2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

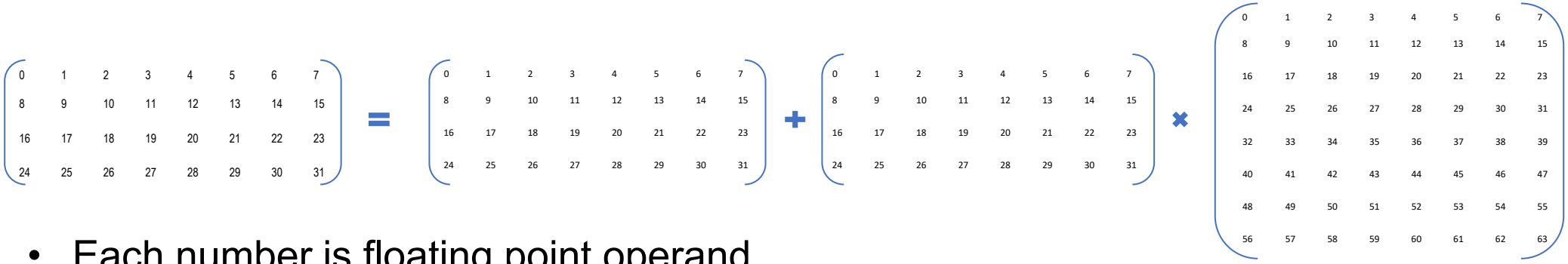
A streamlined approach to validate FP matrix multiplication with formal

Gerardo Nahum, Siemens EDA
Nicolae Tusinschi, Siemens EDA
Seiya Nakagawa, Siemens EDA

accellera SYSTEMS INITIATIVE

Floating point matrix multiply accumulate

Imagine how many operations you require to calculate the following



- Each number is floating point operand
- Simulation methods would take months to start finding bugs
- Exhaustive check for interesting cases and different types of operands / operations is a must
- FPU app includes IEEE 754 floating point building blocks :
  - ADD, SUB, MUL operations and Conversions, and taking in consideration different rounding modes

Single function calculation

- Each element of the resulting matrix is calculated as follows
  - R[i] = ACC[i] + Row Matrix X * Column Matrix Y

- This is a Vector Fuse Multiply and Accumulate operation, which requires to be populated with the relevant Row and Column elements of the Matrixes

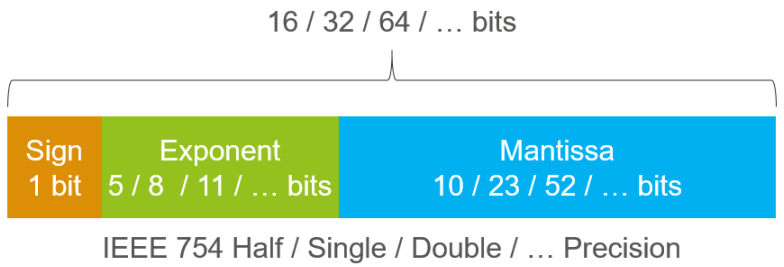
- We've built a new VFMA operation as follows
  VFMA = ACC + X0∗Y0 + X1∗Y1 + X2∗Y2 + X3∗Y3 + X4∗Y4 + X5∗Y5 + X6∗Y6 + X7∗Y7
  - Becomes a basic building block to check each result
  - Support different floating-point types

Floating-Point Unit (FPU) app

- Formally verifying compliance to the IEEE 754 standard

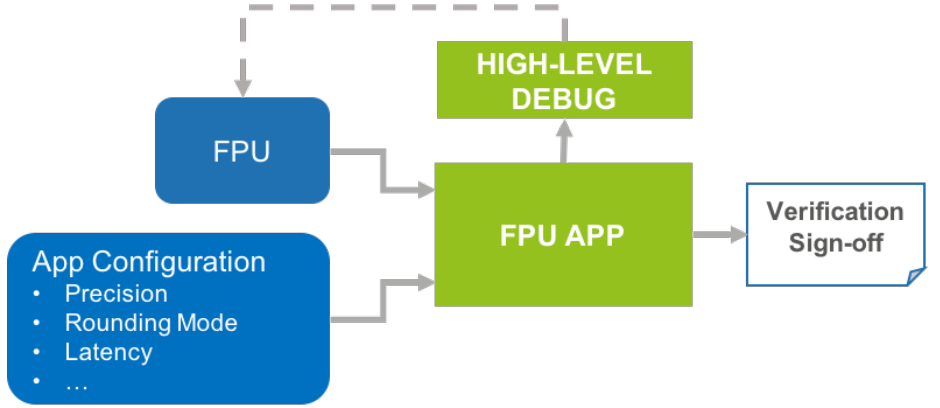
Challenges:

- Floating-point essential for advanced artificial intelligence (AI) applications such as deep learning
- Complex IEEE 754 floating-point specification
  - Arithmetic and comparison operations
  - Bfloat16, half, single, and double precision
  - Five rounding modes
  - Five exception flags
- Simulation cannot guarantee standard has been met
  - Only a formal App can prove compliance



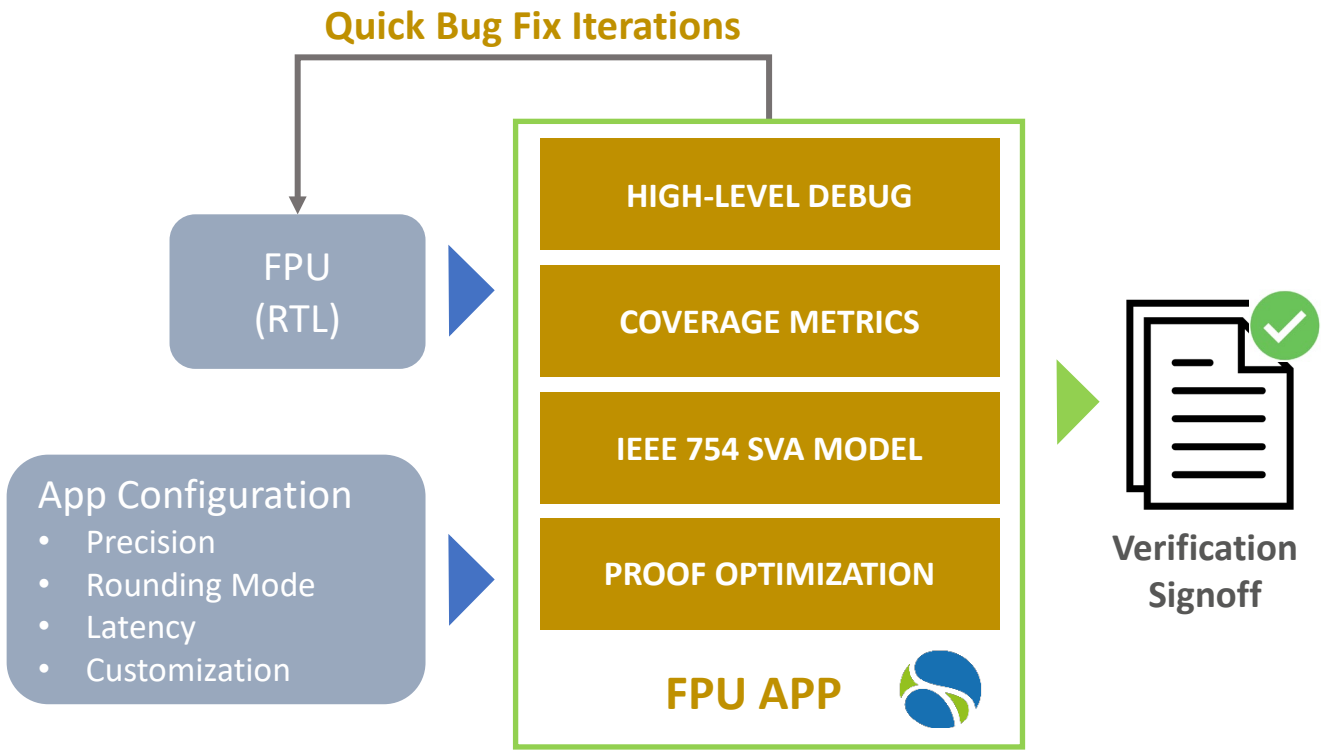
OneSpin Solution:

- Compliance rules captured using standard SystemVerilog Assertions (SVA)
- Supports all operands, rounding modes, and exception flags
- Highly automated formal proof strategies
- Parallel proof engines with network and cloud distribution
- Floating point value view of operands for debugging
- Integrates with RISC-V F/D extensions



OneSpin 360 FPU verification app

Accelerate verification, prove correctness



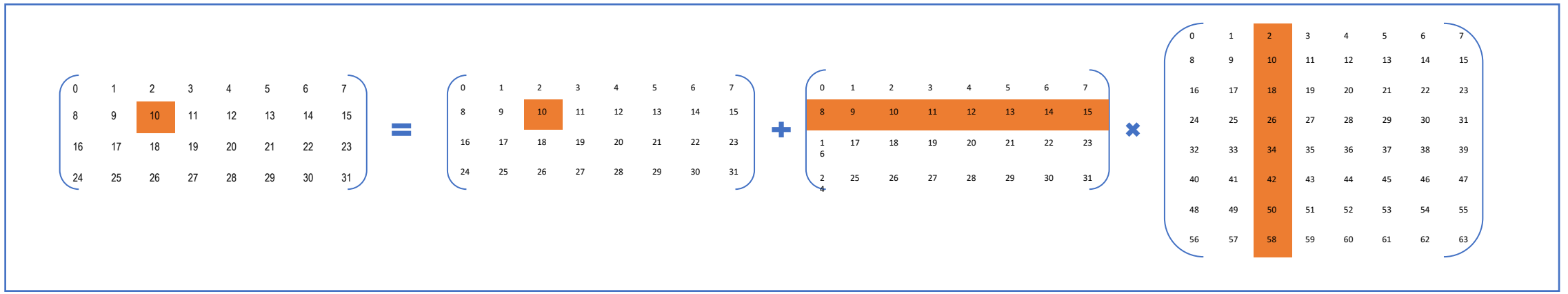
- Easy to setup
- Supports half/single/double bfloat16 and custom precisions
- Supports 10 rounding modes and 5 exceptions flags
- add, sub, mul, fma, abs, neg
- Conversion and comparison operations
- Parameters to specify ambiguities in the standard
- RISC-V configuration
- No need for C++ model of the FPU
- Easy to model intended deviations from the IEEE-754 standard

Matrix Multiplication

Example

In order to calculate Element 10 of the Matrix:

R10 = ACC10 + X8*Y2 + X9*Y10 + X10*Y18 + X11*Y26 + X12*Y34 + X13*Y42 + X14*Y50 + X15*Y58



Use case

- Template Based
  - Simplify property writing
  - Reduces debug time
  - Enables fast transfer of fails to RnD for further debug and fixes
  - created a procedure to download fail vectors

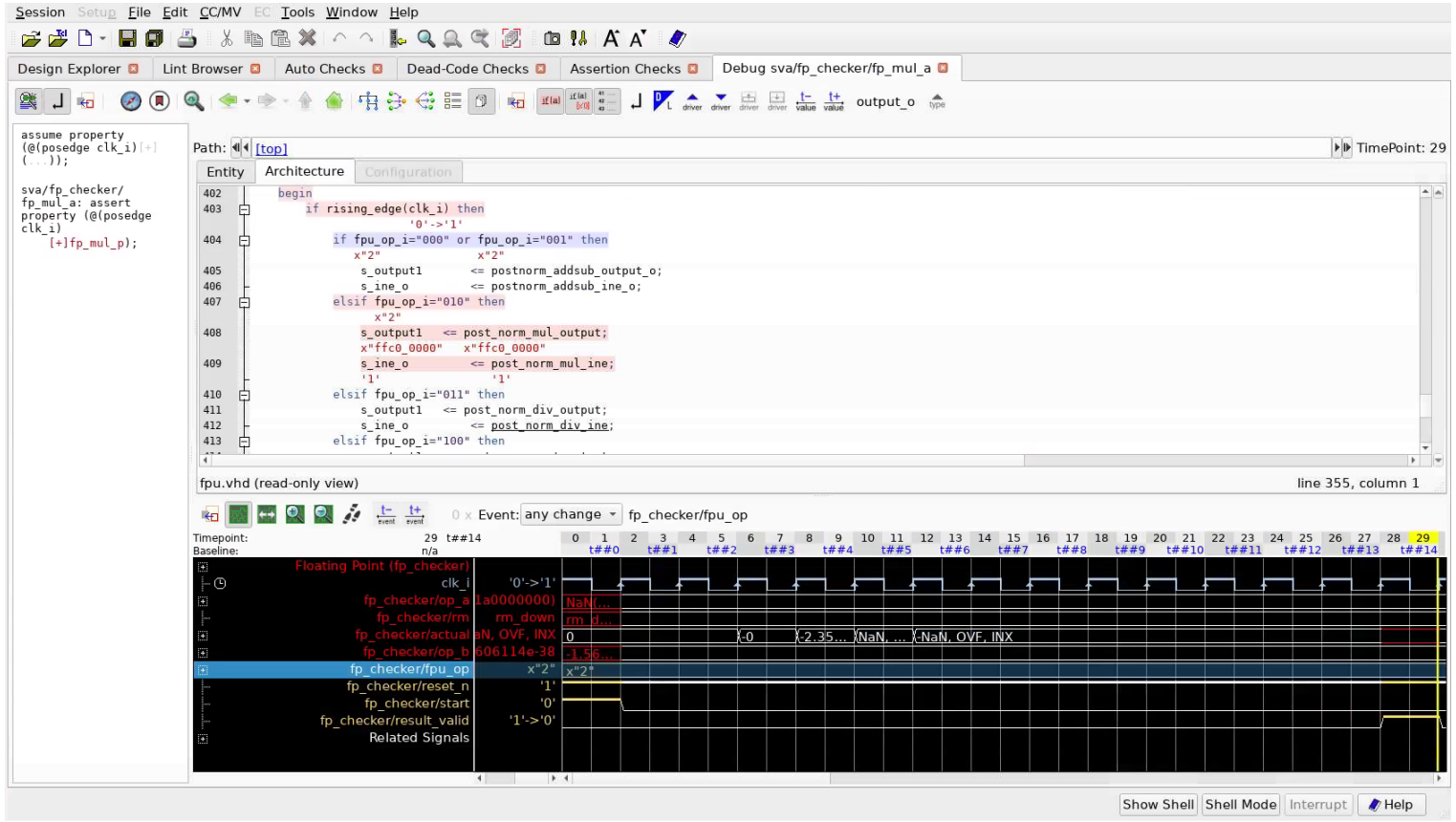
```
property check_op (input integer k) ;
      ieee_t local_prod ;
      ieee_t local_acc  ;
      ieee_t expected   ;
      ##0 operation = MAC
      ##1 (1, local_prod = prod [k])
      ##1 (1, local_acc  = acc[k] )
      ##1 (1,expected = vfma (.op(local_acc), .prod(local_prod), .rm(roundmode) )
      ##X operation = NOP
      |->
      ieee_check_result (.expected(expected), .actual (design_result_with_flags)  );
endproperty

genvar element,i
generate
for (element =0 ; element < 16 ; element++)
      for(i=0;i<8;i=i+1) begin:
                  prod[element][2*i]   = MX[element/8*8+i];
                  prod[element][2*i+1] = MY[element%8+i*8];
                  acc[element] = design_acc_vector[32*(i+1):32*i];
      end
      asrt_element : assert property check_op (element);
endgenerate
```

Property Template

User Data

Debugging fails



- IEEE 754 annotations
- on code and waveform
- Traceability (drivers and loads)
- Property Debugger shows fails
- Active code marking

Results

Prevented a bug escape !

Found an error when having a small accumulator exponent and large product exponent but zeros on mantissa

- We've implemented several operations reusing the same function

  i.e. NEG - Negate the accumulator with no matrix multiplication

```
neg = vfma(.op(acc) , .prod (‘0) , .rm(roundmode) ) ;
```

  MUL- Only calculate the product, ignore the accumulator

```
mul = vfma(.op(‘0), .prod(prod) , .rm(roundmode) ) ;
```

- Were able to fully prove Addition, Negation and other operations
- Full proof on restriction of the multiplication having either all zeros or special numbers (i.e NaN etc)
- Full proof for 2 multiplications being non zero and all other zero's

Runtime Results

Results per lane, before and after fixes

Operation	Unrestricted Before fix	Has special numbers (NaN or Inf)	Unrestricted After fix	Restricted 1 multiplication After fix	Restricted 2 multiplications After fix
VADD/VSUB/VNEG	No fails	20 sec	20sec		
VMUL (Accumulator is zero)	1 min	4min	Hold bounded	10min for full prove	4h for full prove
VFMA	1 min	4min	Hold bounded	9h for full prove	Hold bounded

Summary

- FPU operations are tedious and difficult to verify using simulation
- Bugs are on corner cases
- Questa OneSpin FPU app has the building blocks to construct simple readable properties
- Provers and disprovers performance enables bug finding in minutes
- Full proof is possible on restricted cases
- Bounded proof is available for all cases