Concordia University
Engineering and Computer Science

COEN 6541 Project Report

Winter 2021

Supervised by Dr. Otmane Ait Mohamed

Group 2:

Divyaa Mahalakshmi Guruswamy [40167923]

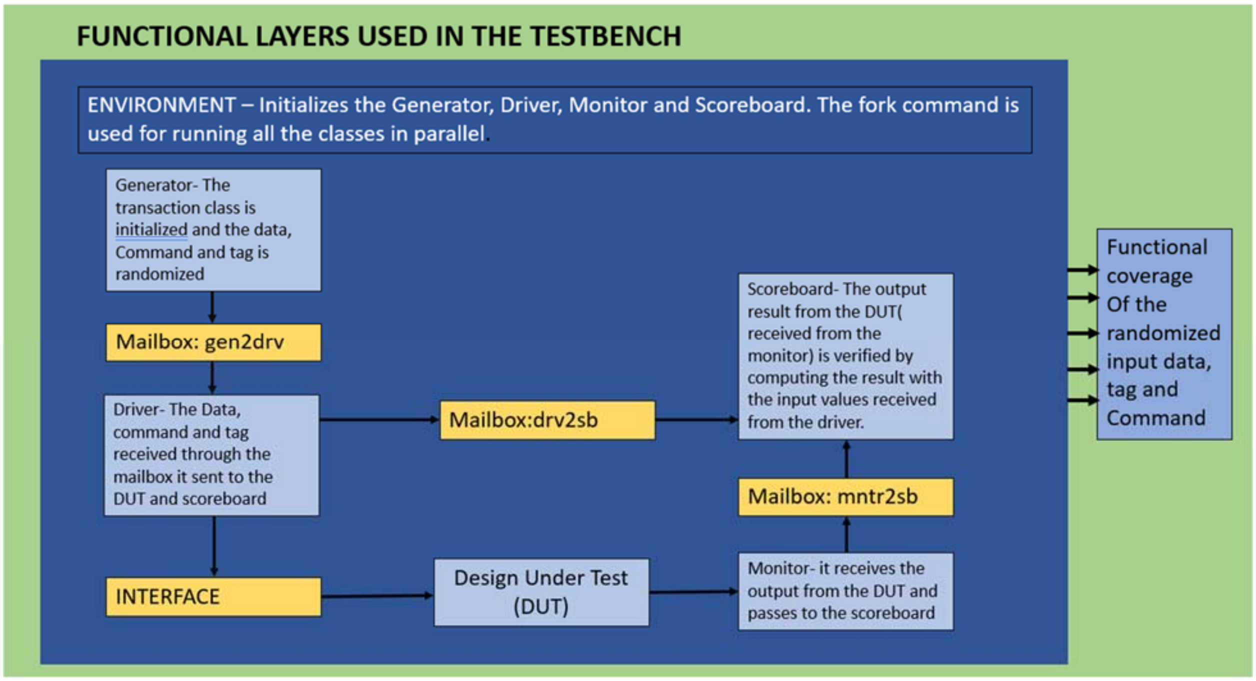
Chirag Jhamb [40169876]

Abdul Rahman Koleilat [40086025]

Contents

INTRODUCTION

The layered testbench as shown in the fig connects various classes to verify the DUT. Breaking the system into different classes helps in reusing the blocks and adding new transactors without changing the other classes. The transactors are passed between the classes, with help of the mailbox. The environment holds all these classes and executes them in parallel.


FUNCTIONAL LAYERS USED IN THE TESTBENCH
ENVIRONMENT – Initializes the Generator, Driver, Monitor and Scoreboard. The fork command is used for running all the classes in parallel
Generator- The transaction class is initialized and the data, Command and tag is randomized
Mailbox: gen2drv
Driver- The Data, command and tag received through the mailbox it sent to the DUT and scoreboard
Mailbox:drv2sb
Scoreboard- The output result from the DUT(received from the monitor) is verified by computing the result with the input values received from the driver.
Mailbox: mntr2sb
INTERFACE
Design Under Test (DUT)
Monitor- it receives the output from the DUT and passes to the scoreboard
Functional coverage Of the randomized input data, tag and Command

The major classes used in the testbench are as follows:

- Transaction Class
- Generator Class
- Driver Class
- Monitor Class
- Scoreboard Class

Transaction

The Transaction class contains all the input variables that are created by the generator and used by the driver to drive the DUT as well as the output variables that are sampled from the DUT by the monitor. The input variables are actually cmd1 which is the input command in the first clock cycle, cmd2 which is the input command in the second clock cycle, data1 which is the input data in the first clock cycle, data2 which is the input data in the second clock cycle, and tag which is the input tag. As for the output variables, they are out_data, output_resp, and out_tag which are values sampled by the monitor. This class will also print the transaction depending on whether it's coming from the driver or the monitor.

Generator

The Generator class creates the transaction arrays for each of the four ports. These transaction arrays have a size between one and four which is generated randomly (if the number is 0 then it will be a transaction with an input command of No_Op and will be given a tag "01") and are filled with transactions that are randomized and then the array is sent to the driver class with help of the mailboxes (gen2drv1, gen2drv2, gen2drv3, gen2drv4), one for each port. The generator will wait for the monitor to be done until it proceeds to the next iteration. It will keep on generating transaction arrays until the maximum amount is reached.

Driver

The Driver class obtains the randomized transaction array from the generator and passes it to the DUT as input. The transaction array received is broken down and the interface is used to send these data to the DUT to stimulate the execution of the test device. This transaction array is also sent to the scoreboard class with help of the mailboxes (drv2sb1, drv2sb2, drv2sb3, drv2sb4), one for each port.

Monitor

The Monitor will sample the output of the DUT. It receives the size of the transaction array in order to know the number of output transactions to expect. It will keep on looping until one of the values of the output signals of the interface (intf.out_tag, intf.out_data, intf.out_resp) is not zero for each port and the loop has a maximum number of 10 clock cycles. If the first output is detected and the number of outputs to expect is more than one, then it will keep on looping until the output with a different tag is detected and is given a maximum number of 5 clock cycles. When an output is detected, it is added to a dynamic array of transactions that is sent to the scoreboard with the help of mailboxes (mntr2sb1, mntr2sb2, mntr2sb3, mntr2sb4), one for each port. After that, the monitor done event is triggered so the generator can proceed to the next iteration.

Scoreboard

The Scoreboard will receive a transaction array that contains the generated input transactions for each port from the driver through the mailboxes we mentioned earlier as well as a transaction array that contains the sampled output transactions for each port from the monitor through the corresponding mailboxes. The Scoreboard will check the size of the arrays in order to know if there are any missing transactions, then it will check the order of the add\sub transactions and the shift transactions. After that, it will print both the arrays and for the driver transactions, if the input signals have certain values, it will print the specific test case it belongs to. Then, the transactions with the same tag from the driver transaction array and the monitor transaction array will be compared with each other. The expected output data for each operation will be calculated and then compared to the actual output data from the monitor. Also, the expected output response is compared with the actual one. If the values are not equal an error message will be printed and the error count is incremented, otherwise, a correct message will be printed and the correct count is incremented.

TEST PLAN

The below are the various test cases verified randomly, by varying the inputs with constrains. The number of inputs to each port ranges from 1 to 4. Giving a maximum 16 outputs in few cases.
In all the corner cases, the randomized data is given each corner case data have a particular weightage, according to the constraints (mentioned in the next section).

Specification /Feature	**Referenc e**	**Test Points**	**Test Scenarios**	**Expected Results**
Operations (in all the 4 ports)	**All the below conditions were verified with varying the number of inputs from 0-4 in each port and providing a tag value for each of them**			
Add				
Basic addition operation the inputs given is varied from 1-4 in each port Getting maximum of 16 outputs (4 in 4 ports)	1.1, 1.2	To check the basic addition operation without overflow	The input command is given as ‘0001’b to the req(x)_cmd_in bus and the operand1 value is provided to the req(x)_data_in input bus in the first cycle. In the second cycle, the operand2 value is given to the req(x)_data_in input bus and input command is given as no operation. In this case, the output computation value does not exceed the maximum output value range.	The out_res(x) is '01'b implying a successful response. The addition of operand1 and operand2 takes place and the output value is obtained in the out_data(x) bus.

addition operaration resulting in a overflow output value (given randomly to the ports using constrains)	1.1,1.3	To verify the addition operation with overflow	In the first cycle, The operand1 value and input command of '0001'b is given to the req(x)_data_in and req(x)_cmd_in bus, respectively. The operand2 input value is given to the req(x)_data_in in the second cycle. In this case, the output computation value exceeds the maximum output value range.	The out_res(x) is '10'b implying a overflow response.
Data dependent corner case with overflow	2.4.1	To verify the output of adding two numbers that results in a overflow by 1	In the first cycle, The req(x)_data_in is driven with a input value of “FFFFFFFF" as operand1 and req(x)_cmd_in is provided with the value of '0001'b for addition operation. In the second cycle, The req(x)_data_in is driven with a input value of 1 with no operation as input command.	The out_res(x) is '10'b implying a overflow response.
Data dependent corner case without overflow (randomly generated, number of inputs varying from 1-4 in each port)	2.4.2	To verify the output of adding two numbers that result in a output sum value of “FFFFFFFF"	The req(x)_cmd_in bus receives a value of ‘0001’b for addition operation. The operand1 and operand2 values are provided in such a way that the output sum is “FFFFFFFF".	The out_res(x) is '01'b implying a successful response. The addition of operand1 and operand2 takes place and the output value of “FFFFFFFF" is obtained in the out_data(x) bus.
adding 2 maximum numbers		To check the additionesult when 2 operands have maximum value	the input command is addition and both the operands values are “FFFFFFFF"	The out_res(x) is '10'b implying a overflow response.
Adding max number with min number, (test case randomly generated)	2.4.2	To check the addition result when 1 operand has maximum value and other has minimum value	the input command is addition. The operand1 takes a value of “FFFFFFFF" and operand2 takes a value of "00000000" in the req(x)_data_in port, in their respective cycle.	The out_res(x) is '01'b implying a successful response. The addition of operand1 and operand2 takes place and the output value of “FFFFFFFF" is obtained in the out_data(x) bus.

adding 2 minimum numbers		To check the addition result when 2 operands have minimum values	the input command is addition and both the operands values are “00000000"	The out_res(x) is a success and the output result is "00000000"
Subtract				
Basic subtraction operation	1.1, 1.2	To check the basic subtraction operation without overflow	The input command is given as ‘0010’b to the req(x)_cmd_in bus and the operand1 value is provided to the req(x)_data_in input bus in the first cycle. In the second cycle, the operand2 value is given to the req(x)_data_in input bus. In this case, the value of operand2 does not exceed the operand 1 value.	The out_res(x) is '01'b implying a successful response. The subtraction of operand2 from operand1 takes place and the output value is obtained in the out_data(x) bus.
Subtraction operaration resulting in a underflow output value	1.1, 1.3	To verify the subtraction operation with underflow	In the first cycle, The operand1 value and input command of '0010’b is given to the req(x)_data_in and req(x)_cmd_in bus, respectively. The operand2 input value is given to the req(x)_data_in bus in the second cycle. In this case, the operand2 value is more than operand1 value.	The out_res(x) is '10'b implying a underflow response.
Data dependent corner case with underflow	2.4.4	To verify the output of adding two numbers that results in a underflow by 1	The req(x)_cmd_in is driven with a input value of '0010’b for subtraction operation and req(x)_data_in is provided with the operand 1 value in the first cycle. In the second cycle, The operand 2 value is given to the req(x)_data_in bus in such a way that it exceeds the operand1 value by 1 .	The out_res(x) is '10'b implying a underflow response.
subtraction of 2 minimum numbers		To check the subtraction result when 2 operands have minimum values	the input command is subtraction and both the operand values are “00000000"	The out_res(x) is a success and the output result is "00000000"

subtraction of 2 maximum numbers		To check the subtraction result when 2 operands have maximum values	the input command is subtraction and both the operand values are “FFFFFFFF"	The out_res(x) is a success and the output result is "00000000"
Subtracting a max number from a min number	2.4.2	To check the addition result when operand1 has minimum value and operand 2 has a maximum value and other has	the input command is subtraction(value of ‘0010’b) . The operand1 takes a value of "00000000" and operand2 takes a value of “FFFFFFFF" in the req(x)_data_in port, in their respective cycle.	The out_res(x) is '10'b implying a underflow response as operand2 is greater than operand1.
Shift left				
Basic shift operation	1.1, 1.2	To check the basic left shift operation	The req(x)_cmd_in bus is driven with a value of ‘0101' b for left shift operation and the operand1 value is provided at the req(x)_data_in input bus in the first cycle. In the second cycle, the operand2 value is given to the req(x)_data_in input bus. In this case, the value of operand2 does not exceed the decimal value of 31.	The out_res(x) is '01'b implying a successful response. The out_data(x) has the result of operand1 left shifted by the number of bits mentioned in by operand2 value.
Data dependent corner case shifting 0 places	2.4.5	To verify the result when operand1 is left shifted by 0 places	The input command is given as ‘0101’b to the req(x)_cmd_in bus for left shift operation and the operand1 value is provided to the req(x)_data_in input bus in the first cycle. In the second cycle, req(x)_cmd_in receives a no operation command and the req(x)_data_in input bus receives a value of "00000000".	The out_res(x) is a successful response indicated by the value '01'b . The out_data(x) result is same as operand1 as it is left shifted by 0 bits.
Data dependent corner case shifting 31 places	2.4.6	To verify the result when operand1 is left shifted by 31 places	The input command is given as ‘0101’b to the req(x)_cmd_in bus for left shift operation and the operand1 value is provided to the req(x)_data_in input bus in the first cycle. In the second cycle, req(x)_data_in input bus	The out_res(x) is a successful response indicated by the value '01'b . The out_data(x) results in "00000000" as the operand1 is left shifted by maximum

			receives a value of "0000001F".	allowable shifting places i.e 31 bits.
Ignoring high order 27 bits in operand2	2.3	To ensure that the MSB 27 bits are not considered in operand2 for left shifting	The req(x)_cmd_in is given as '0101'b for the left shift operation and the operand1 value is given in the req(x)_data_in port.The operand2 is driven with input value greater than 31(decimal). In this case, few of the high order 27 bits are also asserted with 1.	The out_res(x) is a successful response indicated by the value '01'b . In the result, the operand is only left shifted by the decimal value obtained from the lower 5 bits of operand2, rest of the higher bits are ignored.
Shift Right				
Basic shift operation	1.1, 1.2	To check the basic right shift operation	The req(x)_cmd_in bus is driven with a value of '0110' b for right shift operation and the operand1 value is provided at the req(x)_data_in input bus in the first cycle. In the second cycle, the operand2 value is given to the req(x)_data_in input bus. In this case, the value of operand2 does not exceed the decimal value of 31.	The out_res(x) is '01'b implying a successful response. The out_data(x) has the result of operand1 right shifted by the number of bits mentioned in by operand2 value.
Data dependent corner case shifting 0 places	2.4.5	To verify the result when operand1 is right shifted by 0 places	The input command is given as '0110' b to the req(x)_cmd_in bus for right shift operation and the operand1 value is provided to the req(x)_data_in input bus in the first cycle. In the second cycle, req(x)_data_in input bus receives a value of "00000000".	The out_res(x) is a successful response indicated by the value '01'b . The out_data(x) result is same as operand1 as it is right shifted by 0 bits.
Data dependent corner case shifting 31 places	2.4.6	To verify the result when operand1 is right shifted by 31 places	The input command is given as '0110'b to the req(x)_cmd_in bus for right shift operation and the operand1 value is provided to the req(x)_data_in input	The out_res(x) is a successful response indicated by the value '01'b . The out_data(x) results in "00000000" as the

			bus in the first cycle. In the second cycle, req(x)_data_in input bus receives a value of "0000001F".	operand1 is right shifted by maximum allowable shifting places i.e 31 bits.
Ignoring high order 27 bits in operand2	2.3	To ensure that the MSB 27 bits are not considered in operand2 for right shifting	The req(x)_cmd_in is given as '0110'b for the right shifting operation and the operand1 value is given in the req(x)_data_in port.The operand2 is driven with input value greater than 31(decimal). In this case, few of the high order 27 bits are also asserted with 1.	The out_res(x) is a successful response indicated by the value '01'b . In the result, the operand is only right shifted by the decimal value obtained from the lower 5 bits of operand2, rest of the higher bits are ignored.
Dirty state (In all the cases mentioned below the inputs to the port are varied from 1 to 4 (randomly) and the outputs are verified for the different cases)				
addition followed by subtraction	2.1.1	To verify the result when a addition command is followed by a subtraction command	In the first cycle, the Input commad is given as addition and the operand1 value is provided to the req(x)_data_in port. In the second cycle, the input command is given as subtraction instead of no operation and operand2 value is provided to the data in port.	The out_res(x) should be a successful response and the result output data should be the addition of the two operands. The command given in the second cycle should not affect the command given in the first cycle.
Shift left followed by shift right	2.1.1	To verify the result when a shift left command is followed by a shift right command	In the first cycle, the Input commad is given as shift left and the operand1 value is provided to the req(x)_data_in port. In the second cycle, the input command is given as shift right instead of no operation and operand2 value is provided to the data in port.	The out_res(x) should be a successful response and the result output data should be the shifted left by the number of bits mentioned by the operand2. The command given in the second cycle should not affect the command given in the first cycle.

Across all ports, Addition followed by subtraction	2.1.2	To verify the result in all the four ports when a addition command is followed by a subtraction command	the input command of addition and input data as operand1 is given to all the ports concurrently, in the first cycle. The command is subtraction instead of no operation and operand2 data is provided in the second cycle.	The output response should be a success in all the ports and the addition of both the operands should be obtained at the output data result. The second cycle command does not affect the operation
Across all ports, Shift left followed by shift right	2.1.2	To verify the result in all the four ports when a addition command is followed by a subtraction command	the input command of shift left and input data as operand1 is given to all the ports concurrently, in the first cycle. In the second cycle, the input command is given as shift right instead of no operation and operand2 values is provided to the data in ports.	The out_res of all the ports should be a success. The result output data in all the port should be the shifted left by the number of bits mentioned by the operand2 values. The command given in the second cycle should not affect the command given in the first cycle.
invalid data	2.5	To check the response when invalid data is provided	One or both of the input command is given as invalid data	The data which is invalid is ignored.
invalid commands	3.1	To check when invalid commands are given in input	The input command is not given from the command table for operation.	The output response '10' showing that it is an invalid command.
Output response when operand is not mentioned(No-op condition)	3.2	To check when the operand is not given	The input command in both the cycles is 4'b 0000	there should not be any output response produced
Reset	3.3	To check the reset function	Drive the reset input (0:7) but setting it to '11111111'b and hold it for seven cycles	All the input busses except the the clk is set to zero
Testing for the pipelined ADD/Sub and Shift left and Right				
sequence of output		To verify the sequence of outputs for the	The array of inputs is given to the DUT	The output order of the add/ Sub is expected to be the

		pipelined ADD/Sub and Shift left/Right		same as it uses one pipeline And the order of shift left / right Is expected to be as it was given the input
TAG verification		To check the output tag with input tags	The input command and results are given each with a tag value. And the output Tags are obtained	The Tag is used for verifying the results, hence the Tag values obtained in the output are expected to be same as inputs(can be out of order)

CONSTRAINTS GIVEN TO INPUT DATA AND COMMANDS

data1_c (First cycle Data) :

For the input data which is given in the first cycle, minimum value is (00000000) and maximum value is (ffffffff) for the data inputs to each port (a weightage of 20% is given to each of these values. These corner conditions are check in most cases to verify the output data and response. The other corner condition values include 32'h00000001,32'hFFFFFFFE (given a weightage of 20% to each) and 32'h0000001F has 10% weightage. All other input data in the range of [32'h00000002:32'hFFFFFFFD] have 10% weightage. These conditions are given to all the ports, for the first cycle data input.

Data2_c (second cycle Data) :

In the second cycle the data input to all the ports have the following constraints:

32'h00000000(min)=20%,

32'h00000001(corner condition)=20%

32'h0000001F(corner condition)=10%

32'hFFFFFFFF(max):=20%

32'hFFFFFFFE(corner condition)=20%

Other values in range of [32'h00000002:32'hFFFFFFFD] has 10 %.

cmd1_c (First cycle command):

command 1(add):=22%, command 2(sub):=22% , command 5(shift left):=22%, command 6(shift Right):=22% , other commands[3:4]:=2% ,[7:15]:=10%(invalid case)

cmd2_c (Second cycle command):

command 0(add):=90% (for proper execution of the operation) command [1:15] =10% (dirty state)

BUGS DETECTED IN THE DUT

The following were the few bugs detected while testing the DUT with random input data, commands and tag:

- When two minimum values were added in port 2 (with 3 other inputs), the obtained response was 10 whereas it is expected to have a success response. (Test case 1)
- When adding min with max numbers the output obtained is incorrect for all the ports but the response is correct for port 1 and 2 (Actual: 01, Expected: 01) .(Test case 2)
- In test case 3, for overflow (add max with 1) condition is not giving a proper output for port 1 and 2(along with 3 other inputs simultaneously given to the ports)
- In the case of adding 2 max numbers for port for the output response is 01 but is expected to have overflow response of (10) in ports 2,3 and 4. .(Test case 4)
- While subtracting the 2 minimum values (00000000) the expected output data is obtained as zero, but the output response is 2'b10 in port 4,3 and 2'b00 in port1. The response is expected to be a success(2'b01). (Test case 5)
- The outputs are not obtained for the operation of subtraction max with min(in ports 1 and 3, which has simultaneous inputs of 2 and 3 respectively) .(Test case 6)
- In test case 7, subtracting two max numbers the output response is 2'b10 whereas it is expected to have 2'b01 for port2. The output data obtained is incorrect for port4 when it has a tag value of 2'b11.
- When maximum value of (ffffffff) is subtracted by one. The response as well as the data output is incorrect. The Actual value is : ffffffff but Expected: fffffffe. The output response is expected to be 2'b11 due to internal error but it is 2'b01. (Test case 8)

- In case of underflow corner case of subtracting one from minimum value.The output is not obtained, underflow response is not in port1 and port4 the response is not obtained for 5 cycles. (Test case 9)
- In the case of subtracting min with max resulting in underflow. In port it the underflow response is obtained when it is has a tag value of 2’b00, but gives incorrect response when the tag value is 2’b01. The response is also incorrect for port1 (Test case 10)
- In the test case of 11, where shifting minimum number by max, the response of port 2 with a tag of 2’b00 is incorrect. Output data of port4, port3 and port1 with tag 2’b11 and 2’b 01 respectively, is not correct.
- In shifting left of min number by zero digits, port1 gives the response as 2’b00(incorrect) and port4 output is not obtained. (Test case 12)
- In test case 13, left shifting maximum by min. the output data and response obtained is incorrect for port1 and 4.
- For shifting left the maximum number by 1. All the ports give incorrect output. (Test case 14)
- When we are shifting left by one , we get an incorrect output data as the expected output is 80000000 while the actual output is 00000000 whereas the actual response is 10 while the expected response is 01 for the tag 00. When the tag has a value of 01,10 and 11, Both the output data and the output response are different to the expected data and the expected response. (Test case 18)
- When we shift left minimum by one we get an error at port 1, according to which some transactions were missing from the output. In the tag values 00, The output response values are the same while the data (00000001) was different than the data expected (80000000). When the tag is 01 , The expected and the actual output data are identical while the output response (00) is different to the response expected (10). (Test Case 15)
- When we shift left one by 31 we get a Missing transactions error at port 4. When we check these transactions with tag 00 , the output data expected (00000000) and the actual output data (ffffffff) and also the expected response (10) and the obtained response (01) are different. With tag 01 , the actual data (00000003) and the expected data (00000000) are different whereas the actual and expected response are the same (01). With tag 11 , the actual data (fffffffe) and the expected output data (80000000) are different whereas the actual and expected response are the same (01).
- When we shift left max by 31 there is an error at port 2.With tags 00 and 01 the expected and actual output data as well as the expected and actual responses are different to each other. (Test case 20)
- When we shift right min by max we get an error at port 3. While we check with tag 00 we notice that the actual and the expected output data have the same value (00000000) whereas the actual response (10) and the expected response (01) are different values. (Test Case 21)

- When we shift right min by min we get a correct output response at port 4. With both the tags 00 and 01 we get same values for the actual and expected output data as well as for the expected and actual response. (Test Case 22)
- When we shift right max by min we get an error at port 2. With tag 10 , the output data (0000309b) and the expected output data (ffffffff) are different while the expected response (01) and the actual response (01) are same. With tags 01 and 00, the actual data and the expected data and also the expected and actual response are all different. (Test Case 23)
- When we shift right max by one we get an error at port 2. With tag 00 , The actual data (00000000) and the expected data (fffffffe) are different and also the expected response (01) and the actual response (10) are different. With tag 01, The actual output data (00000002) and the expected output data (7fffffff) are different whereas the expected and the actual response (01) are the same. (Test Case 24)
- When we shift right min by one we get an error at port 1.With tags 00 and 10 we get the correct responses whereas with tag 01 we get different expected data output (edb3411d) and actual data output (ffffffe). (Test Case 25)
- When shifting right max by 31 we get an error at port 1 for the tag 00 as the expected data output (00000001)is not equal to the actual data output (00000000) while the actual and expected response values are the same (01). We also get a similar error at port 3 for the tag bits 10 and 01. (Test case 30)
- The test case of 31 includes all the other combination of the of the input data give and the commands: few of the bugs detected in those are (since the number of bugs identified in this test case is lot only few of them are mentioned).
  - In he case of No operation(in which the first command is given as 0), the port 4 and 2 gives a success response expected to get a no response.
  - In the case of dirty state the second command should not affect the operation given by the first command. This is not followed in any ports
- When we shift right one by max we get an error with tag 01 as the actual output data (00000002) and the expected output data (00000000) are different whereas for the tag 10 , the actual output data (ffffffe2) and the expected output data (00000000) are different and also the expected response (01) and the actual response (10) are different. (Test Case 27)
- When we shift right one by min we get an error at port 3 as when we use the tags 00 and 10 we get different values for actual output data and expected output data while we get similar values for the actual response and the expected response.(Test case 26).
- Some transactions were missing in the output and weren't sampled by the monitor.

CONCLUSION

The objective of this project is to build a verification environment with functional coverage that is based on random testing. It has proven to be more effective than direct testing in covering all the possible cases for the input. It is much less time consuming than direct testing when it comes to testing more cases. After simulating our testbench, we were able to identify the errors for the specific test cases as well as other types of errors. Also, functional coverage also helped in keeping track of all the possible inputs that were given to the DUT.

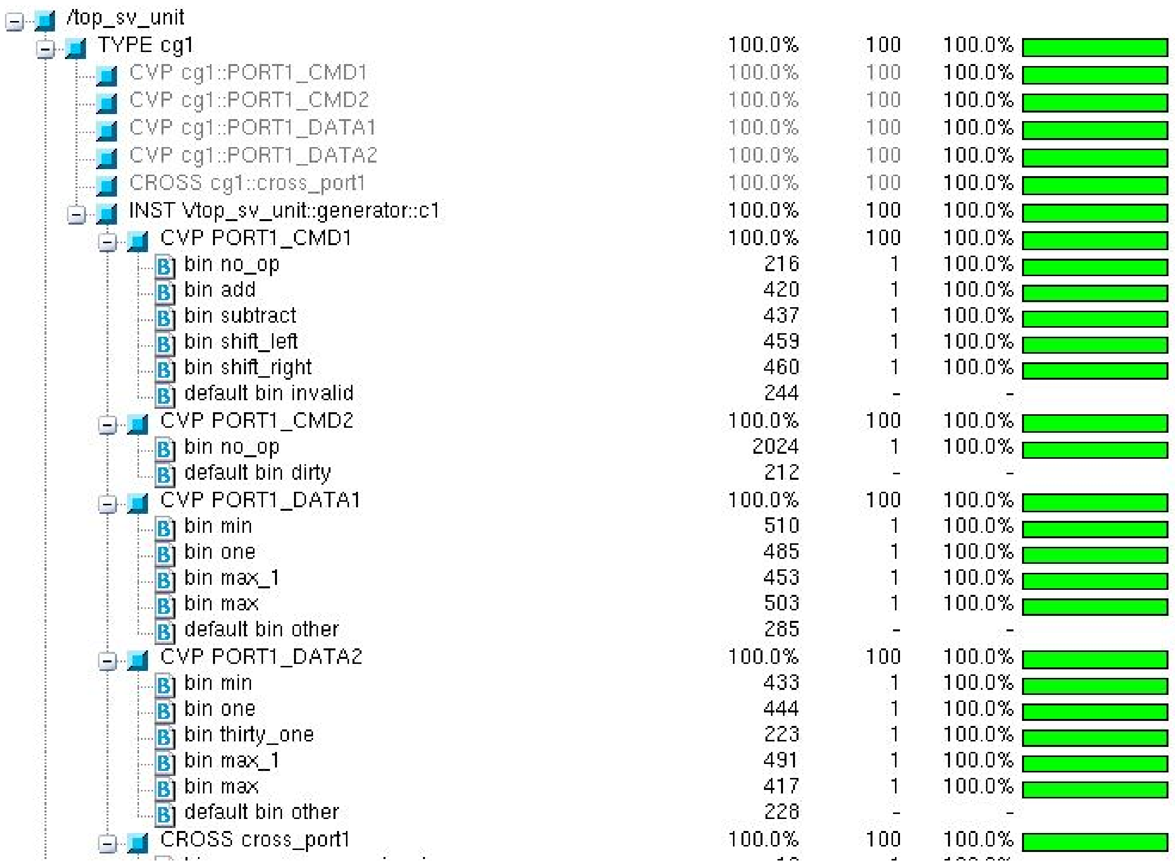
References

1. Lecture notes of Dr. Otmane Ait Mohamed.
2. Notes of the POD Mr. Vivek Bansal.
3. First report of project1 Calc1.

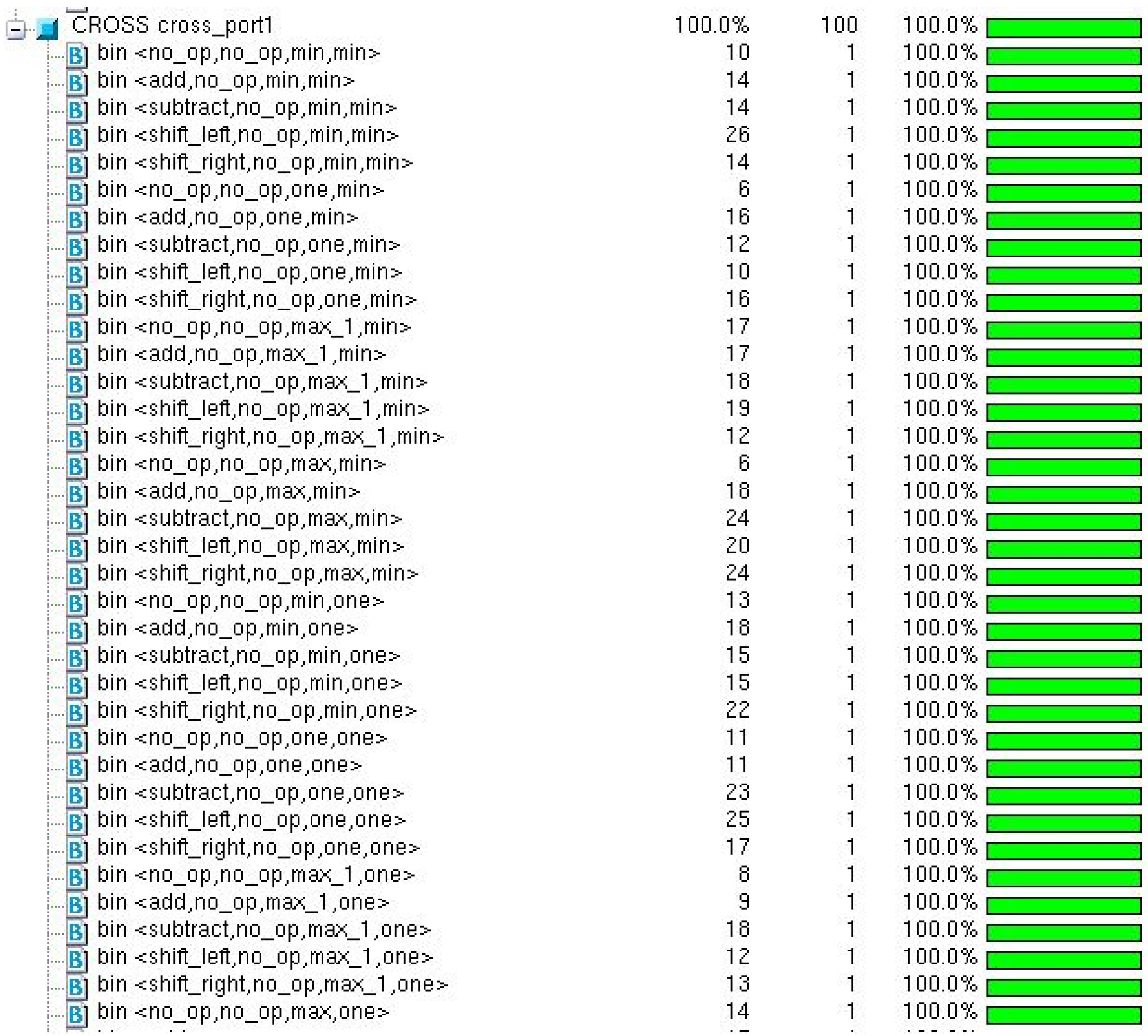
Appendix

Coverage

Because the coverage report is too long and the coverage was 100%, we decided to only show the coverage for port 1. Please note that the max number of iterations for this simulation was 1000.



```
/top_sv_unit
  TYPE cg1                                   100.0%   100   100.0%
    CVP cg1::PORT1_CMD1                      100.0%   100   100.0%
    CVP cg1::PORT1_CMD2                      100.0%   100   100.0%
    CVP cg1::PORT1_DATA1                     100.0%   100   100.0%
    CVP cg1::PORT1_DATA2                     100.0%   100   100.0%
    CROSS cg1::cross_port1                   100.0%   100   100.0%
    INST Vtop_sv_unit::generator::c1         100.0%   100   100.0%
      CVP PORT1_CMD1                         100.0%   100   100.0%
        bin no_op                               216     1   100.0%
        bin add                                 420     1   100.0%
        bin subtract                            437     1   100.0%
        bin shift_left                          459     1   100.0%
        bin shift_right                         460     1   100.0%
        default bin invalid                     244     -        -
      CVP PORT1_CMD2                         100.0%   100   100.0%
        bin no_op                              2024     1   100.0%
        default bin dirty                       212     -        -
      CVP PORT1_DATA1                        100.0%   100   100.0%
        bin min                                 510     1   100.0%
        bin one                                 485     1   100.0%
        bin max_1                               453     1   100.0%
        bin max                                 503     1   100.0%
        default bin other                       285     -        -
      CVP PORT1_DATA2                        100.0%   100   100.0%
        bin min                                 433     1   100.0%
        bin one                                 444     1   100.0%
        bin thirty_one                          223     1   100.0%
        bin max_1                               491     1   100.0%
        bin max                                 417     1   100.0%
        default bin other                       228     -        -
      CROSS cross_port1                      100.0%   100   100.0%
```



```
CROSS cross_port1                        100.0%   100   100.0%
  bin <no_op,no_op,min,min>                 10      1   100.0%
  bin <add,no_op,min,min>                   14      1   100.0%
  bin <subtract,no_op,min,min>              14      1   100.0%
  bin <shift_left,no_op,min,min>            26      1   100.0%
  bin <shift_right,no_op,min,min>           14      1   100.0%
  bin <no_op,no_op,one,min>                  6      1   100.0%
  bin <add,no_op,one,min>                   16      1   100.0%
  bin <subtract,no_op,one,min>              12      1   100.0%
  bin <shift_left,no_op,one,min>            10      1   100.0%
  bin <shift_right,no_op,one,min>           16      1   100.0%
  bin <no_op,no_op,max_1,min>               17      1   100.0%
  bin <add,no_op,max_1,min>                 17      1   100.0%
  bin <subtract,no_op,max_1,min>            18      1   100.0%
  bin <shift_left,no_op,max_1,min>          19      1   100.0%
  bin <shift_right,no_op,max_1,min>         12      1   100.0%
  bin <no_op,no_op,max,min>                  6      1   100.0%
  bin <add,no_op,max,min>                   18      1   100.0%
  bin <subtract,no_op,max,min>              24      1   100.0%
  bin <shift_left,no_op,max,min>            20      1   100.0%
  bin <shift_right,no_op,max,min>           24      1   100.0%
  bin <no_op,no_op,min,one>                 13      1   100.0%
  bin <add,no_op,min,one>                   18      1   100.0%
  bin <subtract,no_op,min,one>              15      1   100.0%
  bin <shift_left,no_op,min,one>            15      1   100.0%
  bin <shift_right,no_op,min,one>           22      1   100.0%
  bin <no_op,no_op,one,one>                 11      1   100.0%
  bin <add,no_op,one,one>                   11      1   100.0%
  bin <subtract,no_op,one,one>              23      1   100.0%
  bin <shift_left,no_op,one,one>            25      1   100.0%
  bin <shift_right,no_op,one,one>           17      1   100.0%
  bin <no_op,no_op,max_1,one>                8      1   100.0%
  bin <add,no_op,max_1,one>                  9      1   100.0%
  bin <subtract,no_op,max_1,one>            18      1   100.0%
  bin <shift_left,no_op,max_1,one>          12      1   100.0%
  bin <shift_right,no_op,max_1,one>         13      1   100.0%
  bin <no_op,no_op,max,one>                 14      1   100.0%
```

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------------------
Covergroup                                   Metric     Goal/ Status
                                                        At Least
----------------------------------------------------------------------------------------------------
TYPE /top_sv_unit/cg1                        100.0%       100 Covered
    Coverpoint cg1::PORT1_CMD1               100.0%       100 Covered
    Coverpoint cg1::PORT1_CMD2               100.0%       100 Covered
    Coverpoint cg1::PORT1_DATA1              100.0%       100 Covered
    Coverpoint cg1::PORT1_DATA2              100.0%       100 Covered
    Cross cg1::cross_port1                   100.0%       100 Covered
Covergroup instance \/top_sv_unit::generator::c1      100.0%       100 Covered
```

```
Coverpoint PORT1_CMD1                    100.0%      100 Covered
    covered/total bins:              5        5
    missing/total bins:              0        5
    bin no_op                   216        1 Covered
    bin add                    420        1 Covered
    bin subtract                437        1 Covered
    bin shift_left              459        1 Covered
    bin shift_right              460        1 Covered
    default bin invalid            244          Occurred
Coverpoint PORT1_CMD2                    100.0%      100 Covered
    covered/total bins:              1        1
    missing/total bins:              0        1
    bin no_op                   2024        1 Covered
    default bin dirty             212          Occurred
Coverpoint PORT1_DATA1                    100.0%      100 Covered
    covered/total bins:              4        4
    missing/total bins:              0        4
    bin min                    510        1 Covered
    bin one                    485        1 Covered
    bin max_1                    453        1 Covered
    bin max                    503        1 Covered
    default bin other             285          Occurred
Coverpoint PORT1_DATA2                    100.0%      100 Covered
    covered/total bins:              5        5
    missing/total bins:              0        5
    bin min                    433        1 Covered
    bin one                    444        1 Covered
    bin thirty_one               223        1 Covered
    bin max_1                    491        1 Covered
    bin max                    417        1 Covered
    default bin other             228          Occurred
Cross cross_port1                 100.0%      100 Covered
    covered/total bins:             100       100
    missing/total bins:              0      100
```

```
bin <no_op,no_op,min,min>              10        1 Covered
bin <add,no_op,min,min>              14        1 Covered
bin <subtract,no_op,min,min>           14        1 Covered
bin <shift_left,no_op,min,min>         26        1 Covered
bin <shift_right,no_op,min,min>          14        1 Covered
bin <no_op,no_op,one,min>               6        1 Covered
bin <add,no_op,one,min>              16        1 Covered
bin <subtract,no_op,one,min>           12        1 Covered
bin <shift_left,no_op,one,min>         10        1 Covered
bin <shift_right,no_op,one,min>          16        1 Covered
bin <no_op,no_op,max_1,min>              17        1 Covered
bin <add,no_op,max_1,min>              17        1 Covered
bin <subtract,no_op,max_1,min>           18        1 Covered
bin <shift_left,no_op,max_1,min>         19        1 Covered
bin <shift_right,no_op,max_1,min>          12        1 Covered
bin <no_op,no_op,max,min>               6        1 Covered
bin <add,no_op,max,min>              18        1 Covered
bin <subtract,no_op,max,min>           24        1 Covered
bin <shift_left,no_op,max,min>         20        1 Covered
bin <shift_right,no_op,max,min>          24        1 Covered
bin <no_op,no_op,min,one>              13        1 Covered
bin <add,no_op,min,one>              18        1 Covered
bin <subtract,no_op,min,one>           15        1 Covered
bin <shift_left,no_op,min,one>         15        1 Covered
bin <shift_right,no_op,min,one>          22        1 Covered
bin <no_op,no_op,one,one>              11        1 Covered
bin <add,no_op,one,one>              11        1 Covered
bin <subtract,no_op,one,one>           23        1 Covered
bin <shift_left,no_op,one,one>         25        1 Covered
bin <shift_right,no_op,one,one>          17        1 Covered
bin <no_op,no_op,max_1,one>               8        1 Covered
bin <add,no_op,max_1,one>               9        1 Covered
bin <subtract,no_op,max_1,one>           18        1 Covered
bin <shift_left,no_op,max_1,one>         12        1 Covered
bin <shift_right,no_op,max_1,one>              13         1 Covered
bin <no_op,no_op,max,one>                      14         1 Covered
bin <add,no_op,max,one>                        17         1 Covered
bin <subtract,no_op,max,one>                   16         1 Covered
bin <shift_left,no_op,max,one>                 17         1 Covered
bin <shift_right,no_op,max,one>                16         1 Covered
bin <no_op,no_op,min,thirty_one>                4         1 Covered
bin <add,no_op,min,thirty_one>                  9         1 Covered
bin <subtract,no_op,min,thirty_one>             5         1 Covered
bin <shift_left,no_op,min,thirty_one>           6         1 Covered
bin <shift_right,no_op,min,thirty_one>          6         1 Covered
bin <no_op,no_op,one,thirty_one>                5         1 Covered
bin <add,no_op,one,thirty_one>                 13         1 Covered
bin <subtract,no_op,one,thirty_one>             9         1 Covered
bin <shift_left,no_op,one,thirty_one>          11         1 Covered
bin <shift_right,no_op,one,thirty_one>         11         1 Covered
bin <no_op,no_op,max_1,thirty_one>              2         1 Covered
bin <add,no_op,max_1,thirty_one>                7         1 Covered
bin <subtract,no_op,max_1,thirty_one>          10         1 Covered
bin <shift_left,no_op,max_1,thirty_one>         4         1 Covered
bin <shift_right,no_op,max_1,thirty_one>        4         1 Covered
bin <no_op,no_op,max,thirty_one>                5         1 Covered
bin <add,no_op,max,thirty_one>                  8         1 Covered
bin <subtract,no_op,max,thirty_one>            12         1 Covered
bin <shift_left,no_op,max,thirty_one>          11         1 Covered
bin <shift_right,no_op,max,thirty_one>          8         1 Covered
bin <no_op,no_op,min,max_1>                     9         1 Covered
bin <add,no_op,min,max_1>                      25         1 Covered
bin <subtract,no_op,min,max_1>                 17         1 Covered
bin <shift_left,no_op,min,max_1>               22         1 Covered
bin <shift_right,no_op,min,max_1>              19         1 Covered
bin <no_op,no_op,one,max_1>                    11         1 Covered
bin <add,no_op,one,max_1>                      16         1 Covered
bin <subtract,no_op,one,max_1>                 23         1 Covered
bin <shift_left,no_op,one,max_1>             23        1 Covered
bin <shift_right,no_op,one,max_1>            13        1 Covered
bin <no_op,no_op,max_1,max_1>                 7        1 Covered
bin <add,no_op,max_1,max_1>                  15        1 Covered
bin <subtract,no_op,max_1,max_1>             15        1 Covered
bin <shift_left,no_op,max_1,max_1>           23        1 Covered
bin <shift_right,no_op,max_1,max_1>          22        1 Covered
bin <no_op,no_op,max,max_1>                   9        1 Covered
bin <add,no_op,max,max_1>                    17        1 Covered
bin <subtract,no_op,max,max_1>               13        1 Covered
bin <shift_left,no_op,max,max_1>             21        1 Covered
bin <shift_right,no_op,max,max_1>            24        1 Covered
bin <no_op,no_op,min,max>                     4        1 Covered
bin <add,no_op,min,max>                      16        1 Covered
bin <subtract,no_op,min,max>                 23        1 Covered
bin <shift_left,no_op,min,max>               16        1 Covered
bin <shift_right,no_op,min,max>              25        1 Covered
bin <no_op,no_op,one,max>                     6        1 Covered
bin <add,no_op,one,max>                      13        1 Covered
bin <subtract,no_op,one,max>                 14        1 Covered
bin <shift_left,no_op,one,max>               23        1 Covered
bin <shift_right,no_op,one,max>              13        1 Covered
bin <no_op,no_op,max_1,max>                   8        1 Covered
bin <add,no_op,max_1,max>                    18        1 Covered
bin <subtract,no_op,max_1,max>               20        1 Covered
bin <shift_left,no_op,max_1,max>             16        1 Covered
bin <shift_right,no_op,max_1,max>            19        1 Covered
bin <no_op,no_op,max,max>                    11        1 Covered
bin <add,no_op,max,max>                      20        1 Covered
bin <subtract,no_op,max,max>                 13        1 Covered
bin <shift_left,no_op,max,max>                9        1 Covered
bin <shift_right,no_op,max,max>              16        1 Covered
```

Transcript Output

Because the transcript output is very long, we decided to show part of it (two transaction arrays).

```
# Time:            344950 Port        1:
**Driver to Scoreboard**:
# Test Case 31: Other
# Input Command1: 0101, Input Command2: 0000, Input Tag: 00, Input Data1: 00000001, Input Data2:
fffffffe
# Test Case 1: Adding min with min
# Input Command1: 0001, Input Command2: 0000, Input Tag: 01, Input Data1: 00000000, Input Data2:
00000000
# Test Case 24: Shifting Right max by one
# Input Command1: 0110, Input Command2: 0000, Input Tag: 10, Input Data1: ffffffff, Input Data2:
00000001
# Test Case 31: Other
# Input Command1: 0010, Input Command2: 0000, Input Tag: 11, Input Data1: ffffffff, Input Data2:
fffffffe
# **Monitor to Scoreboard**:
# Output Response: 10, Output Tag: 00, Output Data: 00000000
# Output Response: 10, Output Tag: 01, Output Data: 00000000
# Output Response: 01, Output Tag: 10, Output Data: ffffffff
# Error: Missing transactions from output
# Checking transaction with tag: 00
# Error: Incorrect Output Data: Actual: 00000000, Expected: 40000000
# Error: Incorrect Output Response: Actual: 10, Expected: 01
# Checking transaction with tag: 01
# Correct Output Data: Actual: 00000000, Expected: 00000000
# Error: Incorrect Output Response: Actual: 10, Expected: 01
# Checking transaction with tag: 10
# Error: Incorrect Output Data: Actual: ffffffff, Expected: 7fffffff
# Correct Output Response: Actual: 01, Expected: 01

# Time:            345950 Port        1:
# **Driver to Scoreboard**:
```

```
# Test Case 31: Other
# Input Command1: 0110, Input Command2: 0000, Input Tag: 00, Input Data1: 2681b6ad, Input Data2:
ffffffff
# Test Case 31: Other
# Input Command1: 0101, Input Command2: 0000, Input Tag: 01, Input Data1: ffffffff, Input Data2:
ffffffff
# Test Case 31: Other
# Input Command1: 0110, Input Command2: 0000, Input Tag: 10, Input Data1: 31c1a780, Input Data2:
ffffffff
# Test Case 31: Other
# Invalid Command 1
# Dirty State (Command 2 isn't no_op)
# Input Command1: 1111, Input Command2: 0100, Input Tag: 11, Input Data1: ffffffff, Input Data2:
00000000
# **Monitor to Scoreboard**:
# Output Response: 01, Output Tag: 10, Output Data: 00000001
# Output Response: 10, Output Tag: 00, Output Data: 00000000
# Output Response: 10, Output Tag: 01, Output Data: 00000000
# Output Response: 01, Output Tag: 11, Output Data: fffffffe
# Checking transaction with tag: 00
# Correct Output Data: Actual: 00000000, Expected: 00000000
# Error: Incorrect Output Response: Actual: 10, Expected: 01
# Checking transaction with tag: 01
# Error: Incorrect Output Data: Actual: 00000000, Expected: 80000000
# Error: Incorrect Output Response: Actual: 10, Expected: 01
# Checking transaction with tag: 10
# Error: Incorrect Output Data: Actual: 00000001, Expected: 00000000
# Correct Output Response: Actual: 01, Expected: 01
# Checking transaction with tag: 11
# Error: Incorrect Output Data: Actual: fffffffe, Expected: 00000000
# Error: Incorrect Output Response: Actual: 01, Expected: 10
```