Verification Report

Overview

- Number of blocks: 31
- Number of assertions: 1716
- Number of assumptions: 234
- Lines of Code: >98 k
- Reported Bugs/Enhancements: 22

This report outlines the formal verification effort and sign-off process.

The goal of the verification effort was to verify the correct functional behavior of the design by using System Verilog assertions. Assumptions ensure that only realistic input stimuli are considered by these assertions. The goal of the sign-off process was to guarantee a thorough exploration of the design behavior by the created property suite. Formal coverage analysis and bound analysis are two examples that were used to achieve this goal.

Formal Effort

The table below shows the summary of the conducted formal verification activities for each block.Its columns contain the following information

- Version of the adamsbridge.
- Module/Block name which has a dedicated Assertion IP.
- The number of assertions implemented
- The number of assumptions and constraints applied
- The Assertion IP (AIP) Lines of Code (LoC) developed
- Any bugs or enhancements identified during verification, along with their corresponding GitHub issue IDs

The goal of this detailed breakdown is to provide information about the verification scope and outcome in a transparent way

	Version	Block	# Assertions	# of Assumptions	AIP LoC	# Bugs/Enhancements	Issue ID
1	1.0	sha3/keccak Round	10	1	722	0	
2	1.0	sha3/keccak	45	35	2262	3	#127,#126,#128
3	1.0	adamsbridge_ctrl	872	60	46178	6	#85, #78, #64, #55,#43,#46
	1.0	sample_in_bal	21	1	2056	1	#62

4		l_ctrl					
5	1.0	exp_mask_ctrl	1	0	166	0	
6	1.0	rej_bounded_ctrl	12	1	1744	0	
7	1.0	rej_sampler_ctrl	12	1	1280	0	
8	1.0	abr_piso	13	3	682		
9	1.0	sib_mem	4	4	156	0	
10	1.0	ntt_shuffle_buffer	3	13	656	1	#93
11	1.0	ntt_ctrl	170	32	11414	3	#89, #86,#90
12	1.0	ntt_butterfly	6	3	378	0	
13	1.0	ntt_hybrid_butterfly_2x2	10	5	660	0	
14	1.0	ntt_masked_BFU_add_sub	6	2	175	0	
15	1.0	ntt_masked_BFU_mult	12	1	228	0	
16	1.0	ntt_masked_butterfly1x2	2	1	185	0	
17	1.0	ntt_masked_gs_butterfly	2	1	115	0	
18	1.0	ntt_masked_pwm	3	2	129	0	
19	1.0	power2round_top	34	2	2124	0	
20	1.0	decompose	37	6	3420	2	#87, #41
21	1.0	skencode	34	1	3194	0	
22	1.0	skdecode_top	124	8	6942	0	
23	1.0	makehint	47	3	2806	1	#95
24	1.0	norm_check_top	17	4	1382	1	#96
25	1.0	sigencode_z_top	16	1	1030	0	
26	1.0	pkdecode	16	1	844	0	
	1.0	sigdecode_z_t	16	1	1018	0	

27		op					
28	1.0	sigdecode_h	32	4	1752	3	#131, #130,#132
29	1.0	sampler_top	54	25	784	0	
30	1.0	ntt_top	50	7	1616	0	
31	2.0	adamsbridge_ ctrl(only stream_msg)	35	5	1924	1	#145(found in simulation and then in FPV)

Formal Sign-off

The sign-off section outlines details about what aspects were considered for the blocks to be said as formally verified within the scope. This starts with the verification strategy chosen, formal coverage, quantitative analysis like reviews, and proof time.

Verification Strategy:

Some formal checks did not conclude within the configured amount of time. This is a typical outcome when formal verification is applied on mathematical complex design blocks with a large sequential depth. Inconclusive proofs are caused by a large search space which increases the runtime exponentially, well-known as state space explosion.

We addressed this issue by applying reduction and abstraction techniques such that the formal check either concludes, or it reaches a sufficiently large sequential depth. In cases where this effort was not enough and in cooperation with our partners, we agreed on the application of simulation to support the verification effort or focus on critical aspects of the design behavior.

We tailored our verification efforts to every block. Overall, we used the following reduction and abstraction techniques:

- Initial-value abstraction
- Counter abstraction
- Non-determinism
- Signal cutting
- Scoreboarding
- Parameter reduction

The application of these techniques reduced the runtime of formal checks significantly. This allowed us to achive a higher formal coverage. Some formal checks are still inconclusive. However, the overall result of this verification effort is in our opinion still robust. This is supported by the achieved coverage metric and the comprehensive reviews.

Formal Coverage:

Formal coverage is one of the key metrics we rely on to judge how complete our formal verification is. It tells us whether:

- We’ve missed any important features or cases
- Our constraints are too tight and hiding valid behavior

Coverage numbers like checker and stimuli coverage help us decide if we’ve verified a block well enough. Higher coverage means we’ve done a good job exploring the design and its behavior.

Code, Constraints Review:

We didn’t rely on the tools alone. All the code, constraints were reviewed manually to make sure they reflect what we want to test and match the expected behavior. This step also helped us avoid over-constraining the design unintentionally.

Bound review(Non-Converging proofs):

This bound determines the maximum number of sequential steps or cycles the tool will analyze to either prove or disprove a property. Evaluate whether the property's behavior within the bound is sufficient to provide meaningful confidence, even if full convergence isn't achieved. If full convergence is achieved, it could be depicted as bound review as complete.

Prove Time:

Each block was given a minimum prove time so the tool could properly explore the design. This helps ensure deeper or more complex behaviors aren't missed just because the proof was cut short.

	Version	Block	Formal Coverage	Coverage Review	Bound Review	Constraint Review	Code Review	Prove time minimum ^
1	1.0	sha3/keccak Round	100%	Yes	Yes	Yes	Yes	< 1h
2	1.0	sha3/keccak	100%	Yes	Yes	Yes	Yes	< 1h
3	1.0	adamsbridge_ctrl	100%	Yes	Yes	Yes	Yes	< 24h
4	1.0	sample_in_ball_ctrl	100%	Yes	Yes	Yes	Yes	<1h
5	1.0	exp_mask_ctrl	100%	Yes	Yes	Yes	Yes	<1h
6	1.0	rej_bounded_ctrl	100%	Yes	Yes	Yes	Yes	<1h
7	1.0	rej_sampler_ctrl	100%	Yes	Yes	Yes	Yes	<1h
8	1.0	abr_piso	100%	Yes	Yes	Yes	Yes	<1h
9	1.0	sib_mem	100%	Yes	Yes	Yes	Yes	<1h
10	1.0	ntt_shuffle_buffer	100%	Yes	Yes	Yes	Yes	<1h
11	1.0	ntt_ctrl	100%	Yes	Yes	Yes	Yes	< 24h
12	1.0	ntt_butterfly	-	No	Yes	Yes	Yes	< 24h
	1.0	ntt_hybrid_	-	No	Yes	Yes	Yes	< 24h

13		butterfly_2x2						
14	1.0	ntt_masked_BFU_add_sub	-	No	Yes	Yes	Yes	< 24h
15	1.0	ntt_masked_BFU_mult	-	No	Yes	Yes	Yes	< 24h
16	1.0	ntt_masked_butterfly1x2	-	No	Yes	Yes	Yes	< 24h
17	1.0	ntt_masked_gs_butterfly	-	No	Yes	Yes	Yes	< 24h
18	1.0	ntt_masked_pwm	-	No	Yes	Yes	Yes	< 24h
19	1.0	power2round_top	100%	Yes	Yes	Yes	Yes	< 4h
20	1.0	decompose	100%	Yes	Yes	Yes	Yes	< 3h
21	1.0	skencode	100%	Yes	Yes	Yes	Yes	< 1h
22	1.0	skdecode_top	100%	Yes	Yes	Yes	Yes	< 3h
23	1.0	makehint	100%	Yes	Yes	Yes	Yes	< 1h
24	1.0	norm_check_top	100%	Yes	Yes	Yes	Yes	< 1h
25	1.0	sigencode_z_top	100%	Yes	Yes	Yes	Yes	< 1h
26	1.0	pkdecode	100%	Yes	Yes	Yes	Yes	< 1h
27	1.0	sigdecode_z_top	100%	Yes	Yes	Yes	Yes	< 1h
28	1.0	sigdecode_h	100%	Yes	Yes	Yes	Yes	< 4h
	1.0	sampler_to	100%	Yes	Yes	Yes	Yes	< 2h

29		p						
30	1.0	ntt_top	100%	Yes	Yes	Yes	Yes	< 2h
31	2.0	adamsbridge_ctrl(stream_msg)	100%	Yes	Yes	Yes	Yes	< 24h

*) excluding unreachable and deadcode.

^) Proof time with 24h have some non-converging proofs

And the ntt compute modules like butterfly formal coverage is inconclusive since it involves arithmetic operations like multiplication, modulo which are tough to solve in formal environment.