Concordia University
Engineering and Computer Science

Winter 2019

COEN 6541: Hardware Functional Verification
Dr. Otmane Ait Mohamed

Project Report
Testbench for calculator design (CALC1,CALC2,CALC3)

Team Members

Amulya Prabakhar (**40089026**)
Arjun Ravindran (**40069977**)
Sibi Ravichandran (**40076654**)

TABLE OF CONTENTS

1. INTRODUCTION

This report is a detailed description of the verification plan, Verification environment, algorithm used and simulation results of calc1 protocol. Specification of calc1 Design Under Test is given below, we have used QuestaSim for compilation and simulation of our test.

1.1. CALC1

CALC1 design performs 4 arithmetic operations namely Add, Subtract, Shift left and Shift right. The calculator has 4 input ports and 4 output ports, where it can perform 4 requests in parallel at a time. All 4 requestors use separate input signals which have equal priority. Each port must wait till the current request completes before sending the new one. When using a cycle simulator, the clock should be held high and to be toggled when using event simulator.Priority logic allows for 1 add or subtract at a time and one shift operation at a time, Figure 1 shows the block diagram of CALC1 design. Protocol used for CALC1 is shown in the Table 1 below

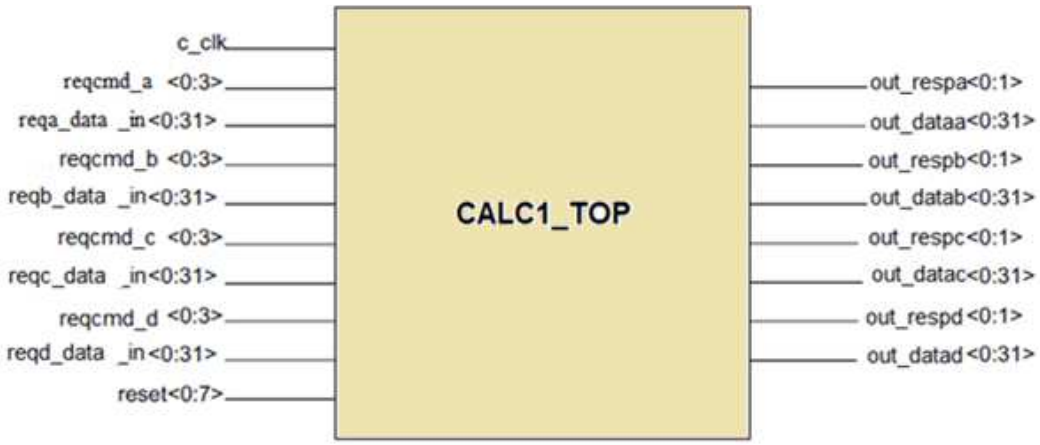

Figure 1. Block diagram of CALC1 design

PARAMETERS	PROPERTIES
OPCODE	0 - No-op 1 - Add operand1 and operand2 2 - Subtract operand2 from operand1 5 - shifts contents of operand1 to the left operand2 (27:31) 6 - shifts contents of operand1 to the right operand2 (27:31) Others – Invalid Command
DATA INPUT	32’H 00000000 to 32’H FFFFFFFF
RESET	Hold reset(1:7) to '1111111'b at start of testcase for seven cycles
RESPONSE	0 - no response 1 - successful operation completion 2 - invalid command or overflow/underflow error 3 - Internal error
DATAOUTPUT	32’H 00000000 to 32’H FFFFFFFF

Table 1. Properties of CALC1 protocol

1.2. CALC2

CALC2 design is built on CALC1 design, in CALC1 only one request from each port was taken care at a time, each port requestor sends 4 commands at a time and hence could handle 16 commands at a time.There are two arithmetic pipelines, one performs add/subtract and another performs shift operation. The 2 arithmetic pipelines work in parallel and hence commands can be executed out of order, but within the queue it should be in the order it came in. In order to correlate the responses to the correct commands, a 2 bit tag is being added to the input and output protocols. Figure2 shows the block diagram of CALC2. Protocol used for CALC2 is shown in Table 2.

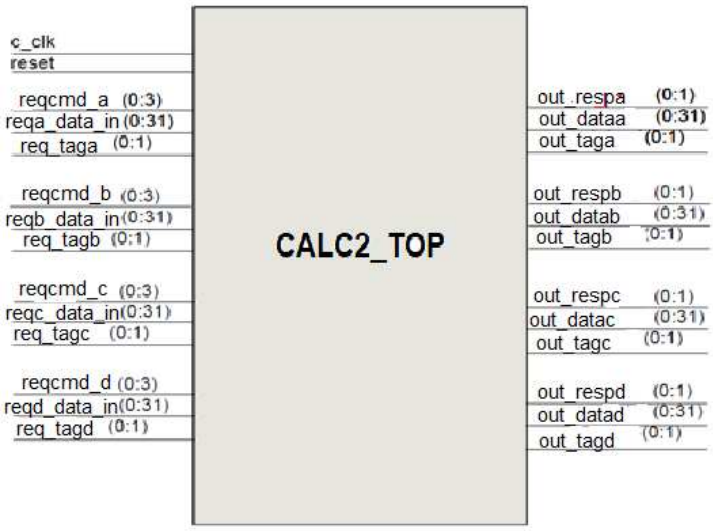

Figure 2. Block diagram of CALC2 design

PARAMETERS	PROPERTIES
C_clk	Main clock
Reset	needs to be held high for the three cycles at the start of the testcase. Must remain low during functional testing. Similarly, all input ports need to be driven low from the start of simulation.
reqcmd_X(0:3)	0 - No-op 1 - Add operand1 and operand2 2 - Subtract operand2 from operand1 5 - shifts contents of operand1 to the left operand2 (27:31) 6 - shifts contents of operand1 to the right operand2 (27:31) Others – Invalid Command
reqX_data_in(0:31)	The operand data is sent one cycle after another. Operand1 data accompanies command, and operand2 data follows.
reqtag_X(0:1)	Two bit identifier for each command from the port. Can be reused as soon as the calculator responds to the command.
out_respX(0:1)	0 - no response 1 - successful operation completion 2 - invalid command or overflow/underflow error 3 - Internal error
out_dataX(0:31)	This is the data that accompanies a good response.
out_tagX(0:1)	Corresponds to the command tag sent by the requester. Used to identify which command the response if for.

Table 2. Properties of CALC2 Protocol.

1.3. CALC3

In CALC3 design, we have 16 internal registers from where we read the operand data unlike we send it manually as per the previous design. Commands Fetch and store access the registers whereas branch commands causes next command from port to be skipped if evaluated to true. Each port requestor can send upto 4 commands, 2-bit tag on request. Duplicate tags are not allowed. Since each port requestor is sending an instruction stream and the arithmetic ops reference operand registers internal to the design, instruction ordering concepts must be obeyed by the design so that within each port, the commands may only proceed out of order when the operand registers do not conflict.

PARAMETERS	PROPERTIES
reqX_cmd(0:3)	1: adds contents of d1 to d2 and stores in r1 2: subtracts contents of d2 from d1 and stores in r1 5: shifts contents of d1 to the left d2(27:31) places and stores in r1 6: shifts contents of d1 to the right d2(27:31) places and stores in r1 9: stores reqX_data(0:31) into r1 10: fetches contents of d1 and outputs it on out_dataX(0:31) 12: branch if zero: skip next valid command if contents of d1 are 0 13: branch if equal: skip next valid command if contents of d1 and d2 are equal
reqX_d1(0:3)	Corresponds to register which holds operand 1
reqX_d2(0:3)	Corresponds to the register which holds operand 2
reqX_r1(0:3)	Corresponds to register r1 where the result is stored
reqtag_X(0:1)	Two bit identifier for each command from the port. Can be reused as soon as the calculator responds to the command.
reqX_data(0:31)	Operand data
outX_resp(0:1)	1: Successful completion 2: overflow/underflow error 3: Command skipped due to branch
outX_tag(0:1)	Corresponds to the command tag sent by the requester. Used to identify which command the response if for.
outX_data(0:31)	This is the data that accompanies a good response.

Table 3. Properties of CALC3 Protocol.

2. VERIFICATION ENVIRONMENT

The verification environment used to test CALC (CALC1, CALC2, CALC3 designs) designsconsists of transactor, generator, driver, monitor, scoreboard, environment, test and top files. It also has mailboxes for transmit data between generator to driver, generator to scoreboard, monitor to scoreboard. It has a interface file which connects DUT to test environment. Functional coverage help us to cover the maximum test points. Figure 3 shows the test environment implemented to verify the calculator design and Figure 4 shows the interface between design and testbench.

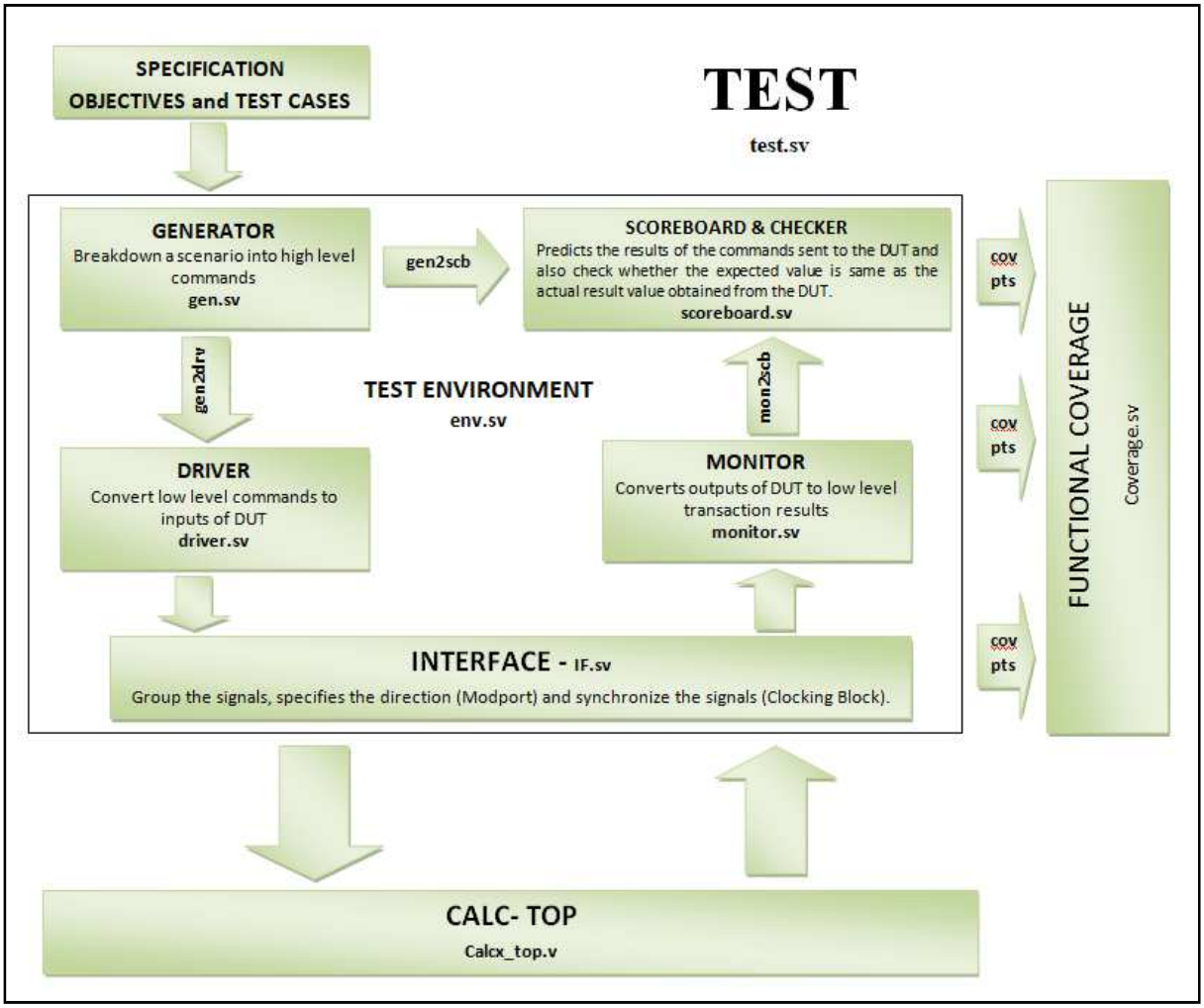

Figure 3. Verification environment of CALC

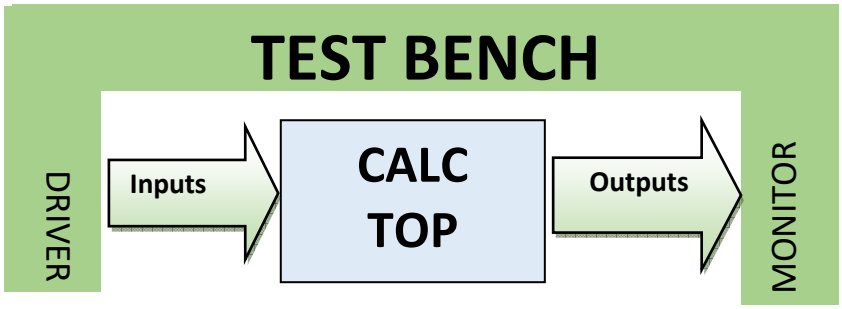

Figure 4. Interface between CALC design and test bench.

3. TEST BENCH IMPLEMENTATION

3.1. Test bench implementation for CALC1:

File name	Type	Description	Purpose
Transaction	Class	This holds the fields required to generate the stimulus are declared in the transaction class. Transaction class can also be used as placeholder for the activity monitored by monitor on DUT signals. This trasaction file is for the generator to driver.	• Declare and randomize the request command with constraints(0,1,2,5,6), randomize the input data, declare reset bit. • Display request command and request data at each port. • Create a copy of the object of class Transaction.
gen (Generator)	Class	This is used to breakdown a scenario into high level commands	• Randomizes the class transaction • Create and randomize the transaction packets and the maximum transaction count. • Send data from generator to driver, generator to scoreboard through mailbox.
driver	Class	This is used to convert low level commands to inputs of DUT	• Create mailbox to receive data from generator and pass the same to scoreboard • Perform the reset operation for seven cycles • Send command and data on all the ports of DUT and display the same
IF (Interface)	Class	Interface will group the signals, specifies the direction (Modport) and Synchronize the signals(Clocking Block).	• Connects driver and monitor to DUT • Create clocking block for driver and monitor • Defines the modports for driver, monitor and DUT.
dut_out_a	Class	This provide the output data and response from the DUT at port A	• Define output data and response for Port A • Create a copy of the object of the class dut_out_a • Print the value received from the DUT at port A
dut_out_b	Class	This provide the output data and response from the DUT at port B	• Define output data and response for Port B • Create a copy of the object of the class dut_out_b • Print the value received from the DUT at port B
dut_out_c	Class	This provide the output data and response from the DUT at port C	• Define output data and response for Port C • Create a copy of the object of the class dut_out_c • Print the value received from the DUT at port C
dut_out_d	Class	This provide the output data and response from the DUT at port D	• Define output data and response for Port D • Create a copy of the object of the class dut_out_d • Print the value received from the DUT at port D
monitor	Class	Monitor converts outputs of	• Monitor out_data, out_resp everytime

		DUT to low level transaction results	out_resp has a valid value. • Put the transaction in mailbox monitor to scoreboard • Display the data and response received
scoreboard	Class	This predicts the results of the commands sent to the DUT and also check whether the expected value is same as the actual result value obtained from the DUT.	• Calculate the expected data corresponding to commands and operands. • Get the transaction from mailbox generator to scoreboard and monitor to scoreboard. • Compare the expected data with the output data received from monitor. • Display Error when there is a mismatch
env (Environment)	Class	The environment file acts as the top file for the entire test bench environment.	• Create all the mailboxes • Instantiate all the classes
test	Class	This file is used to kickoff the test	• Instantiate the environment • Kick off the test
top	Class	This file is used to integrate the test bench with the DUT	• Set the simulation_cycle • Provide clock and synchronize Interface, test bench and DUT
coverage	Class	After randomization, functional coverage is used to cover maximum number of test points.	Define a covergroup which includes coverpoint for all inputs at all the ports.

3.2. Test bench implementation for CALC2:

File name	Type	Description	Purpose
Transaction	Class	This holds the fields required to generate the stimulus are declared in the transaction class. Transaction class can also be used as placeholder for the activity monitored by monitor on DUT signals. This trasaction file is for the generator to driver.	• Declare and randomize the request command withconstraints(0,1,2,5,6,), randomize the tag bits with constraints(0,1,2,3) in cyclic order , declare reset bit. Randomize the input data. • Display request command, request tag and request data at each port. • Create a copy of the object of class Transaction.
gen (Generator)	Class	This is used to breakdown a scenario into high level commands	• Randomizes the class transaction • Create and randomize the transaction packets and the maximum transaction count. • Create tags for each transaction and ensures that there are no duplicate tags and individual port tags are also in cyclic order. • Send data from generator to driver, generator to scoreboard through mailbox.
driver	Class	This is used to convert low level commands to inputs of DUT	• Create mailbox to receive data from generator and pass the same to scoreboard • Perform the reset operation for three cycles • Send command and data on all the ports of DUT and display the same
IF (Interface)	Class	Interface will group the signals, specifies the direction (Modport) and	• Connects driver and monitor to DUT • Create clocking block for driver and monitor considering tag along.

		Synchronize the signals(Clocking Block).	• Defines the modports for driver, monitor and DUT.
dut_out_a	Class	This provide the output data and response from the DUT at port A	• Define output data and response at port A • Create a copy of the object of the class dut_out_a • Print the value received from the DUT at port A
dut_out_b	Class	This provide the output data and response from the DUT at port B	• Define output data and response at port B • Create a copy of the object of the class dut_out_b • Print the value received from the DUT at port B
dut_out_c	Class	This provide the output data and response from the DUT at port C	• Define output data and response at port C • Create a copy of the object of the class dut_out_c • Print the value received from the DUT at port C
dut_out_d	Class	This provide the output data and response from the DUT at port D	• Define output data and response at port D • Create a copy of the object of the class dut_out_d • Print the value received from the DUT at port D
monitor	Class	Monitor converts outputs of DUT to low level transaction results	• Monitor out_data, out_resp, out_tag everytime out_resp, out_tag has a valid value. • Put the transaction in mailbox monitor to scoreboard • Display the data, tag and response received
scoreboard	Class	This predicts the results of the commands sent to the DUT and also check whether the expected value is same as the actual result value obtained from the DUT.	• Calculate the expected data corresponding to commands and operands. • Calculate the expected tag corresponding to the input tag. • Get the transaction from mailbox generator to scoreboard and monitor to scoreboard. • Compare the expected data with the output data received from monitor. • Compare the expected tag with the output tag received from monitor. • Display Error when there is a mismatch in data as well as in tag.
env (Environment)	Class	The environment file acts as the top file for the entire test bench environment.	• Create all the mailboxes • Instantiate all the classes
test	Class	This file is used to kickoff the test	• Instantiate the environment • Kick off the test
top	Class	This file is used to integrate the test bench with the DUT	• Set the simulation_cycle • Provide clock and synchronize Interface, test bench and DUT
coverage	Class	After randomization, functional coverage is used to cover maximum number of test points.	Define a covergroup which includes coverpoint for all inputs at all the ports.

3.3. Test bench implementation for CALC3:

File name	Type	Description	Purpose
Transaction	Class	This holds the fields required to generate the stimulus are declared in the transaction class. Transaction class can also be used as placeholder for the activity monitored by monitor on DUT signals. This trasaction file is for the generator to driver.	• Declare and randomize the request command with constraints(0,1,2,5,6,9,10,12,13), randomize the tag bits with constraints(0,1,2,3) in cyclic order , declare reset bit. Randomize the internal registers d1,d2 and r1 at each port. • Display request command, request tag and request data at each port. • Create a copy of the object of class Transaction.
gen (Generator)	Class	This is used to breakdown a scenario into high level commands	• Randomizes the class transaction • Create and randomize the transaction packets and the maximum transaction count. • Create tags for each transaction and ensures that there are no duplicate tags and individual port tags are also in cyclic order. • Send data from generator to driver, generator to scoreboard through mailbox.
driver	Class	This is used to convert low level commands to inputs of DUT	• Create mailbox to receive data from generator and pass the same to scoreboard • Perform the reset operation for three cycles • Function to store initial register values for port 1, 2, 3 and 4 • Send randomly generated commands, tags and operands to the DUT through the Interface
IF (Interface)	Class	Interface will group the signals, specifies the direction (Modport) and Synchronize the signals(Clocking Block).	• Connects driver and monitor to DUT • Create clocking block for driver and monitor considering tag along. • Defines the modports for driver, monitor and DUT.
dut_out_a	Class	This provide the output data and response from the DUT at port A	• Define output data and response at port A • Create a copy of the object of the class dut_out_a • Print the value received from the DUT at port A
dut_out_b	Class	This provide the output data and response from the DUT at port B	• Define output data and response at port B • Create a copy of the object of the class dut_out_b • Print the value received from the DUT at port B
dut_out_c	Class	This provide the output data and response from the DUT at port C	• Define output data and response at port C • Create a copy of the object of the class dut_out_c • Print the value received from the DUT at port C
dut_out_d	Class	This provide the output data and response from the DUT at port D	• Define output data and response at port D • Create a copy of the object of the class dut_out_d

			• Print the value received from the DUT at port D
monitor	Class	Monitor converts outputs of DUT to low level transaction results	• Monitor out_data, out_resp, out_tag, everytime out_resp, out_tag has a valid value. • Put the transaction in mailbox monitor to scoreboard • Display the data, tag and response received
scoreboard	Class	This predicts the results of the commands sent to the DUT and also check whether the expected value is same as the actual result value obtained from the DUT.	• Calculate the expected data corresponding to commands and operands. • Calculate the expected tag corresponding to the input tag. • Get the transaction from mailbox generator to scoreboard and monitor to scoreboard. • Compare the expected data with the output data received from monitor. • Compare the expected tag with the output tag received from monitor. • Display Error when there is a mismatch in data as well as in tag.
env (Environment)	Class	The environment file acts as the top file for the entire test bench environment.	• Create all the mailboxes • Instantiate all the classes
test	Class	This file is used to kickoff the test	• Instantiate the environment • Kick off the test
top	Class	This file is used to integrate the test bench with the DUT	• Set the simulation_cycle • Provide clock and synchronize Interface, test bench and DUT
coverage	Class	After randomization, functional coverage is used to cover maximum number of test points.	Define a covergroup which includes coverpoint for all inputs at all the ports.

4. TEST REPORT

Detailed test report in excel sheet "calc_1.xls calc_2.xls and calc_3.xls" is attached along with the report.

5. BUG REPORT

5.1. BUG Report of CALC1 Design:

Sl.no	TEST PARAMETER	RESULT	Analysis
1.	Reset	PASS	As per test point 10, When reset operation is performed the Outputs of the calculator are ignored.
2.	Addition	FAIL	1. **Pass** for the least four bits.(as per 2.1,4.2) 2. **Pass** when sum is equal to "Xffffffff" (as per 4.1) 3. **Fails** for the data greater than 4 bits. (as per 1.1, 4.4 5th bit of output bus is dead) 4. **Overflow condition: Pass** DUT generates output response properly (as per 3.3 and 4.3)

3.	Subtraction	PASS	1. **Pass** for Data 1 > Data 2 and Data 1 = Data 2. 2. **Underflow condition: Pass** DUT generate output response. (as per 2.3, 5.2)
4.	Shift Operation	FAIL	1. Shift Left Operation **passes** for 31 places of shift 2. Shift Right Operation **passes** for all 31 places of shift but it **doesn't work** for 0^{th} position (as per 7.1)
5.	Invalid Command	FAIL	When an invalid command is generated the output response should hold the value '2' but it is '0'. So the calc is not able to recognize an invalid command and generate the response accordingly. (as per 9.1,9.2,9.3,9.4)
6.	Priority	PASS	The logic allows for only 1 add or subtract and one shift operation at a time. They are executed in the order of first come first serve basis.

5.2. BUG Report of CALC2 Design:

Sl.no	Test Parameter	Result	Analysis			
			PORT 1	**PORT 2**	**PORT 3**	**PORT 4**
1.	Basic Add Command	Pass	• **Pass**as per 1.1,1.2,1.3,1.4			
2.	Basic Subtract Command	Pass	• **Pass**as per 2.1,2.2,2.3,2.4			
3.	Basic Shift Left	Fail	Shift Left Operation fails as per 3.1,3.2 due to 15^{th} bit of output which is dead or always 0, it passes when the 15^{th} bit of output should be zero as per 3.3,3.4			
4.	Basic Shift Right	Pass	Shift Right Operation **passes** for 31 places of shift as per 4.1,4.2,4.3,4.4			
5.	All four tags	Pass	**Pass,** they are cyclically assigned and no duplication			
6.	Variable timing between commands	Not tested	Considered only fixed timing between commands			
7.	Invalid Commands	Fail	Output response received is 0 instead of 2(as per 6.1,6.2,6.3,6.4)			
8.	Overflow and underflow cases	Pass	• **Overflow** : **Pass**DUT produces the right output response (as per 1.3) • **Underflow** : **Pass**DUT generate output response.(as per 2.2)			
9.	Various lengths of shift values	Fail	Shift left fails due to 15^{th} bit is dead			
10.	Reset	Pass	When reset operation is performed the Outputs of the calculator are ignored.			

5.3. BUG Report of CALC3 Design:

Sl.no	Test Parameter	Result	Analysis			
			PORT 1	**PORT 2**	**PORT 3**	**PORT 4**
1.	Adds contents of d1 to d2 and stores in r1	Fail	• Fail as per 1.1,2.1,3.1,4.1			
2.	subtracts contents of	Fail	• Fail as per 1.2,2.2,3.2,4.2			

	d2 from d1 and stores in r1		
3.	Result in Shift left which shifts contents of d1 to the left d2(27:31) places and stores in r1	Pass	• Pass as per 1.3,2.3,3.3,4.3
4.	Shift right shifts contents of d1 to the right d2(27:31) places and stores in r1	Pass	• Pass as per 1.4,2.4,3.4,4.4
5.	Overflow/underflow	Pass	**Pass** as per 1.5,2.5,3.5,4.5,1.6,2.6,3.6,4.6
6.	fetches contents of d1 and outputs it on out_dataX(0:31)	Fail	Fail as per 3.8,1.8,2.8,4.8,
7.	Each port many have four commands given at a time but executes only one operation and a given clock.	Pass	Pass as per 6
8.	Reset	Pass	When reset operation is performed the Outputs of the calculator are ignored.
9.	stores reqX_data(0:31) into r1	Pass	Pass as per 1.7,2.7,3.7,4.7

6. SIMULATION RESULT AND COVERAGE REPORT

Figure 5, Figure 6 and Figure 9 illustrates the operation of Reset, priority logic consideration and the simulation results obtained in the questasim for the inputs provided for CALC1, CALC2 and CALC 3 respectively. Figure 7 and Figure 8 provides the coverage report of the CALC1 and CALC2.

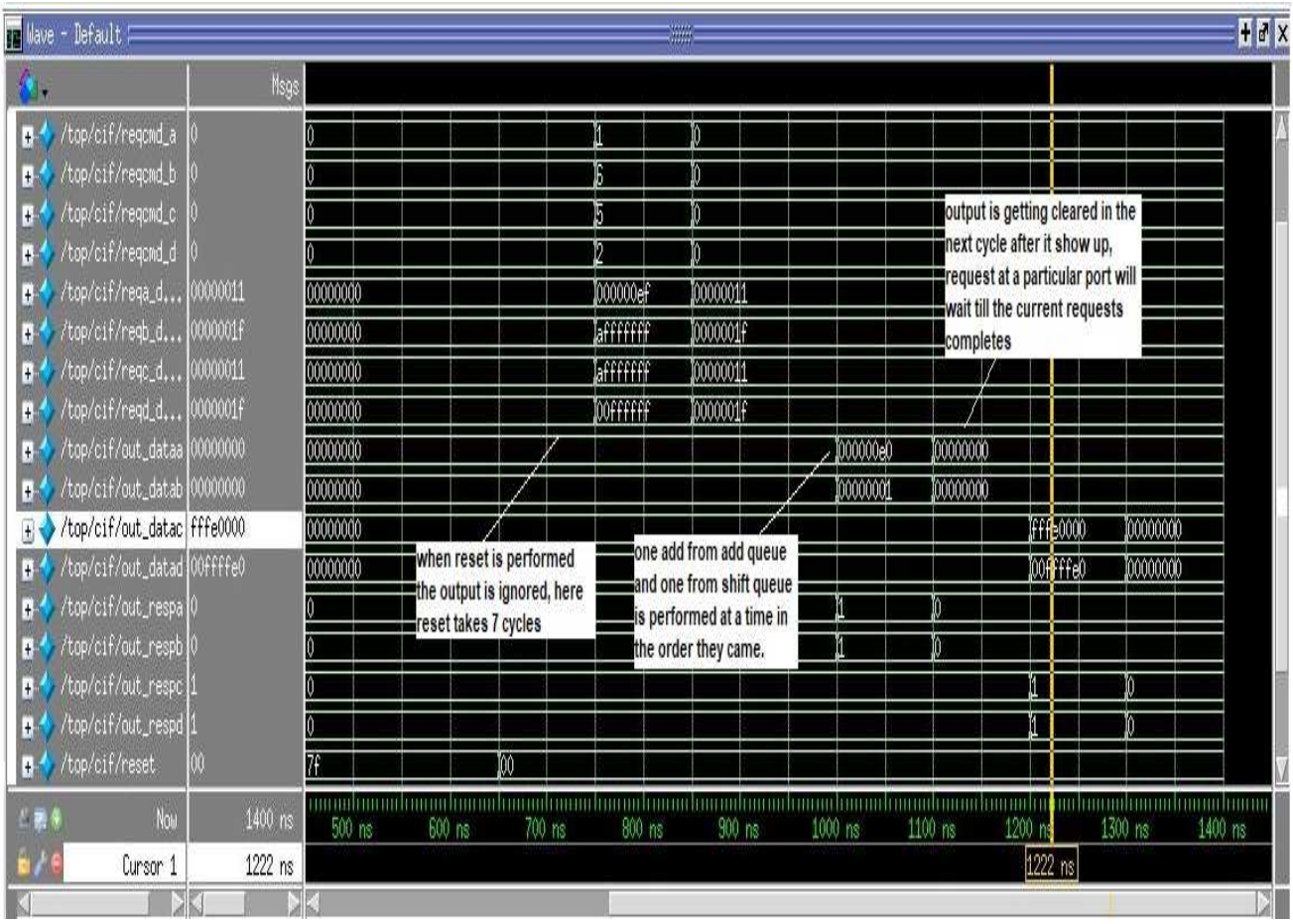

Figure 5. Simulation Result of CALC1

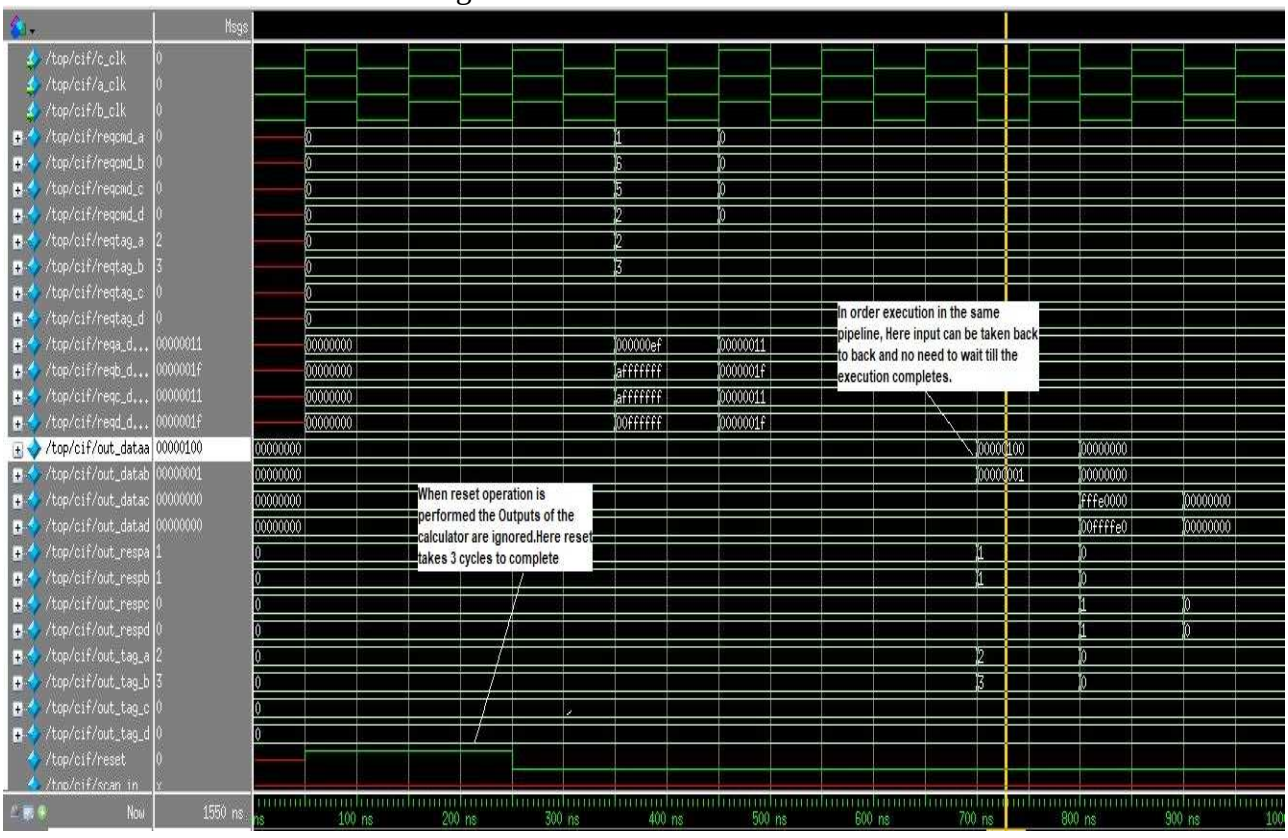

Figure 6. Simulation Result of CALC2

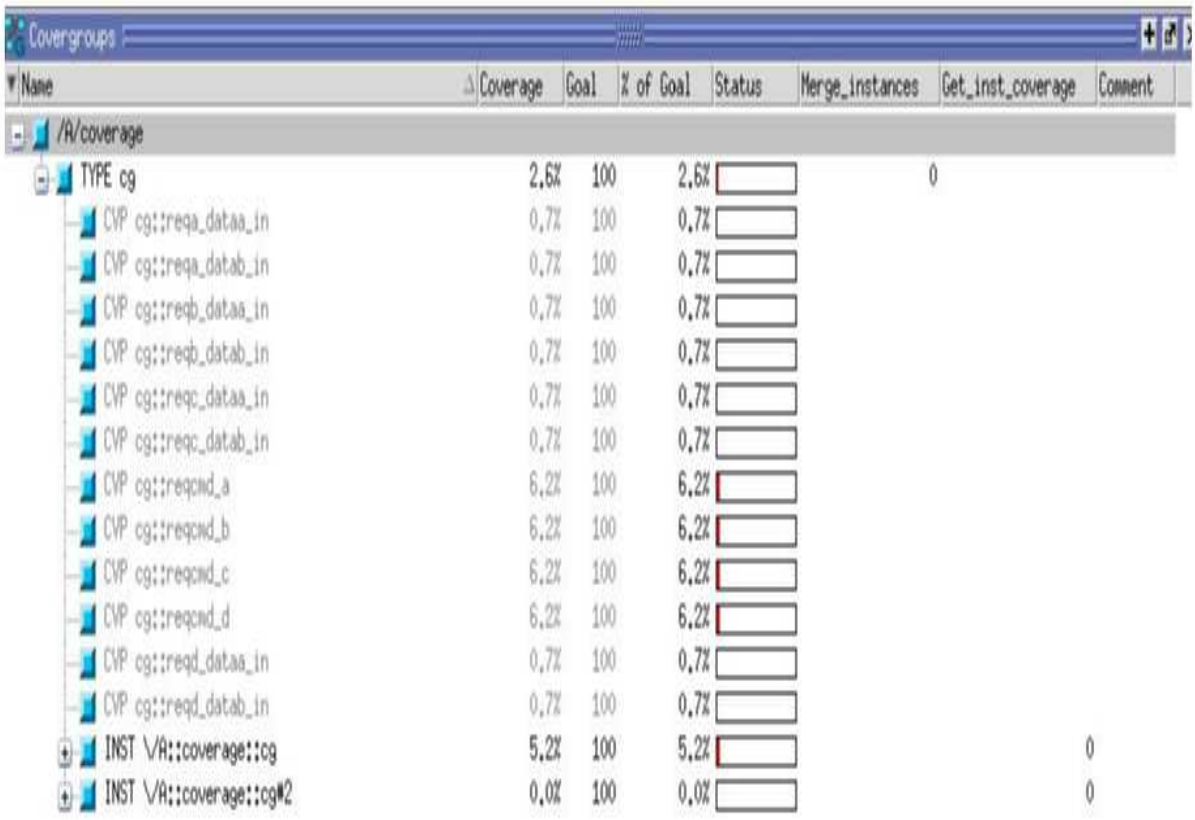

Figure 7. Coverage report of CALC1 Design

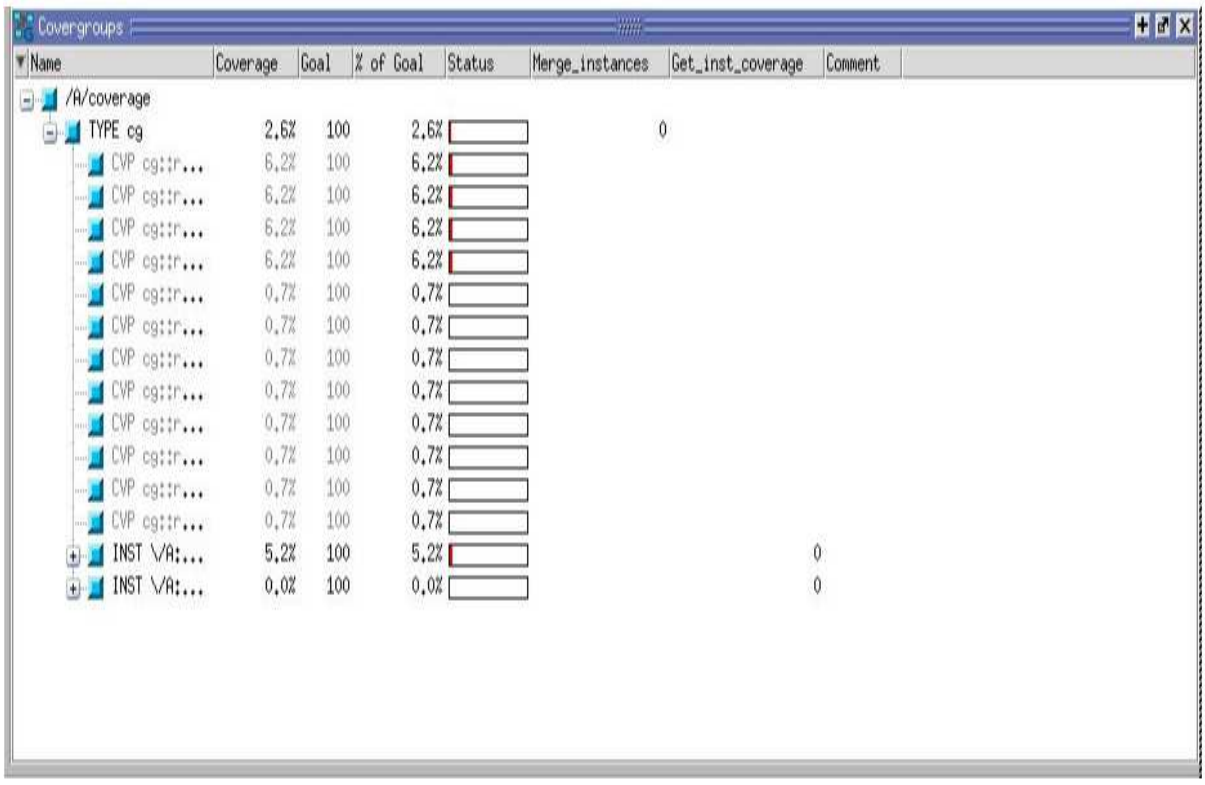

Figure 8. Coverage report of CALC2 Design

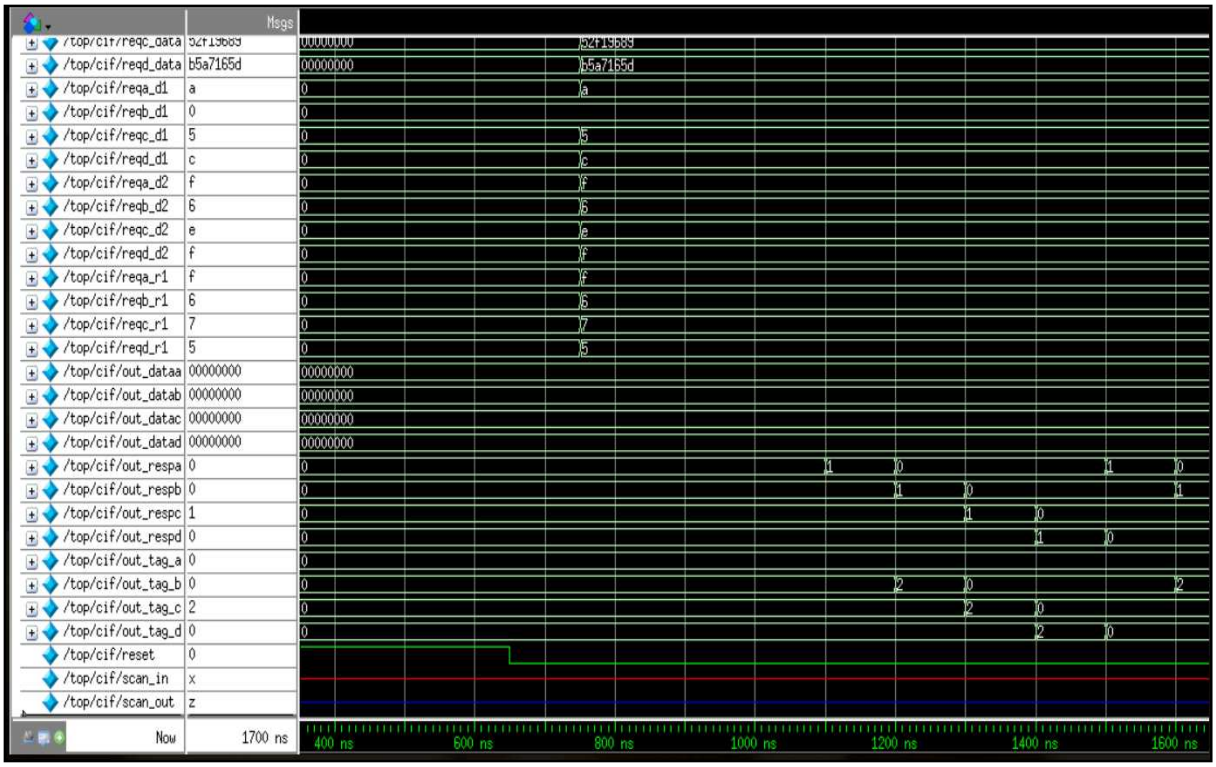

Figure 9. Simulation Result of CALC3

7. CONCLUSION

All the calculator designs CALC1, CALC2 and CALC3 are verified using test bench in system verilog using object oriented concepts. The test plan which includes the verification environment, implementation, classes used and bug report of the design are explained in the report above. Detailed test report in excel sheet “TEST_REPORT.xls” is attached with the report. Zipped folder containing test bench code along with the transcripts, waveforms and makefiles for each design are also attached. All the codes used in project are submitted as .sv files.