Bringing DataPath Formal to Designers' Footsteps

M, Achutha KiranKumar V, Intel Technology India Pvt Ltd, Bangalore
(achutha.kirankumar.v.m@intel.com)

Disha Puri, Intel Technology India Pvt Ltd, Bangalore (disha.puri@intel.com)

Shriya Dharade, Intel Technology India Pvt Ltd, Bangalore(shriya.dharade@intel.com)

***Abstract* - Aggressive demands in the Graphics Industry force the need for a shift left approach on all the aspects of design life cycle. This compels the designers to adopt better arithmetic implementations and also optimize the design to gain better power and performance. Such changes often have a short time to market schedule and hence, need to be quickly validated. In Intel Graphics, we have been successful in completely validating complex data path designs by functionally validating the algorithmic specifications (specs) in a high level language (C/C++), also known as System Level Model (SLM) against the RTL implementation in a Formal Verification (FV) tool. With these new demands, however, confining only to this approach is not sufficient. We propose extending DataPath FV to include three comparatively low effort and high ROI methodologies: Design Exercise, RTL-RTL and Corroborating C Specification model. We share success stories where these methodologies helped in proliferating FV within Intel Graphics and have helped to quickly validate new feature changes.**

Keywords - Formal Validation, Datapath, SLM-RTL, RTL-RTL, Design Exercise

I. NEW METHODOLOGIES AND THEIR IMPACT AT INTEL GRAPHICS

Graphics Industry is a highly competitive market which requires extensive changes and enhancements in arithmetic algorithms every generation to meet the aggressive demands of area, timing, performance and power. In Graphics Processors (GT), there are several deep rooted algorithms which range from simple tasks such as format conversions and filters to very complex algorithms such as (de)compression, encoders and decoders, blend optimizations, image segmentation, media enhancements etc. The state space for these algorithms is immense with several corner cases related to deeply nested loops/branches. Additionally, there is a need for shift left on all aspects of the design life cycle including quick validation to stay ahead of the competition.

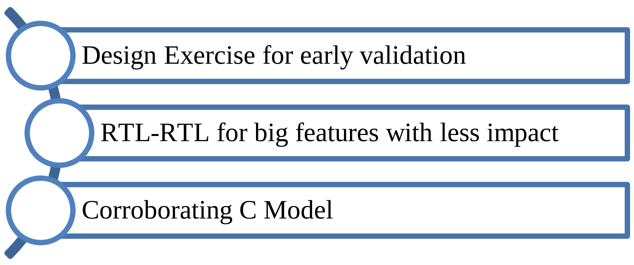
Design Exercise for early validation
RTL-RTL for big features with less impact
Corroborating C Model

Figure 1: Three New Methodologies for Datapath FV

The current established methodology (SLM-RTL) for such algorithms involves taking a mature SLM Model against RTL using a Datapath FV tool [1].

We collaborated extensively with R&D team to get support for most of the C++11/14 constructs, Vectors etc. Moreover, we now have a Formal friendly interface to support all BitVec, Float and Fixed legacy libraries. This has enabled us to consume native C++ code with only minimum and essential modifications. Current SLM-RTL methodology has allowed us to spread the gamut of DataPath Verification and expose several corner case bugs. However, it still requires an involved effort from a DataPath FV expert at times for compilation (synthesizing the C++ code in an FV tool) to remove/replace non Formal friendly constructs in the core algorithm. The functional models for C++ and RTL may be implemented very differently which hinders convergence in absence of intermediate mapping signals. This would require effort and expertise of a DataPath FV engineer to try several techniques of case splitting and customized tuning of tool to close the verification. To support shift left, we require quick turn-around time in validation.

To support shift left, we require quick turn-around time in validation which can be achieved by active collaboration between designers and FV experts at the preliminary stage of design development itself and by empowering the designers to use the Formal tool themselves. We propose three new methodologies which are relatively low effort and high ROI and have enabled us to successfully take DataPath Formal to Designers (Figure 1) – Design Exercise, RTL-RTL and Corroborating C model with Specs.

Design Exercise using Formal Property Verification (FPV) is a methodology applied for testing Control Paths in the RTL [1]. Design Exploration helps verifying the basic functionality of the design in the early stages of design cycle by proving properties adhering to the specification.This methodology cannot be directly used for algorithmic designs as the datapath designs verification is not just about proving basic properties but also to achieve input output equivalence with a specification model.

Design Exercise in DataPath designs entails starting the C-RTL verification even before the designers have full confidence in either the C++ or the RTL code instead of waiting for the mature/Golden specification. In Section II.1, we share a case study where market realignment requirements forced the designer to optimize and architect a new algorithm, which had a very short runway for design, specification and verification requirements. The C++ reference model would have taken considerable time to move to the golden state. Aggressive schedule challenged the current flow and the feature was questioned to be accommodated. This inspired us to propose Design Exercise through Datapath FV tool which quickly found the counter-examples starting initially with simple mismatches to complex bugs later. With Architect's help, the designers made a call on whether C++ needs to be fixed or RTL needs to be modified. In less than a week's effort and 18 bugs later (including both C++ and RTL), we had a high quality design and a happy team of designers, architects and management.

There are several designs which undergo changes every generation for power savings and efficiency such as enabling clock gating or adding more pipeline stages. RTL-RTL is a widely accepted methodology for proving design equivalence [2], however in our experience it is difficult to converge algorithmic designs in normal Sequential Equivalence tools. We have proposed using RTL-RTL using the DataPath FV tool in Data Path oriented designs. We share a case study in Section II.2 where a complex arithmetic algorithm had undergone a simple case of manual re-pipelining. Sequential equivalence tools could not converge for more than a month, whereas through Datapath FV tool, the design converged in less than 5 min.

Current SLM-RTL is dependent on the correctness of Spec (C code) to find failures in RTL. However, this limits the efficiency of verification to the correctness of the C code. There have been several instances where the C++ code (in some cases legacy code as well) has been found non-compliant to architectural specifications. In Datapath FV tools, we can write assertions in software to validate behavioral properties expected from the code. Also, there are several automated checks such as bound checks on array and loops to validate sanctity of C code. In Section II.3, we share a recent experience where we wrote additional assertions on the SLM model while executing Design Exercise. We found that the assumptions under which the architect had written the algorithm was incorrect in a corner case scenario. The design was rectified by architect and the entire design was verified with collaboration with architect, designers and FV engineers in less than 2 weeks.

To ensure shift-left in validation, it is imperative that more and more algorithmic implementations are validated using DataPath FV. We have shown how non-traditional methodologies can help us realize that aim.

(1) Design Exercise Methodology for DataPath Designs

General Design exploration aims at navigating the design through some cover properties to get a preliminary confidence on the functionality, with a relatively lightweight verification effort. However, we still require a clear-cut plan on what to check and when to declare that design exercise is done. In control path, we can decide on exit criteria on the basis of coverage report or time limit. In DataPath Designs, we have used Design Exercise in a slightly different context. The aim is still to check functionality of the design in early design development, but it differs from Control Path as DataPath eventually conforms to 100% coverage.

The process starts from simple wiggling to exploring complex behaviors formally. This allows designers to create a simple formal test environment and helps reduce design development and debug effort. There are various tools in the industry which allow creating such formal test environment without writing too many properties. The idea of design exercise is that we are in a preliminary stage of design development and want to quickly gain confidence in its functionality, with a relatively lightweight verification effort. However, we still require a clear-cut plan on what to check and when to declare that design exercise is done. For example, when dealing with a state machine design in control path, the most obvious goal is to check that we can visit every legal state. We may also want to look at specification document and make sure that all paths are asserted and all required behaviors are exercised. In control path, we can decide on exit criteria on the basis of coverage report or time limit.

	Golden SLM to RTL Verification	Design Exercise
Stage of Design Development	SLM is tested, RTL is preliminary or mature	SLM and RTL both are preliminary
Coverage	Requires 100% coverage to claim confidence	Aim is to find initial bugs to ensure confidence.
Verification Effort	Requires designers help to write constraints, FV team help to ensure convergence	Active collaboration from designer for constraints, convergence is required much later when all bugs have been fixed and RTL is mature.

Table 1: Tradition Approach Vs Design Exercise

In DataPath Designs, we have used Design Exercise in a slightly different context. The aim is still to check functionality of the design in early design development, but it differs from Control Path as DataPath eventually conforms to 100% coverage. Current SLM-RTL methodology involves waiting for SLM model to be fully developed and tested using simulation to establish its validity. After that, SLM-RTL FV is done to find bugs and prove that the RTL implementation is correct.

In Design Exercise, we begin validation even before the simulation environment is fully up and SLM has been standardized. The RTL is also, of course, in development stage. This helps us to find bugs in both Fulsim and RTL quickly to enable shift-left.

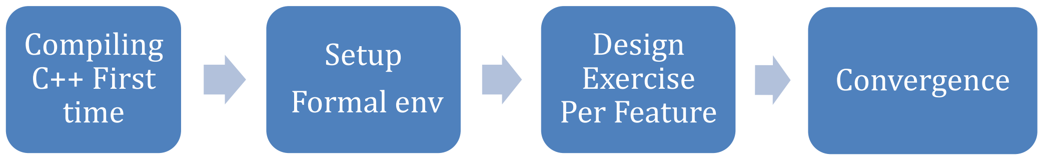

Figure 2 : Design Exercise Flow

The Design Exercise Flow (Figure 2) can be broadly categorized in following steps:

- *Compiling SLM (C++) in DataPath Tool first time*: Despite a lot of tool/vendor support, compiling a C++ specification in a Formal tool can at times be very tedious due to non-formal friendly constructs. In our experience, this results more from the style of coding of a designer rather than actual arithmetic intent of core algorithm. In Design Exercise, we have an added advantage that the C++ code is in a preliminary stage and we are in active sync with designer, so the core algorithm can be written in a formal friendly fashion for relatively quicker compilation. As a one time effort, we also have to work on resolving linker errors arising from interaction between multiple header files but in our experience, once this step is complete, further iterations of C++ get compiled with ease.
- *Set up Formal Environment*: We define the mappings between SLM and RTL. Direct mappings may not exist so this requires defining a correct wrapper around either SLM or RTL to match the interface. We also define the constrained setup under which SLM-RTL input output equivalence is expected to be established.
- *Design Exercise per Feature*: The design exploration maximizes its return on investment if targeted one feature at a time. Once the basic setup is ready, the onus is on designers to debug and resolve failures. If there is a gap in their understanding with respect to intent of the algorithm, prompt help from architect can be taken. During each iteration, a counter example is generated with shortest path to failure which makes it easy to debug whether C++ or RTL needs to be changed.
- *Convergence*: Once all failures are resolved, we may face convergence issues because of huge gate count of design. This requires specialized treatment using path analysis, cutpoints, case-splits etc. Formal guarantees 100% coverage for DataPath algorithm.

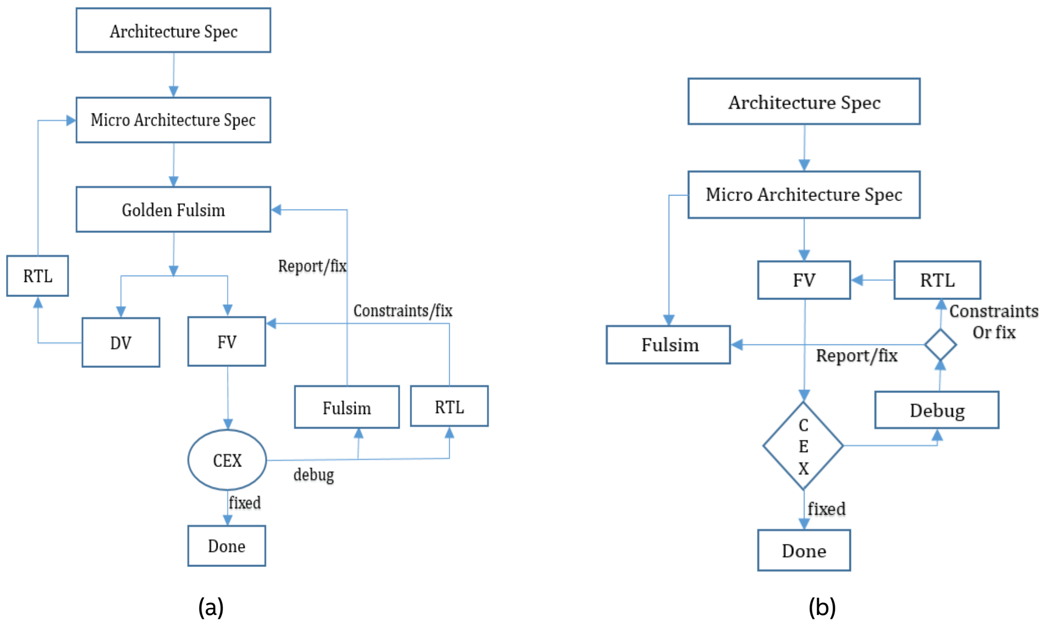

Figure 3: Validation flow differences between (a) SLM to RTL and (b) Design Exercise

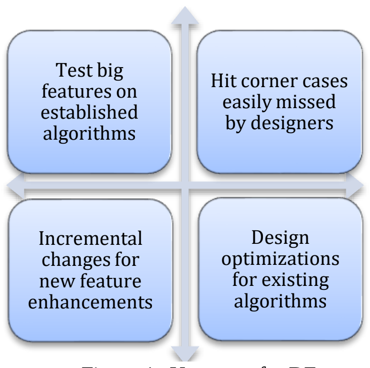
Test big features on established algorithms
Hit corner cases easily missed by designers
Incremental changes for new feature enhancements
Design optimizations for existing algorithms

Figure 4 : Use cases for DE

Conventionally, the norm is to take a golden specification and use it to validate a piece of implementation (RTL) mostly at an algorithmic level. We switched our method without changing our goal with the new methodology of Design Exercise (DE). As explained in Figure 3, with DE in place, we start validation even before either specification or implementation have been standardized. A huge advantage of DE is that it is capable of vanquishing errors in coding specification model. In DE, designs are modeled in a Formal friendly way since the designer understands the expectations of Formal from an early phase of design cycle. We have also observed the turnaround time for DE is shorter than the current (yet powerful and widely executed) SLM-RTL methodology. Design Exercise has several use cases as shown in Figure 4. In Results, we discuss how we used Design Exercise successfully to close on a new feature in very less amount of time.

(2) RTL-RTL FOR DATAPATH DESIGNS

The chip design requires aggressive power and area targets with much stricter time to market constraints. This has led to manifold increase in design complexity, not only with regards to algorithm changes but also in terms of other incremental feature changes which do not affect the functionality of the algorithm, but immensely affect the power and timing fixes. These features need to be quickly developed to meet market demands and even more quickly validated to make sure that the changes have not affected the design behavior or added any unintended bugs.

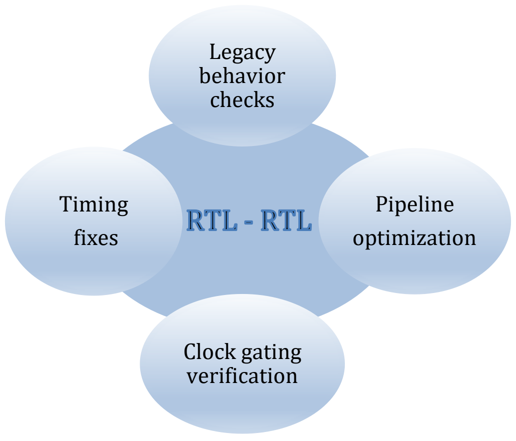

Figure 5 : Use Cases for RTL-RTL

For algorithmic designs, the presence of Formal was limited:
1. Sequential Equivalence Tools do not converge on algorithmic designs
2. C++ environment is not up and the bandwidth required for the setup is not present due to requirement of quick validation

In such a scenario, we have now shown that we can use a DataPath FV tool to do RTL-RTL for such designs. DataPath tools work using DFG's and BDD's and ensure better coverage capability for such designs.

The vendor tool employs abstractions for multipliers, for example, booth multiplier and grade point multliplier. The concept of cutpoints and case-splitting is used to enable optimizations during the run-time. For example, when computing a*b+c we are looking at whether the design computes this value correctly, rather than whether a, b and c arrive and depart at the right time. By reducing this problem to that of transactional equivalence, we can restrict our focus to checking one single transaction as we are checking for all possible input values. Since data-flow-graphs (DFG) are optimized for such analysis, Datapath tools converge much more quickly than their SEC counterparts.

(3) CORROBORATING C MODEL

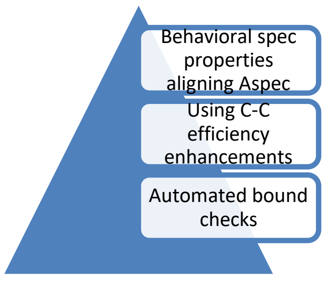
Behavioral spec properties aligning Aspec
Using C-C efficiency enhancements
Automated bound checks

Figure 6: Use Cases

Formal tools allow us to write behavioral properties expected from the code in terms of assertions. Once a setup is ready in a Formal tool, checking such behavioral specifications with respect to expectations from the C++ code can be very quickly achieved with very good results.

Moreover, Formal tools come with built in automated checks for errors like div by zero, array bound checks, null pointer, max iteration checks etc which help to ensure that the Fulsim models are correct with regards to standard. We experimented with several tools for the same. Once a C code is verified and the setup is ready, ensuring that future C models are correct is extremely easy. The use cases for this methodology are presented in Figure 6. In Section II.3, we share an interesting case study.

II. RESULTS/CASE STUDIES

(1) **Design Exercise** – Blend modes are used in image editing to determine degrees of darkness, contrast, inversion, etc. Intel Graphics has an advanced Blend Optimization Algorithm wherein market requirements forced the designer to drive a complex feature to superior quality in a short amount of time. We proposed DataPath Formal to test the algorithm using this version of C++ model. Through FV tool, we found simple mismatches initially and then corner case bugs. In case of some corner case bugs and ambiguity in specification, quick help from Architect was also taken. In the process, we were able to validate RTL as well as a C++ model. Within a week's time and 18 bugs later (including both C++ and RTL) (Figure 7), we had a high quality design and a happy team of designers, architects and management.

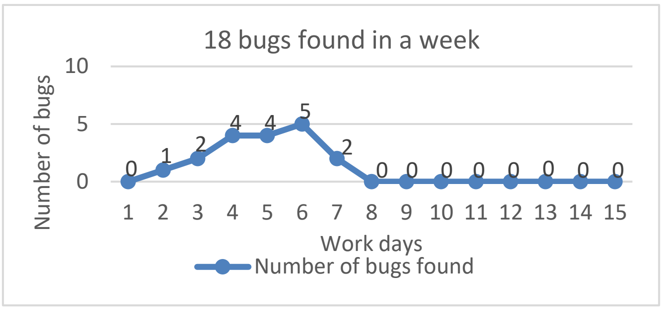

Figure 7: Bugs Found Vs Number of Days

(2) **RTL-RTL** - We have been able to test RTL-RTL methodology on various designs such as integer multiplication, integer fused multiply and add, fixed to float conversion (Table). Not unlike many RTL designs, it had a large number of multiplication and shift operations. These designs could not converge on Sequential Equivalence (SE) Tools for more than a month but they converged on the Datapath FV tool within an hour. See Section 3 for more details.

Algorithm	Datapath FV tool	Traditional Formal Equivalence tool
Int mul	<1 hour*	No convergence
Fix2flt (manual repipelining)	<5min	No convergence
Int mad	<1hour*	No convergence

Table 2: Comparison of convergence between SE and DataPath Formal tools

(3) **Corroborating C Specification Model** – In the DataPath tool, we can quickly test "assert" properties on specification. In one of the designs containing a feedback loop, output was being fed back into the main algorithm as an input. It was incorrectly assumed that when input is positive, output is positive too. Hence, unsigned data type was used for optimizing the design. We extorted the "assert" feature and defined a property on C model to disprove this assumption by a failure of this property with a counterexample. The entire process took less than 2 weeks. The code was revamped to fit the new specifications.

III. SUMMARY

Intel Graphics is abound with deep-rooted complex algorithmic implementations. It has been established that DataPath Formal Verification plays a key role to enable shift left in validation cycle. However, the current SLM-RTL methodology requires a lot of time and effort from an FV engineer in each complex design which limits its scope. We have proposed the use of three new methodologies in Datapath FV – Design Exercise, RTL-RTL and Corroborating Specification model which has the potential to enable widespread use of FV and bringing the benefits of Formal to more and more designers and architects. We have shown through several case studies how use of these methodologies has enabled validating new feature changes, design enhancements etc. in less amount of time with 100% confidence and hence high ROI.

In future, we want to work with C modeling team to ensure that for all new algorithmic designs which are good candidates for FV, core algorithms are written by the designer using formal friendly constructs which can be easily compiled in a DataPath tool. If this can be achieved, the bring up for Formal setup would have a short runway. With the help of trainings and active collaboration, we envision empowering the designers to own these three new methodologies themselves to gain 100% confidence in their designs with no need for simulation. We are actively working on automated scripts for compilation and templates on RTL-RTL to support this vision.

IV. REFERENCES

[1] M V Achutha KiranKumar, Aarti Gupta, and Ss Bindumadhava, "Formal Datapath Verification Against SLM," SNUG 2016.

[2] *RTL2RTL Formal Equivalence : Boosting the Design Confidence.* **M V AchutaKiran Kumar, Aarti Gupta, S S Bindumadhava.** Singapore : s.n., 2014. FSFMA.