2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Complexity Conquered: Pioneering Formal Verification Methods for Systolic Controllers in Advanced Computing

Sarsij Saurabh, Rahul Dabur, Tushar Agarwal, Vichal Verma
Intel Corporation
sarsij.saurabh@intel.com, rahul.dabur@intel.com, tushar.agarwal@intel.com, vichal.verma@intel.com

***Abstract*- Dot Product Accumulator Systolic (DPAS) is a pivotal component in graphics processing units (GPUs), predominantly utilized for matrix computations that are crucial for artificial intelligence (AI) and machine learning (ML) tasks. The architecture of DPAS is distinguished by its parallel and pipelined design, which is optimized for the high computational demands of AI and ML workloads. The verification of the DPAS design poses significant challenges due to its intricate algorithmic implementations and sophisticated control logic. Formal Property Verification (FPV) stands out as a robust method to ensure design compliance with its intended specifications. Nonetheless, FPV faces hurdles such as state space explosion, which complicates the verification process. To surmount these obstacles, we have implemented abstraction and convergence techniques to reduce the complexity by minimizing the number of state elements under consideration. This approach enabled us to effectively validate complex properties and uncover hard-to-find corner case bugs, including a critical flaw that could have impaired DPAS functionality. Without the use of advanced convergence methods, such defects might have remained undetected. The verification strategies detailed in this paper can be generalized to other complex design architectures, facilitating thorough validation, and achieving high-assurance formal verification closure.**

I. INTRODUCTION

This work presents an overview and verification challenges of the complex computational block taking case study of DPAS unit, a hardware component designed for efficient matrix multiplication and accumulation. The DPAS unit operates in parallel across SIMD lanes for a particular instruction and can execute consecutive instructions due to its pipelined architecture. It interfaces with the Thread Controller, Granular Register File banks, and Read and Write Controllers to perform computations and handle the resulting matrix. Many complex algorithms are implemented within it to increase the efficiency and throughput which make it very difficult to verify.

Despite its less than 200k gates and 10k flop count, the DPAS unit's wide instruction set and sideband information create a large input space, posing significant verification challenges. These include verifying all valid input combinations and managing complexities introduced by features like Read Suppression and Cache Mechanism, which affect coherency and consistency.

The verification process utilizes constraints and assertions to validate the DPAS unit's operations, with checkers categorized into four groups to cover various aspects of the unit's functionality. However, initial formal proofs showed only 27% convergence of properties, leading to the implementation of optimization techniques such as parameter reduction, case splitting, helper assertions, and abstract modeling. These strategies significantly improved the convergence rate and helped uncover 14 bugs in the RTL, demonstrating the effectiveness of the optimization techniques in the verification process. The report includes figures that illustrate the improved convergence rates and the distribution of bugs found.

II. DESIGN OVERVIEW

DPAS unit runs instructions that multiply two matrices (src1 & src2), accumulate the result with a third matrix (src0) and produces a result matrix. DPAS Unit consists of multiple, Single Instruction Multiple Data (SIMD) lanes that operate in parallel and is pipelined to support multiple DPAS instructions back-to-back. DPAS module receives instructions from the Thread Controller (TC) and fetches operand data from the Granular Register File (GRF) banks through the Read Controller (RC). It then **kick-off datapath** computation by sending the operand data to the DPAS unit. Once the matrix multiplication and accumulation are completed, the resulting matrix is either sent back to the Write Controller (WC) to be written to memory or retained in local storage. The ability to keep the resultant matrix in local storage for reuse is called **Read Suppression**, which is a critical feature that enhances performance, especially in scenarios where subsequent operations can utilize the previously computed data. This approach reduces memory bandwidth requirements and latency associated with memory access.

A. Design Complexities causing Verification Challenges

The complexity of formal verification tools can be significantly influenced by factors beyond gate count, such as the breadth of the input space. Design Under Test (DUT), with less than 200k gates and flop count in range of 10K, initially seems to be a good candidate for formal verification but its wide instruction set augmented with sideband information results in a large input space. This presents several verification challenges, including the need to verify all valid input combinations and the complexity introduced by advanced features like **Read Suppression or Cache Mechanism**, which improves performance but complicates verification due to coherency, consistency, and integration with other components. **Data-path Kick-off** verification requires complex abstraction models due to dependencies like operand availability and pipeline collisions. Additionally, verifying the **Variable Execution time of Instructions** is complex, as it involves tracking operand fetch status and computational status amidst concurrent instruction execution.

III. VERIFICATION STRATEGY

Typical formal verification environment consists of constraints written on signals at the input interface to drive valid values and assertions or checkers to check the validity of the output signal. In the current formal verification setup, multiple constraint models were crafted to generate valid instructions, considering format rules and sideband information. Additionally, an abstract model of RC's arbitration logic was created to manage grant signals according to priority and bank conflicts from various pipelines.

Output checkers are categorized into four categories with complexities involved as described in table 1:

TABLE I
CHECKERS AND ASSOCIATED COMPLEXITIES

Checker Type	**Complexity Involved**
Checkers on the read request sent to RC	Applied for each source operand, the read transaction is supposed to be initiated only if read suppression is not detected.
Checkers on the pre-processed instruction sent to DPAS	Require a significant amount of helper code for field estimation
Checkers on DPAS collision signals	Require predicting result matrix write timing
Checkers on the back pressure signals	Need to track the completion of the instruction. As the instruction span is dynamic, this tracking needs to consider many factors, making the checkers complex

The formal verification test environment with different constraints/checker models is shown below in Fig. 1.

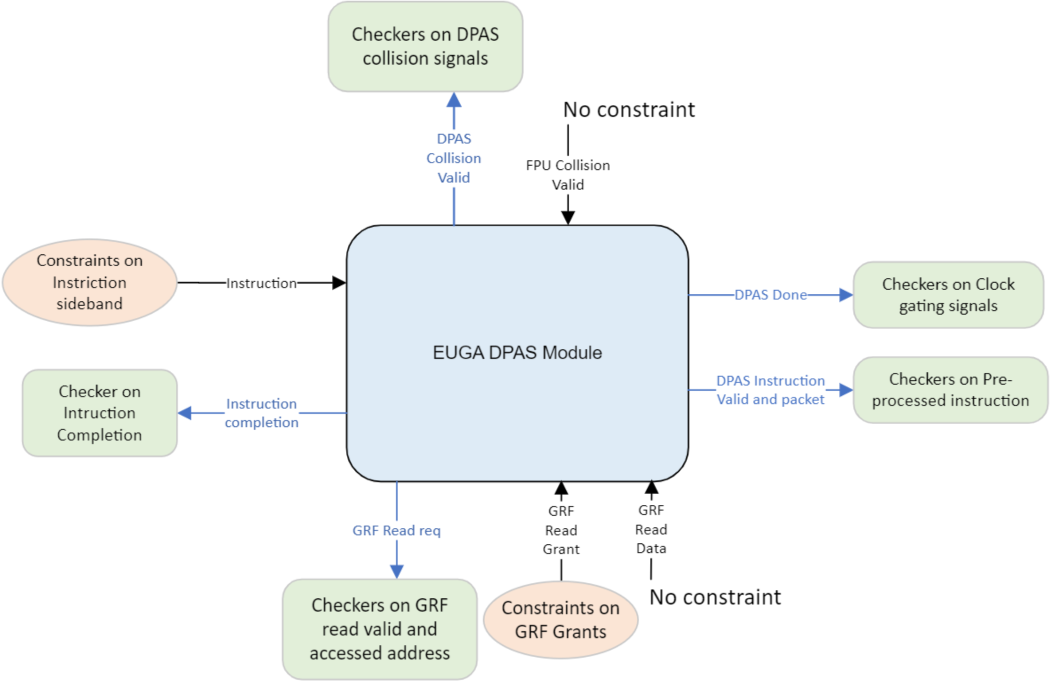
Checkers on DPAS collision signals
DPAS Collision Valid
No constraint
FPU Collision Valid
Constraints on Instriction sideband
Instruction
EUGA DPAS Module
DPAS Done
Checkers on Clock gating signals
DPAS Instruction Valid and packet
Checkers on Pre-processed instruction
Checker on Intruction Completion
Instruction completion
GRF Read req
GRF Read Grant
GRF Read Data
No constraint
Checkers on GRF read valid and accessed address
Constraints on GRF Grants

Fig. 1: Formal Verification environment for DPAS module

Implementing these checkers required complex models and in-depth feature knowledge. Three major abstraction models were essential:

- **Read Suppression or Cache Mechanism**: An abstract cache model to validate operand data retention and reuse.
- **GRF Reading**: Counter-based models to verify correct RC read requests, adjusting for instruction parameters and multi DPAS mode operand port priorities.
- **Data Path Kick Logic**: An abstract model incorporating various rules to accurately check the initiation of MAC operations in the DPAS controller.

A. *Optimization Techniques*

After conducting formal proofs, it was found that only 26% of properties converged. To enhance the quality of sign-off, the minimum proof depth (MPD) was calculated based on the RTL pipeline depth. Additionally, cover properties were added, and RTL cover points were integrated into the formal verification (FV) environment. Despite these efforts, non-converged properties did not satisfy the bounded sign-off criteria. To address this, several optimization techniques were employed:

- **Helper Assertions**: For complex kick checks, simpler assertions on intermediate column tracking logic as helpers were used.
- **Assume Guarantee**: DPAS module was divided into three subsystems (Pre-ingress FIFO, Ingress FIFO, Post-ingress FIFO) and each subsystem was verified independently.
- **Parameter Reduction**: Simplifying FIFOs and Cache by reducing depth parameters improved bounds for read suppression properties.
- **Case Splitting**: Imposing temporary constraints for specific scenarios, like DPAS mode and input formats (integer, single, and double precision floating point), helped some properties converge.

- **Abstract Modelling**: To manage the complex data-path kicking mechanism, an abstract model capturing essential behaviors for better convergence was applied.

V. RESULTS

Prior to the implementation of the various convergence strategies that have been discussed, a substantial proportion of properties, specifically 74%, failed to achieve convergence. After the implementation of these strategies, there was a notable improvement in convergence, which is depicted in Fig. 2.

Additionally, after applying case splitting and helper assertions, minimum proof depth for properties that remained undetermined is illustrated in Fig 3. This increase in the proof bound gave confidence to proceed with bounded sign-off for such properties.

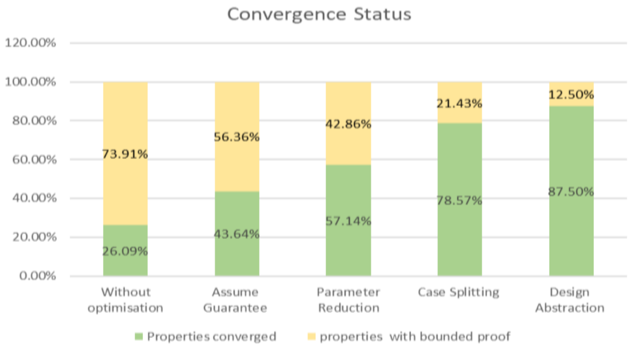

Fig 2 Property Convergence status

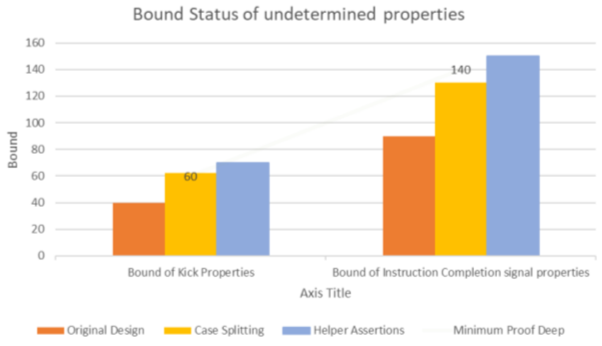

Fig 3: Bound Status of undetermined properties

During the verification process, a total of 14 defects were identified within the RTL. Out of these, 9 were detected in the early stages prior to applying any optimizations, while the remaining 5 were corner case bugs that emerged at different points during the application of convergence techniques, as depicted in Figure 4.

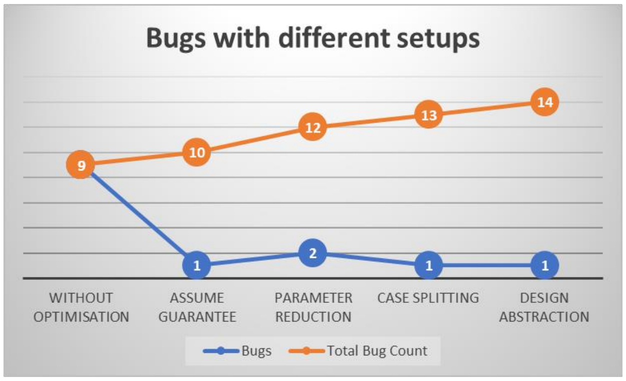

Fig 4: Bugs with different setups

Bug1: Data corruption on wrong switching of operand fetching ports.

In multi DPAS mode, multiple ports are used for Src1 operand data fetching. Based on operand fetch status, valid is sent to RC on available port. But it was discovered that the port switching was not functioning correctly, leading to the transmission of corrupted data to the DPAS Unit.

Bug2: Start of new instruction when there are no ports available for operand fetching

In the multi DPAS mode, the pipeline can only accommodate limited instructions simultaneously. However, under this bug, it was observed that maximum supported instructions were already occupying the pipeline, with all ports engaged in reading data. Despite this, the design was erroneously accepting another instruction, which it was incapable of processing.

Bug3: Violation of kick-off mode.

Design must follow certain rules when a float instruction is followed by integer instruction. But it was noted that the DUT was not adhering to this protocol.

Bug4: Erroneous data path kick-off on collision detection from other processing units

DPAS module on detecting collision from other processing units must stall data path kick-off. However, it was detected that kicks were not properly stalled leading to improper data path operations..

Bug5: Invalid design specification

DUT was designed under the specific assumption that once read conflicts are resolved, the sources will receive consecutive grants in accordance with the arbitration policy of the arbiter in the RC. When the grant was driven through basic constraints using the same design assumptions, no failures were detected. However, with the abstraction model of RC, the Formal Test Bench introduced a scenario where the current arbitration policy could create bubbles while accessing GRFs for all sources. This scenario contradicted the initial design assumptions, prompting a change in the priority order to minimize the bubbles. Furthermore, specification was updated to introduce a new feature to safeguard against data path corruption.

IV. CONCLUSION

Our examination of the design's input state space and gate counts indicates that these factors contribute to the complexity of formal verification, as evidenced by the DPAS module of the GPU. Despite the intricate nature of this design, we have successfully employed formal verification, achieved good convergence, and uncovered critical vulnerabilities. Our approach has not only identified bugs but has also enhanced the resilience of the Systolic Controller. This endeavour has significantly advanced formal verification methods for high-performance computing applications. Looking ahead, we aim to achieve full convergence on existing bounded proofs and extend our verification efforts to other complex GPU components, including the TC, FPU, pipeline controllers, and assembler.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to the RTL team Elliot, Alex, and Paulo, for their invaluable contributions in the design discussions to deep dive into the design for coming up with checks for complex design features and for resolving the convergence issues. Furthermore, we would like to extend our thanks to Vichal, Graphics Formal Manager for providing continuous guidance and support throughout every phase of the project and making this wok a success.

REFERENCES

[1] E. Seligman, T. Schubert and M.V.A. Kiran Kumar, Formal Verification: An Essential Toolkit for Modern VLSI Design, Morgan Kaufman, 2015.

[2] Disha Puri, Madhurima E, Shravya Jampanna; Raising the Bar: Achieving Formal Verification Sign-Off for Complex Algorithmic Designs, with a Dot Product Accumulate Case Study, DVCON, India, 2023