Add icon Add cover

Verification Report

Overview

- Number of blocks: 17
- Number of assertions: 881+263
- Number of assumptions: 111
- Lines of Code: >23 k
- Reported Bugs/Enhancements: 9

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	2.0	abr_ctrl(mlkem)	549	31	12095	2	#215,#224
2	2.0	ntt_butterfly(mlkem add,mult,gs,1X2)	11	6	416		
	2.0	ntt karatsuba,	15	13	480		

3		ntt_masked_pairwm, Barret reduction masked/unmasked					
4	2.0	CBD	5	1	114	1	#163
5	2.0	ntt_top	76	7	3074		
6	2.0	ntt_ctrl(mlkem)	130	10	5383	2	#180,#225
7	2.0	Sampler_top, rej_sampler(mlkem)	60	26	895		
8	2.0	Compress, Decompress	35	17	1137	4	#166, #167,#165,#189(Wont fix)
		Sum 881		Sum 111	Sum 23594	Sum 9	

The table below is an additional set of assertions which were added to breakdown the compute module into several steps. These involves several intermediate assertions which were added in c2rtl verification.

	Block	# DPV Assertions
1	ntt_butterfly(mlkem mult,gs,1X2,2X2)	152
2	ntt karatsuba, ntt_masked_pairwm, Barret reduction masked	111

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
- Data Path Verification(c2rtl)

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 comprehensive reviews.

The application of data path verification is comparing the outputs of the arithmetic rtl units with the high-level model, and the formal engines used by commercial tools differ from the general property verification, which are better for such operations. The technique used here is primarily the computation algorithm steps where split in multiple intermediate checkers and driving the results to the primary assertion. This helps in gaining confidence in the cases for arithmetic units where proofs are inconclusive.

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	2.0	abr_ctrl(mlkem)	100%	Yes	Yes	Yes	Yes	< 24h
	2.0	ntt_butterfl	-	No	Yes	Yes	Yes	< 24h

2		y(mlkem add,mult,gs,1X2)						
3	2.0	ntt karatsuba, ntt_masked_pairwm, Barret reduction masked/unmasked	-	No	Yes	Yes	Yes	< 24h
4	2.0	CBD	100%	Yes	Yes	Yes	Yes	< 1h
5	2.0	ntt_top	100%	Yes	Yes	Yes	Yes	< 2h
6	2.0	ntt_ctrl(mlkem)	100%	Yes	Yes	Yes	Yes	< 24h
7	2.0	Sampler_top, rej_sampler(mlkem)	100%	Yes	Yes	Yes	Yes	< 4h
8	2.0	Compress, Decompress	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.