2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Revolutionizing Proof Convergence for Algorithmic Designs: Combining Multiple Formal Verification Tools

Disha Puri (disha.puri@intel.com), Suraj Kamble (suraj.jyotiram.kamble@intel.com), Rajib Lochan Jana (rajib.lochan.jana@intel.com)
Intel Corporation

***Abstract*-Arithmetic designs play a key role in GPU architecture. As the market demand for power/performance increases, so does the onus on architects to come up with optimized and understandably more complex algorithmic designs. Formal Verification particularly C vs RTL formal methodology has been recently accepted as an industry standard to verify such data-centric designs by formal experts. Success of Datapath verification tools in recent times have encouraged multiple vendors to invest in this area. However, even though all vendor tools are investing heavily in their R&D, it is reasonable to say that we cannot expect one single tool to achieve complete convergence every time on all types of newly architected complex data-oriented designs.**

In this paper, we showcase that instead of trying deep convergence techniques with one tool alone over multiple months, it might be more lucrative in these competitive times to adopt a tool agnostic flow to validate an algorithm. This involves splitting an algorithm into multiple disjoint blocks and proving them in tools which work best on those individual blocks. We then show how to seamlessly stitch the algorithm back together to get complete convergence. We use single/double precision fused multiply-add (FMA) as case study for this paper.

I. INTRODUCTION

C vs RTL and RTL vs RTL (illustrated in Fig. 1) are very common Datapath Formal Verification methodologies that we have been leveraging at Intel from many years with successful signoff on various generation projects. As designs are becoming more and more complex, we have been collaborating with multiple vendors on tool strategies, improvements, and enhancements to get support on complex Datapath designs such as compression, fused multiply add, hashing etc. The reason that we need to be forthcoming in solving the problem of convergence for Datapath designs is that in Datapath, we cannot afford even a single bug miss. Even if we run the design for >48 hours on a tool without any failure, the unconverged status still implies that there is a possibility of uncovering a deep-rooted corner case bug later in the design cycle or in silicon which can be catastrophic. This problem becomes more prevalent when we are dealing with floating point arithmetic designs such as Fused Multiply Add where we must revert to various convergence techniques like case splitting, assume guarantee, cut points etc. due to huge state space as well as internal complexity of the algorithm. In this paper, we are talking about those designs where we are applying numerous convergence techniques but still, we are not able to get a convergence. With this scenario, the next step to getting a convergence is to reduce the abstraction difference by creating multiple intermediate models so that we can prove them step by step and get complete convergence. These kinds of activities are usually taken by Datapath formal engineers towards the end of the design cycle once the design becomes stable, because the convergence efforts will be useless if the design changes in the meantime due to various optimizations including gate count reduction. However, applying formal verification convergence activity at the end of design cycle can lead to a major effect on overall product timelines, as missing the corner case deep bugs will not be unearthed till end of design cycle.

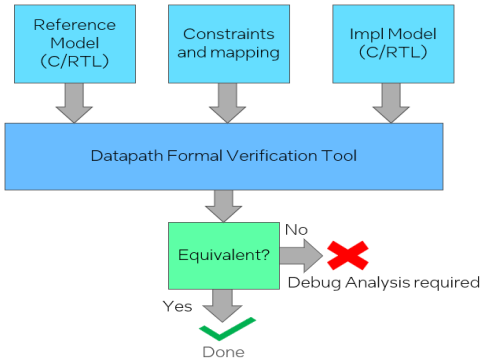

Fig.1 Datapath Formal Equivalence

In this work, we are leveraging the expertise and engines of different Datapath tools to come up with a strategy that can do a huge left shift in convergence of the design. With this approach, we have not only converged on 7 complex Datapath designs including SPFMA and DPFMA but also unearthed 3 deep corner case bugs very early in the design cycle. The message here is not to compare and contrast the various Datapath tools, but to emphasize that we can create an environment where we can complement these tools and come up with strategies where we can adopt tool agnostic flow for every complex design. In this paper, we have showcased how we have done it successfully on FMA Unit and we have a plan to adopt this for every other project at Intel.

II. Related work

Formal verification is becoming more and more popular in industry [1] especially with the increase in scalability which formal verification tools can now support. With the development of formal equivalence checking techniques in recent years [2], it is now possible to check the functional equivalence at higher abstraction levels, such as C to RTL, RTL to RTL, and C to C, both on combinational and sequential logics. Commercial EDA tools are available in applications such as sequential equivalence checking and transaction-level equivalence checking [2] [3][4]. There are various papers which talk about C vs RTL equivalence verification of complex algorithms such as ALU floating point operations [5-6]. A collective study of these papers can help a user understand various convergence techniques in Datapath domain. Full proof of complex designs often leverages deep understanding of internals of designs to manually reduce abstraction difference. Needless to say, it requires design expertise and a lot of effort which spans over multiple months in many cases. In this paper, we leverage the efficiency of different tools to solve different sub-problems and provide End to end solution by stitching the proofs seamlessly.

III. Application

3.1. Design Overview:

The Fused Multiply Add (FMA) Unit performs A * B + C operation, depicted in Fig. 2, where A, B and C can be single or double precision floating-point numbers. The mantissa of A and B are multiplied and stores the product P in carry save format. At the same time, the mantissa of C is aligned with the product of A and B according to computed value of Exponent (A) + Exponent (B) - Exponent (C). The product P and mantissa of C are added in form of two vectors using carry save adder. The two vectors are added and complemented if result is negative. Otherwise, the vectors are sent to the leading zero anticipator to perform normalization on final mantissa along with rounding. The processed mantissa, sign, and exponent forms the result. The FMA performs not only multiply-add (A * B + C) but also may emulate a floating-point adder (A + C) with B equal to 1.0 and floating-point multiplier (A * B) with C equal to 0.0.

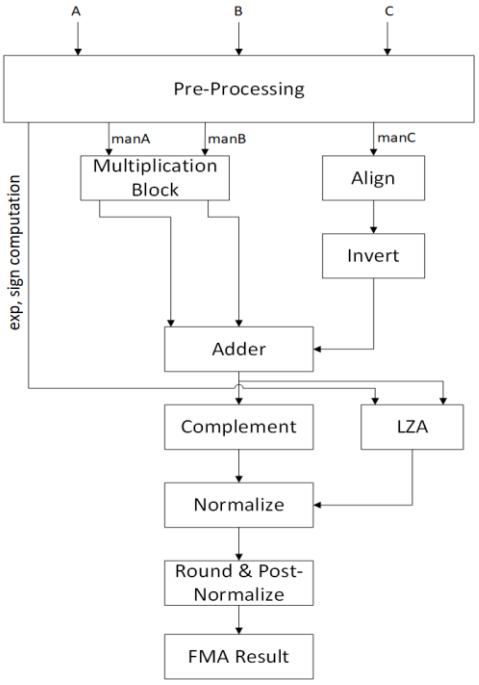

Fig.2 FMA Design

3.2. Problem Statement:

In this work, our intention is to prove functional correctness of a highly optimized FMA unit of Intel graphics processor design for single/double precision floating point numbers. At the time the work has been done, no single state of art industrial tool could prove the complete unit using C vs RTL equivalence checking. The formal verification complexity comes from the complex algorithms like multiplication block and renormalization style with different rounding modes. We are not claiming that an FV expert with deep knowledge of design/tool and ample time at their disposal to write multiple properties at internals of the design would never be able to prove this design in a single tool. However, a practical challenge is to find a methodology that can formally prove the correctness of FMA unit in reasonable time frame with minimal manual intervention.

3.3. Methodology Overview:

We first run E2E flow on all tools available to us for C vs RTL to ensure that no one tool is enough to provide complete convergence with standard convergence techniques. We have then divided our tool agnostic flow, depicted in Fig. 3, into following 4 intuitive.

3.3.1. Complexity Analysis:

The first step is to understand the structure of the design and determine the portion of logic/modules that are most likely to introduce the formal complexity. In this case study, verification of FMA operation, we identify the multiplication block and renormalization logic as a main source of formal complexity based on prior knowledge and tool generated complexity reports.

3.3.2. Partitioning Into Sub-Blocks:

The sub-blocks identified as source of complexity are neatly partitioned and assertions and assumptions are written surrounding it to isolate them from rest of the design. For example, in this case study, Fig.3 we have isolated the multiplier block in both C and RTL by proving that output of multiplier block is equivalent to multiplication of inputs for C as well as for RTL. This technique allows us to Blackbox the multiplier from end-to-end proof. This is a standard convergence methodology adopted by many verification experts to apply different kinds of abstraction techniques, but we have leveraged it to use tool agnostic strategy in next step.

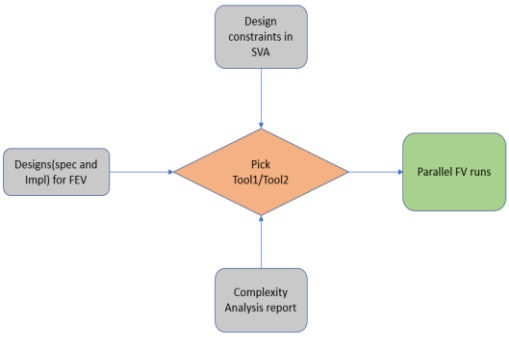

Fig.3 Tool Agnostic Flow

3.3.3. Pick the Best Tool for Each Sub-Block:

Based on previous analysis of FV runs (analysis of unconverging proofs), we have isolated the sub blocks which introduces complexity for FV proofs, we can run proofs on multiple tools and select one which works best for corresponding logic. To save time and engineering efforts we modelled the design constraints as System Verilog Assertions (SVA) that are portable in most of the commercial formal tools and help to enable tool agnostic flow, only input mapping and internal checkers are written in tool specific language, refer Fig.2. For the sake of using FMA Unit, we proved internal logic in multiple tools (refer Table 3). The selection of tool can also be dependent on the prior knowledge of tool efficiency. For FMA case study, combination of two tools worked best than executing end 2 end proof in single tool.

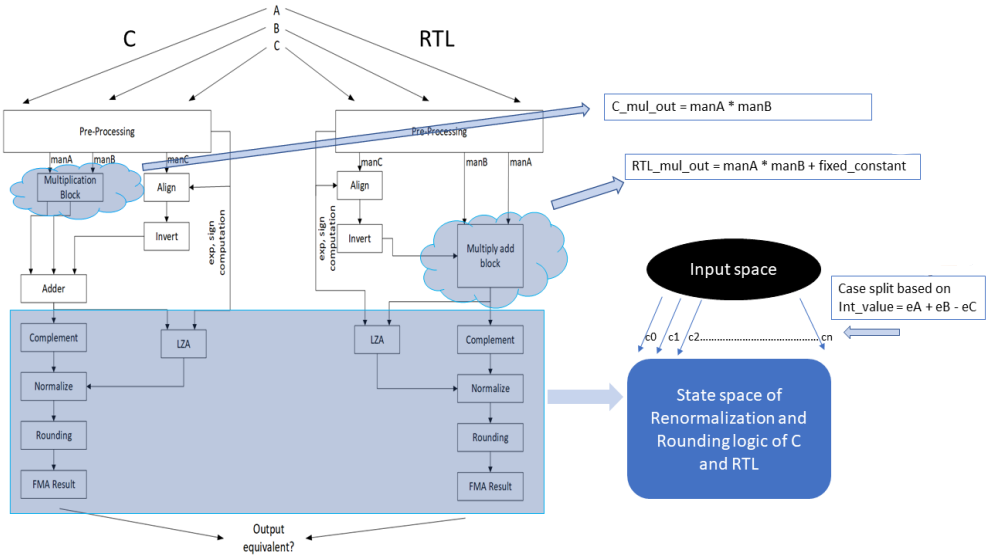

Fig.4 Methodology Overview (C vs RTL)

3.3.4. Stitching the E2E Proof:

The proved assertions on different internal blocks of the design using multiple tools are used as assumptions in our End-to-End (E2E) setup that is used to prove the correctness of functional behavior of the complete FMA unit. We used case-split and assume guarantee techniques to assist final proof for convergence and finally, prove the correctness of function behavior of FMA unit for SP and DP floating point numbers within 15 hours and 25 hours respectively.

IV. RESULT

We exploit the different solvers facilitated by multiple tools to prove the helper assertions as well as end 2 end proof on our FV setup. We reported >10 bugs including some corner case scenarios on FMA unit for SP and DP floating point numbers. 2 of them are explained below.

A	0x0B381E20	3.5459775E-32
B	0x2DF2067E	2.7515098E-11
C	0x808002B8	-1.1755919E-38
Expected output	0x80800000	-1.1754944E-38
Impl output	0x0	0.0

Table 1. Corner case bug-1 (SPFMA)

This is one of the corner cases in SPFMA where RTL did not correctly determine whether to flush the result to zero or keep it as a minimum normal number when the mantissa after renormalization before rounding is negative (sign = 1), leading bit of 1, and all remaining 23 mantissa bits are 0s. This case can occur only once when sign bit is 1, mantissa MSB 1 and all remaining bits 0 (1 in 2^32 (billion) combinations) of an intermediate signal.

A	0xA3E59B1BB76CF3A6	-9.289325019823287E-136
B	0x1F39C4CE02FEB9A5	2.932621087873144E-158
C	0x0331660EC998B4B7	2.724207044524136E-293
Expected output	0x8010000000000000	-2.2250738585072014E-308
Impl output	0x8000000000000000	-0.0

Table 2. Corner case bug-2 (DPFMA)

This is one of the corner cases reported after application of convergence techniques on DPFMA where the result after renormalization and rounding becomes minimum normal number but due to the wrong mantissa shift value calculation RTL determining it as negative zero.

For complete convergence on C vs RTL setup where we are verifying RTL (Gen1) using golden (softfloat) C model (Normal/Denormal floating point inputs for SPFMA and Normal floating-point inputs for DPFMA) we have used convergence techniques along with tool agnostic flow where best of multiple tools are explored to prove and get complete confidence on FMA design with less engineering efforts and minimal manual interventions. Similarly, for next Gen RTL we have enabled RTL vs RTL flow and proved (E2E algorithm) SPFMA and DPFMA with same approach. All proofs are run at least for 48 hours before application of any convergence technique.

	Gen 1(C vs RTL)	
	SPFMA	DPFMA
E2E setup	Unconverged	Unconverged
Standard case split	Unconverged	Unconverged
Tool Agnostic flow	Proved (~15h)	Proved (~25h)

Table 3. C vs RTL Gen1 Results

	Next Gen (RTL vs RTL)	
	SPFMA	DPFMA
E2E setup	Unconverged	Proved (~5h)
Standard case split	Unconverged	NA
Tool Agnostic flow	Proved (~3h)	NA

Table 4. RTL vs RTL next Gen Results

V. Conclusion

This paper shows how the benefit of tool agnostic flow can be leveraged to achieve faster convergence of proof on complex arithmetic-centric algorithm like FMA units. We do understand that as tools would be exposed to more structures, they may add abstraction support so that this case study can be done in a single tool itself. However, as new algorithms are being architected every day to bring an edge to semiconductor industry, we can say that there would be multiple designs which would fall in same category. Our approach suggests using a tool agnostic flow to leverage the benefits of multiple tools at once and contradicts the traditional mindset of going deep in the design with single tool and doing extensive manual interventions to get full proofs where convincing complete solution itself is a complex task.

VI. Acknowledgment

We would like to express our sincere gratitude to the RTL designers, Hurd, Kevin, Wu, shifu and Nakagawa, Takashi, for their invaluable contributions in the design discussions to deep dive into the design for resolving the convergence issues as well as resolving bugs and providing fixes as early as possible. We would also like to acknowledge the support and understanding of Vichal from the Graphics management team for understanding the algorithm's complexity and bandwidth requirements and thus helped in successfully carrying out the activity. Furthermore, we extend our thanks to Kiran, the FVCTO Manager, for providing continuous guidance throughout the project ensuring the project's success.

References

[1] H. Foster, Trends in Functional Verification-A 2014 Industry Study, Design Automatic Conference (DAC) 2015.
[2] A. Koelbl, R. Jacoby, H. Jain, and C. Pixley, Solver Technology for System-level to RTL Equivalence Checking, Proc. Design Automation and Test Conference of Europe (DATE) 2009.
[3] V. M. A. KiranKumar, A. Gupta and R. Ghughal, "Symbolic trajectory evaluation: The primary validation vehicle for next generation Intel processor graphics fpu", Formal Methods in Computer-Aided Design (FMCAD) 2012, pp. 149-156, Oct 2012.
[4] SLEC, https://www.mentor.com/products/fv/questa-slec.
[5] T. W. Pouarz, and V. Agrawal, Efficient and Exhaustive Floating-Point Verification Using Sequential Equivalence Checking, Design and Verification Conference and Exhibition (DVCON), 2017.
[6] J. Wang and J. Zhu, conquer difficult C-RTL formal verification problems using recursive compositional reasoning, design/IP track, Design Automation Conference (DAC) 2017