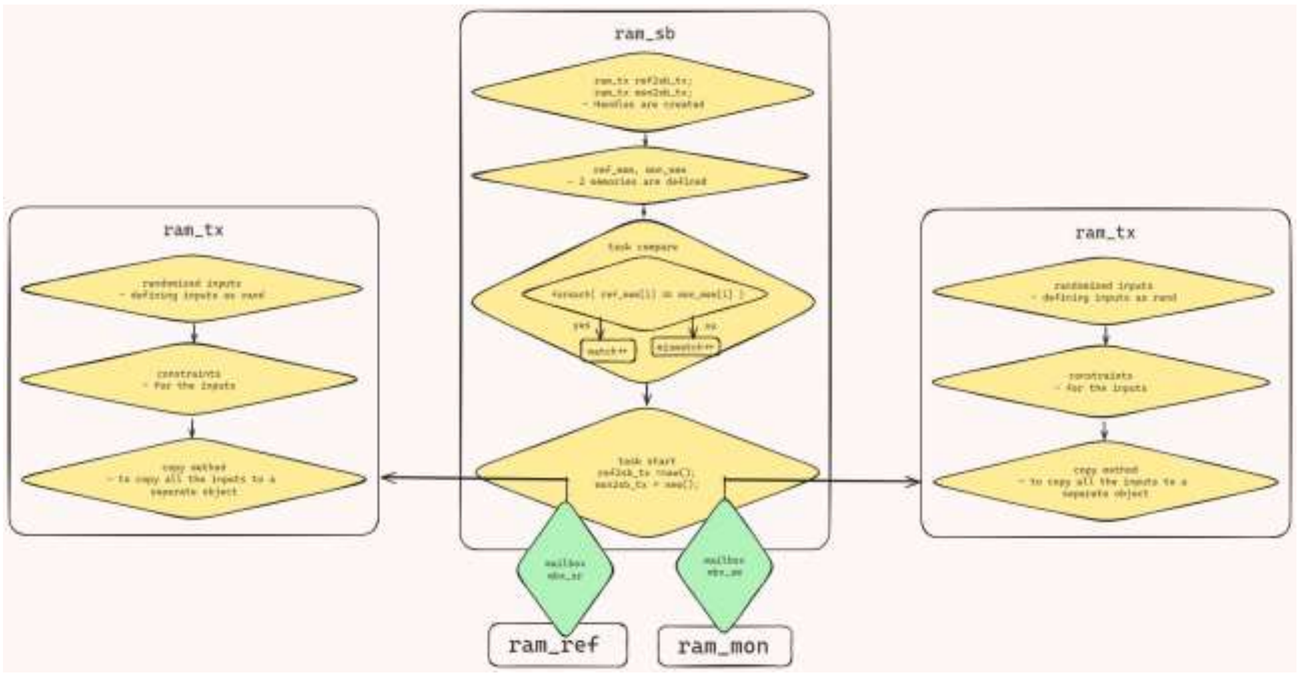
- Handle (mon_tx) is created for the class ram_tx
- Mailbox (mbx_ms) is defined which is used to transfer data between monitor and scoreboard
- Virtual Interface v_intf_mon is defined to observe the outputs of DUT
- Coverage for the output variables of DUT is performed
- A start task is defined to observe the outputs of DUT using v_intf_mon and assign it to the properties of object, mon_tx. Then it is put into mailbox, mbx_ms.

3.6 Reference Model: (ram_ref)



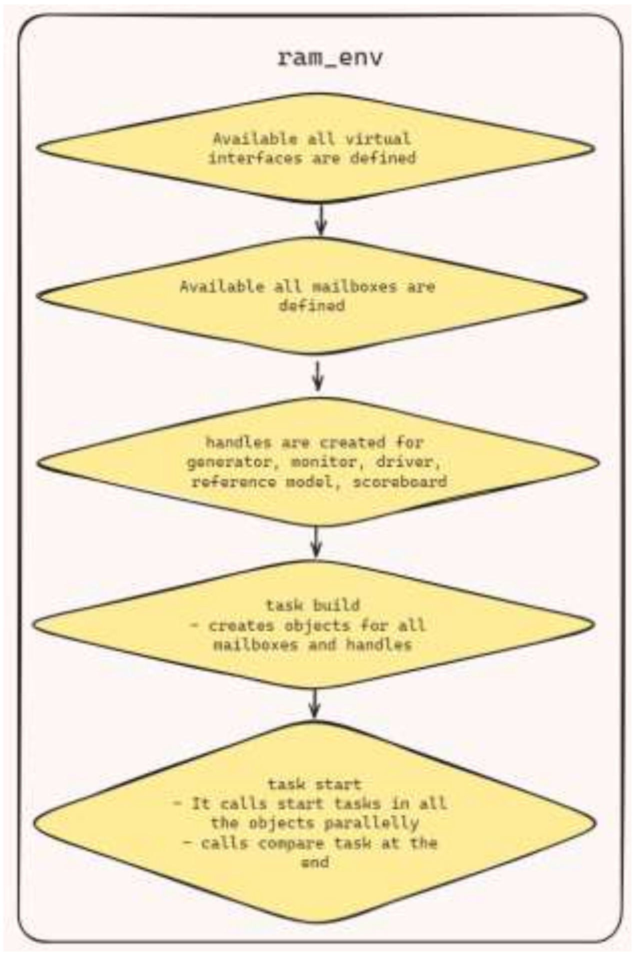
- Handle (ref_tx) is created for the class ram_tx
- Mailbox (mbx_rd) is defined which is used to transfer data between reference model and driver
- Mailbox (mbx_rs) is defined which is used to transfer data between reference model and scoreboard
- Virtual Interface v_intf_ref is defined to include skew to the clock
- A local memory (mem) is defined to depict the RAM
- A start task is defined to get input values from driver using mailbox, mbx_rd and obtain the output for obtained inputs using skewed clock through virtual interface (v_intf_ref). Then put the obtained values to mailbox, mbx_rs using object ref_tx

3.7 Scoreboard: (ram_sb)



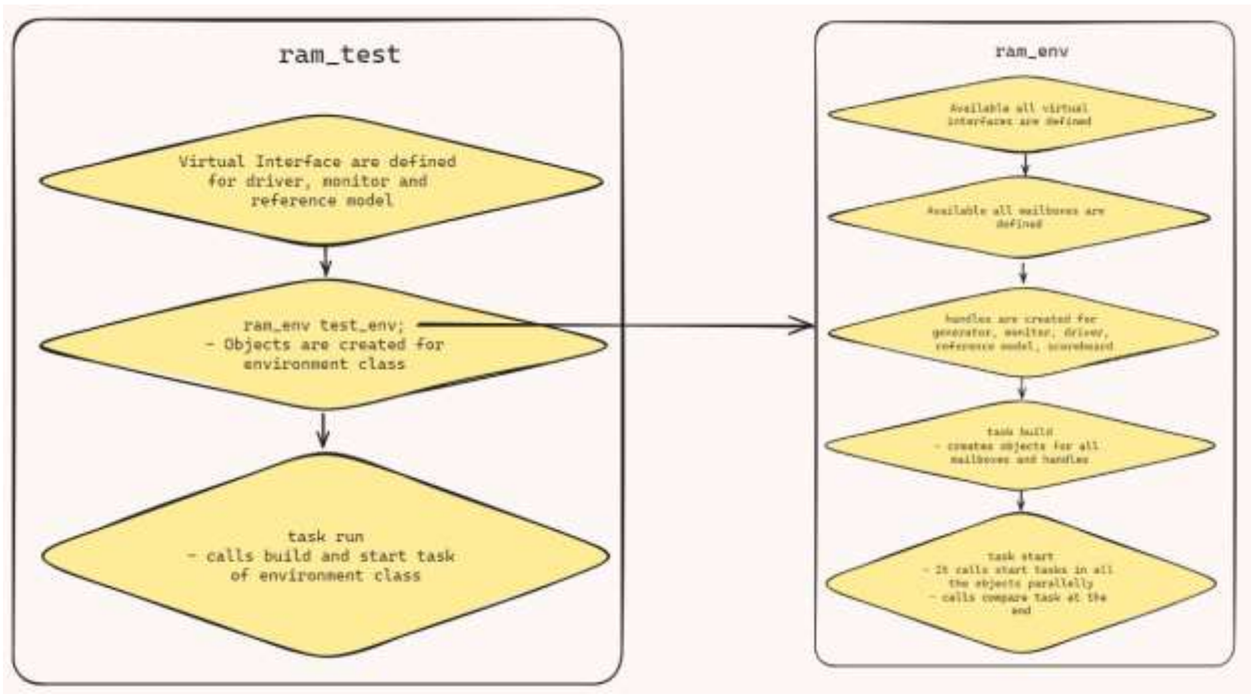
- Handle (ref2sb_tx) is created for the class ram_tx
- Handle (mon2sb_tx) is created for the class ram_tx
- Mailbox (mbx_sr) is defined which is used to transfer data between scoreboard and reference model
- Mailbox (mbx_sm) is defined which is used to transfer data between scoreboard and monitor
- Two local memories (ref_mem and mon_mem) are defined to store values from reference model and monitor
- Two variables match and mismatch are declared to provide number of matches and mismatches between the actual and expected results
- A start task is defined to get values from mailbox, mbx_sr and mbx_sm into memories ref_mem and mon_mem.
- A compare task is defined to compare the memories and increment match and mismatch variables

3.8 Environment: (ram_env)


ram_env
Available all virtual interfaces are defined
Available all mailboxes are defined
handles are created for generator, monitor, driver, reference model, scoreboard
task build - creates objects for all mailboxes and handles
task start - It calls start tasks in all the objects parallelly - calls compare task at the end

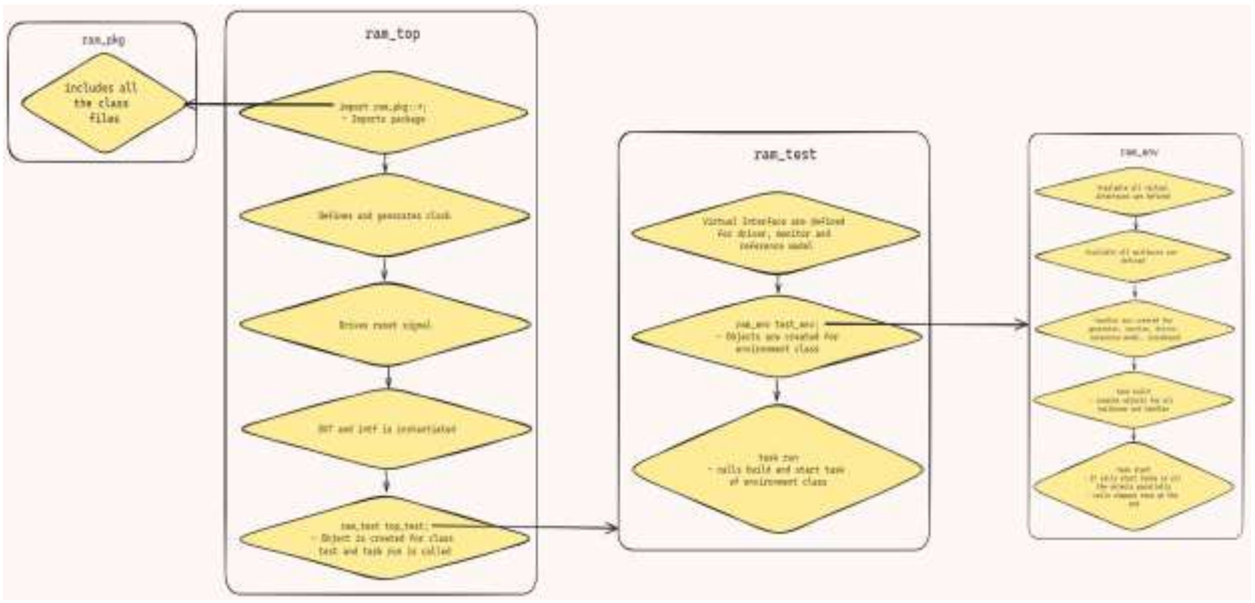
- Virtual Interface v_intf_drv, v_intf_mon and v_intf_ref are defined for driver, monitor and reference model
- Mailbox (mbx_gd) is defined for generator to driver connection
- Mailbox (mbx_dr) is defined for driver to reference model connection
- Mailbox (mbx_ms) is defined for monitor to scoreboard connection
- Mailbox (mbx_rs) is defined for reference model to scoreboard connection
- A build task is defined to create objects for all mailboxes and objects env_gen, env_drv, env_mon, env_ref, env_sb for classes generator, driver, monitor, reference model, scoreboard
- A start task is defined to call all the start task of each class objects parallelly
- Finally calling the compare task of scoreboard

3.9 Test: (ram_test)


ram_test
Virtual Interface are defined for driver, monitor and reference model
ram_env test_env; - Objects are created for environment class
task run - calls build and start task of environment class
ram_env
Available all virtual interfaces are defined
Available all mailboxes are defined
handles are created for generator, monitor, driver, reference model, scoreboard
task build - creates objects for all mailboxes and handles
task start - It calls start tasks in all the objects parallelly - calls compare task at the end

- Virtual Interface v_intf_drv, v_intf_mon and v_intf_ref are defined for driver, monitor and reference model
- Object is created for class environment by passing all the virtual interfaces defined
- A run task is defined to call the build and start tasks of environment class

3.10 Top: (ram_top)


ram_pkg
includes all the class files
ram_top
import ram_pkg::*; - Imports package
Defines and generates clock
Drives reset signal
DUT and intf is instantiated
ram_test top_test; - Object is created for class test and task run is called
ram_test
Virtual Interface are defined for driver, monitor and reference model
ram_env test_env; - Objects are created for environment class
task run - calls build and start task of environment class
ram_env

- Imports package (ram_pkg) to include all the classes.
- Defining clock (clk) and reset (rst)
- Generating clock and driving the reset signal
- DUT and Interface (ram_intf) are instantiated
- Object for test class is defined and the task run is called

CHAPTER 4 – TEST PLAN AND TEST CASES

S.No	TestName	Test Description	Priority	Status
1	write_with_write_en	We will test writing to the RAM with 25MHz, write_en high and low reset	1	Pass
2	read_with_read_en	We will test reading from the RAM with 25MHz, read_en high and low reset	1	Pass
3	read_or_write_with_reset	We will test reading and writing from and to the RAM with 25MHz and high reset	1	Pass

CHAPTER 5 – DESIGN VERIFICATION COMPONENTS

The Github link to the components (codes) of the Design Verification is given below:

https://github.com/kvjk2001/Single_Port_RAM

CHAPTER 6 – USER GUIDE

6.1 Simulating in QuestaSim vlog 10.6c Compiler 2017.07 Jul 25 2017

Questa-sim opening command: vsim $

Compilation command: vlog <testbench file>

End-Of-Elaboration command: vsim -novopt -suppress 12110 <top level modules-object>

Waveform command: add wave -position insertpoint sim:/ <top level modules-object>/duv/*

Logfile command: add log -r sim:/ <top level modules-object>/*

Run command: run -all

6.2 Coverage using QuestaSim vlog 10.6c Compiler 2017.07 Jul 25 2017

The below commands are used to get code coverage and functional coverage report:

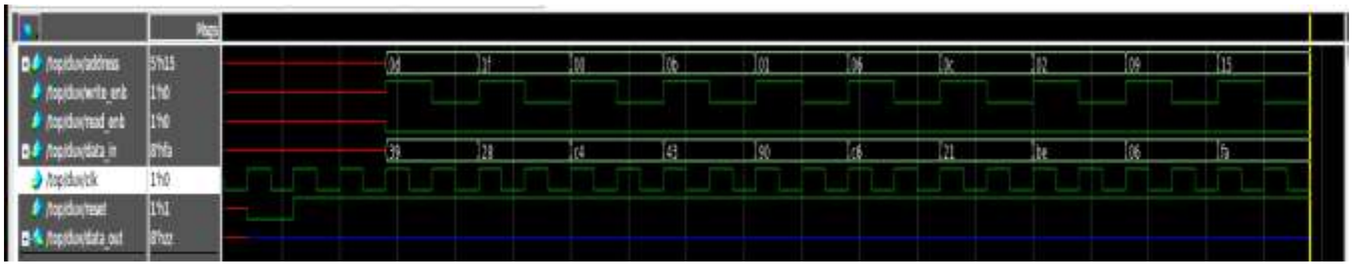
```
vlib work
vlog -sv +acc +cover +fcover -l ram_top.log ram_top.sv
vsim -vopt work.top -voptargs=+acc=npr -assertdebug -l simulation.log -coverage -c -do
"coverage save -onexit -assert -directive -cvg -codeAll coverage.ucdb; run -all; coverage report -
detail; exit"
vcover report -html coverage.ucdb -htmldir covReport -details
cd covReport/
firefox index.html
```

CHAPTER 7 – SIMULATION RESULTS

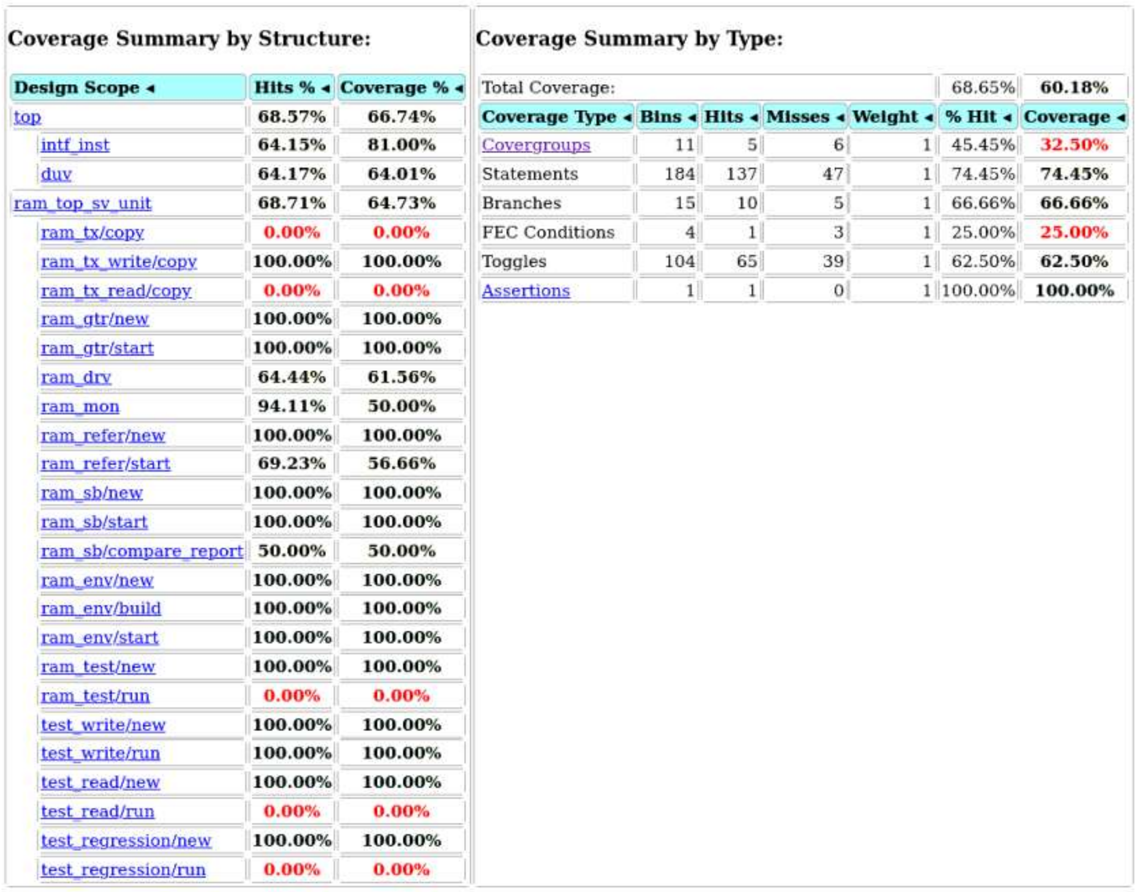
7.1 Test cases

7.1.1 write_with_write_en

Waveform:



Coverage Report:



Coverage Summary by Structure:

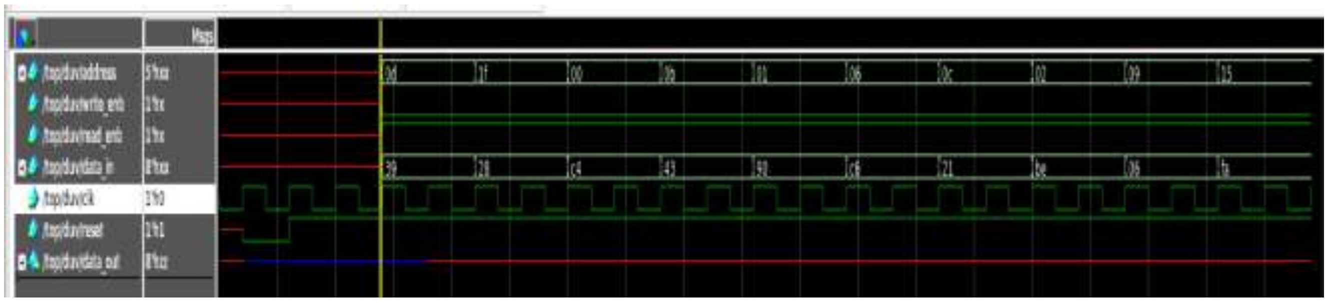
Design Scope	Hits %	Coverage %
top	68.57%	66.74%
intf_inst	64.15%	81.00%
duv	64.17%	64.01%
ram_top_sv_unit	68.71%	64.73%
ram_tx/copy	0.00%	0.00%
ram_tx_write/copy	100.00%	100.00%
ram_tx_read/copy	0.00%	0.00%
ram_gtr/new	100.00%	100.00%
ram_gtr/start	100.00%	100.00%
ram_drv	64.44%	61.56%
ram_mon	94.11%	50.00%
ram_refer/new	100.00%	100.00%
ram_refer/start	69.23%	56.66%
ram_sb/new	100.00%	100.00%
ram_sb/start	100.00%	100.00%
ram_sb/compare_report	50.00%	50.00%
ram_env/new	100.00%	100.00%
ram_env/build	100.00%	100.00%
ram_env/start	100.00%	100.00%
ram_test/new	100.00%	100.00%
ram_test/run	0.00%	0.00%
test_write/new	100.00%	100.00%
test_write/run	100.00%	100.00%
test_read/new	100.00%	100.00%
test_read/run	0.00%	0.00%
test_regression/new	100.00%	100.00%
test_regression/run	0.00%	0.00%

Coverage Summary by Type:

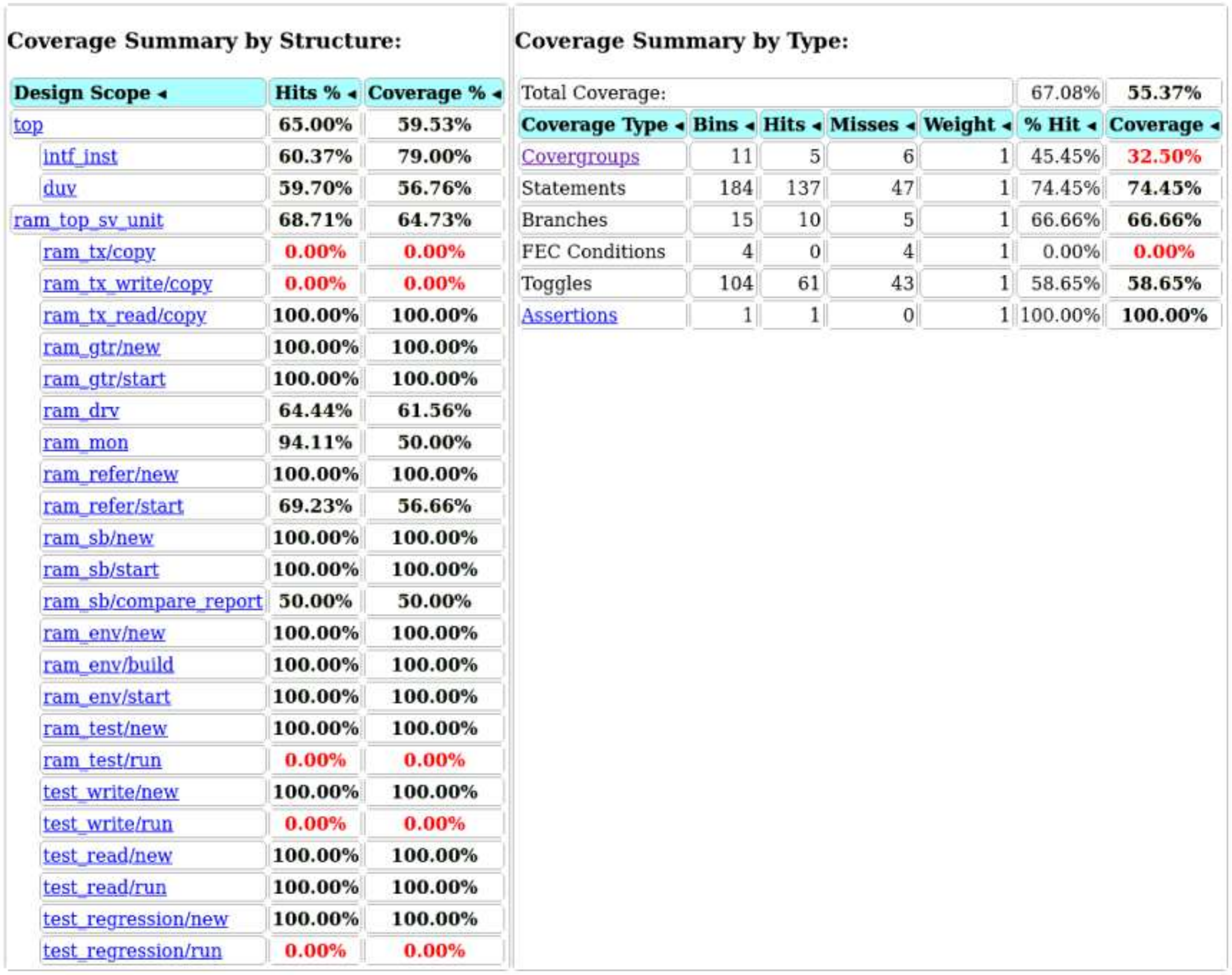
Total Coverage:					68.65%	60.18%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	11	5	6	1	45.45%	32.50%
Statements	184	137	47	1	74.45%	74.45%
Branches	15	10	5	1	66.66%	66.66%
FEC Conditions	4	1	3	1	25.00%	25.00%
Toggles	104	65	39	1	62.50%	62.50%
Assertions	1	1	0	1	100.00%	100.00%

7.1.2 read_with_read_en

Waveform:



Coverage Report:



Coverage Summary by Structure:

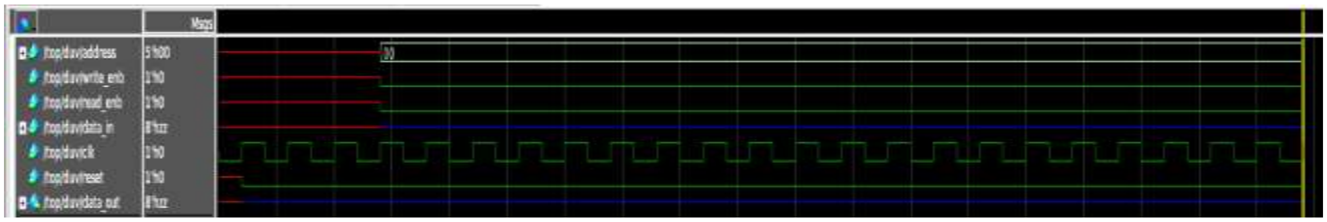
Design Scope ◂	Hits % ◂	Coverage % ◂
top	65.00%	59.53%
intf_inst	60.37%	79.00%
duv	59.70%	56.76%
ram_top_sv_unit	68.71%	64.73%
ram_tx/copy	0.00%	0.00%
ram_tx_write/copy	0.00%	0.00%
ram_tx_read/copy	100.00%	100.00%
ram_gtr/new	100.00%	100.00%
ram_gtr/start	100.00%	100.00%
ram_drv	64.44%	61.56%
ram_mon	94.11%	50.00%
ram_refer/new	100.00%	100.00%
ram_refer/start	69.23%	56.66%
ram_sb/new	100.00%	100.00%
ram_sb/start	100.00%	100.00%
ram_sb/compare_report	50.00%	50.00%
ram_env/new	100.00%	100.00%
ram_env/build	100.00%	100.00%
ram_env/start	100.00%	100.00%
ram_test/new	100.00%	100.00%
ram_test/run	0.00%	0.00%
test_write/new	100.00%	100.00%
test_write/run	0.00%	0.00%
test_read/new	100.00%	100.00%
test_read/run	100.00%	100.00%
test_regression/new	100.00%	100.00%
test_regression/run	0.00%	0.00%

Coverage Summary by Type:

Total Coverage:					67.08%	55.37%
Coverage Type ◂	**Bins ◂**	**Hits ◂**	**Misses ◂**	**Weight ◂**	**% Hit ◂**	**Coverage ◂**
Covergroups	11	5	6	1	45.45%	32.50%
Statements	184	137	47	1	74.45%	74.45%
Branches	15	10	5	1	66.66%	66.66%
FEC Conditions	4	0	4	1	0.00%	0.00%
Toggles	104	61	43	1	58.65%	58.65%
Assertions	1	1	0	1	100.00%	100.00%

7.1.3 read_or_write_with_reset

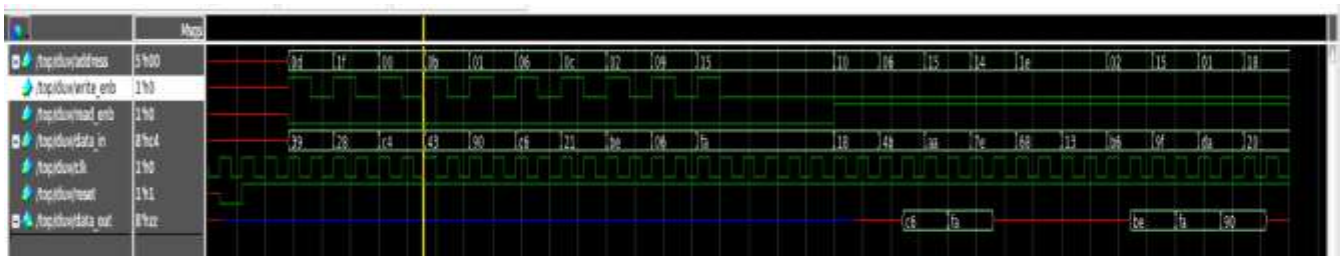
Waveform:



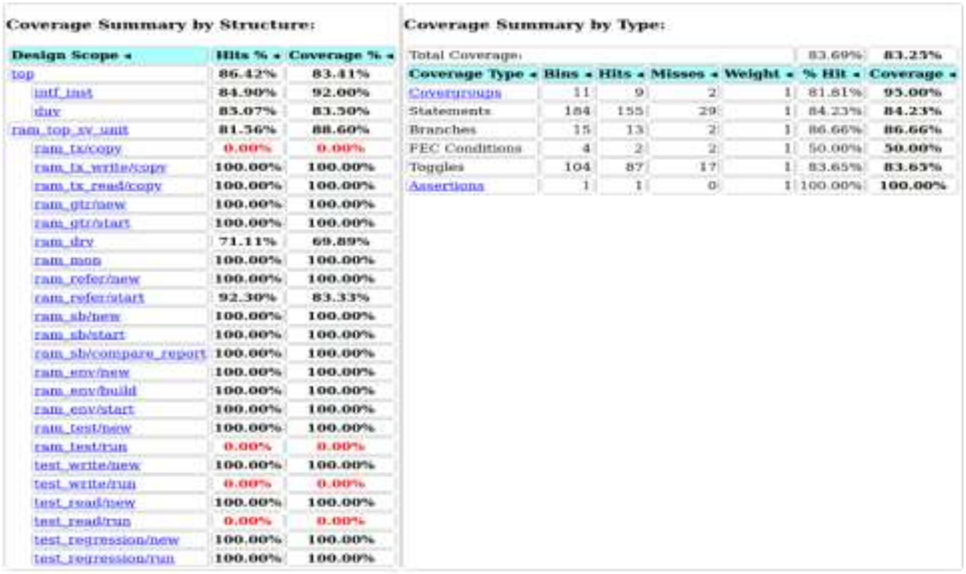
7.2 Regression

Writing and then reading from the RAM

Waveform:



Coverage Report:



Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
top	86.42%	83.41%
intf_inst	84.90%	92.00%
duv	83.07%	83.50%
ram_top_sv_unit	81.56%	88.60%
ram_tx/copy	0.00%	0.00%
ram_tx_write/copy	100.00%	100.00%
ram_tx_read/copy	100.00%	100.00%
ram_gtr/new	100.00%	100.00%
ram_gtr/start	100.00%	100.00%
ram_drv	71.11%	69.89%
ram_mon	100.00%	100.00%
ram_refer/new	100.00%	100.00%
ram_refer/start	92.30%	83.33%
ram_sb/new	100.00%	100.00%
ram_sb/start	100.00%	100.00%
ram_sb/compare_report	100.00%	100.00%
ram_env/new	100.00%	100.00%
ram_env/build	100.00%	100.00%
ram_env/start	100.00%	100.00%
ram_test/new	100.00%	100.00%
ram_test/run	0.00%	0.00%
test_write/new	100.00%	100.00%
test_write/run	0.00%	0.00%
test_read/new	100.00%	100.00%
test_read/run	0.00%	0.00%
test_regression/new	100.00%	100.00%
test_regression/run	100.00%	100.00%

Coverage Summary by Type:

Total Coverage:					83.69%	83.25%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	11	9	2	1	81.81%	95.00%
Statements	184	155	29	1	84.23%	84.23%
Branches	15	13	2	1	86.66%	86.66%
FEC Conditions	4	2	2	1	50.00%	50.00%
Toggles	104	87	17	1	83.65%	83.65%
Assertions	1	1	0	1	100.00%	100.00%