



JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA



Breaking the Formal Convergence Barriers of a Floating-Point Dot-Product Block for AI/ML Accelerators

Satyabrata Sarangi¹, Neelabja Dutta², Sai Ma¹, Ashish Kapoor², Reily Jacoby²

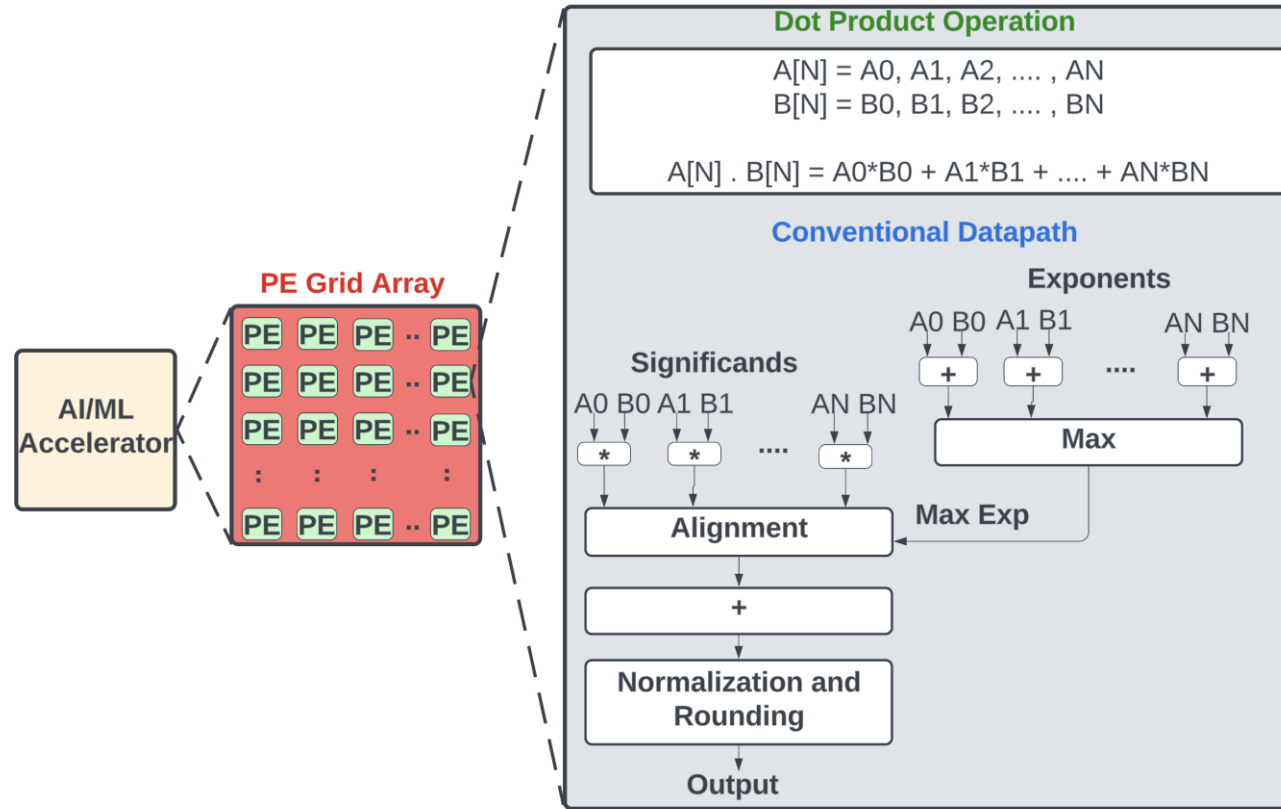
Adrian Lewis¹, Rohan Mallya¹, Eda Sahin¹

¹ Meta, Sunnyvale, USA

² Synopsys, Sunnyvale, USA




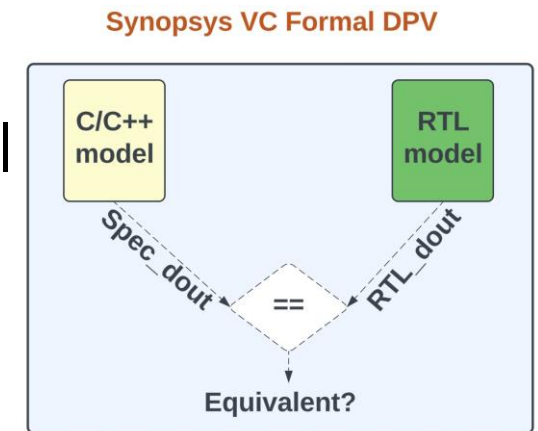
Dot-Product Compute Engine



A multi-input and multi-precision floating-point dot-product block is pivotal to AI/ML hardware accelerators!

C vs RTL Formal Verification

- C vs RTL formal verification is necessary for a multi-input and multi-precision dot-product block to do datapath verification signoff.
 - **Exhaustive analysis – corner-case bugs** 
- We used Synopsys VC Formal DPV to perform transactional equivalence check between C/C++ and RTL models.

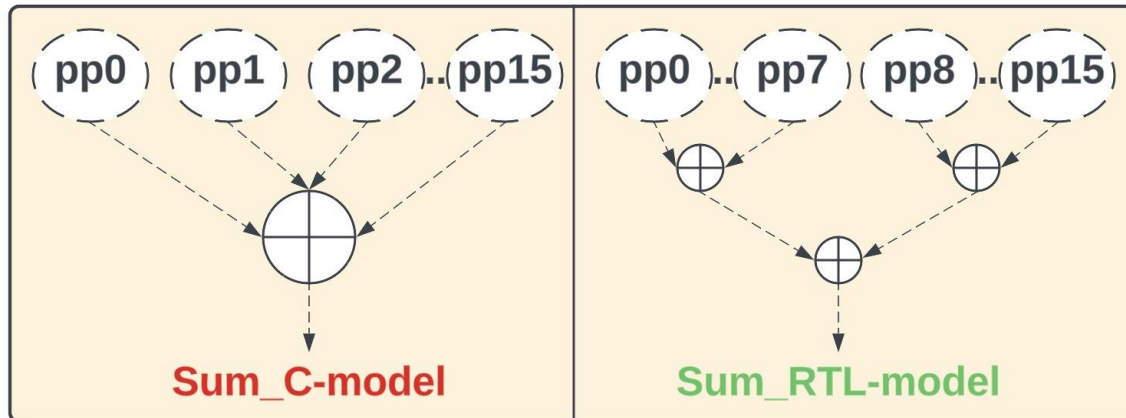


C vs RTL formal verification for a multi-input and multi-precision dot-product block is challenging!

Motivation

- Challenges in achieving formal verification convergence

- Datapath complexity makes the tool harder to converge.
 - Multi-input and multi-precision floating point(fp) dot-product support
- Finding equivalence points due to different C/RTL adder trees



RTL - model

- Partial products (pp) in a shared datapath

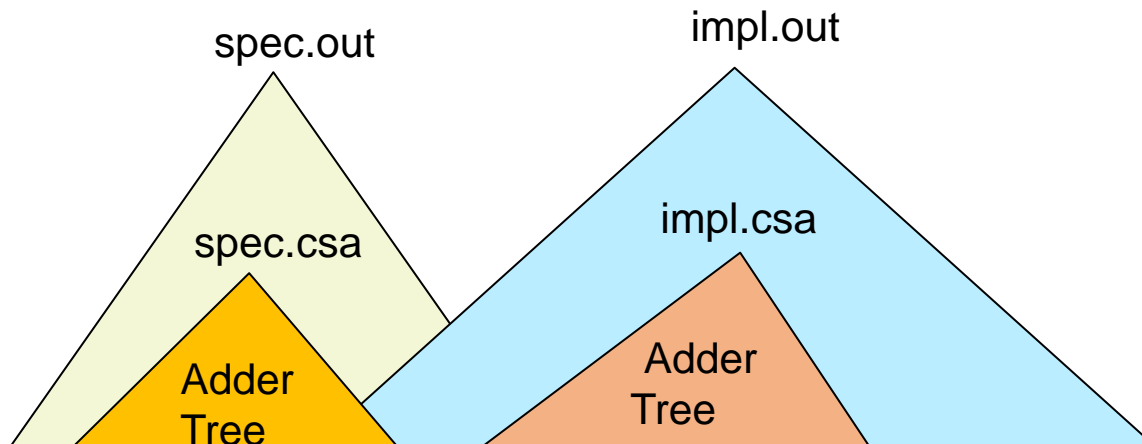
C - model

- Different adder sizes due to bit width-inaccuracy at the intermediate compute stages

How to break the formal convergence barriers?

Assume-Guarantee

- Intermediate nodes are proven and then assumed to structurally simplify the proof.
- Adder trees for C and RTL were proven independently.
- Once the adder trees were proven equivalent, they were assumed to prove the final dot-product output.



Step 1:

Prove $\text{spec.csa} == \text{impl.csa}$

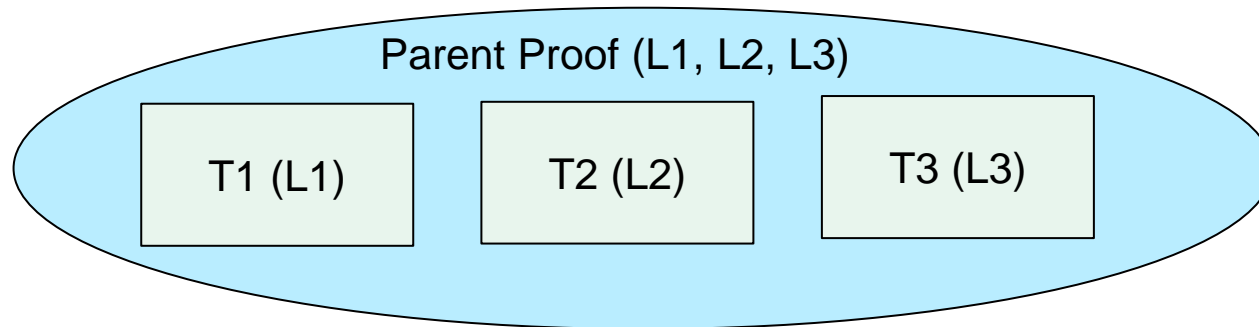
Step 2:

Assume $\text{spec.csa} == \text{impl.csa}$

Prove $\text{spec.out} == \text{impl.out}$

Lemma Partitioning

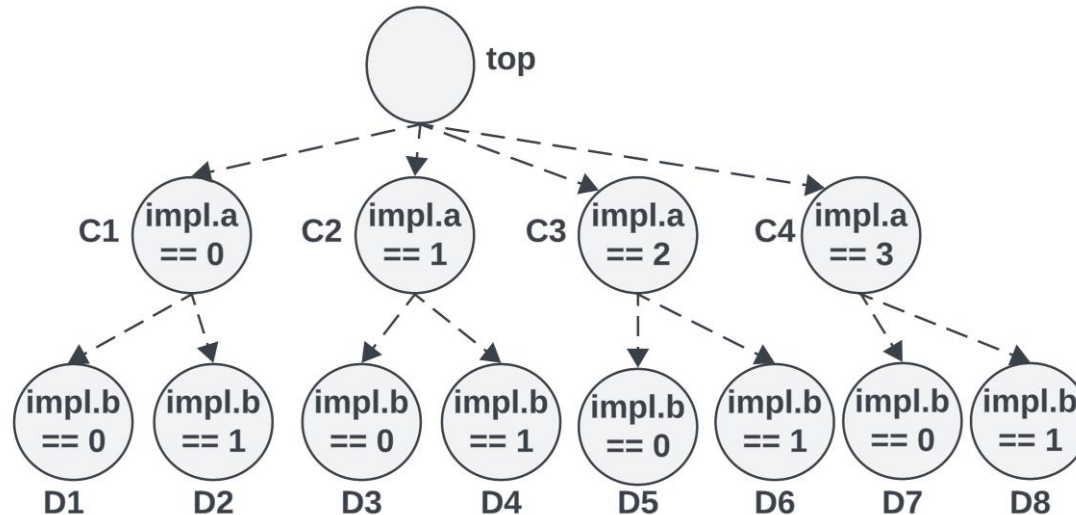
- Partition/cluster lemmas so that they can be solved independently.
 - Faster run-time as each partition can be solved in parallel with separate tasks.



- Parent Proof is verifying 3 assertions: L1, L2, and L3
- The assertions can be partitioned among 3 tasks: T1, T2, and T3

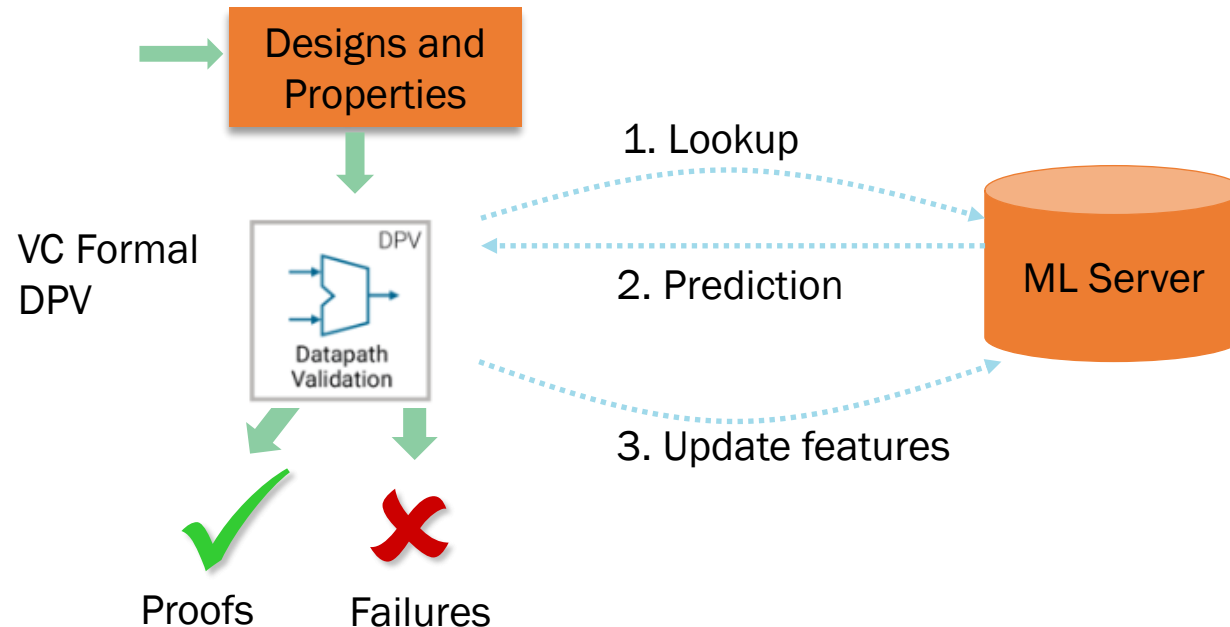
Case Splitting

- Break complex formal verification proofs into several easier sub-problems.
- Each sub-problem will have one or more additional assumptions, reducing the state space of the problem.
- Example: split the proofs based on input operands type (sub-normal, NaN/Infinity etc.) or product output (zero).



RMA (Regression Mode Acceleration)

- Learn information about proof/falsification and apply the saved information in the subsequent runs.
- The successful scripts are applied first when there is no or minimal change in the design/setup.



Adder Tree Mismatches and Recommendations

- **Problem**

- How to achieve formal convergence amidst different C/RTL adder tree structures?
- Can the VC Formal DPV tool automatically detect different adder trees from C/RTL?

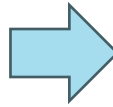
- **Solution**

- We enabled helper lemmas using adder tree expressions at individual-adder-level to find equivalency between C/RTL adder tree structures.
- Synopsys is currently working on the flow to identify different adder trees between C/RTL and automatically resolve them in the VC Formal DPV flow.

Results

```
1513 vcf>listtask 1
1514 =====
1515 Task 1
1516 Proof p
1517 Lemmas # 98
1518 Covers # 0
1519 Vacuities # 0
1520 Witnesses # 0
1521 Task Status finished
1522 Machine
1523 Task Result killed
1524 Solve Time 60542
1525 Converged% 0
1526 Solve Script orch multipliers
1527 Task Start Time Tue Aug 29 16:18:21 2023
1528 [0:0:1] Info COMREW-001: Start REC nodes 37802, mult 256, div 0, udiv 0, mod 0, unod 0, add 1321, sub 0, mux 2838
```

Non-convergence (run-time: 16+hrs)



Name	Progress	Result
final_nan_inf	34:34:0:0	
mul	16:16:0:0	
aligned_bits	16:16:0:0	
individual_adders	3:3:0:0	
rtl_full_adder	1:1:0:0	
sumc_rtl_adder_relation	1:1:0:0	
final	34:34:0:0	
prod_zero	34:34:0:0	

status	name	vacuity	witness	type	class	engine	elapsed time
✓	out_data_check_0			lemma	user	orch_satonly	00:00:04 ...re
✓	out_data_check_1			lemma	user	orch_satonly	00:00:04 ...re
✓	out_data_check_10			lemma	user	orch_satonly	00:00:03 ...ad
✓	out_data_check_11			lemma	user	orch_satonly	00:00:04 ...ad
✓	out_data_check_12			lemma	user	orch_satonly	00:00:03 ...ad
✓	out_data_check_13			lemma	user	orch_satonly	00:00:03 ...ad
✓	out_data_check_14			lemma	user	orch_satonly	00:00:04 ...ad
✓	out_data_check_15			lemma	user	orch_satonly	00:00:03 ...ad
✓	out_data_check_16			lemma	user	orch_satonly	00:00:04 ...ad
✓	out_data_check_17			lemma	user	orch_satonly	00:00:04 ...ad
✓	out_data_check_18			lemma	user	orch_satonly	00:00:04 ...ad
✓	out_data_check_19			lemma	user	orch_satonly	00:00:05 ...ad

name	vacuity
1_scv_assume_0	
2_scv_assume_1	
3_v_assume_10	
4__assume_100	
5__assume_101	
6__assume_102	
7__assume_103	...temp_5 == 0) && (spec.prodP_temp_6 == 0) && (spec.prodP_temp_7 == 0) && (spec.prodP_
8_v_assume_11	
9_v_assume_12	
10_v_assume_13	
11_v_assume_14	
12_v_assume_15	

Total Properties: 34 - passed[34] - failed[0] - disabled[0] - Constraints Enabled: 104 Run Time: 0:29:40

Convergence (run-time: 29 mins)



- Run-time for a 16-input FP32-type dot-product : <30 mins
- Run-time across several other floating-point-types : <11 hrs
- In addition to achieving 100% convergence and 100% C-model coverage, critical bugs (output data/exception) were flagged.

Conclusion

