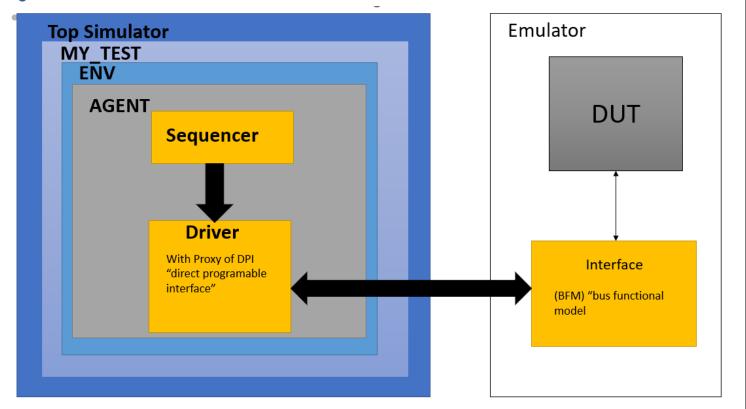


# Contents

Question 1:	3
Question 2:	3
Question 3:	4
Question 4:	5
Top Module of the design:	6
Inputs & output signals of the top module:	6
UVM Environment:	7
Environment Components:	7
Results:	8
1 <sup>st</sup> test:	8
2 <sup>nd</sup> test:	9
3 <sup>rd</sup> Test:10	0
4 <sup>th</sup> test:	1
5 <sup>th</sup> test:	2
6 <sup>th</sup> test:1	3
7 <sup>th</sup> test:	4
Display of the monitor every 10 ns	5
Tcl script used to preload memory1	5
TCL script used to compare final memory values with the expected values and save	
whether they are identical or not in report file:1	6
Notes on the Project & Process:	7

# **Question 1:**



- -Emulator is used to accelerate the UVM testbench so using emulator will saves a lot of time than using the simulation
- -We need to connect the simulator with the emulator and that happens by writing a proxy of a direct programable interface "DPI" and connecting it with BFM interface in the emulator
- -DPI is interface which can connect and initialize connection between System Verilog and any other language

# **Question 2:**

The main criteria of developing the UVM with Emulator is to save time as doing the simulation on FPGA "Emulator" of course will be faster than doing it on CPU "Simulator"

So using Emulator will accelerate the UVM testbench

Also we connect the simulator with the Emulator by SCE-MI interfaces by one of the following 3 ways

- 1-Macro-based: where we send data from simulator to emulator by Macros
- 2-Function-based: use the system Verilog to call RTL functions running in the Emulator
- 3-Pipes-based: use of predefined System Verilog APIs to send the messages to the emulator

# **Question 3:**

1-System Specifications: in this step we decide the specs that we want to achieve in the design before start coding

## 2-Functional Design & Simulation:

- -in this step we start coding the design using HDL languages whether it's VHDL or Verilog, where we will define the behavioral function of the design
- -Then we will start our simulation using testbench or UVM to check the functionality of the design

## 3- Synthesis & Simulation:

- -In this step we turn our design to the gate-level
- -Then we will make logic simulation on the design after gate-level logic

## 4-Technology mapping & Simulation:

-Here we will add the gate delays depending on the technology used and simulate to check the functionality and timing "Setup & hold violations" after adding the gate delays

#### 5-Place & route with Simulation:

- -Here we place the design components from the library and start routing "wiring", adding wires delay
- -Then we start simulation to check that the timing of the design is met after adding wiring delays

## 6- Post-layout simulation "Verification of the design"

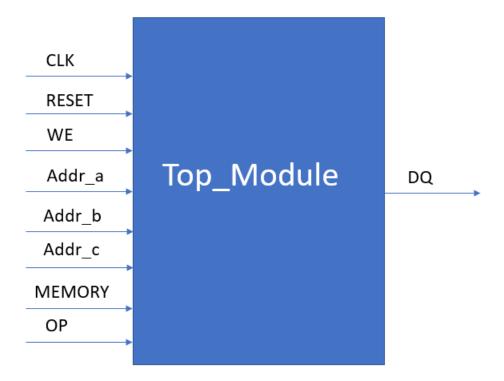
-Before the fabrication we check that the system specs are met

#### 7-Fabrication:

-we could fabricate the design on ASIC "Application specific integrated circuit" or programable devices like: FPGA

Question 4:	
The functional verification process passes by many steps:	
1-set DUT specs and putting verification plan	
2-Creating the testbench environment	
3-writing testcases to test the functionality of the design	
4-creating code coverage and functional coverage modules to measure scenarios, corner cases & the system specifications as the random test and focused test are added to the coverage module then the verification will be more accurate	•

# **Top Module of the design:**



#### **Inputs & output signals of the top module:**

CLK: input signal to drive the clock to the design

RESET: input signal used to reset the design when it's equal 0

WE: write enable signal used to decide whether DQ is input or output (when we=1  $\rightarrow$  DQ is input, when we=0  $\rightarrow$  DQ is output)

Addr a: input signal where the user choose which address A he needs to make

Addr b: input signal where the user chooses which address B he needs to make

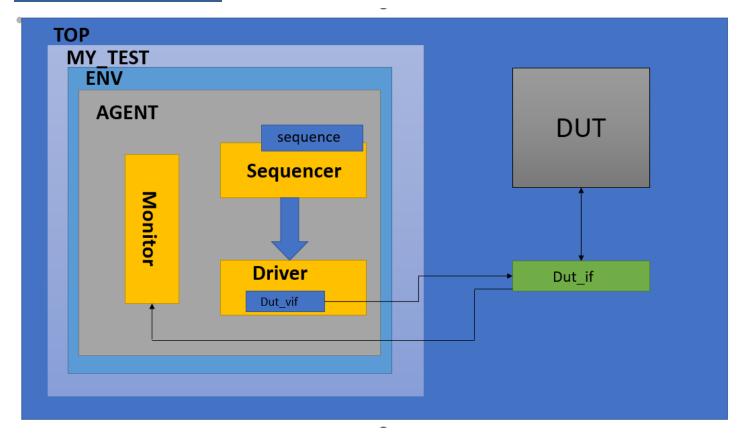
Addr c: input signal where the user chooses which address C he needs to make

MEMORY: The memory of the design which is preloaded by a file produced through a TCL script and it contains 16 locations each one of 16 bit (we could change the size of the memory by the parameters address width & data width in the code)

OP: input signal used to decide which operation we are going to do while WE=0 (when OP=0 →DQ=MEMORY[address C], when OP=1 → MEMORY[address c]= MEMORY[address a]+ MEMORY[address b], when OP=2 → MEMORY[address c]= MEMORY[address a]- MEMORY[address b]

DQ: is inout signal depending on the value of the WE.

## **UVM Environment:**



### **Environment Components:**

**Top:** Contains the testbench package, DUT & interface where it connects the components of the environment to the DUT through the interface also here we preload the memory and intiate the clock

Test package: Contains the environment and here we start the process of each component

**Environment**: Responsible for managing all the components also it initiate the agent and the scoreboard if found

**Agent**: Responsible for connecting the sequencer, the driver & the monitor also it provides ports for the monitor to send the transaction to the coverage and scoreboard modules

**Monitor**: it monitors the transactions seen on the interface

**Sequencer**: Responsible for creating the transactions through different random sequences

**Driver**: Responsible for decoding the transactions taken from the sequencer and driving it to the DUT through the actual and virtual interfaces.

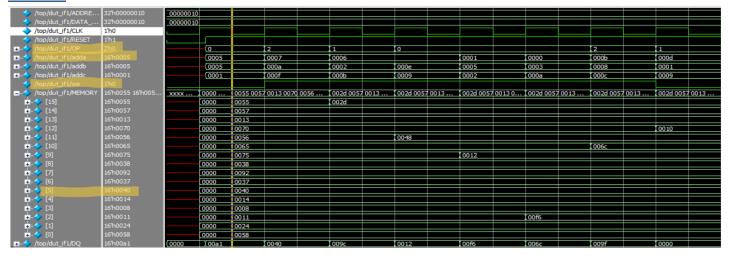
# **Results:**

We firstly set Reset=0 at 5 ns to reset our memory then at 10 ns we start to preload our memory through the input file produced from the TCL script with 16 random values.

Note: values in the simulation are in hexadecimal, we also generate 8 random sequences through the sequencer

- In each test case we have two figures where the first figure show the transaction before the positive edge clock while the second figure show the result of this transaction at the positive edge clock

#### 1st test:



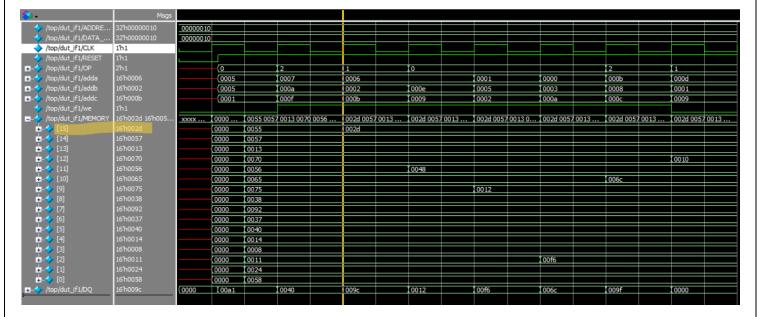


Our first test where in the transaction the WE=0, ADDRESS  $A = 5 \rightarrow$  so in the next posedge CLK we will have DQ=Memory[Address A]=16'h0040 as shown in both figures in the highlighted parts.

Here we will discard the value of OP as the write enable is zero so in this case we will have DQ as an output port and we will not write in the memory

#### 2<sup>nd</sup> test:

<b>≨</b> 1 →	Msgs																	
/top/dut_if1/ADDRE	. 32'h00000010	00000010																
/top/dut_if1/DATA	32'h00000010	00000010																
/top/dut_if1/CLK	1'h1																	
/top/dut_if1/RESET	1'h1																	
+ /top/dut_if1/OP	2'h2		0		2		1		0						2		1	
+			0005		0007		0006				0001		0000		000b		000d	
+	16'h000a		0005		000a		0002		000e		0005		0003		0008		0001	
+			0001		000f		000Ь		0009		0002		000a		000c		0009	
/top/dut_if1/we	1h1																	
=	16'h0055 16'h005	XXXX	0000	0055 005	0013 0070	0056	002d 0057	0013	002d 0057	0013	002d 0057	00130	002d 0057	0013	002d 0057	0013	002d 0057	0013.
<del>. j {</del>	16'h0055		0000	0055			002d											
<u> </u>	16'h0057	I	0000	0057														
<u>+</u> -🔷 [13]	16'h0013	I	0000	0013														
<u>+</u>	16'h0070		0000	0070													0010	
<b></b>	16'h0056		0000	0056					0048									
÷- <b>/&gt;</b> [10]		I	0000	0065											006c			
<u>+</u> <b>/</b> [9]	16'h0075		0000	0075							0012							
<u>+</u> <b>/&gt;</b> [8]	16'h0038		0000	0038														
<b>+-</b> [7]	16'h0092		0000	0092														
<u>+</u> <b>/</b> [6]	16'h0037		0000	0037														
<u>+</u> <b>/</b> [5]	16'h0040	I	0000	0040														
<b></b>	16'h0014		0000	0014														
<b></b>	16'h0008		0000	0008														
<b></b>	16'h0011		0000	0011									00f6					
<b></b>	16'h0024		0000	0024														
<u>+</u>	16'h0058		_	0058														
+	16'h00 <del>4</del> 0	0000	(00a1		0040		1009c		0012		00f6		006c		009f		0000	



Our second test where in the transaction the WE=1, OP=2, ADDRESS A=7, ADDRESS B = 10, ADRESS C=  $15 \rightarrow$  so in the next posedge CLK we will have

MEMORY[Address C]=Memory[Address A]-Memory[Address B]=16'h0092-16'h0065=16'h002d as shown in both figures in the highlighted parts

## 3rd Test:

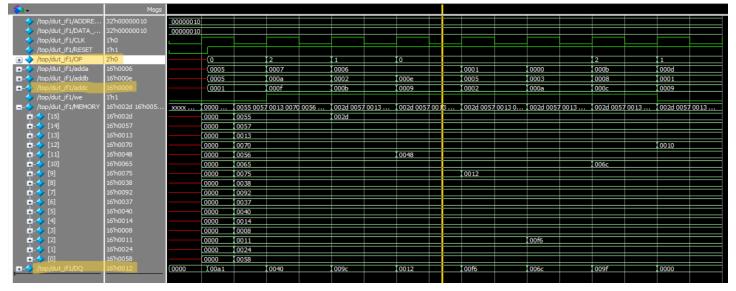
<u>}</u>	Msgs																		
/top/dut_if1/ADDRE	32'h00000010	00000010																	
/top/dut_if1/DATA	32'h00000010	00000010																	
/top/dut_if1/CLK	1'h0							ш											
	1'h1																		
+ /top/dut_if1/OP	2'h1		-(0		2		1			0						2		1	
+	16'h0006		0005		0007		0006					0001		0000		000b		000d	
	16'h0002		0005		000a		0002			000e		0005		0003		8000		0001	
/top/dut_if1/addc	16'h000b		0001		000f		000b		T (	0009		0002		000a		000c		0009	
/top/dut_if1/we	1'h1																		
	16'h002d 16'h005				0013 0070	0056	002d 005	0013	3	002d 0057	0013	002d 0057	0013 0	002d 0057	0013	002d 005	0013	002d 0057	0013
	16'h002d		0000	0055			002d	4											
	16'h0057		0000	0057				H	_										
	16'h0013		0000	0013				┷	=										
	16'h0070		0000	0070					$\Rightarrow$									0010	
	16'h0056		0000	0056						0048									
T	16'h0065 16'h0075		0000	0065					=			2242				006c			
	16'h0075 16'h0038		0000	0075					=			0012							
T 7 1 1	16'h0038 16'h0092		0000	0038					=										
[6]	1610092 16'b0027		0000	10092					=										
- V	16'h0040		0000	10037					=										
	16'h0014		0000	10014															
- V	16'h0008		0000	10008															
[2]	16'h0011		0000	10011										00f6					
	16'h0024		0000	10024															
	16'h0058		0000	0058															
		(0000	(00a1		0040		I 009c			0012		00f6		006c		009f		0000	

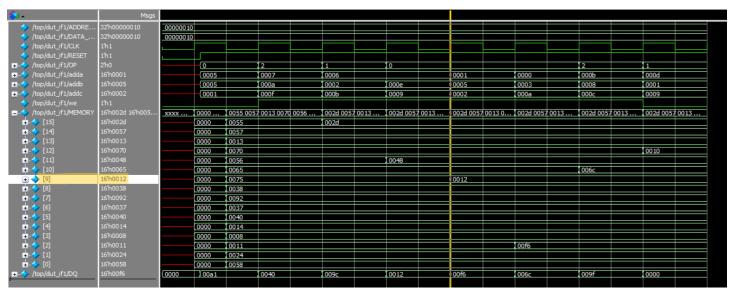
<b>ૄ</b>	Msgs																	
/top/dut_if1/ADDRE	32'h00000010	00000010																
/top/dut_if1/DATA	32'h00000010	00000010																
/top/dut_if1/CLK	1h1																	
/top/dut_if1/RESET	1h1		li—															
+/> /top/dut_if1/OP	2'h0		0		2		1		0						2		1	
<b>∓</b> - <b>/</b> /top/dut_if1/adda	16'h0006		0005		0007		0006				0001		0000		000b		000d	
<b>∓</b> - /top/dut_if1/addb	16'h000e		0005		000a		0002		000e		0005		0003		0008		0001	
	16'h0009		0001		000f		000b		0009		0002		000a		000c		0009	
/top/dut_if1/we	1'h1																	
-/ /top/dut_if1/MEMORY	16'h002d 16'h005	XXXX	0000	0055 0057	0013 0070 0	056	002d 005	0013	002d 0057	0013	002d 0057	0013 0	002d 005	0013	002d 0057	0013	002d 0057	0013
±- <b>/&gt;</b> [15]	16'h002d		0000	0055			002d											
<b>-</b> d-√ [14]	16'h0057		0000	0057														
<u>+</u>	16'h0013		0000	0013														
<u>+</u> -4 [12]	16'h0070		0000	0070													0010	
<b>n</b> - <b>4</b> [11]	16'h0048		0000	0056					0048									
+	16'h0065		0000	0065											006c			
<u>+</u>	16'h0075		0000	0075							0012							
<u>+</u>	16'h0038		0000	0038														
<u>+</u> - <b>(</b> 7]	16'h0092		0000	0092														
<u>+</u> -🔷 [6]	16'h0037		0000	0037														
<u>+</u> - <b>/&gt;</b> [5]	16'h0040		0000	0040														
<u>+</u> - <b>♦</b> [4]	16'h0014		0000	0014														
<u>+</u>	16'h0008		0000	10008														
<u>+</u> -4 [2]	16'h0011		0000	0011									00f6					
<u>+</u> -4 [1]	16'h0024		0000	0024														
<u>+</u> > [0]	16'h0058		0000	0058														
→ /top/dut_if1/DQ	16'h0012	(0000	(00a1		0040		009c		0012		00f6		006c		009f		0000	

Our third test where in the transaction the WE=1 , OP=1 ,ADDRESS A=6, ADDRESS B = 2, ADRESS C=  $11 \rightarrow$  so in the next posedge CLK we will have

MEMORY[Address C]=Memory[Address A]+Memory[Address B]=16'h0037

+16'h0011= 16'h0048 as shown in both figures in the highlighted parts

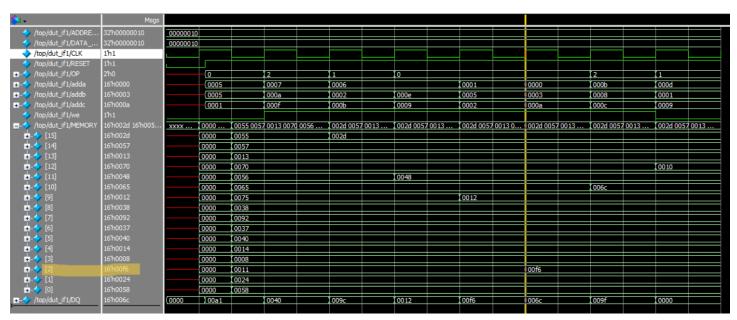




Our fourth test where in the transaction the WE=1 , OP=0 , ADRESS C= 9, DQ=16'h0012  $\rightarrow$  so in the next posedge CLK we will have

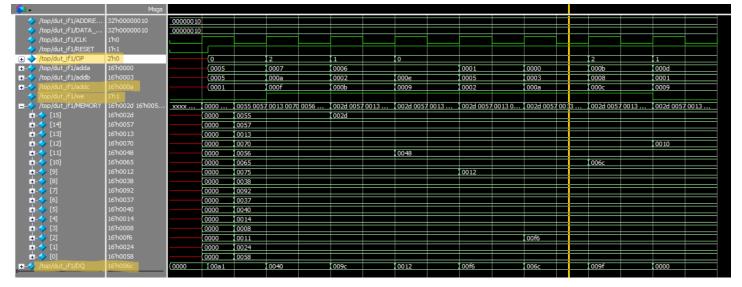
MEMORY[Address C]=DQ =16'h0012 as shown in both figures in the highlighted parts

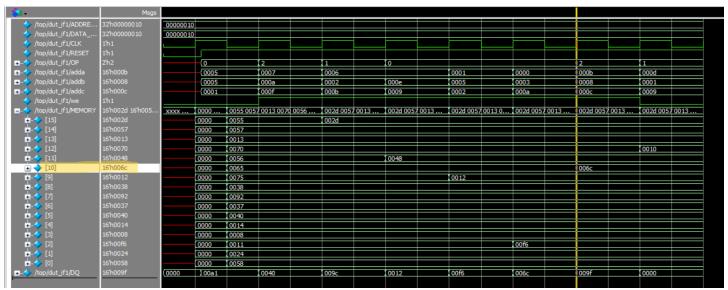
<b>≨1</b> •	Msgs																		
/top/dut_if1/ADDRE	32'h00000010	00000010																	
<pre>/top/dut_if1/DATA</pre>	32'h00000010	00000010																	
/top/dut_if1/CLK	1'h0																		
/top/dut_if1/RESET	1'h1		╚																
+	2'h0		0		2		1		0							2		1	
	16'h0001		0005		0007		0006				0001			0000		000b		000d	
	16'h0005		0005		000a		0002		000e		0005			0003		8000		0001	
+	16'h0002		0001		000f		000Ь		0009		0002		_	000a		000c		0009	
7	1'h1												_						
- Y '. '' - '	16'h002d 16'h005			1	7 0013 0070	0056		0013	002d 005	7 00 13	002d 0057	00131	D I	002d 0057	0013	002d 0057	0013	002d 0057	0013
[15]	16'h002d		0000	0055			002d						=						
	16'h0057 16'h0013		0000	0057									=						
	16'h0070		0000	0013									=					10010	
T X : : :	16'h0048		0000	0070					00.40				=					0010	
I A 5.03	16'h0065		0000	0056					0048				=			1006c			
	16'h0012		0000	10005							0012		=			, 000C			
[8]	16'h0038		0000	10038							10012		=						
±- <b>4</b> [7]	16'h0092		0000	0092									=						
	16'h0037		0000	0037									=						
	16'h0040		0000	10040									=						
	16'h0014		0000	10014									$\equiv$						
	16'h0008		0000	0008															
	16'h0011		0000	0011										00f6					
<u>+</u> -4 [1]	16'h0024		0000	0024															
<u>+</u> - <b>♦</b> [0]	16'h0058		0000	0058															
+/top/dut_if1/DQ	16'h00f6	0000	00a1		0040		009c		0012		00f6			006с		009f		10000	



Our fifth test where in the transaction the WE=1 , OP=0 , ADRESS C= 2 , DQ=16'h00f6  $\rightarrow$  so in the next posedge CLK we will have

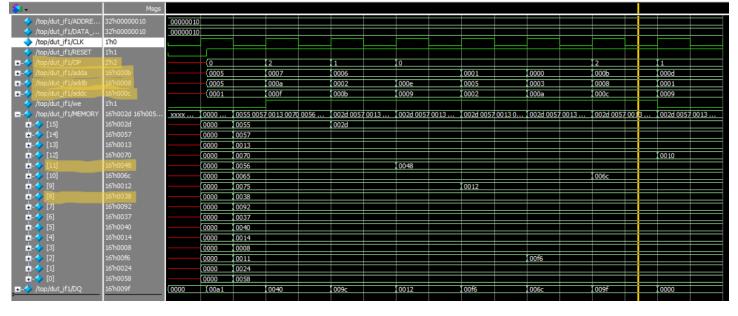
MEMORY[Address C]=DQ =16'h00f6 as shown in both figures in the highlighted parts

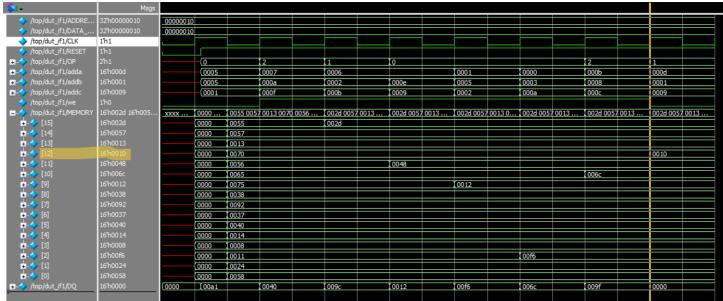




Our sixth test where in the transaction the WE=1 , OP=0 , ADRESS C= 10 , DQ=16'h006c  $\rightarrow$  so in the next posedge CLK we will have

MEMORY[Address C]=DQ =16'h006c as shown in both figures in the highlighted parts





Our third test where in the transaction the WE=1, OP=2, ADDRESS A=11, ADDRESS B = 8, ADRESS C=  $12 \rightarrow$  so in the next posedge CLK we will have

MEMORY[Address C]=Memory[Address A]-Memory[Address B]=16'h0048

-16'h0038=16'h0010 as shown in both figures in the highlighted parts

#### Display of the monitor every 10 ns

```
UVM_INFO monitor.sv(43) @ 5: uvm_test_top.env.agent.m0 [MON] Saw address A = 0 Saw address B = 0 Saw address C
                                                                                                                          [MON] Saw OPERATION = 0
                                                                                                                                                                                Saw write_enable = 0 Saw DQ = 0
 UVM_WARNING D:/uvm_examples/my_testbench_pkg.sv(86) @ 10: uvm_test_top [] Hello World!
UVM_INFO monitor.sv(43) @ 15: uvm_test_top.env.agent.m0 [MON] Saw address A = 5 Saw address B = 5 Saw address C = 1
  UVM_INFO monitor.sv(44) @ 15: uvm_test_top.env.agent.m0 [MON] Saw OPERATION = 0
                                                                                                                                                                                 Saw write_enable = 0 Saw DQ = 161
  UVM_INFO monitor.sv(43) @ 25: uvm_test_top.env.agent.m0 [MON] Saw address A = 7
                                                                                                                                                                                 Saw address B = 10 Saw address C = 15
  UVM_INFO monitor.sv(44) @ 25: uvm_test_top.env.agent.m0
                                                                                                                            [MON] Saw OPERATION = 2
                                                                                                                                                                                Saw write_enable = 1 Saw DQ = 64
                                                      @ 35: uvm_test_top.env.agent.m0
                                                                                                                                                                                 Saw address B = 2 Saw address C = 11
  UVM_INFO monitor.sv(44) @ 35: uvm_test_top.env.agent.m0 [MON] Saw OPERATION = 1
                                                                                                                                                                                 Saw write_enable = 1 Saw DQ = 156
  UVM_INFO monitor.sv(43)
                                                      @ 45: uvm_test_top.env.agent.m0
                                                                                                                            [MON] Saw address A = 6
                                                                                                                                                                                 Saw address B = 14
                                                                                                                                                                                                                           Saw address C = 9
                                                      @ 45: uvm_test_top.env.agent.m0
                                                                                                                             [MON] Saw OPERATION = 0
                                                                                                                                                                                 Saw write_enable = 1 Saw DQ = 18
  UVM_INFO monitor.sv(43) @ 55: uvm_test_top.env.agent.m0 [MON] Saw address A = 1
                                                                                                                                                                                 Saw address B = 5 Saw address C = 2
  UVM_INFO monitor.sv(44) @ 55: uvm_test_top.env.agent.m0 [MON] Saw OPERATION = 0
                                                                                                                                                                                 Saw write_enable = 1 Saw DQ = 246
                                                      @ 65: uvm_test_top.env.agent.m0
 UVM_INFO monitor.sv(44) @ 65: uvm_test_top.env.agent.m0 [MON] Saw OPERATION = 0 Saw write_enable = 1 Saw DQ = 108
UVM_INFO monitor.sv(43) @ 75: uvm_test_top.env.agent.m0 [MON] Saw address A = 11 Saw address B = 8 Saw address C =
                                                                                                                                                                                                                            Saw address C = 12
                                                      @ 75: uvm_test_top.env.agent.m0
                                                                                                                            [MON] Saw OPERATION = 2 Saw write_enable = 1 Saw DQ = 159
 UVM_INFO monitor.sv(43) @ 85: uvm_test_top.env.agent.m0 [MON] Saw address A = 13 Saw address B = 1 UVM_INFO monitor.sv(44) @ 85: uvm_test_top.env.agent.m0 [MON] Saw OPERATION = 1 Saw write_enable =
                                                                                                                                                                                                                           Saw address C = 9
 OVM_INFO monitor.sv(44) @ 85: uvm_lest_top.env.agent.mo [MON] Saw dutiess = 15 Saw address 
--- UVM Report Summary ---
  ** Report counts by severity
UVM INFO :
 UVM_WARNING :
UVM_ERROR : 0
```

Here the monitor display every signal every 10 ns by uvm\_info command also we could display the memory here

# Tcl script used to preload memory

puts "\*\*\*\*\*\*\* Scripting file using TCL to Preload Memory of 16 random values\*\*\*\*\*\*\*\*

```
set fh [open input_file w+ ]
for {set x 0} {$x<16} {incr x} {
set MEMORY($x) [exp int ([exp rand()*100])]
puts $fh $MEMORY($x)
}
close $fh</pre>
```

- We also print output file of the last memory values by \$fdisplay command in the testbench code

# TCL script used to compare final memory values with the expected values and save whether they are identical or not in report file:

```
puts "****Comparing 2 files****"
set fh_0 [open expectedoutput_file.txt r]
set fh_1 [open output_file r]
set fh 2 [open report.txt w+]
set file data 0 [read $fh 0]
set file data 1 [read $fh 1]
set x [string compare $file_data_0 $file_data_1]
if \{x=0\}
set str "Both outputs are identical"
puts "Both outputs are identical"} else {
set str "Both outputs are different"
puts "Both outputs are different"}
puts $fh 2 $str
close $fh 0
close $fh 1
close $fh 2
```

## **Notes on the Project & Process:**

- -The process in this position was a very good learning opportunity as I was interested in studying the verification methodologies because it was a missing part in the digital design flow that I haven't learn it yet.
- -So that was my first UVM project that was very challenging and it also expands my mind to invest my time in completing studying the UVM and its connectivity with the emulation.
- -All of the report and the code of the project are due to self-education & searching over the internet without the help of anyone in writing the code or explaining commands.
- -Future work on this project: Adding the coverage module and scoreboard module, but in this project, we have covered 8 random different cases including the 4-corner case also we have done a comparison between the final output of the memory and the expected final memory using the TCL Scripting