Verification Plan

Calc 2 Design

Design Ref: Comprehensive Functional Verification (Text)

ECE 506 Research Project

Sukrut Kelkar

938621692

Design under Verification:

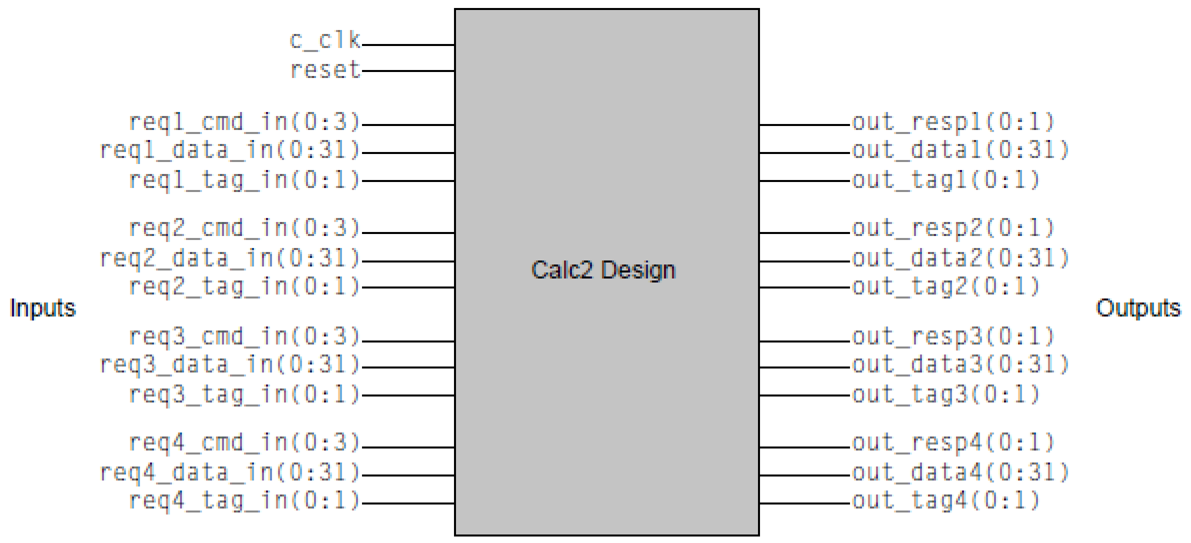

Fig 1: The input output signals for Calc 2 design grouped by ports.

Description:

The design is a calculator having three basic functions. Add, Subtract and shit. This design is based on the original Calc 1 design which also had multiple ports. The main difference between the second version i.e. our DUV is that this can handle up to 4 requests per port and has two pipelines for shift and Add/Subtract operations each. The base design consists of input ports, 2 ALU in which one is for Add/Subtract and one is from Shift operations. This design has a pipeline queue which can hold up 16 commands. It means that each port can send up to four commands without receiving an acknowledgement for the same. In order execution takes place for commands in one queue and out-of-order execution can take place between both the queues. The design has few timing constraints like the command comes first with the first operand. The second operand follows the next clock cycle. Up to 4 commands can be consecutively sent from each port. These commands queue up in the pipeline according to the type of command i.e. add/subtract or shift operations. The design has one output port through which we receive the response signal, the output data signal and the tag signal. This design has an unique tag identifier for each of the input on each of the ports.

Verification Levels:

The design requires to be tested on two different levels. The first one is unit level in which there are 3 different units that require extensive testing. First one and the most important one is the priority logic unit testing. This is the most complicated part of the design. The other two units to be tested are the 2 ALU units. As this design is based on the Calc 1 design it can be assumed that the testing of the two ALU is complete as the same modules are used again for this design. However, in case needed these units can be tested again. The second level of testing is the Top level which will be our focus in verifying this system.

Required tools:

Single/Team of verification engineers. Simulation work stations for the team or the individual engineer. As the Calc2 design involves complexity it needs to have and high level verification language. If a team is involved in this verification a platform is required to share the source codes and documents.

Risks and dependencies:

Verification of the priority unit requires a unit level specification including the input and output specifications. Chip level verification requires completion of the unit level verification. It is very important to have the required tool and license availability for the same.

Functions to be verified:

Basic Tests on the functionality of the design:

Calc2 basic function tests

Test reference number	Test Description
1.1	Check the basic command-response protocol on each of the four ports.
1.2	Check the basic operation of each command on each port.
1.3	Check overflow and underflow cases for add and subtract commands.

Calc2 advanced function tests

Test reference number	Test description
2.1.1	For each port, check that each command can have any command follow it without leaving the state of the design dirty, such that the following command is corrupted, such that the following command is corrupted.

2.1.2	Across all ports (e.g., four concurrent adds do not interfere with each other), check that each command can have any command follow it without leaving the state of the design dirty, such that the following command is corrupted, such that the following command is corrupted.
2.2	Check that there is fairness across all four ports such that no port has higher priority than the others.
2.3	Check that the high-order 27 bits are ignored in the second operand of both shift commands.
2.4.1	Data dependent corner case: Add two numbers that overflow by 1 (“FFFFFFFF”X + 1).
2.4.2	Data dependent corner case: Add two numbers whose sum is “FFFFFFFF”X.
2.4.3	Data dependent corner case: Subtract two equal numbers.
2.4.4	Data dependent corner case: Subtract a number that underflows by 1 (Operand2 is one greater than Operand1).
2.4.5	Data dependent corner case: Shift 0 places (should return Operand1 unchanged).
2.4.6	Data dependent corner case: Shift 31 places (the max allowable shift places).
2.5	Check that the design ignores data inputs unless the data are supposed to be valid (concurrent with the command and the following cycle). Remember that “00000000”X is a data value just as any other 32- bit combination. Here, the check must include verifying that the design latches the data only when appropriate, and does not key off nonzero data.

Generic tests and checks

Test reference number	Test Description
3.1	Check that the design correctly handles illegal commands.
3.2	Check all outputs all of the time. Calc1 should not generate superfluous output values.
3.3	Check that the reset function correctly resets the design.

Tests to be done other than the basic tests:

Calc 2 expansin test cases:

Test reference number	Test Description
4.1	Send Multiple commands with variable timing between commands from the same port.
4.2	Send commands using variable tags for each command.
4.3	Send multiple invalid commands

Calc2 corner case tests cases

Test reference number	Test Description
5.1	Send only a mix of add and subtract commands to fill the add queue.
5.2	Send only a mix of shift commands to fill the shift queue.
5.3	Verify mixes of overflow, underflow and good response cases across back to back port commands.
5.4	Verify all lengths of shift cases across back to back port commands.

5.5	Verify the DUV does not allow output collisions from both pipelines sending results to the same port simultaneously.

All lengths of shift operands should be tested but it is not necessary to test all operand combinations for the add/subtract operations.

Specific checks for interesting cases:

Test reference number	Test Description
6.1	Check that the response value matches expected response based on command and data.
6.2	Check that every command gets response.
6.3	Check for unmatched tags on the command port.
6.4	Check that result data matches expected result based on command and data.
6.5	Check for correctness of out-of-order responses across command pipeline types but never across the same command type.
6.6	Verify in-order execution of all adds/subtracts or shifts, no matter which port sent the command.
6.7	Check that the response tag matches the data for the command that was sent.
6.8	Check that there are no unexpected or stray values on the output.

Specific tests and methods: Environment

Type of verification: Grey Box

Most of the verification will happen at the top level. The checkers and monitors will also operate on top level but as this design has queues in it some of the verification will happen inside the design to monitor the internal states of the queues.

Verification environment: Random based verification. (Generation of random values places less burden on the test writer and also provides a broader range of data and control stimulus.)

Although early simulation will be restricted to simple scenarios to test the basic functionality of the design.

Randomization controls:

To have control over the randomization it needs to have some constrains on random values of:

- Operand Data
- Command types
- Tag values
- Delay between commands

The same constrained random environments of one ports can be used for testing all the ports. The verification environment should be in such a way that it gives a statistical measurement of how many cycles each operation took.

Output checking:

A transaction based checking approach will provide efficient result to a packet-driven stimulus as each and every transaction verification will yield us results. Using this model we can verify:

- Responses have corresponding tag values in scoreboard.
- Responses have correct values based on command and data in scoreboard.
- Output of correct data on successful responses.
- No noise on the outputs. (Output signal zero unless a valid response)
- Maintain order with each of the ALU queues.

The environment will have separate monitors to track the state of the queues. The verification team can use this information for coverage feedback as well as checking that the queue does not overflow.

Test Scenarios: Matrix

The team must run few of the basic tests noted in the table below on each and every port to test the basic functionality of all the ports.

Calc2 Basic test matrix:

Function	Port 1	Port 2	Port 3	Port 4
Basic add command				
Basic subtract command				
Basic shift left				
Basic shift right				
All four tags				
Variable timing between commands				
Fill add queue (All four ports involved)				
Fill shift queue (All four ports involved)				
Invalid commands				
Overflow and underflow cases				
Various lengths of shift values				

Coverage requirements:

There are many things that need to be covered w.r.t this design. Few of the Models that should be covered are:

Model 1: Number of commands in each queue.

Model 2: Permutation of ports at the top of each queue.

We can also check the coverage to check if all the commands are covered or not and Filling of the pipeline to its optimum capacity.

Resource requirement:

One single engineer with a work station and the required tools, simulation engines, compilers and debuggers.

Schedule details:

A single engineer might require about 40 hours of total work time. The schedule will include working 8 hours per week. Looking at the time frame the verification must be finished in about 5 weeks.

Week no.	Task
Week 1	Studying the design thoroughly and structuring tasks according what all things need to be done.
Week 2	Creating a test environment for verifying basic functions.
Week 3	Designing test cases for advanced tasks.
Week 4	Verifying the whole top level system using a well-built test environment
Week 5	Confirming the verification of the whole design from the coverage, noting down the bugs and submitting a final report.

References:

- Textbook: Comprehensive functional verification.
- ECE 510: Pre-silicon Functional verification class notes.