Functional Hardware Verification - COEN6541

Design under Verification
A simple calculator: Calc

Submitted by

Ragul Nivash Rangasamy Sekar **(40169564**)
Shivendra Arulalan **(40197455)**
Thenmozhie Rajan (**40192527**)

1. Introduction

This report shows the verification test plan and results for Calc design. Further sections go through the specs of each design that is being tested.

2. Calc: Design Specification

Calc is a simple 4-port calculator.

- Each port may have up to 4 outstanding commands at a time.
- The commands consist of add, subtract, shift left, and shift right.
- The operand data accompanies the commands.
- Since there are 4 ports, there can be up to 16 outstanding commands at once.
- Each command from a port is sent in serially (one at a time), but the calc log may respond "out of order," depending upon the internal state of the queues.
- As a result, each command is accompanied by a 2-bit tag, which identifies the command when the response is received.
- **reqX_cmd_in (0:3):**

COMMANDS		
0	0000	No operation
1	0001	Addition
2	0010	Subtraction
5	0101	Right shift
6	0110	Left Shift

- **reqX_data_in(0:31):** The operand data is sent one cycle after another. Operand1 data accompanies command, and operand2 data follows.

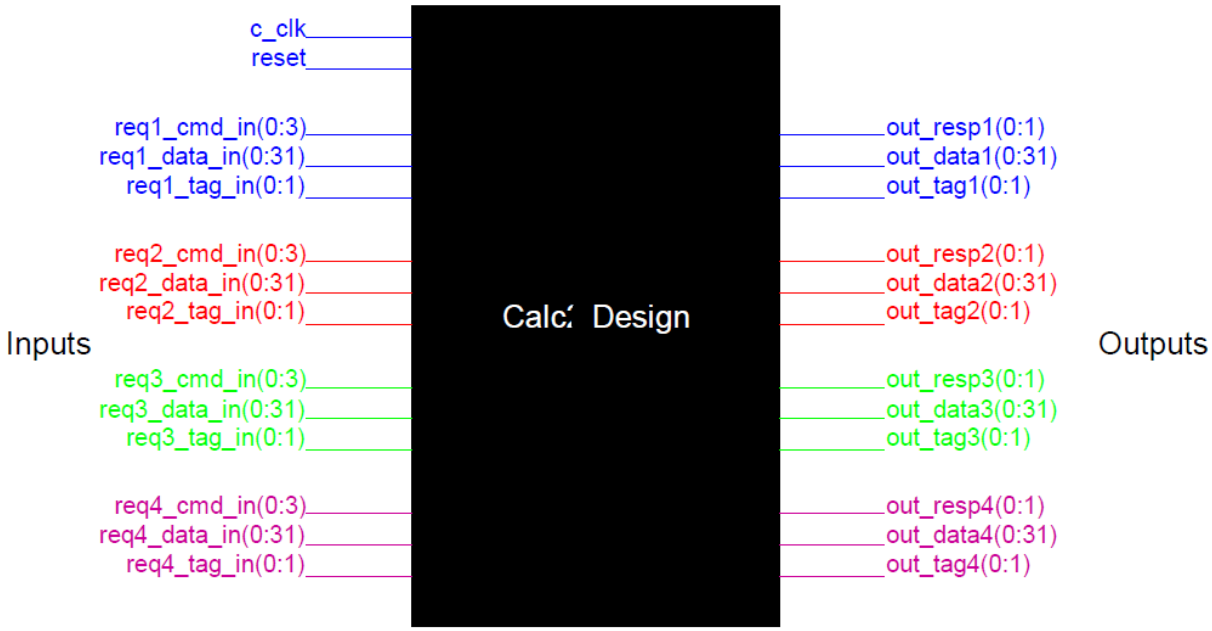
- **reqX_tag_in(0:1):** Two bit identifier for each command from the port. Can be reused as soon as the calculator responds to the command.

- **out_respX(0:1):**

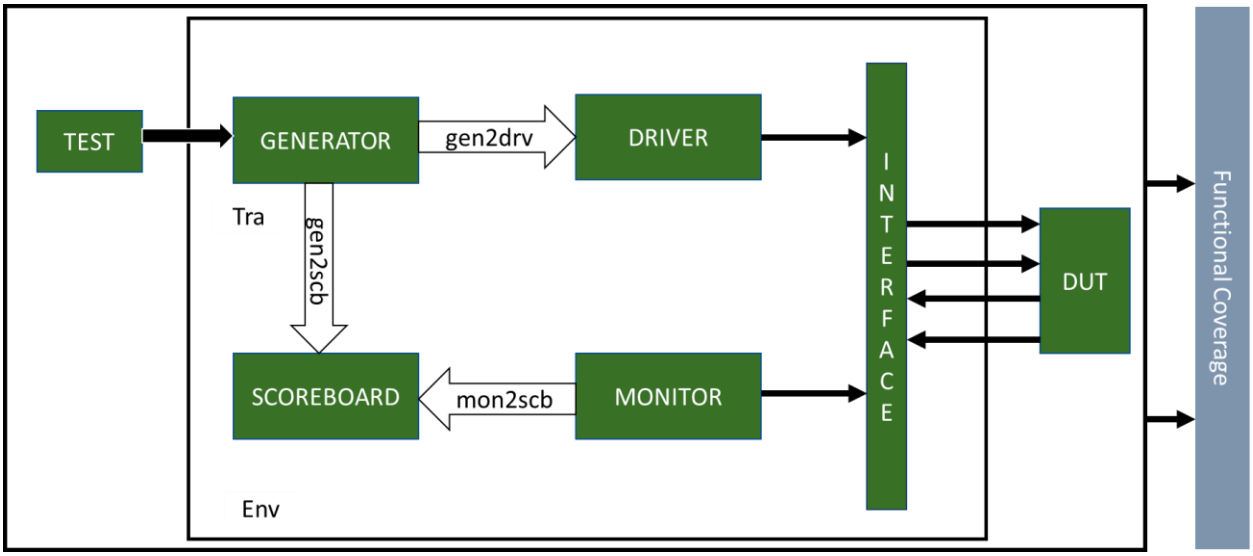
RESPONSE		
0	00	No response
1	01	Success
2	10	Invalid command or over/underflow
3	11	Unused response value

- **out_dataX(0:31):** This is the output data that accompanies a response.

- **out_tagX(0:1):** Corresponds to the command tag sent by the requester. Used to identify which command the response is for.



3. Test Plan



Class Transaction

- Declare the inputs in the transaction class
- Randomize
  - ✓ Commands
  - ✓ Tag
  - ✓ Input data

DUT_out

- All the outputs are declared

Class Generator

- Create ‘max_trans_cnt’ number of transactions packets.
- Generate random stimulus for input.
- Use coverage function to send generated random data.
- Put transactions in mailbox gen2drv.
- Put transaction in mailbox gen2scb.

Class Driver

- Receive the randomizes transactions (from generator) through mailbox gen2drv.
- Pass the data to DUT through interface.

Interface

- Declare all the input and output with logic datatype.
- Connects Driver and monitor to DUT.
- Clocking Block for driver and monitor helps in synchronization of signals.
- The modport is used to group signals and specify directions.

Class Monitor

- Receive the processed data from DUT through interface.
- Put the transaction in mailbox mon2scb.

Class Scoreboard

- Get the transaction from mailbox gen2scb.
- Get the transaction from mailbox mon2scb.
- Calculate the expected data from gen2scb and compare with DUT output data from mon2scb.
- Display Error when the expected data and the actual data from DUT does not match.
- Display Success when the expected data and the actual data from DUT matches.

Class Environment

- It is a container class
- Create a random number for max transaction count.
- Instantiate all the classes and mailbox.
- Run method is defined which has pre_test, test, post_test tasks.

Testbench

- Instantiate the Environment class.
- Start the test.

Top module

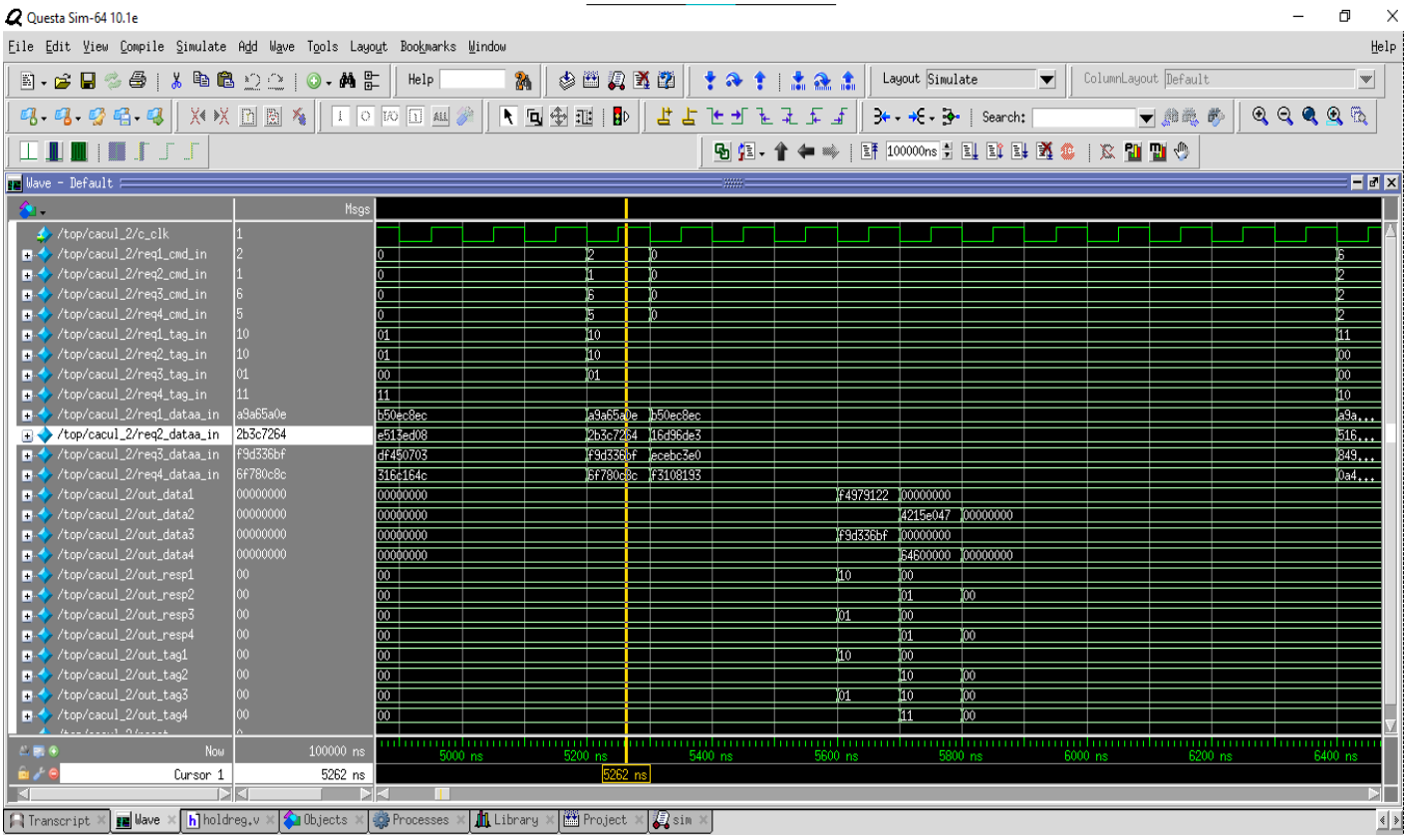
- It is the topmost file that connects the testbench and DUT.
- Set clock frequency.

Functional Coverage

- Used to measure how much of the design specification is measured in testbench.

- Covergroups are created with bins.
- Data generated is sampled.

Output Waveform:



Functional Coverage:

coverage	PA(fast)	SVClass	+acc=<full>	50.0%	50.0%
new	PA(fast)	Function	+acc=<full>		
#covgr#	PA(fast)	SVCovergroup	+acc=<full>	50.0%	50.0%
req1_cmd_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req2_cmd_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req3_cmd_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req4_cmd_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req1_dataa_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req1_datab_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req2_dataa_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req2_datab_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req3_dataa_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req3_datab_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req4_dataa_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
req4_datab_in	PA(fast)	SVCoverpoint	+acc=<full>	100.0%	100.0%
funcov	PA(fast)	Function	+acc=<full>		

Bug Report:

Bug#	Test	Explanation
1	Addition to port 1,2,3,4	Addition works well for all the ports but sometimes without even overflow the **out_respX is** 2 and **out_dataX** is 0.
2	Subtraction to port 1,2,3,4	Subtraction works for all ports but sometimes without even underflow the **out_respX is** 2 and **out_dataX is 0 .**
3	Left shift to port 1,2,3,4	Left shift operation performs well for all ports but sometimes **out_dataX** is set to 0 and **out_respX is** 2
4	Right shift to port 1,2,3,4	Right shift operation performs well for all ports but sometimes **out_dataX** is set to 0 and **out_respX is** 2.
5	Left shift	As per test_case no.4 there is an error in the 15th bit of output line which is set to 0.
6	No operation	No operation gives response as 2 whereas correct response should be 0.
7	Addition overflow	Addition overflow produces correct response
8	Invalid operation	Invalid operation sometimes gives wrong response. i.e., response will be 1 whereas the correct response is 2 as per test case no5.
9	Subtraction underflow	Subtraction underflow produces correct response.
10	Addition overflow	Addition overflow produces correct response

Corner case testing:

- ✓ Sent a mix of add and subtract commands.
- ✓ Sent only a mix of shift commands.
- ✓ Verified mix of overflow, underflow, and good response.
- ✓ Verified all lengths of shift cases.
- ✓ Check output value matches expected value.
- ✓ Check every command gets response.
- ✓ Check output data matches expected result based on command and data.
- ✓ Check the response tag matches the data for the command that was sent.
- ✓ Send command using variable tags for each command.
- ✓ Send multiple invalid command.

4. Task Distribution

S.No	SystemVerilog Code	Shivendra Arulalan	Ragul Nivash	Thenmozhie Rajan
1	Test Plan/Transaction/Generator	✓		
2	Driver/Monitor/Interface/Scoreboard			✓
3	Environment/Functional Coverage/Top/Bug report		✓	

5. Conclusion

A simple 4-port calculator design was verified using SystemVerilog. The test plan, functional coverage, output, and bug report are discussed in the report.