Table of Contents

List of Figures

Calc 2: An Introduction to the Design Under Verification

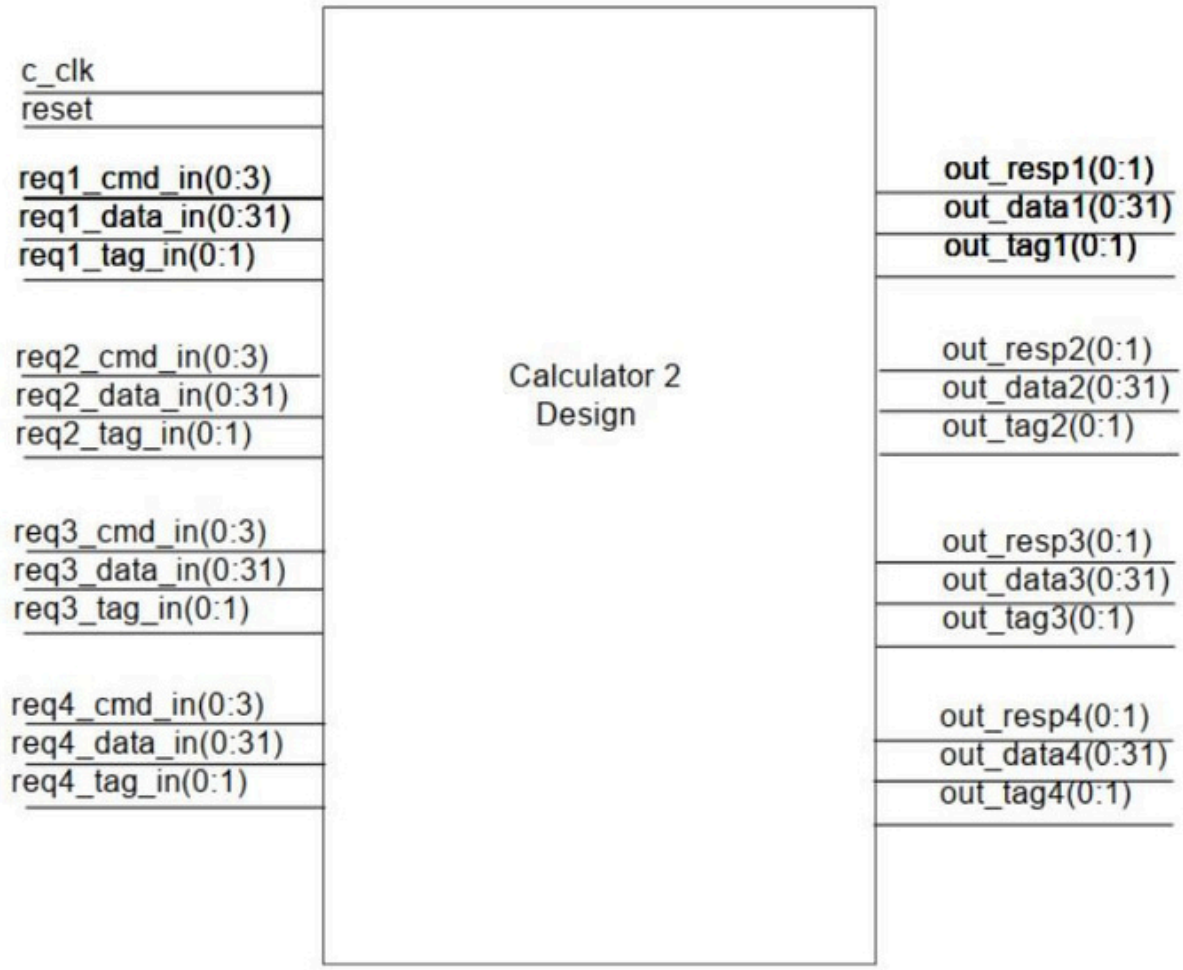

Figure 1: Calc 2 Pin Diagram

Calc 2 offers an improvement over the initial Calc 1 calculator design by introducing "tags" for each of the four sets of ports. A tag is a two-bit value that associates a set of input signals with their respective output signals over a port pair. This allows the calculator to queue up to four transactions on each port irrespective of timing considerations, because a sound design in this case will keep the input data and output data married together via their shared tag.

Familiar Calc 1 features are also present. The device is still a 32-bit, two-operand, four-port calculator with add, subtract, and left/right shift functionalities. Command opcodes are 4-bit values that control data flow through one of two ALUs inside the calculator. Data buses are driven over two clock cycles to deliver each of the two operands, with branching behaviour depending on the operation. At the output, a response signal alerts the environment to the result status of each port, with corresponding output data (if applicable) alongside. The tag functionality introduced in the Calc 2 design now allows multiple commands to be fed to each port instead of waiting for transactions to clear one at a time.

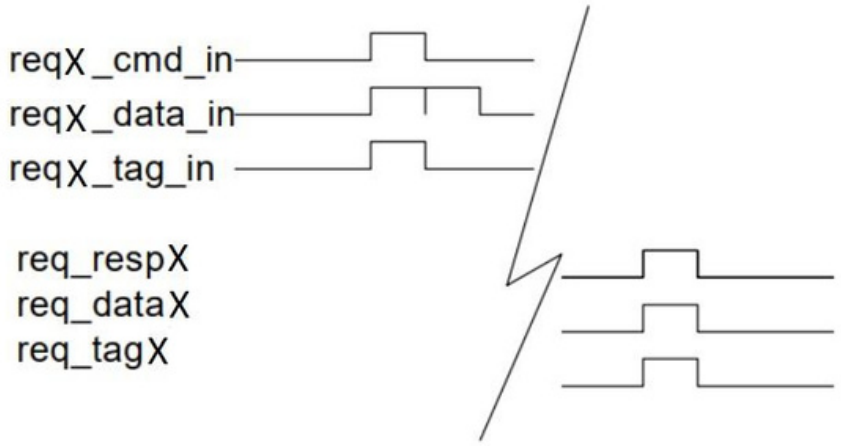

Figure 2: Calc 2 Sample Transaction Timing Diagram

Calc 2 Project: Verification Environment

The verification environment is composed of several discrete, intercommunicating modules responsible for managing the data flow into and out of the DUT. The inner abstraction layer consists of standard and hybrid UVM-style verification classes.

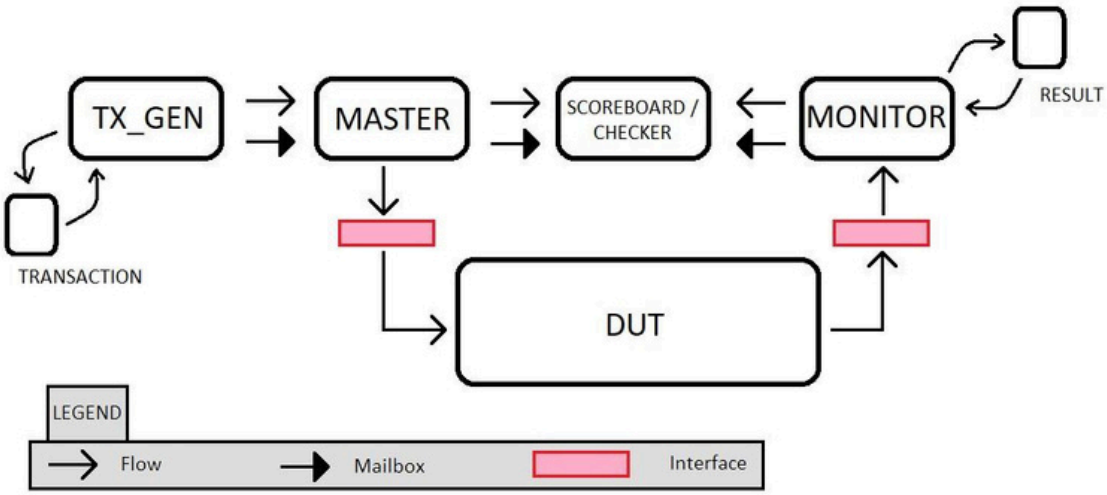

Figure 3: Calc 2 Verification Environment - Transactor Layer

Component	Primary Objects and Variables	Primary Methods
Generator	• Instantiated Transaction object • Mailbox w/ Master • Opcode Constraints	• Randomize instances of transaction object • Drive transactions to Master
Master (includes Agent)	• Mailboxes w/ Generator and Scoreboard • Drive-delay Constraints (x2)	• Pull transactions from Generator • Drive transactions to DUT • Drive transactions to Scoreboard • Reset the DUT
Scoreboard (includes Checker)	• Inbound Transaction and Result objects • Mailboxes w/ Master and Monitor • Input and output covergroups	• Pull Transactions from Master and determine correct output • Pull Results from Monitor and compare with correct output • Present or display test results • Effect functional coverage analysis
Monitor	• Instantiated Result object • Mailbox w/ Scoreboard	• Check DUT port status and pull completed transactions • Create Result objects from stimulus responses and drive them to the Scoreboard

At the environment abstraction layer, the environment is accompanied by the two classes for data structures (Transactions and Results). These are wrapped in a top level layer that also includes the clock and a testbench class that instantiates the environment.

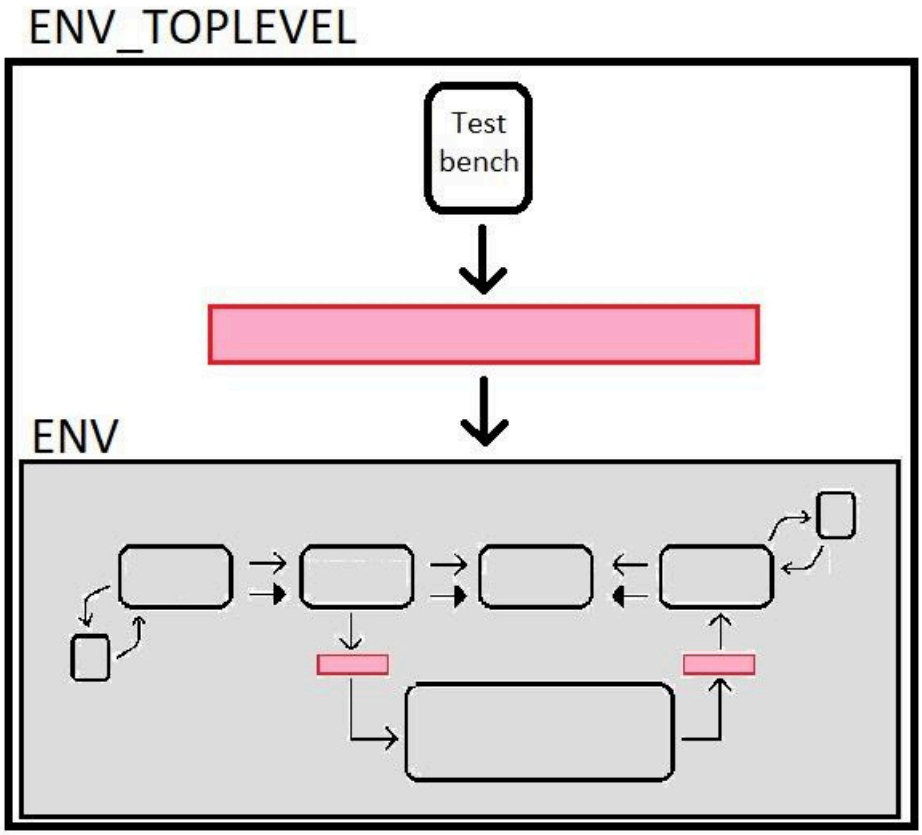

Figure 4: Calc 2 Verification Environment - Top Layer

Additional constraints for data operand values are located in the Transaction class, and for the volume (number) of transactions per test iteration in the transactor-level Environment class.

Calc 2: Test Plan

Specification/Feature	Reference	Test Points *	Test Scenarios	Expected Results
Check basic port responses	1.1	Top-level command and response	Check responses on every port by cycling through every valid command Match expected output with monitored output over valid opcodes	Proper responses: no response, successful, Overflow/Underflow, unused response value, error for unidentified response
Check basic operations of each command on each port	1.2	ALU1, ALU2, adder, shifter, output ports	All ports: force overflow and underflow conditions (with ~5% properly bounded test inputs, for comparison)	Expected values match calculated values and DUT shows working arithmetics
Add/subtract overflow and underflow	1.3	Top-level over/underflow, ALU1, adder		Response = 10 on all ports
Commands do not corrupt following commands (serial)	2.1.1	I/O Ports, Priority logic	Storm the ALUs with commands, one port at a time, one command after the other, with zero delay and variable tags.	Port response and output should match long delay results
Commands do not corrupt following commands (parallel)	2.1.2	I/O Ports, Priority logic	Storm the ALUs with commands into two ports simultaneously, one command after the other, with varying delay and tags.	Port response and output should match long delay results
Priority Logic Fairness: All ports have equal priority	2.2	I/O Ports, Priority logic	Perform fully unconstrained random testing across all ports to observe behaviour	Random port access; commands execute consecutively and according to their ID tag

Specification/Feature	Reference	Test Points *	Test Scenarios	Expected Results
Shift commands ignore operand 2 MSBs	2.3	I/O Ports, ALU2, Shifter	Test any and all MSBs: they should not interfere with performing shift operations	For any values of MSBs, both shift operations across all ports work normally
Data dependent corner case: Addition with overflow of one	2.4.1	Top-level overflow, I/O ports, ALU1, Adder	Constrained random: Set (op1 + op 2) equal to (ffffffff + 1), along with appropriate addition command	Expected overflow: resp = 10 on all ports. If output is sampled, it should read h'00000000
Data dependent corner case: Addition with max sum	2.4.2	I/O ports, ALU1, Adder	Constrained random: Set op 2 equal to (ffffffff - op1), along with appropriate addition command	Response = 01 and Output = ffffffff on all ports
Data dependent corner case: Subtract two equal numbers	2.4.3	I/O ports, ALU1, Adder	Constrained random: Set op1 & op 2 equal to each other, along with appropriate subtraction command	Response = 01 and Output = 0 on all ports
Data dependent corner case: Subtraction with underflow of one	2.4.4	Top-level underflow, I/O ports, ALU1, Adder	Constrained random: Set op 2 equal to (op1 + 1), along with appropriate subtraction command	Expected underflow: resp = 10 on all ports. If output is sampled, it should read h'ffffffff
Data dependent corner case: Zero shift	2.4.5	I/O ports, ALU2, Shifter	Perform left and right shift with zero places	Data_out[i] = op1 for any port i
Data dependent corner case: Max shift	2.4.6	I/O ports, ALU2, Shifter	Perform left and right shift with max (31) places	LSB becomes MSB, or MSB becomes LSB, depending on shift direction, with remaining output zeroed, for all ports

Specification/Feature	Reference	Test Points *	Test Scenarios	Expected Results
Data dependent corner case: continuously switching response states	2.4.7	Top-level over/underflow, I/O ports, ALU1, Adder, Priority logic	Constrained random: overflow, underflow, and good response states	Outputs and responses should not depend on / be affected by previous outputs / responses, even (especially) if they are different.
Command dependent corner case: consecutive add and subtract requests	2.4.8	I/O ports, ALU1, Adder, Priority logic	Storm ALU1 with commands, all ports simultaneously, with minimal delay	Requests are handled correctly with matching tags intact
Command dependent corner case: consecutive SHL and SHR requests	2.4.9	I/O ports, ALU2, Shifter, Priority logic	Storm ALU2 with commands, all ports simultaneously, with minimal delay	Requests are handled correctly with matching tags intact
Cross-ALU data isolation	2.5	I/O ports, ALU1, ALU2, Adder, Shifter, Priority Logic	Attempt to drive a port from both ALUs at the same time	No output collisions from ALU1, ALU2 both trying to drive the same port
Invalid Operands: 4-state Inputs	2.6	I/O Ports, Priority logic	Performing operations using 'Z' and 'X' as inputs	resp[i] = 00 and data_out[i] = 0 for any port i
Illegal Commands are Handled Properly	3.1	Response ports, Priority logic	Check response of all 12 unused commands	All ports produce resp = 10 and output = 0 for all illegal commands
No Superfluous Output at Inactive Ports	3.2	Output ports	Send valid commands to one port, and check output of all ports	Output only appears on output port matched to input port

Specification/Feature	Reference	Test Points *	Test Scenarios	Expected Results
Reset Functionality is Working	3.3	Top-level reset, Output ports	Reset after environment loads	Data and response ports cleared
Generated tag fidelity	4.1.1	I/O Ports, Priority logic	(Nearly) unconstrained random testing	Transaction tag matches data sent
Outgoing tag fidelity	4.1.2	I/O Ports, Priority logic	(Nearly) unconstrained random testing	Result tag from monitor (slave) matches master data
One-to-one tag referencing	4.1.3	I/O Ports, Priority logic	(Nearly) unconstrained random testing	All tags are matched: no data lacks ID at output
Out-of-order responses are handled correctly	4.2	I/O Ports, Priority logic	(Nearly) unconstrained random testing	Tag-matched transactions are handled correctly at every port regardless of when they arrive

NOTES * MUX not listed

Calc 2: Test Cases

Specification/Feature	Reference	Proposed Test Scenarios	Test Cases Employed
Basic port responses	1.1	check responses on every port by cycling through every valid command	Test all legal commands plus one illegal command 1111; test all illegal commands; both on all ports
Check basic operations of each command on each port	1.2	match expected output with actual output across all ports	Test all arithmetic on all ports, without timing/parallelism concerns, with constrained inputs
Add/subtract overflow and underflow	1.3	All ports: force overflow and underflow conditions	2500 cases @ 95% over/underflow // 5% legal result on all ports
Commands do not corrupt following commands (serial)	2.1.1	Unconstrained random data with all valid opcodes and zero delay; separate attack for each port	2500 cases with even split between valid opcodes; separate attack per port with other ports zeroed (and therefore not sampled). Variable delay. Variable tags.
Commands do not corrupt following commands (parallel)	2.1.2	Feed the ALU commands into 2 different ports at the same clock edges with varying delays between transactions	All valid opcodes for all permutations of (N = 2) ports. Variable delay. Variable tags.
Priority Logic Fairness: All ports have equal priority	2.2	Perform arbitrary valid command in parallel across all ports for extended operation time, with varying delays between transactions	Unconstrained random data; random valid opcodes commands with tracking of results. Several tests with several sets of data should be used to obviate the set random seed. Variable delay.
Shift commands ignore operand 2 MSBs	2.3	Set op2 MSBs randomly with constrained LSBs and perform shift operations	2500 cases of unconstrained random data1; constrained data2 LSBs = 00100, or 01000, or 01100 for easy perusal

Specification/Feature	**Reference**	**Proposed Test Scenarios**	**Test Cases Employed** 2500 cases of
Data dependent corner case: Addition with overflow of one	2.4.1	Add op1 & op2 to achieve an overflow of 1	constrained data2 = (ffffffff - data1 + 1), and then data1 + data2. Verify on all ports.
Data dependent corner case: Addition with max sum	2.4.2	Add op1 & op2 to have the maximum sum of FFFFFFFF	2500 cases of constrained data2 = (ffffffff - data1), and then data1 + data2. Verify on all ports.
Data dependent corner case: Subtract two equal numbers	2.4.3	set op1 & op 2 equal to each other and subtract	2500 cases of constrained data2 = data1, and then data1 - data2. Verify on all ports.
Data dependent corner case: Subtraction with underflow of one	2.4.4	set op 2 = (op1 + 1) and subtract	2500 cases of underflow by 1 on all data_in ports and verify on all response ports
Data dependent corner case: Zero shift	2.4.5	Perform left and right shift with zero places	2500 cases each of unconstrained random data 1; data2 = ???00000
Data dependent corner case: 31 shift	2.4.6	Perform left and right shift with max places	2500 cases each of unconstrained random data 1; data2 = ???fffff
Over/underflow and good response blitz	2.4.7	Storm all ports with dramatically changing states that produce one of these three responses	Constrained, conditional random testing to enable desired state and response (and rate of change)
ALU1 blitz (add/subtract)	2.4.8	Stress test ALU1 with rapid consecutive add and subtract commands across all ports	Constrained random testing with cmd == {0001, 0010}
ALU2 blitz (SHL/SHR)	2.4.9	Stress test ALU2 with rapid consecutive SHL and SHR commands across all ports	Constrained random testing with cmd == {0101, 0110}
Cross-ALU data isolation	2.5	Stress test both ALUs with rapid consecutive valid opcodes across all ports	(Nearly) unconstrained random testing
Four-State Inputs	2.6	Performing operations using 'Z' and 'X' as inputs	Test all operations in parallel across all ports using as input 'X' and 'Z', with varying delay

Specification/Feature	**Reference**	**Proposed Test Scenarios** Check	**Test Cases Employed**
Illegal Commands are Handled Properly	3.1	response of all 12 unused commands with different delays	Test every illegal command on every port and observe behaviour
No Superfluous Output at Inactive Ports	3.2	Send valid commands to one port, and check output of all ports	Test every valid command on every port and observe behaviour
Reset Functionality is Working	3.3	Reset before booting main() methods	Instantiation of DUT followed by reset; check all output and response ports
Generated tag fidelity	4.1.1	Fire many random (valid) tag values with varying delay and observe response	(Nearly) unconstrained random testing
Outgoing tag fidelity	4.1.2	Fire many random (valid) tag values with varying delay and observe response	(Nearly) unconstrained random testing
One-to-one tag referencing	4.1.3	Fire many random (valid) tag values with varying delay and observe response	(Nearly) unconstrained random testing
Out-of-order response handling	4.2	Fire many transactions with varying delay and ensure tag-matched results remain intact	(Nearly) unconstrained random testing

Calc 2: Bug Report

Specification/Feature	Reference	Expected Results	Bugs Identified
Check Responses	1.1	No command 0000 gives resp = 00 on all ports	None
		all valid commands (with valid results) give resp = 01 on all ports	None
		all invalid commands gives resp = 10 on all ports	Irregular *
Basic Arithmetic operations	1.2	Add: Expected = Monitored (all ports)	None
		Sub: Expected = Monitored (all ports)	None
		SHL: Expected = Monitored (all ports)	The 16th bit of the output (counting from LSB) is always zeroed on all ports
		SHR: Expected = Monitored (all ports)	None
Overflow / Underflow	1.3	Additon overflow: response = 10 on all ports	None
		Valid addition during overflow dump still produces response = 01 on all ports	None
		Subtraction underflow: response = 10 on all ports	None
		Valid subtraction during underflow dump still produces response = 01 on all ports	None
Serial Command Attack	2.1.1	Appropriate response and output on active port regardless of opcode, data, or delay	Irregular *

Specification/Feature	Reference	Expected Results	Bugs Identified
Parallel Command Attack	2.1.2	Appropriate response and output on active port regardless of opcode, data, or delay	None **
Priority check	2.2	Appropriate response and output on every active port with tag integrity preserved regardless of opcode, data, or delay	For threshold delay test batteries EITHER: ports 1 and 2 have priority at the start of the battery OR port 3 and/or 4 reverse the order of two of their first few transactions. See also Notes 1* and 2** .
Shift ignores op2 MSB	2.3	resp = 01 and expected = monitored, all ports	None
Addition - Overflow by 1	2.4.1	resp = 10 on all ports	None
		Extension testing: The (somewhat worthless) output is 00000000 on all ports resp = 01, data_out = ffffffff on all ports	None
Addition - Max Sum	2.4.2	resp = 01, data_out = 0 on all ports	None
Subtraction - Equality case	2.4.3	resp = 10 on all ports	None
Subtraction - Underflow by one	2.4.4	Extension testing: The (somewhat worthless) output is ffffffff on all ports SHL: Response = 01;	None
		output = data1 (for all ports) SHR: Response = 01;	None
Shift zero	2.4.5	output = data1 (for all ports) SHL: Response = 01, output = 80000000 or 00000000, depending (for all ports)	Indirect ***
		SHR: Response = 01, output = 00000001 or 00000000, depending (for all ports)	None
Shift max	2.4.6		None

Specification/Feature	Reference	Expected Results	Bugs Identified
Over/underflow; good response blitz	2.4.7	Standard operation regardless of request pattern / timing	None
ALU1 (add/subtract) blitz	2.4.8	High delay: standard operation regardless of request pattern / timing	None
		Low delay: standard operation regardless of request pattern / timing	None
ALU2 (SHL/SHR) blitz	2.4.9	High delay: standard operation regardless of request pattern / timing	Indirect ***
		Low delay: standard operation regardless of request pattern / timing	Indirect ***
Cross-ALU Data Isolation	2.5	Cross-ALU responses produce valid output and no data collisions	Certain timing patterns result in data collisions and device state lockup. Direct testing can locate problematic patterns; one example is a guaranteed collision when there is a delay of 3n cycles between transaction transmissions
Invalid Operands (Four-State Inputs)	2.6	Response = 00 and Output = 0 on all ports	None
Illegal Command Handling	3.1	High delay: All illegal commands produce resp = 10 and output = 0 on all ports	None
		Low delay: All illegal commands produce resp = 10 and output = 0 on all ports	None
No Superfluous Output	3.2	~~Output = 0 for inactive ports~~	None, but see 4.1.3
		Proper output for active input	None

Specification/Feature	Reference	Expected Results DUT	Bugs Identified
Reset Functionality	3.3	resets properly after loading test environment	None
Generated Tag fidelity	4.1.1	Input tag remains with data regardless of port or arrival time	None, but see 4.1.3
Outgoing Tag fidelity	4.1.2	Response tag matches input transaction tag regardless of port or arrival time	None, but see 4.1.3
One-to-one Tag referencing	4.1.3	Input data always has a valid associated tag	Unmatched tag values detected at beginning of transaction train.
		Output data always has a valid associated tag	Valid tags with no associated data detected at the end of transaction train, particularly for larger N
Out-of-order (OOO) response handling	4.2	OOO responses in ALU1 (add, subtract) are correct	Indirect ***
		OOO responses in ALU2 (SHL, SHR) are correct	Indirect ***, especially in ALU2
		OOO responses during tandem ALU usage are correct	Indirect ***, especially in ALU2

NOTES

* Ports with a known bad operation (i.e. resp should not be 01) sometimes give false positive resp=01 and corrupted data on the fourth transaction. This doesn't seem to affect earlier or later transaction tests and its cause is unknown.

** Threshold delays (2 cycles or less between transactions) cause on-port data collisions in ALUs, especially ALU2 when waiting for shifts to complete. This is because this period moves faster than the transactions are driven, and is therefore expected behaviour.

*** The irregular behaviour observed in e.g., Cases 1.2 and 2.2 serve to obfuscate the results of this test.

References

1. Spear, C. and Tumbush, G., *System Verilog for Verification*, 3rd ed. New York, NY: Springer, 2012, various sections particularly those covered in lecture.

2. Wile, B., Goss, J., and Roesner, W., *Comprehensive Functional Verification*, San Francisco, CA: Elsevier, 2005, ch. 4-8.