Achieving End-to-End Formal Verification of Large Floating-Point Dot Product Accumulate Systolic Designs

Emiliano Morini, Bill Zorn, Disha Puri, Madhurima Eranki, Shravya Jampana

Intel Corporation

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA MARCH 4-7, 2024

Mathematical Functionality

$$\underbrace{\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mp} \end{pmatrix}}_{R^{m,p}} = \underbrace{\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}}_{A^{m,n}} \cdot \underbrace{\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}}_{B^{n,p}} + \underbrace{\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}}_{C^{m,p}}$$

DPn.5

$$r_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} + c_{ij}$$

$2n+1$ floats with precision t
4 rounding modes
$\Rightarrow$ $2^{(2n+1)t+2}$ input values for each r_{ij}

Usually implemented in Dot Product Accumulate Systolic units

DPn.5	DP4.5	DP4.5	DP4.5	DP8.5	DP8.5
Precision	16	32	64	32	64
Input Space	2^{146}	2^{290}	2^{578}	2^{546}	2^{1090}

Hard to verify due to the **massive input space** and the **arithmetic** involved

Golden Reference Models: iFP Library

- Fully specified, IEEE 754 compliant, FV friendly
- Uses template metaprogramming to produce multiprecision operands with a single code base
- Based on the open-source FPCore language

Example: Mixed Precision Multiplication

Native C++

```
float x; double y;
float result = x * y;
```

Incorrectly rounded (double rounding)

iFP

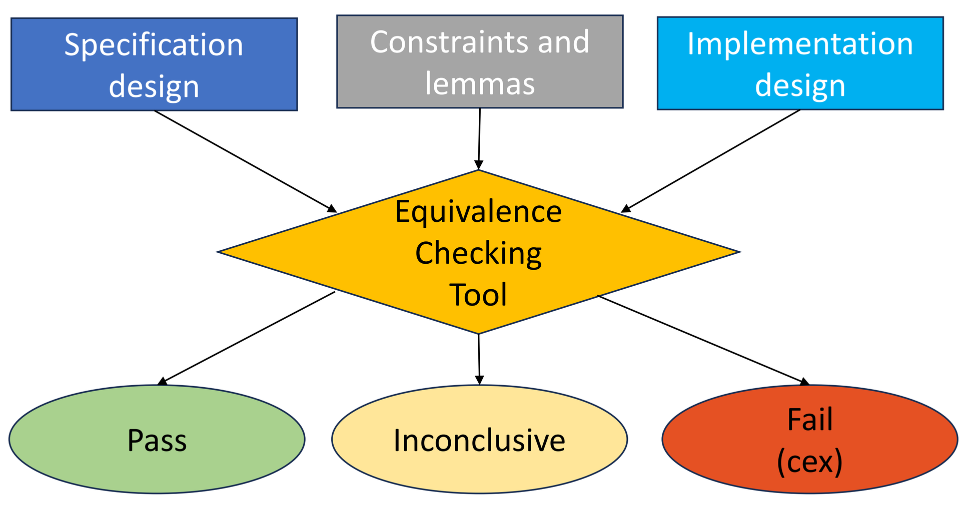
```
using FP32 = RoundingContext<8, 32>;
using FP64 = RoundingContext<11, 64>;
FP32::Unpacked x; FP64::Unpacked y;
auto result = fp_mul<FP32>(x, y);
```

Correctly rounded (single rounding)

iFP Validation

- Sim (MPFR)
- FV (SoftFloat)
- C++ Asserts (Checked in Sim & FV)

Formal Verification Approach: Equivalence Checking



C2RTL, RTL2RTL, C2C

"SYMMETRY INVARIANCE" METHODOLOGY

1. Perform Equivalence Checking C2RTL for r_{11} against iFP DPn.5 model
2. Prove that the other values r_{1j} in the first row are computed as r_{11} with an RTL2RTL **(column symmetry invariance for the first row)**
3. Prove that the values r_{ij} in the following rows are computed as r_{1j} with an RTL2RTL **(row symmetry invariance for other rows)**

Parallel approach
(the 3 steps progress at the same time)
+
Risk diversified

Initial single transaction verification

- Only a single instruction into the pipeline
- Over constrained environment, helpful to create the final correct setup and show arithmetic complexities

Correctness of the setup

- Write the properties in the RTL
- Test them in Sim and FPV
- Assume them in C2RTL

Final multiple transactions verification

- Back-to-back transaction enabled
- Over constraints removed

C2RTL for r_{11}: Double Precision Complexity

Inclusive results out of the box, **proof decomposition** required

Internal equivalence points between C++ and RTL are unlikely, due to the RTL optimizations.

Find **equivalent relations** among multiple signals:

$$impl.prod == (spec.prod << spec.ren)$$

Abstract the design introducing **cutpoints** based on the relations proven

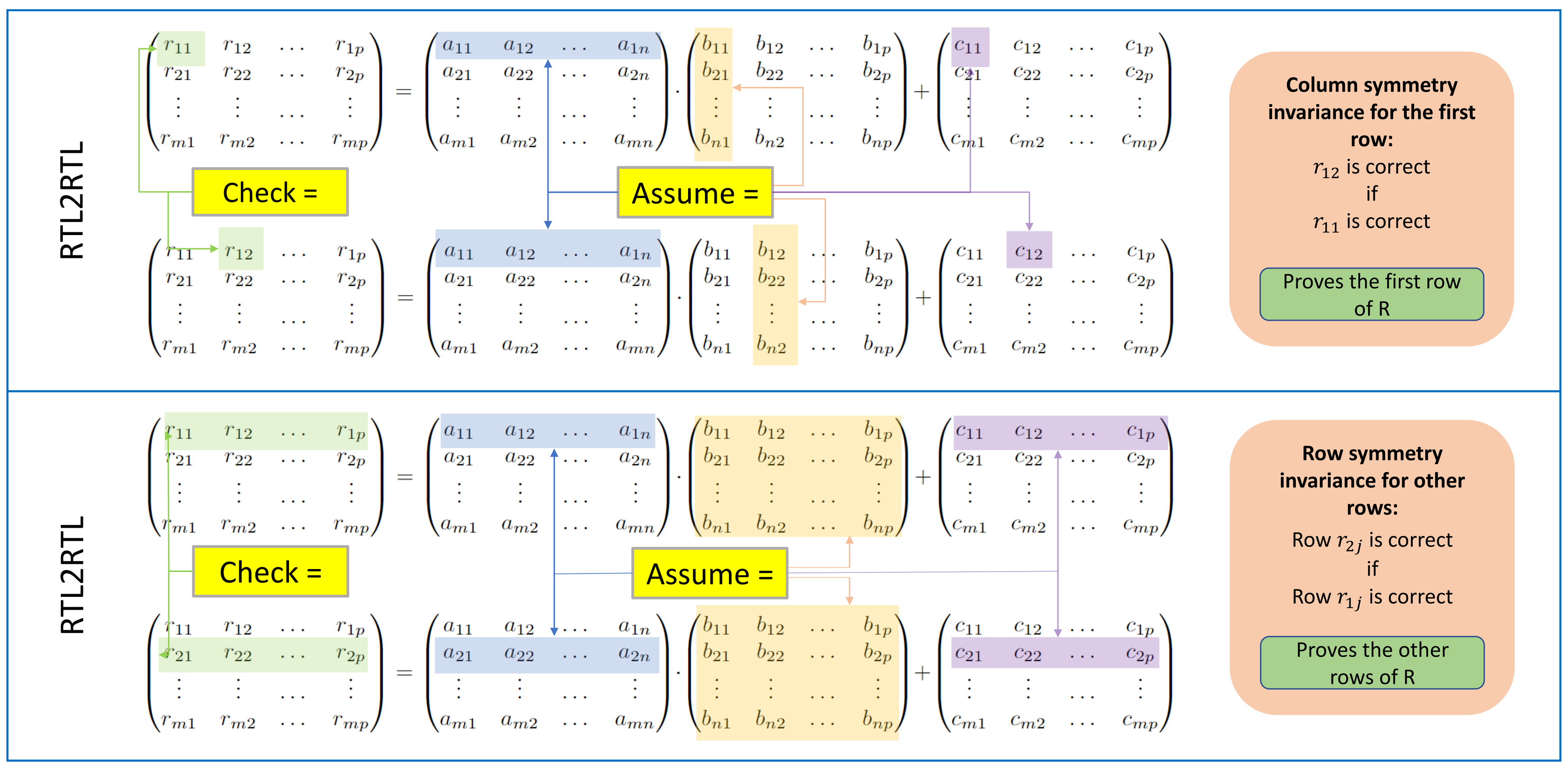
Vast usage of **case-splits** and **assume-guarantee** approaches

Due to RTL optimizations, the double precision multiplication involved in a DPn.5 was inconclusive.

Convergence achieved using the EC tool as an automatic theorem, proving these steps:

- **Encoding**
- **Array creation**
- **Array reduction**
- **CSA form**

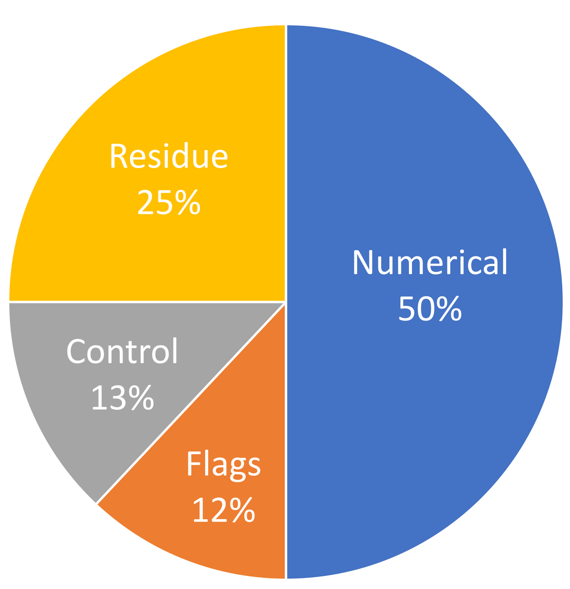
RTL2RTL Symmetry Invariance



Results & Conclusions

Bugs found during the process

- Good initial results with bug hunting approaches (e.g., constrain most of the mantissas of the inputs to 0)
- Detected hard corner cases, happening with a probability of less than 2^{-50}, **practically impossible to find via random sim!**



Proven the absence of bugs in the final RTL

Completed formal equivalence between RTL and iFP C++ model (**independent**, **validated** and **FV friendly**) + Symmetry techniques

Our approach

- Scales up to designs with **hundreds of input bits**
- Can be reused to **verify similar components**
- Enables a **shift left** for every future generation of RTL
- Fully verifies large double precision DPAS in **< 24h**
- Is used in for **regressions** (smoke testing and weekly)

We developed advanced proofs leveraging RTL, computer arithmetic and FV skills, **obtaining results never been achieved before**