2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Formal-driven assurance of RISC-V Cores with AI-Ready FPUs

CY Chuang – Siemens EDA

Nicolae Tusinschi – Siemens EDA

accellera SYSTEMS INITIATIVE

Challenges of processor verification

Complex architecture

- (Custom) extensions
- Exceptions/ Interrupts

Very complex µArchitecture

- Continuous PPA optimizations
- Pipelined implementation

Verification – high effort task

- Writing functional coverage model
- Simulation cannot hit all pipeline corner-cases
- Slow debug process
- Functional and structural coverage closure
- Customization introduces bugs in existing functionality

Challenges with FPU

- Floating-point essential for advanced AI applications such as deep learning, image recognition and more
- Complex IEEE 754 floating-point specification
  - Vector, Arithmetic, Conversion, and comparison operations
  - Wide and Narrow Datatypes
  - Different Rounding modes
  - Handling exception flags
- IEEE 754 design and verification experts are in short supply

Legacy verification methodology

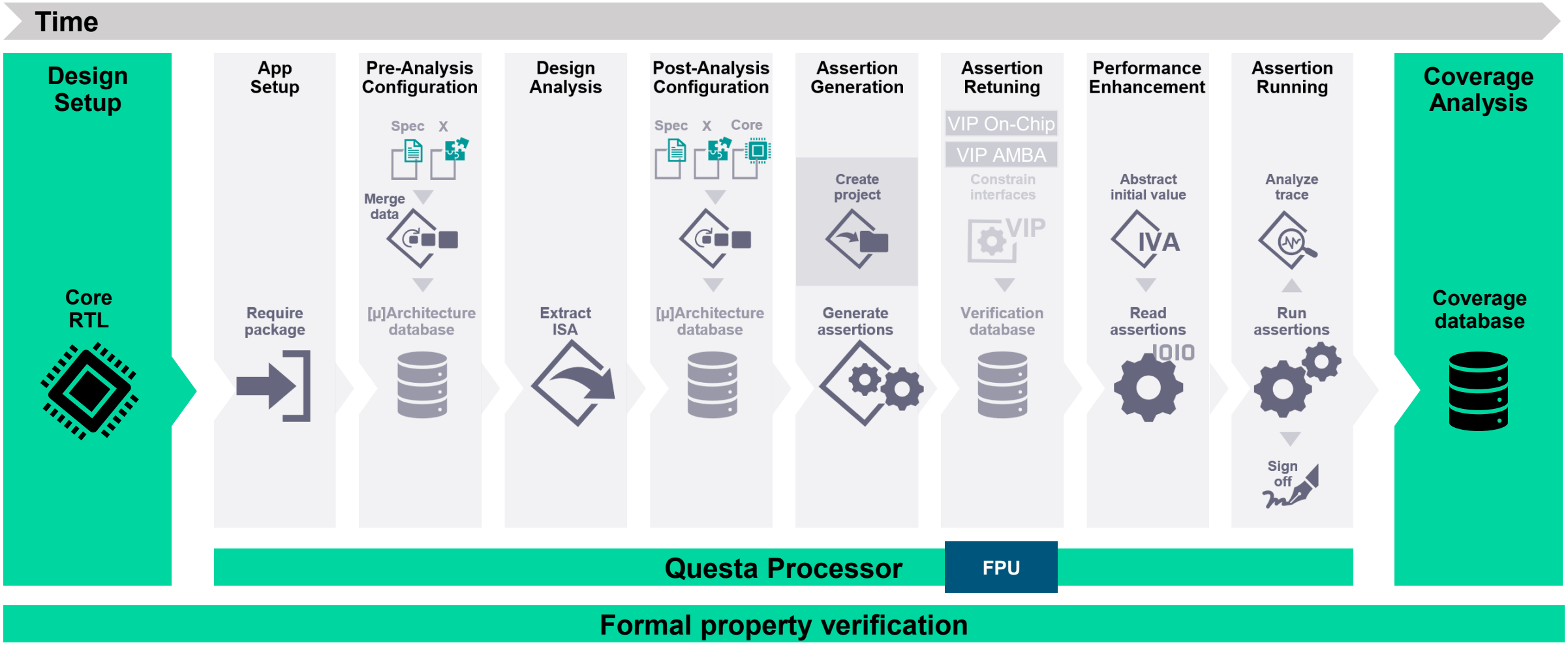
Simulation

- The target is to build a testbench that generates different scenarios of operands, round modes and sequences
  - Not exhaustive, time consuming
  - Scenario based, leaving corner cases uncovered

Equivalence to C-model

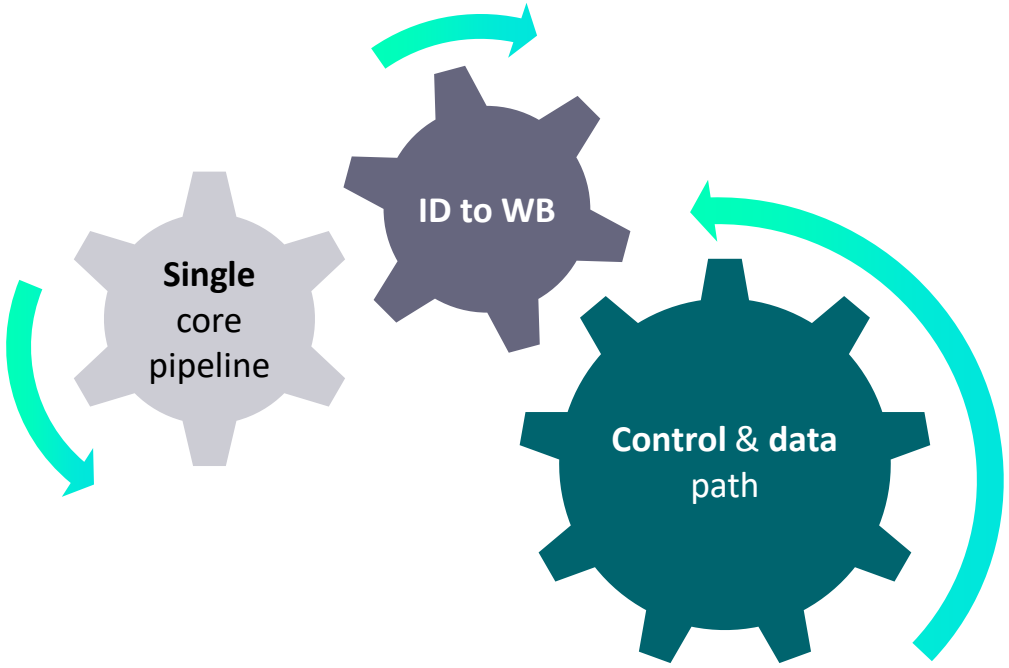
- The methodology is to formally compare the RTL with a C-model representation of the FPU
  - Not always succeed
  - Difficult to perform the mapping
  - Never work as expected
    - creating false positives
  - Complex debug effort in case of a failure

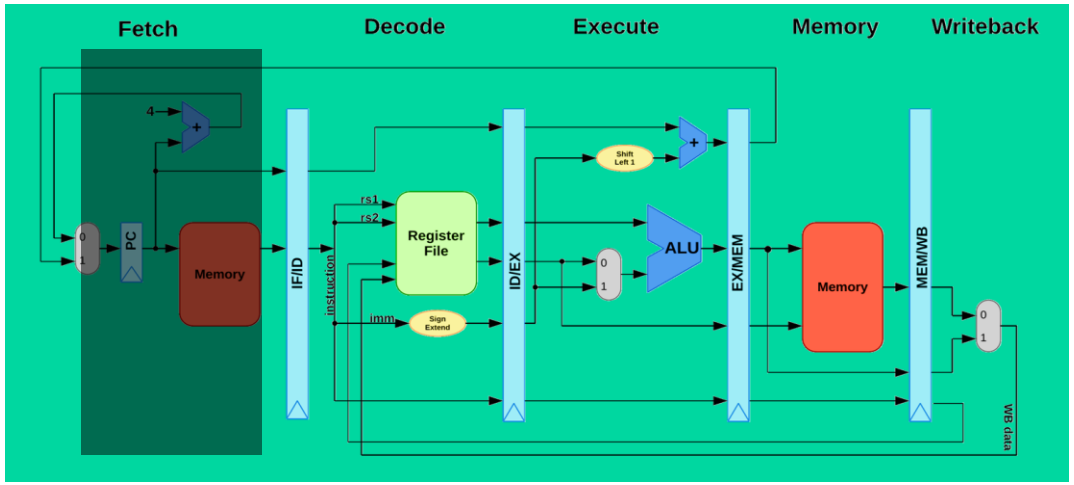
Complete flow to verify RISC-V cores



Questa Processor
Enables highest confidence in the trust and security of processor implementations

- Ideal for in-order execution core implementations with single & multi-issue capabilities
- Detect any inconsistencies between RTL core implementation and the ISA
- Address the needs of core providers and core integrators



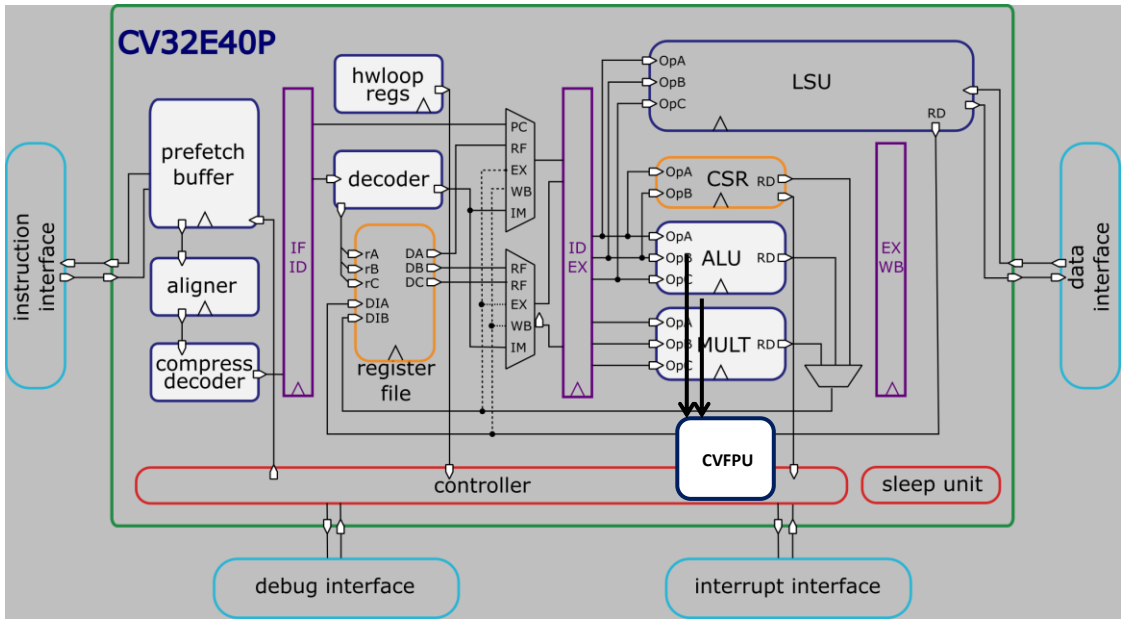


Application example
32-Bit CV32E40Pv2 Core

+ X custom instruction set extensions

- Post-Incrementing Load & Store
- ALU
- Multiply Accumulate
- Single Instruction Multiple Data (SIMD)
- Hardware Loops (zero-overhead loops)

RV32IM[F]C_Zicsr_Zifencei[_Zfinx][_Xpulp][_Xcluster]



31

https://github.com/openhwgroup/cv32e40p/issues

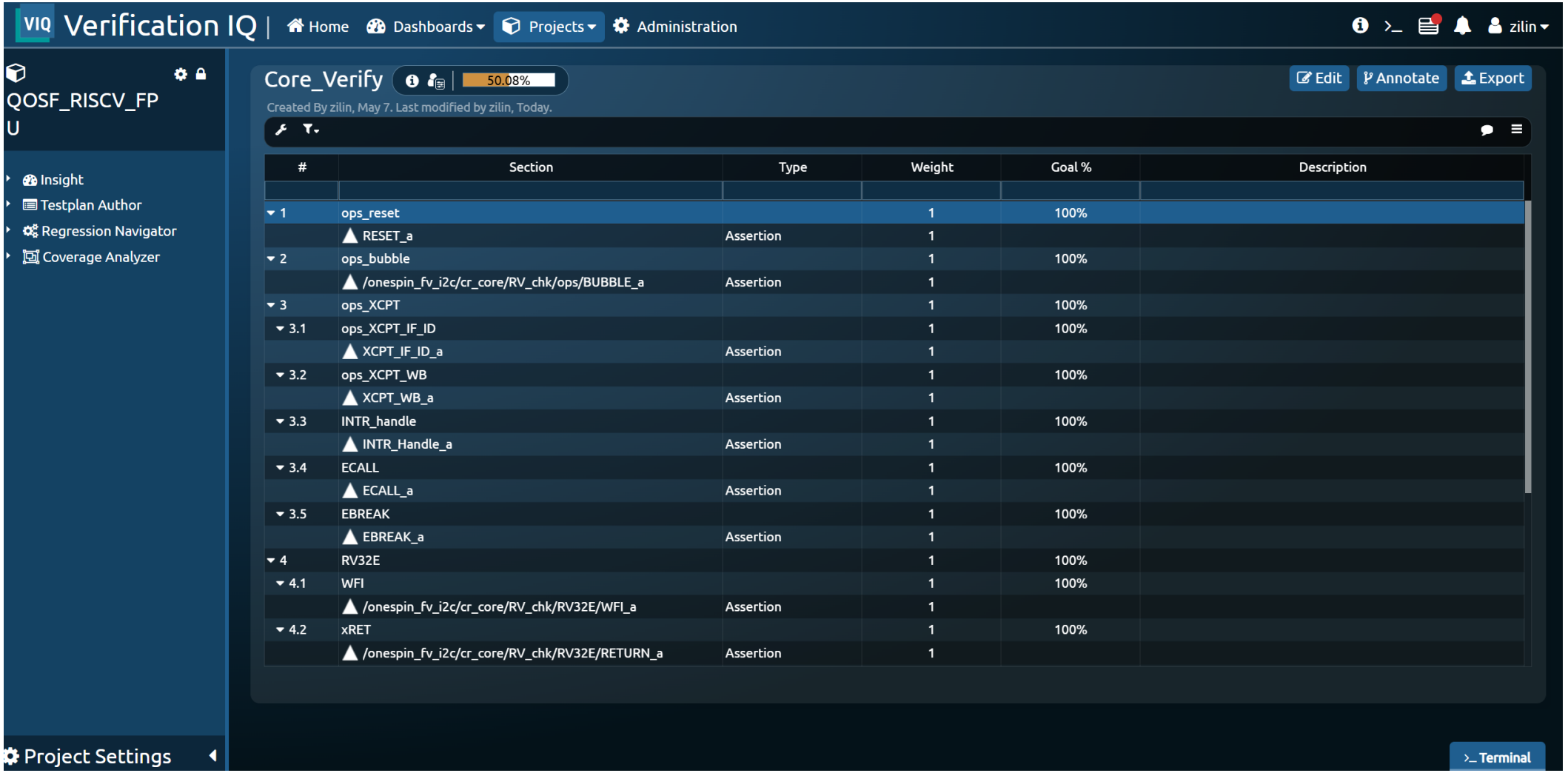
https://github.com/openhwgroup/cv32e40p

7 Standard RISC-V Extensions
+1 **Custom RISC-V Extension**

114 Standard Instructions
+320 **Custom Instructions**

21 Standard CSRs
+8 **Custom CSRs**

Tight integration with test planning

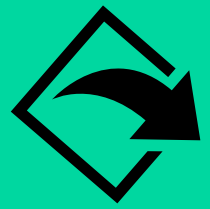


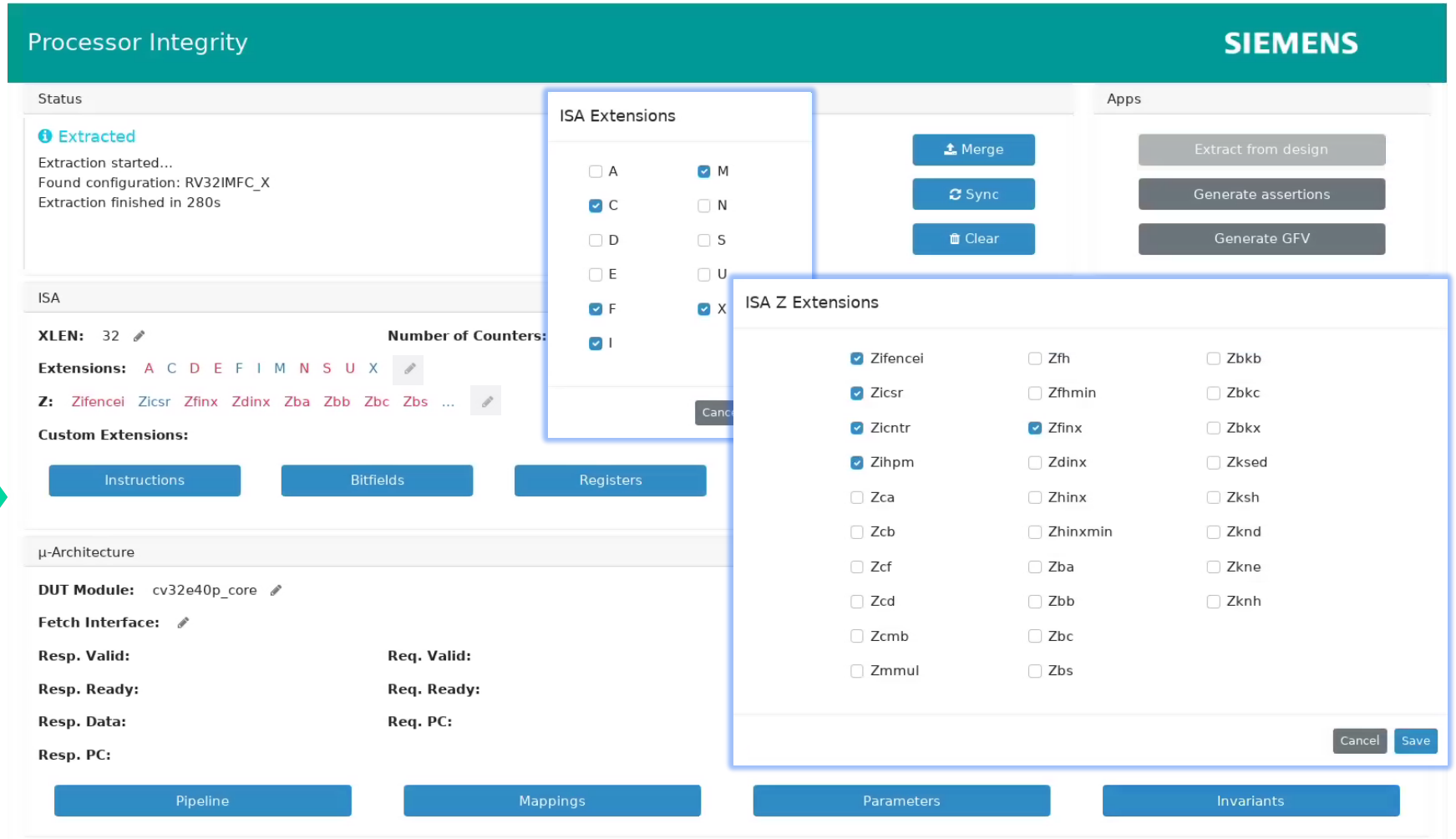
#	Section	Type	Weight	Goal %	Description
1	ops_reset		1	100%	
	RESET_a	Assertion	1		
2	ops_bubble		1	100%	
	/onespin_fv_i2c/cr_core/RV_chk/ops/BUBBLE_a	Assertion	1		
3	ops_XCPT		1	100%	
3.1	ops_XCPT_IF_ID		1	100%	
	XCPT_IF_ID_a	Assertion	1		
3.2	ops_XCPT_WB		1	100%	
	XCPT_WB_a	Assertion	1		
3.3	INTR_handle		1	100%	
	INTR_Handle_a	Assertion	1		
3.4	ECALL		1	100%	
	ECALL_a	Assertion	1		
3.5	EBREAK		1	100%	
	EBREAK_a	Assertion	1		
4	RV32E		1	100%	
4.1	WFI		1	100%	
	/onespin_fv_i2c/cr_core/RV_chk/RV32E/WFI_a	Assertion	1		
4.2	xRET		1	100%	
	/onespin_fv_i2c/cr_core/RV_chk/RV32E/RETURN_a	Assertion	1		

Performing an automated design analysis and configuration extraction

Design Analysis

Extract ISA

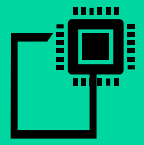




Post-Analysis Configuration

Post-Analysis Configuration

Core details



[µ]Architecture database

Custom extensions

Processor Integrity — SIEMENS

Status

Customized

Merged data for RV32IMFCZicsr_X from file '/bata/shetalan/cve/test/shell/RISCV/CV32E_V2/Xpulp.json'

Merge

Apps

Extract from design

ISA

XLEN: 32

Extensions: A C D E F

Z: Zifencei Zicsr Zfinx Zd

Custom Extensions:

Instructions

µ-Architecture

DUT Module: cv32e40p_cor

Fetch Interface:

Resp. Valid:

Resp. Ready:

Resp. Data:

Resp. PC:

ISA Custom Extensions - Instructions

	Mnemonic	Decoding	Restrictions	Disassembly	Execution
☐	CV.LB.I	imm[11:0] rs1/rd2 000 rd 0001011		cv.lb.i {rd},{imn	let addr : xlenbits = X(rs1) + EXTS(imm); X
☐	CV.LH.I	imm[11:0] rs1/rd2 001 rd 0001011		cv.lh.i {rd},{imn	let addr : xlenbits = X(rs1) + EXTS(imm); X
☐	CV.LW.I	imm[11:0] rs1/rd2 010 rd 0001011		cv.lw.i {rd},{imn	let addr : xlenbits = X(rs1) + EXTS(imm); X
☐	CV.EXTHS	0110000 00000 rs1 011 rd 010101		cv.exths {rd},{r	X(rd) = EXTS(X(rs1)[15..0])
☐	CV.EXTHZ	0110001 00000 rs1 011 rd 010101		cv.exthz {rd},{r	X(rd) = EXTZ(X(rs1)[15..0])
☐	CV.DOTSI	1001000 rs2 rs1 001 rd 1111011		cv.dotsp.b {rd},	X(rd) = EXTS(muls(X(rs1)[7..0],X(rs2)[7..0]
☐	CV.SDOTI	1001100 rs2 rs1 001 rd/rs3 111101		cv.sdotup.b {rd}	X(rd) = X(rs3) + EXTZ(mul(X(rs1)[7..0],X(rs

Add Instruction | Remove Instruction(s)

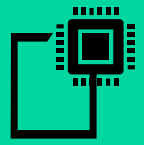
Cancel | Save

Pipeline | Mappings | Parameters | Invariants

Post-Analysis Configuration

Post-Analysis Configuration

Core details



[µ]Architecture database

Custom extensions

Processor Integrity

SIEMENS

Status

ISA Custom Extensions - Registers

	Name	Type
☐	lpstart<0:1>	XLEN
☐	lpend<0:1>	XLEN
☐	lpcount<0:1>	XLEN
☐	uhartid	XLEN

Add Register

ISA Custom Extensions - CSR Map

	Name	Address	Restrictions	Read
☐	lpstart<0:1>	0x800+<4*i>		lpstart<i>
☐	lpend<0:1>	0x801+<4*i>		lpend<i>
☐	lpcount<0:1>	0x802+<4*i>		lpcount<i>
☐	uhartid	0xcc0		uhartid
☐	privlv	0xcc1		curPrivilege

Add CSR | Remove CSR(s)

Cancel | Save

Instructions | Bitfields | Registers | Register Files | CSR Map | CSR Attributes

µ-Architecture

DUT Module: cv32e40p_core **DUT Instance:** core_i

Fetch Interface: **Data Memory Interface:**

Resp. Valid: **Req. Valid:** **Req. Valid:** **Resp. Valid:**

Resp. Ready: **Req. Ready:** **Req. Ready:** **Resp. Ready:**

Resp. Data: **Req. PC:** **Req. Address:** **Read Data:**

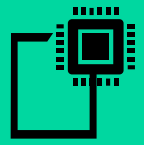
Resp. PC: **Write Data:** **Req. Cancel:**

Pipeline | Mappings | Parameters | Invariants

Post-Analysis Configuration

Post-Analysis Configuration

Core details



[µ]Architecture database

Custom extensions

Processor Integrity — SIEMENS

Status

Customized

Merged data for RV32IMFCZicsr_Zifencei_X from configuration 'cv32e40p'

Merge | Sync | Clear

Apps

Extract from design | Generate assertions | Generate GFV

ISA

XLEN: 32 **Number of Counters:** 4 **Number of PMPs:** 0

Extensions: A C D E F I M N S U X **Reset PC:** 0

Z: Zifencei Zicsr Zfinx Zdinx Zba Zbb Zbc Zbs ... **S:** Smepmp Smstateen

Advanced

Custom Extensions:

Instructions | Bitfields | Registers | Register Files | CSR Map | CSR Attributes

µ-Architecture

DUT Module: cv32e40p_core **DUT Instance:** core_i

Fetch Interface: **Data Memory Interface:**

Resp. Valid: if_stage_i.fetch_valid **Req. Valid:** instr_req_o **Req. Valid:** data_req_o **Resp. Valid:** data_rvalid_i

Resp. Ready: 1'b1 **Req. Ready:** instr_gnt_i **Req. Ready:** data_gnt_i **Resp. Ready:** 1'b1

Resp. Data: if_stage_i.instr_aligned **Req. PC:** instr_addr_o **Req. Address:** data_addr_o **Read Data:** data_rdata_i

Resp. PC: pc_if **Write Data:** data_wdata_o **Req. Cancel:** 1'b0

Pipeline | Mappings | Parameters | Invariants

Automated assertion generation

Assertion Generation

Generate assertions

Processor Integrity

SIEMENS

Status

Generated

Generating assertions for RV32IMFCZicsr_Zifencei_Zfinx_X
426 assertions generated for 427 instructions and 29 CSRs

Merge | Sync | Clear

Apps

Extract from design | Generate assertions | Generate GFV

ISA

XLEN: 32 **Number of Counters:** 4 **Number of PMPs:** 0

Extensions: A C D E F I M N S U X **Reset PC:** 0

Z: Zifencei Zicsr Zfinx Zdinx Zba Zbb Zbc Zbs ... **S:** Smepmp Smstateen

Advanced

Custom Extensions:

Instructions | Bitfields | Registers | Register Files | CSR Map | CSR Attributes

µ-Architecture

DUT Module: cv32e40p_core **DUT Instance:** core_i

Fetch Interface: **Data Memory Interface:**

Resp. Valid: if_stage_i.fetch_valid **Req. Valid:** instr_req_o **Req. Valid:** data_req_o **Resp. Valid:** data_rvalid_i

Resp. Ready: 1'b1 **Req. Ready:** instr_gnt_i **Req. Ready:** data_gnt_i **Resp. Ready:** 1'b1

Resp. Data: if_stage_i.instr_aligned **Req. PC:** instr_addr_o **Req. Address:** data_addr_o **Read Data:** data_rdata_i

Resp. PC: pc_if **Write Data:** data_wdata_o **Req. Cancel:** 1'b0

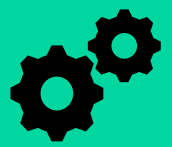
Pipeline | Mappings | Parameters | Invariants

RISC-V with FP extension checkers

Assertion Running

Sign off

Run assertions



Analyze trace

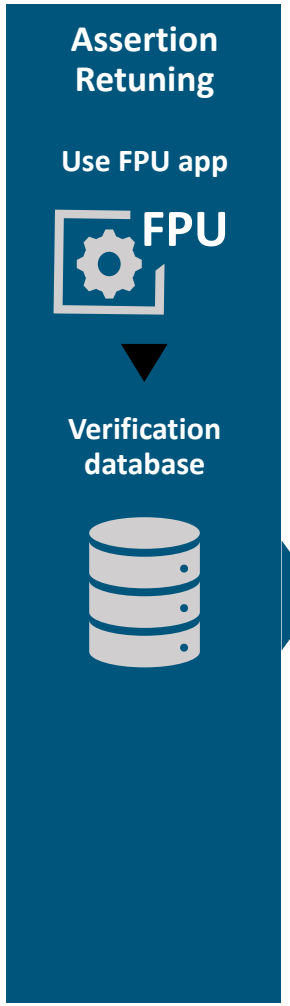
Name	Proof Status	/itness Statu	se Split Stat	Validity
!	! <any status>	! <any	! <any	! <any validity>
Constraints				
Cover Statements				
Properties				
RV_chk.ops.BUBBLE_a	hold	open	open	up_to_date
RV_chk.ops.RESET_a	fail (1)	...eachable	open	assertion_changed
RV_chk.RV32C.ARITH_a	hold	open	open	up_to_date
RV_chk.RV32C.BRANCH_a	hold	open	open	up_to_date
RV_chk.RV32C.JUMP_a	hold	open	open	up_to_date
RV_chk.RV32C.MEM_a	fail (4)	open	open	up_to_date
RV_chk.RV32C.MEM_MultiAccess_a	fail (4)	open	open	up_to_date
RV_chk.RV32E.ARITH_a	hold	open	open	up_to_date
RV_chk.RV32E.BRANCH_a	hold	open	open	up_to_date
RV_chk.RV32E.FENCE_a	hold	open	open	up_to_date
RV_chk.RV32E.JUMP_a	hold	open	open	up_to_date
RV_chk.RV32E.MEM_a	fail (4)	open	open	up_to_date
RV_chk.RV32E.MEM_MultiAccess_a	fail (4)	open	open	up_to_date
RV_chk.RV32E.RETURN_a	hold	open	open	up_to_date
RV_chk.RV32E.WFI_a	hold	open	open	up_to_date
RV_chk.RV32M.DIV_a	hold	open	open	up_to_date
RV_chk.RV32M.MUL_a	hold	open	open	up_to_date
RV_chk.RV32Zicsr.CSRx_a	fail (4)	open	open	up_to_date
RV_chk.xcpt.EBREAK_a	hold	open	open	up_to_date
RV_chk.xcpt.ECALL_a	hold	open	open	up_to_date
RV_chk.xcpt.INTR_Handle_a	vacuous	...eachable	open	up_to_date
RV_chk.xcpt.XCPT_IF_ID_a	fail (4)	open	open	up_to_date
RV_chk.xcpt.XCPT_WB_a	vacuous	...eachable	open	up_to_date
SVA Named Properties				
SVA Sequences				

RV32EMC_Zicsr
Captured in

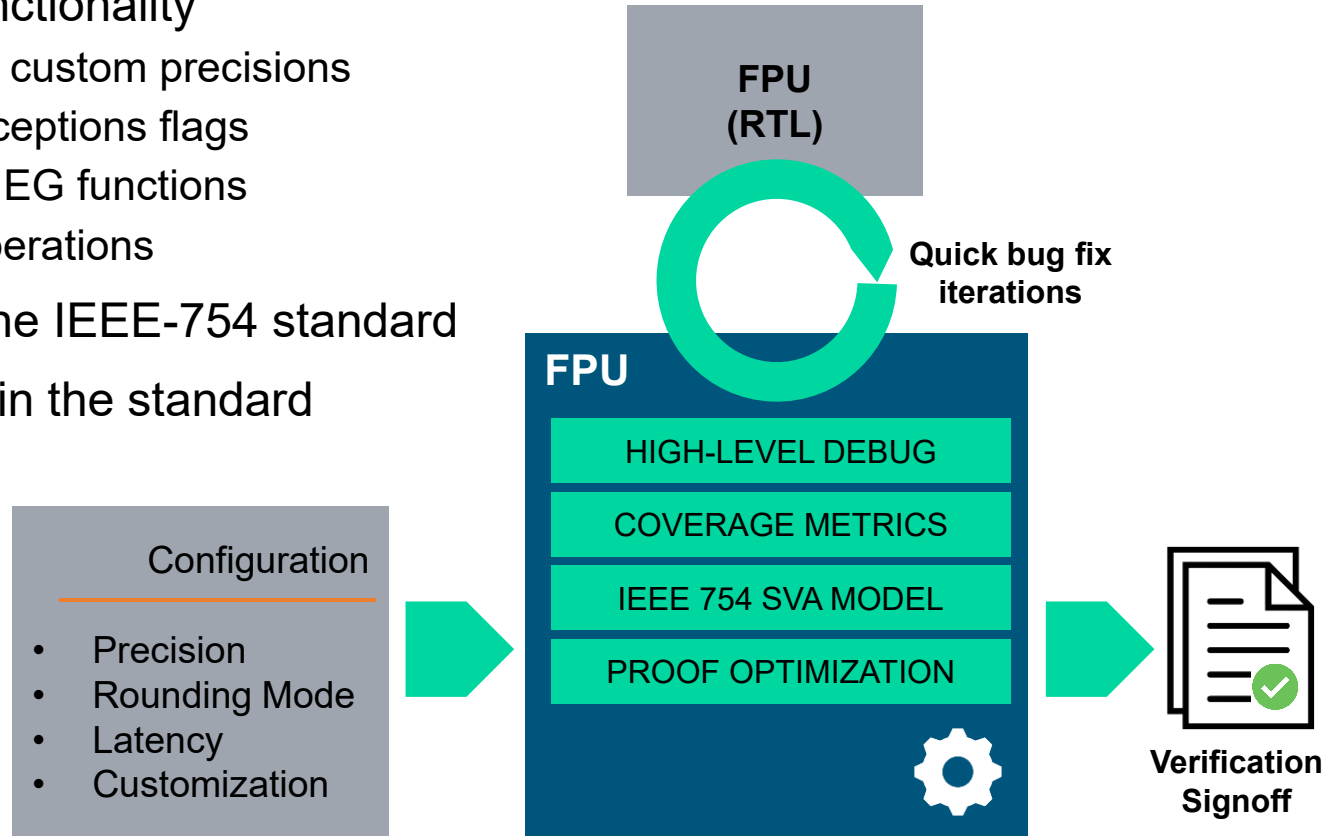
23

Assertions

Prove correctness with easy setup, without C++ model



- Supports broad array of FPU functionality
  - Half/single/double bfloat16 and custom precisions
  - All 5 rounding modes and 5 exceptions flags
  - ADD, SUB, MUL, FMA, ABS, NEG functions
  - Conversion and comparison operations
- Easy to model deviations from the IEEE-754 standard
- Parameters specify ambiguities in the standard
- RISC-V configuration*



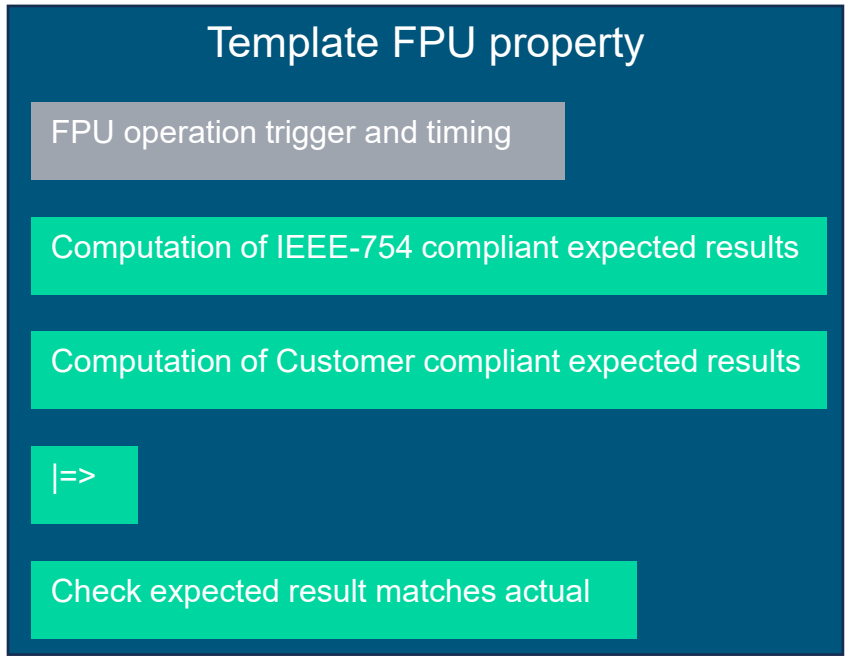
Questa FPU ensures FPU designs meet expectations

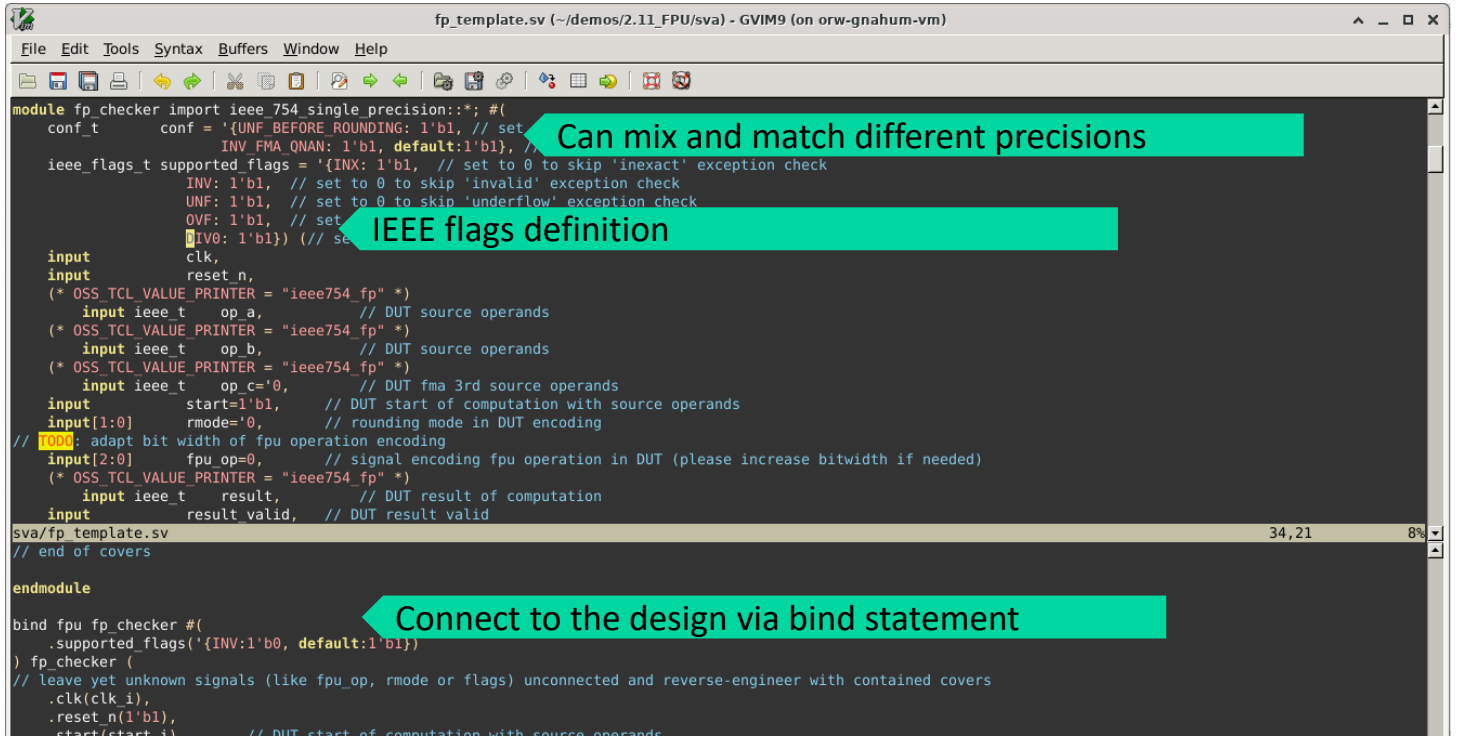
- Reduces verification from months down to days
  - Easy to set up and use: pre-packaged, debug features
  - Finds design bugs much faster than simulation
  - No need to develop reference models, testbench, or test cases
    - No manual effort to try to hit corner cases in simulation

- Provides a complete verification
  - Formal provides proofs that no amount of simulation can deliver
  - Complete external reference model increases verification confidence
    - Avoids same assumptions made in design and verification

Template example

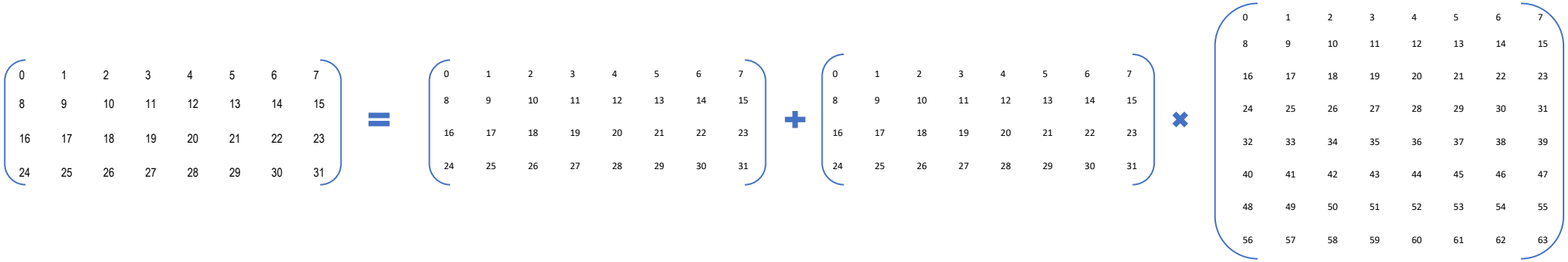
- All standard-derived information part of template
  - User provides mapping of FPU DUT to SVA module
  - Design specifics like encoding of operations, modes, and concrete protocol added by user
  - Extraction supported by numerous covers to extract timing and encoding





FP matrix multiply and accumulate is massive and complex

- 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 – Vector Fuse Multiply

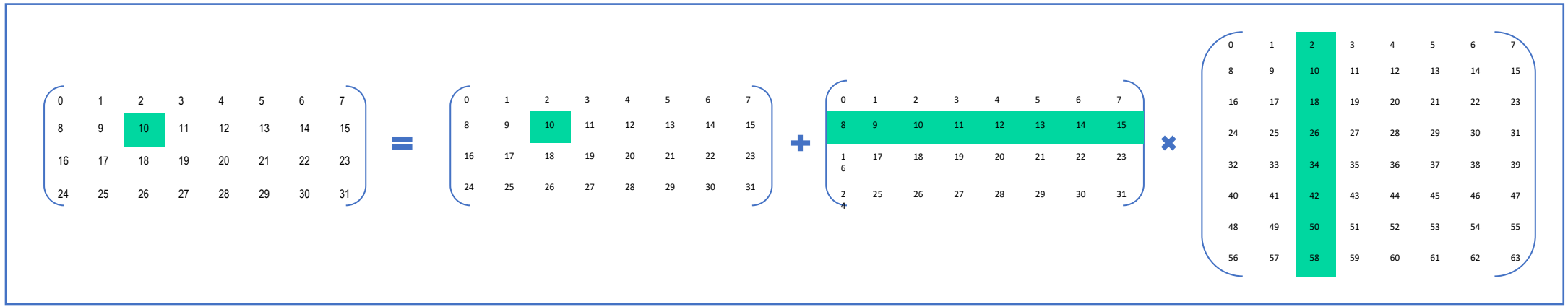
- Each element of the resulting matrix is calculated as follows
  - R[i] = ACC[i] + Row Matrix X * Column Matrix Y

- 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 + Xc0*Yr0 + Xc1*Yr1 + Xc2*Yr2 + Xc3*Yr3 + Xc4*Yr4 + Xc5*Yr5 + Xc6*Yr6 + Xc7*Yr7
  - Becomes a basic building block to check each result
  - Support different floating-point types

Matrix multiplication example

- 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) )
             ##10 operation = NOP
             |->
             ieee_check_result (.expected(expected), .actual (design_result[k])  );
      endproperty

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

Data provided by user

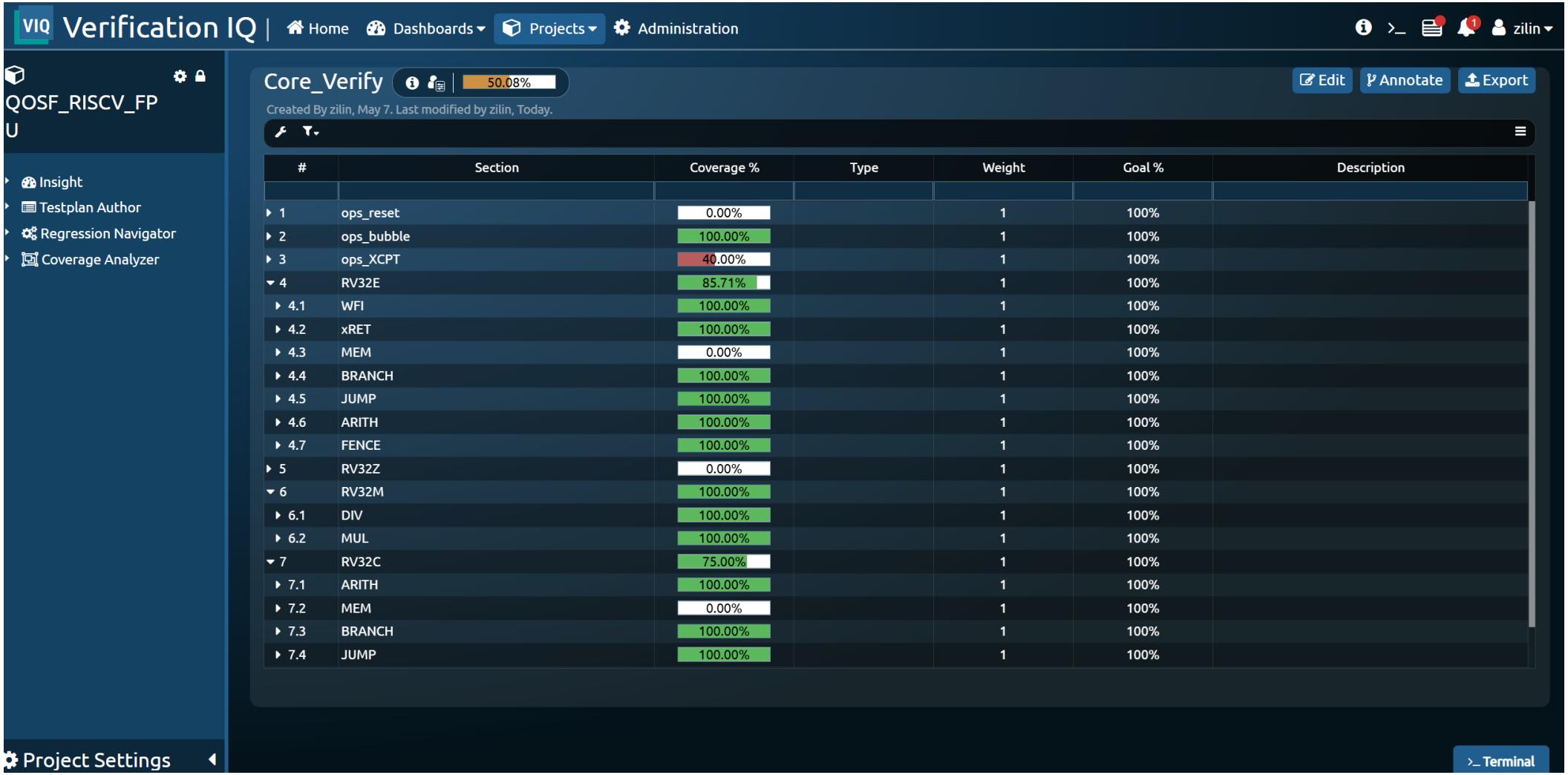
Prevented a bug escape!

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, e.g.:
  - 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 nonzero and all other zero's

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 proof	4h for full proof
VFMA	1 min	4min	Hold bounded	9h for full proof	Hold bounded

Update the Test Plan with the results from Processor and FPU runs



#	Section	Coverage %	Type	Weight	Goal %	Description
1	ops_reset	0.00%		1	100%	
2	ops_bubble	100.00%		1	100%	
3	ops_XCPT	40.00%		1	100%	
4	RV32E	85.71%		1	100%	
4.1	WFI	100.00%		1	100%	
4.2	xRET	100.00%		1	100%	
4.3	MEM	0.00%		1	100%	
4.4	BRANCH	100.00%		1	100%	
4.5	JUMP	100.00%		1	100%	
4.6	ARITH	100.00%		1	100%	
4.7	FENCE	100.00%		1	100%	
5	RV32Z	0.00%		1	100%	
6	RV32M	100.00%		1	100%	
6.1	DIV	100.00%		1	100%	
6.2	MUL	100.00%		1	100%	
7	RV32C	75.00%		1	100%	
7.1	ARITH	100.00%		1	100%	
7.2	MEM	0.00%		1	100%	
7.3	BRANCH	100.00%		1	100%	
7.4	JUMP	100.00%		1	100%	

Summary

Over 10x improvement

On verification setup and runtime

High Degree of Automation

Enables automated verification of core RTL

Exhaustive and Complete

Leaves no bugs and exposes vulnerabilities

Superior and Unique

Unrivalled expertise and leverage unique solutions