Taking Formal Verification to a Higher-Level with Jasper C

Cadence Jasper C team

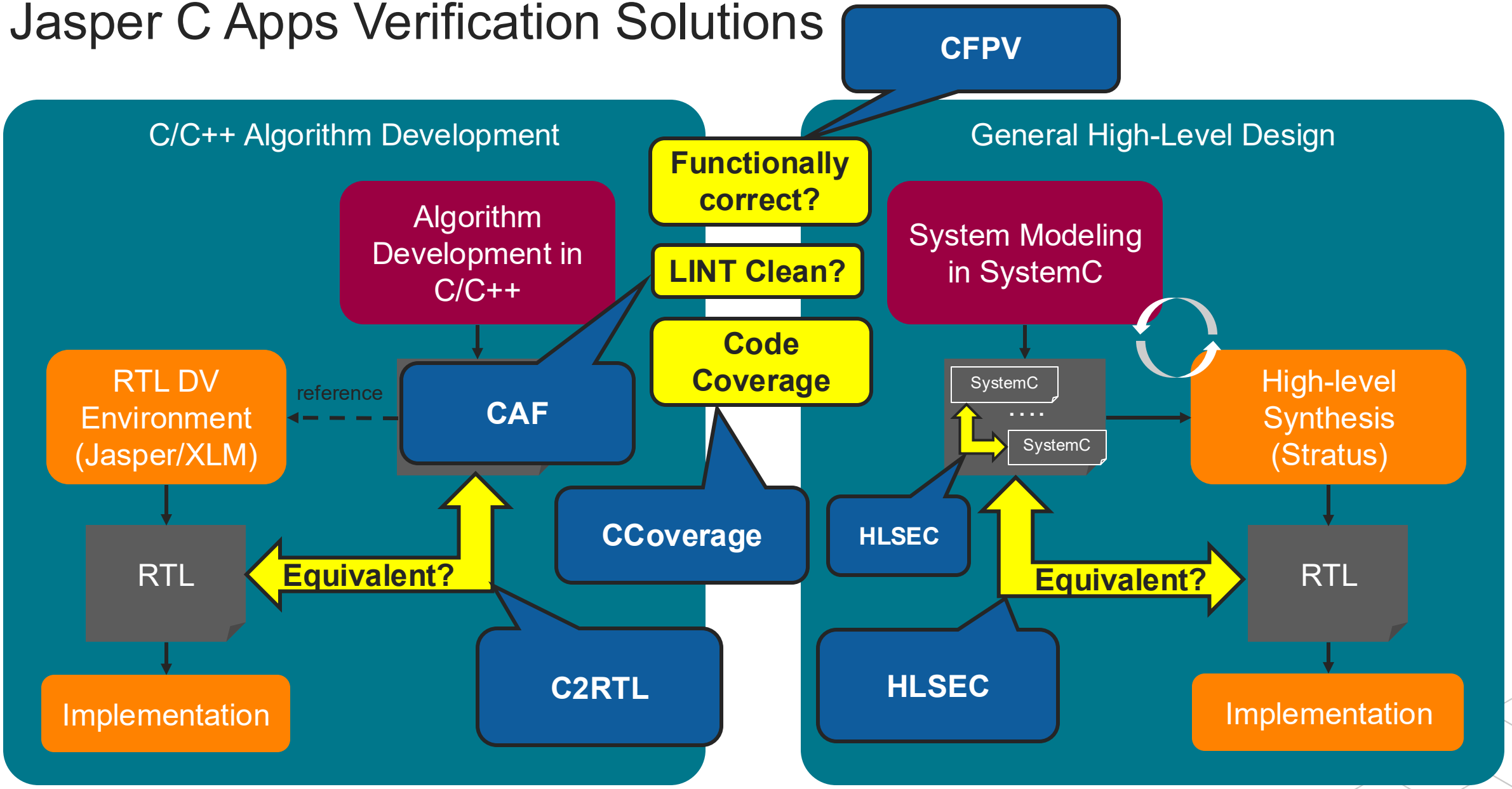
November 18, 2025

cadence®

Agenda

- Jasper C Apps Overview
- Key Components
- Jasper C Apps
  - C2RTL
  - HLSEC
  - CAF
  - CFPV + CCoverage

Jasper C Apps Verification Solutions



Summary

CCoverage
Stimuli coverage for C++ code

Early Access Q1'2026

C Auto Formal
Lint + Advanced formal checks for C/C++/SystemC models and HLS designs

Early Access Available

CFPV
Property checking for C/C++/SystemC Models

Early Access Available

C2RTL
Functional equivalence between C/C++ and RTL

Production

HLSEC
Verify correctness of High-Level Synthesis (SystemC vs RTL)

Restricted Release

Jasper® C Apps

C2RTL | CAF | CFPV | HLSEC

C++ Frontend(s)

C++ Synthesis Engine

Broadest Range of Formal Solvers

Leading Performance and Capacity

Jasper® Visualize™ Interactive Environment

Leading Ease-of-Use

Jasper Formal Verification Platform

This slide contains forward-looking statements regarding Cadence's business or products. Actual results may differ materially from the information presented here.

cadence

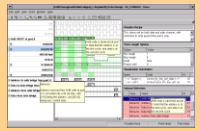
Key Components

C++ Frontend(s)

Broadest Range of Formal Solvers

Leading Performance and Capacity

Jasper® Visualize™ Interactive Environment



Leading Ease-of-Use

Jasper Formal Verification Platform

Jasper C/C++ Frontend

- Excellent C/C++ language constructs and library support
  - C99 and C++98, 03, 11, 14, 17 are all supported well, including template functions and classes
  - C/C++ standard library (including STL containers)
  - 400+ customer C++ models synthesized without any code change (mostly GPU and CPU)
- Dynamically sized structures are handled well:
  - Unbounded loops, recursive function calls
  - Dynamic memory allocation, Variable Length Arrays (VLAs)
- Autogenerated checks for C/C++ model
  - Helps catch bugs in the reference model itself
  - Examples: Unresolved function pointers, C++ undefined behaviors, etc.
- Support for C/C++ user asserts

Jasper SystemC Frontend

- C++ language constructs and SystemC support
  - C++ 98, 03, 11, 14 support
  - SystemC 2.3.3 support
  - 100+ customer HLS models synthesized without any code change
- Stratus™ High Level Synthesis (HLS) integration
  - Stratus HLS pragmas (HLS_INLINE_MODULE, METAPORT, etc.)
  - Stratus libraries (cynw_p2p, cynw_float, etc.)
- Autogenerated checks for SystemC model
  - Helps catch bugs in the reference model
  - Example: Arithmetic overflow, invalid pointer dereference, etc.
- Support for SystemC asserts

Setting Up Your Code

```
1  #ifdef JASPER_C
2    #include “jasperc.h"
3  #endif
4
5  int main() {
6
7    unsigned short int ina[32];
8    unsigned short int inb[32];
9    unsigned short int out[32];
10   bool op;
11
12 #ifdef JASPER_C
13   JASPER_INPUT_ARRAY(ina);
14   JASPER_INPUT_ARRAY(inb);
15   JASPER_INPUT(op);
16 #endif
17
18   for(int i = 0; i < 32; ++i) {
19     if(op)
20       out[i] = ina[i] + inb[i];
21     else
22       out[i] = ina[i] - inb[i];
23   }
24
25 #ifdef JASPER_C
26   JASPER_OUTPUT_ARRAY(out);
27 #endif
28
29 }
```

Jasper header file to be included

Optional:
Macro defined within C2RTL while parsing C++ code

- JASPER_C

Jasper methods to identify inputs:

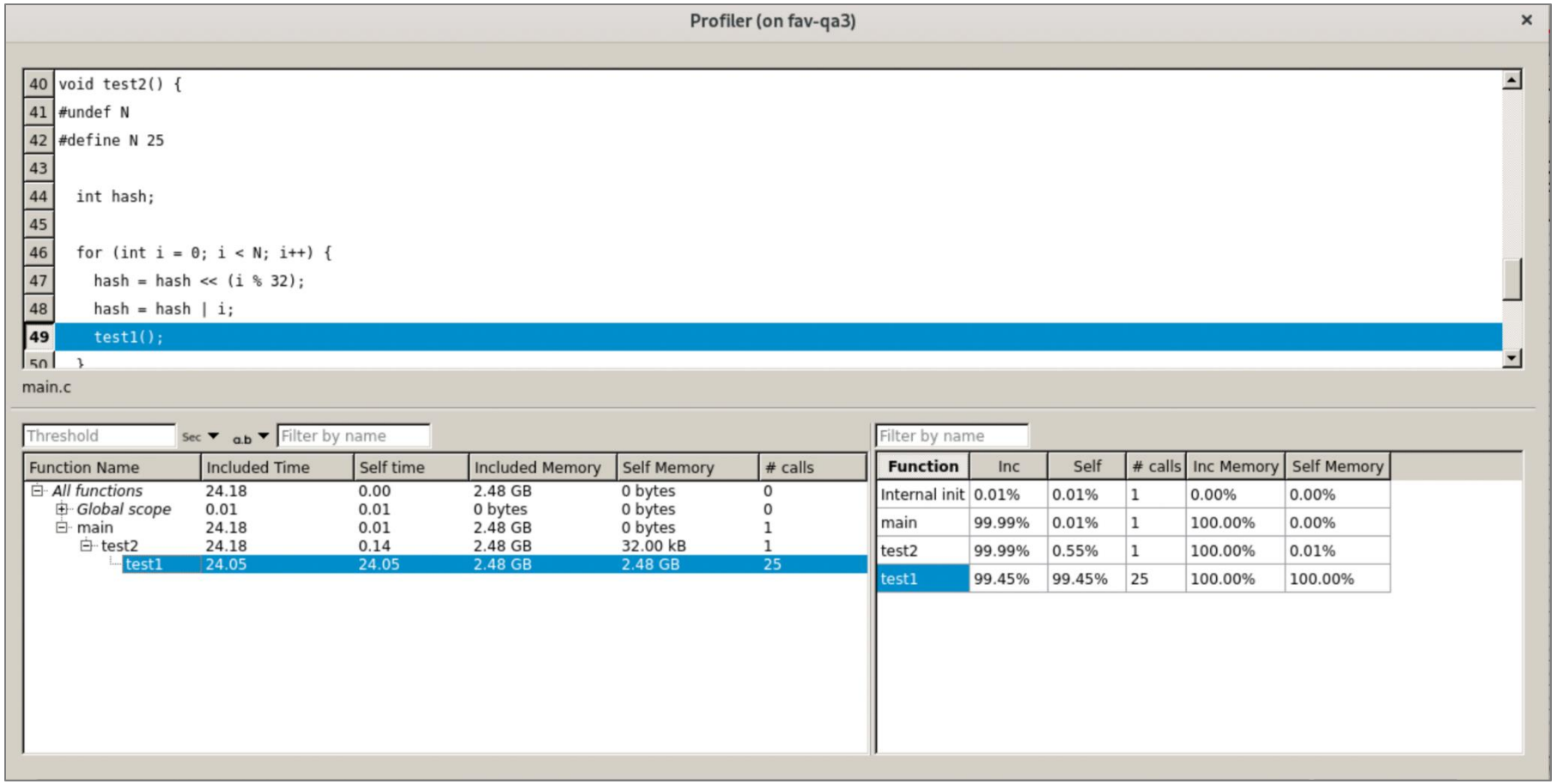
- JASPER_INPUT (var_name)
- JASPER_INPUT_ARRAY (var_name)
- JASPER_INPUT_LVAL (LHS_expression, var_name)

JG methods to identify outputs:

- JASPER_OUTPUT (name)
- JASPER_OUTPUT_ARRAY (name)
- JASPER_OUTPUT_RVAL(RHS_expression, var_name)

New Capabilities

- C++ compilation profiler time and memory



New Capabilities

- C++ compilation profiler time and memory
- Support arrays of structs or nested structs as I/O of the formal model
- Conditional interface macros

```
#include <jasperc.h>

struct A { int x; int y;};
Struct B { A a; }

int main()
{
  B b;
  JASPER_INPUT(b);

  A arr[10];
  JASPER_INPUT(arr);
  ...
  return 0;
}
```

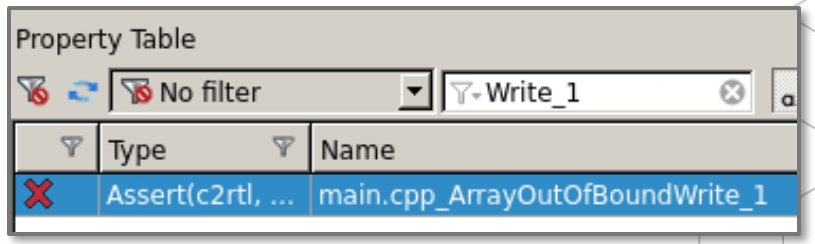
```
int main() {
  int x, y;
  JASPER_INPUT(x);
  if (x < 10) {
     y = x;
     JASPER_OUTPUT(y);
  }
}
```

```
Formal model behaves as follows:
y_jasper_valid = x < 10;
y = y_jasper_valid ? y_original : free;
```

New Capabilities

- C++ compilation profiler time and memory
- Support arrays of structs or nested structs as I/O of the formal model
- Conditional interface macros
- Support for additional math functions (asin*, sincos*)
  - set_cfe_compile_extended_math true/false (enable/disable support for difficult math.h fcts)
- Support for arithmetic datatypes (ac_int)
  - Recommendation to switch from SystemC to arithmetic datatypes (sc_int/sc_bigint → ac_int)
- Pedantic mode in Jasper® C compilation
- Sanity assertion for array out-of-bound write

```
int array[8];
int index = 8;
array[index] = 12;   // ArrayOutOfBoundWrite violation: index is too large
array[index-9] = 12; // ArrayOutOfBoundWrite violation: index is negative
array[index-1] = 12; // OK
```



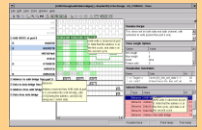
Key Components

C/C++/SC Frontend(s)

Broadest Range of Formal Solvers

Leading Performance and Capacity

Jasper® Visualize™ Interactive Environment

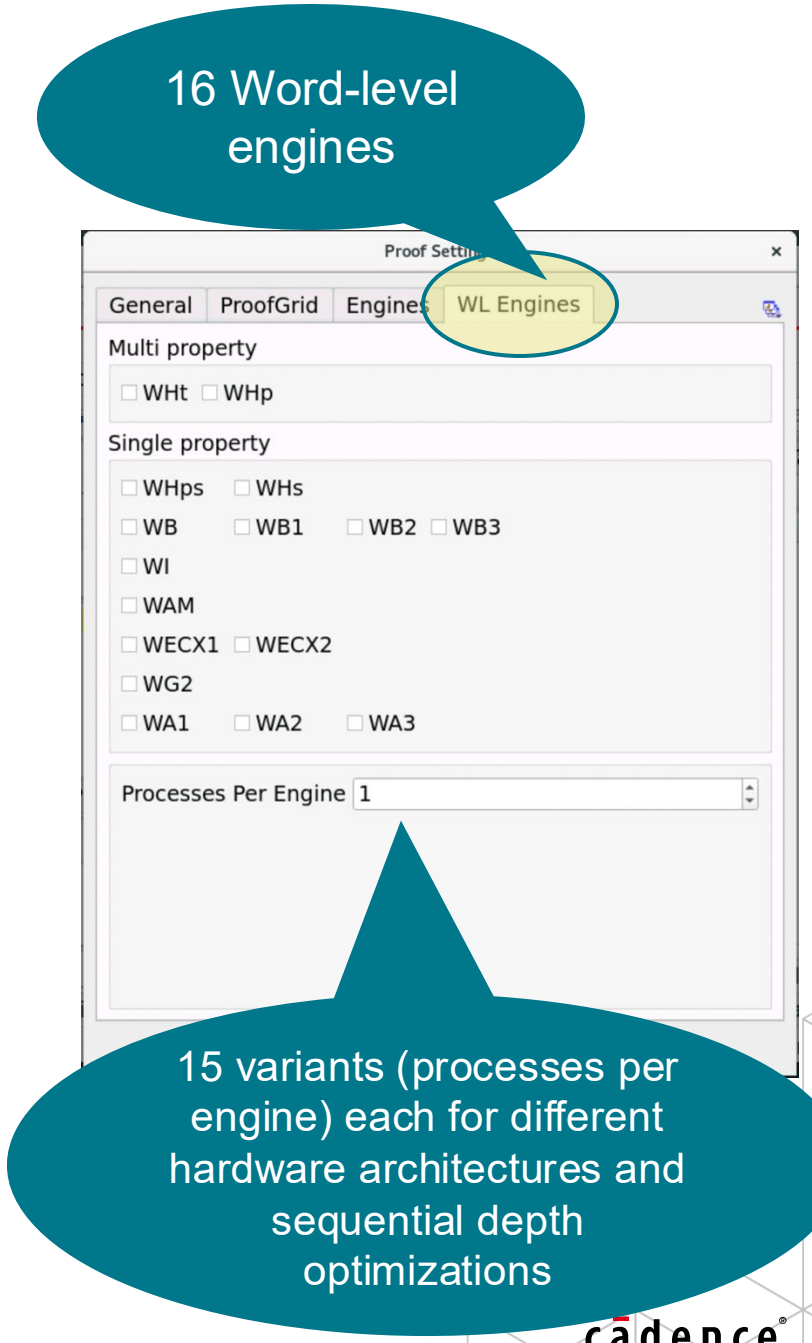


Leading Ease-of-Use

Jasper Formal Verification Platform

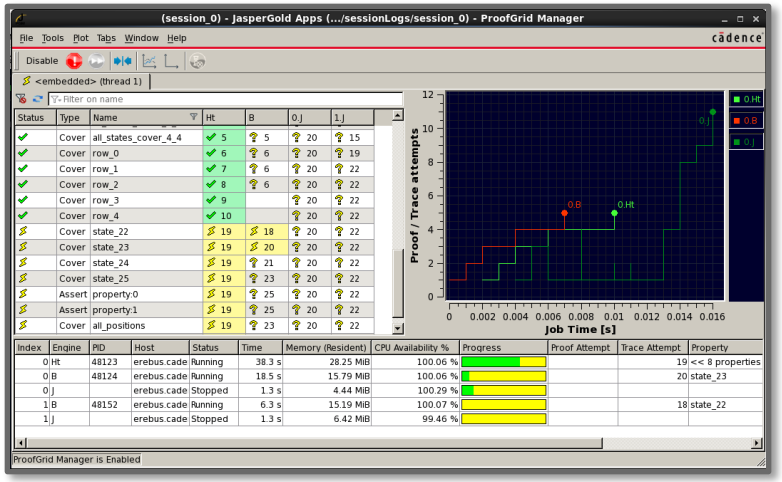
Datapath Proof Engines

- State-of-the-art datapath proof stack
  - Powerful bit-level and word-level solvers
  - Industry-leading powerful engines
- Datapath-specific optimizations:
  - Support for floating-point multiplication to minimize manual case splits
  - Dedicated handling of large integer multipliers implemented using smaller multipliers and adders
  - Word-level arithmetic abstractions, term-rewriting and normalisation, and reductions on the size of vectors
- Specialized bit-level multiplier handling
  - Special engines to verify bit-level implementations of array multipliers, radix2, radix4 booth multipliers, with variants like Baugh Wooley

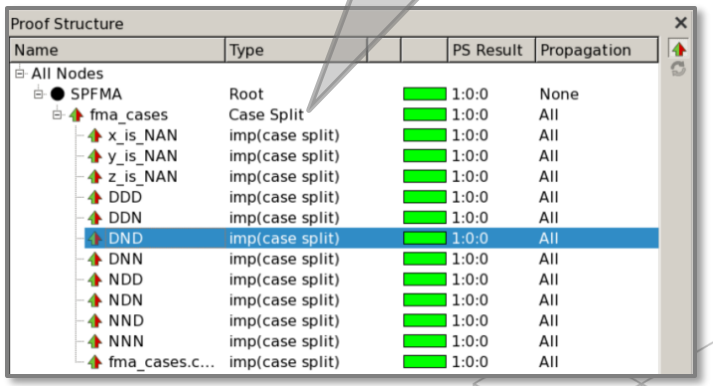


Datapath Proof

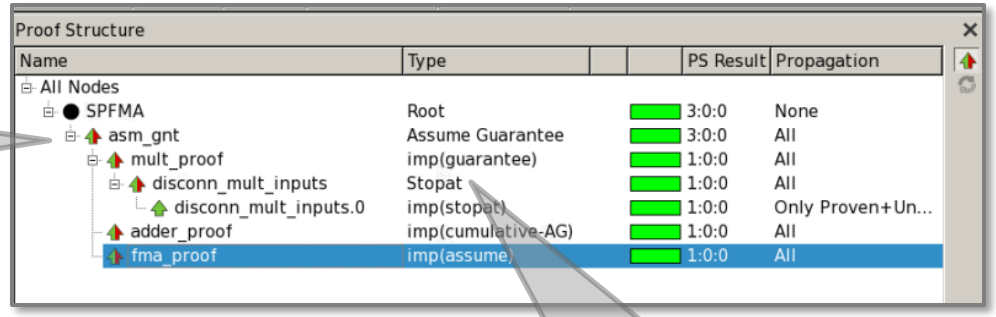
- Proofgrid support
  - Manages 1000+ parallel proof jobs on a server farm
- ML-based proof orchestration
  - Auto engine/solver selection, ML-based engine optimizations
- Prove cache and PPD
  - Use learnings from previous runs for engine choices
  - Save compute resources if design/env changes do not impact the assertion
- Powerful Jasper interactive proof cockpit through proof structure
  - Assume-guarantee, cutpoints, and intermediate helper lemmas
  - All environment modifications without the need to recompile



Case split operation



Multiple Levels of Assume-guarantee



Stopat Operation

Latest Improvements

- New engine WHs focusing on proof search
  - Use it if engines find a high bound but no proof
- Engine improvements focused on handling
  - Many different bit-level multiplier designs
  - More complex general designs, e.g., FMAs, dot-products, etc.
- Comparison of bit-level vs datapath engines on customer testcases:

	Details	Jasper Bit-Level Engines	Datapath Engines
Tcase - 1	Fixed-point multiply with variable decimal location	10 mins	20 seconds
...	...	...	...
Tcase-10	Dot product (8 products of fp16/bf16/fp32 being added together)	NA	~95 mins
Tcase-11	2048-point FFT	NA	~1 day (proof decomposed)
Tcase-12	Dot product (64 products of fp16/bf16 being added together)	NA	3.3 days

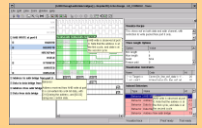
Key Components

C/C++/SC Frontend(s)

Broadest Range of Formal Solvers

Leading Performance and Capacity

Jasper® Visualize™ Interactive Environment

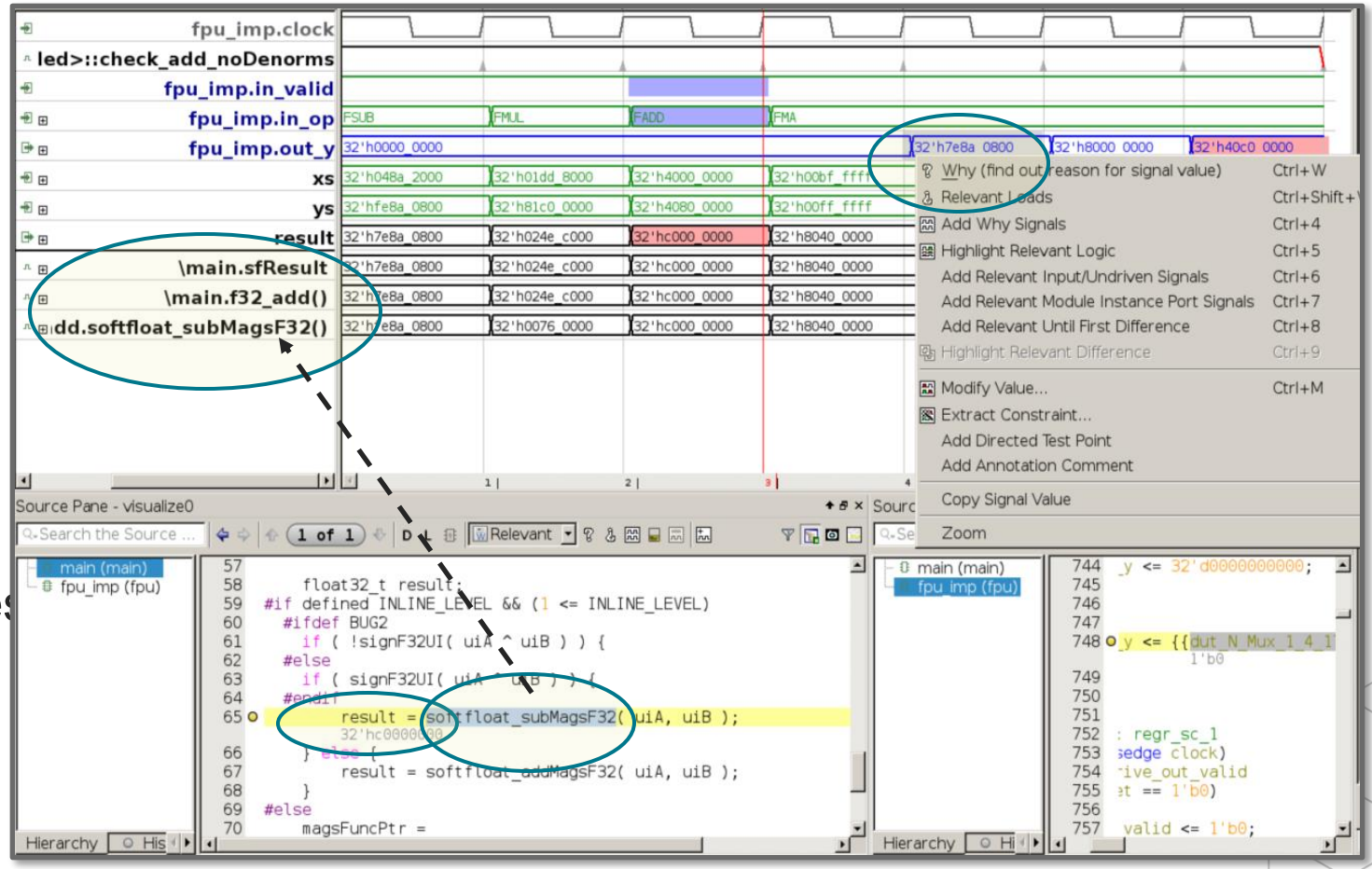


Leading Ease-of-Use

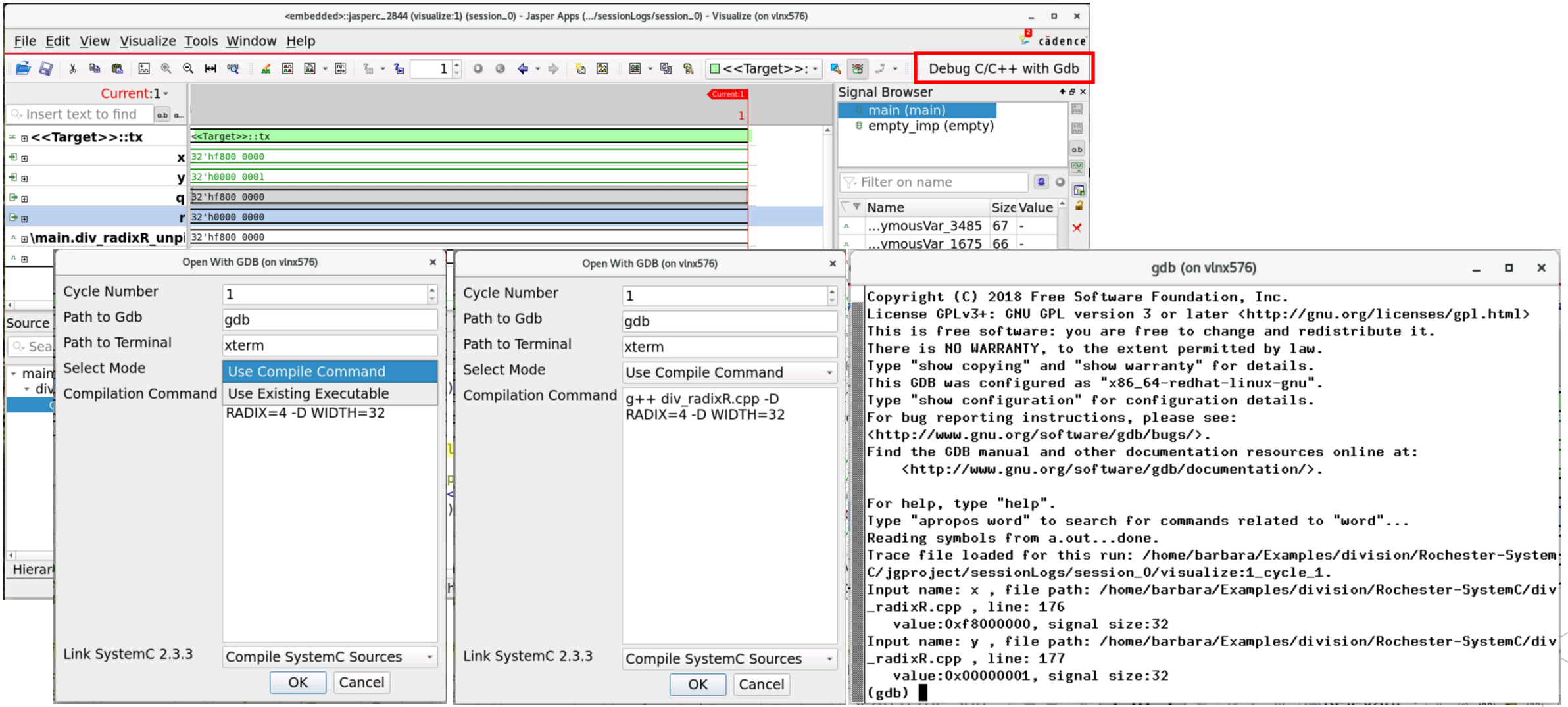
Jasper Formal Verification Platform

Jasper C Debug

- Dual debug for C/C++/SystemC vs RTL
- RTL-like waveform debug
  - Why tracing in C
  - Driver/load tracing
  - Value annotation in source
- Specialized debug
  - All variables/function-calls are uniquely identifiable and plottable
  - Full call stack in variable names
- Powerful Jasper® Visualize™ Interactive Debug Environment features integrated
  - What-if analysis
  - Quiet trace
  - Freeze and extend, etc.



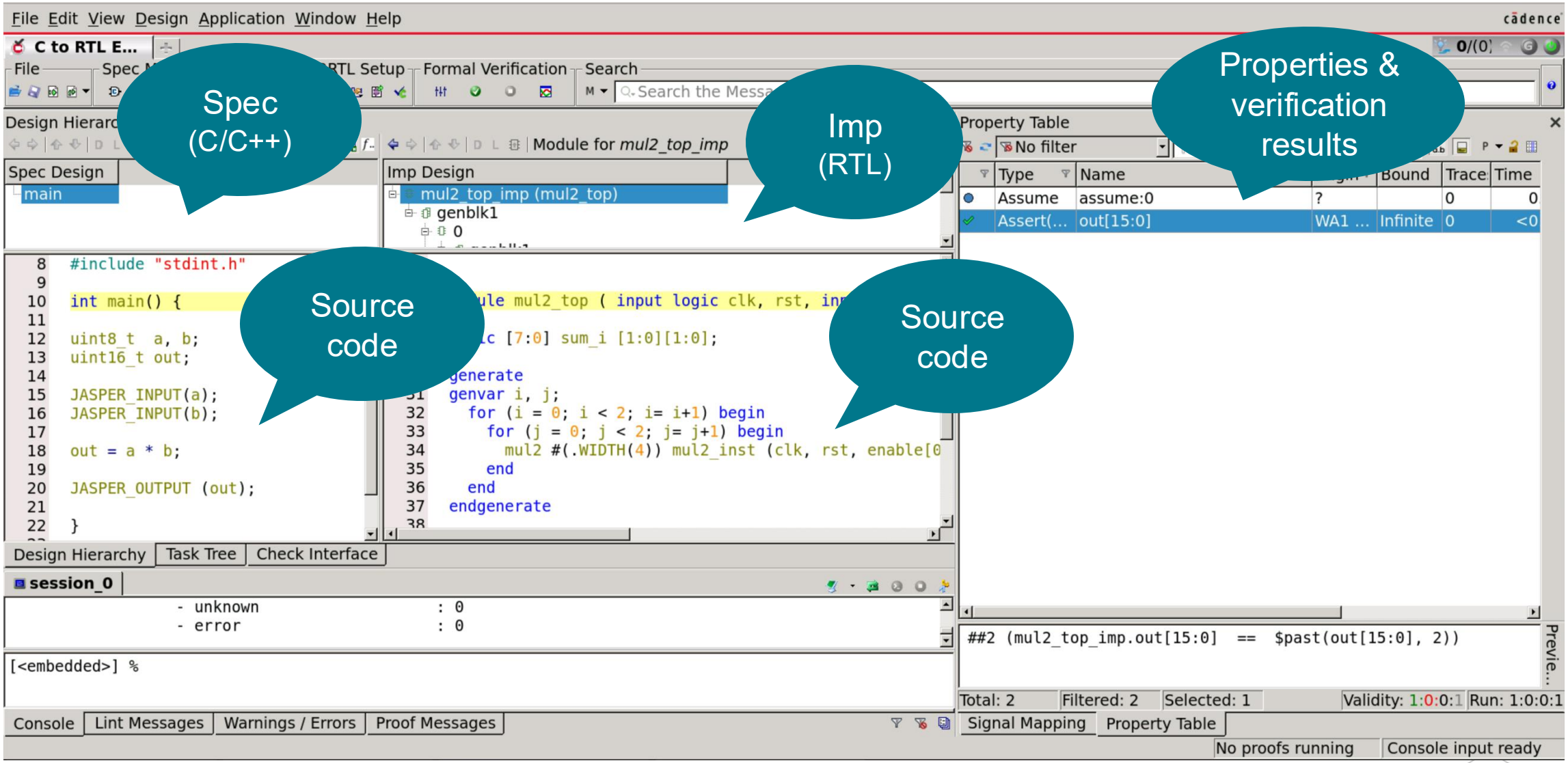
New Capabilities: GDB Debug



Jasper C2RTL

Datapath Verification App

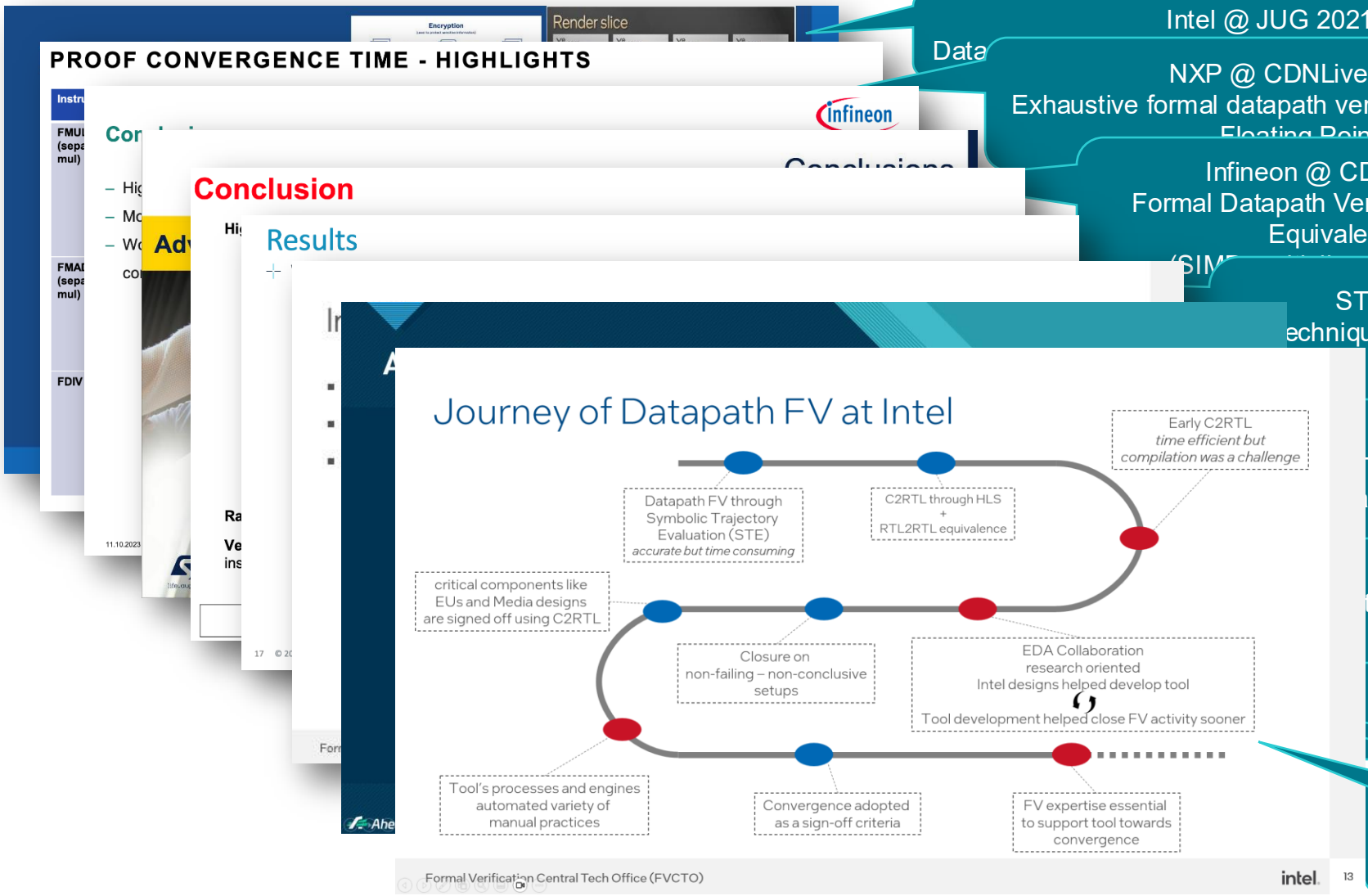
C2RTL App GUI



Key Features

- Integration in Jasper® Platform
  - Same setup, same look-and-feel, easy to get started
- Best-in-class bit-level and word-level solvers and datapath-specific abstractions
  - Reduces manual work (like proof decomposition)
- Broad C++ language support with C/C++ standard library support, including STLs
  - A large class of C programs is handled without code changes
- Verification of control and data transformations simultaneously
  - No need to separate datapath (reduces required expert knowledge)

Jasper C2RTL: Essential for Datapath Verification



Intel @ JUG 2021

NXP @ CDNLive India 2022
Exhaustive formal datapath verification of RISCV-based

Infineon @ CDNLive EMEA 2023
Formal Datapath Verificaiton using High-Level Equivalence Checking

ST @ CDNLive EMEA 2023
echniques to check equivalence between
C model from Matlab vs RTL for

TI @ CDNLive India 2023
L gives ~40x productivity & exhaustive
tion on Floating Point and Trigonometric
operations

ARM @ JUG 2023
tional Verification of prediction algorithms
with C vs RTL

Intel @ JUG India 2024
Software formal for Firmware

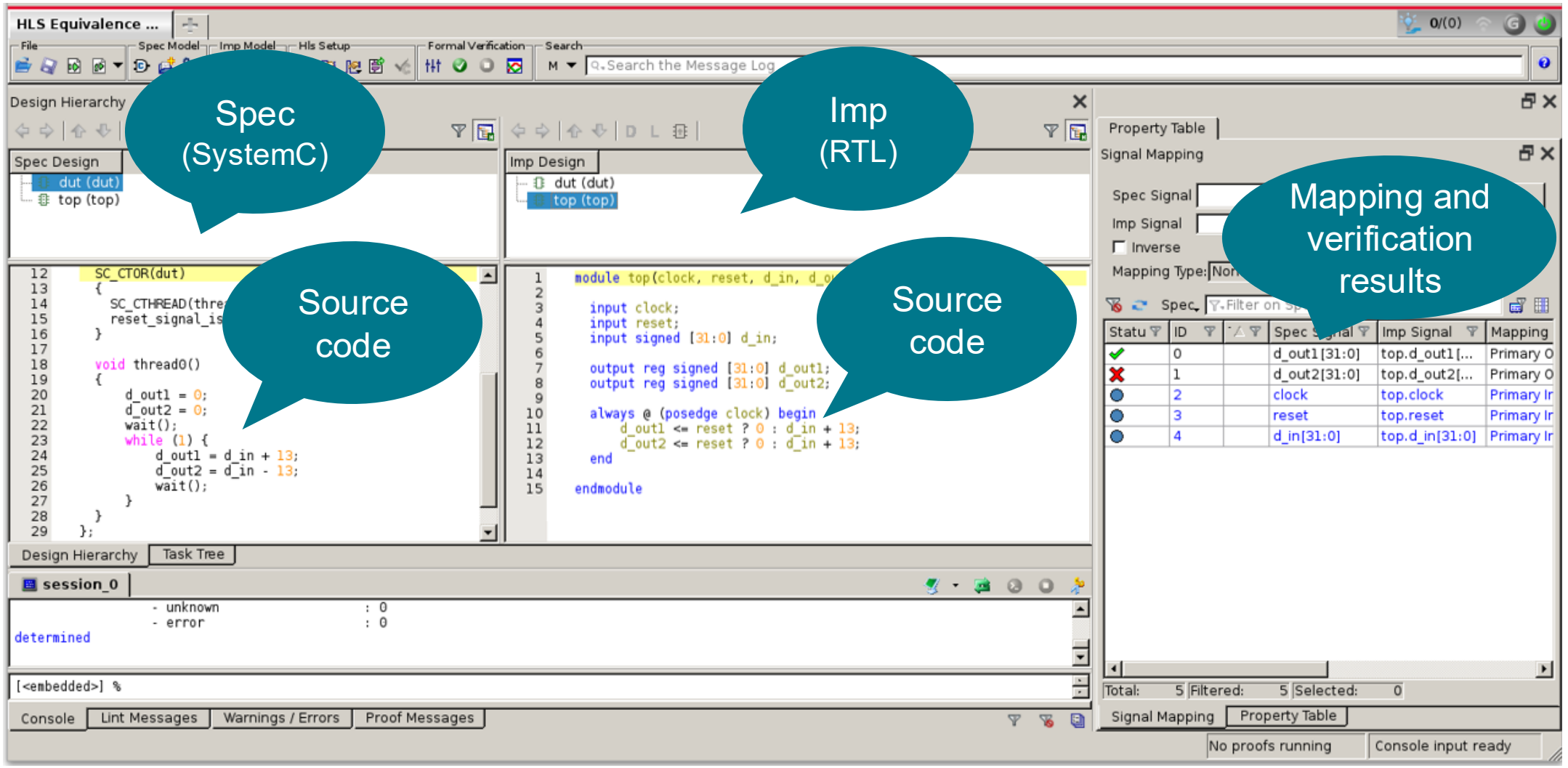
AheadComputing @ JUG 2025
RISC-V
FV Shift Left of Bug Finding

Intel @ JUG 2025
Symbiotic relation of tool and design:
Advancing C2RTL methodology

Jasper HLSEC

Equivalence check between SystemC design and HLS generated RTL

HLSEC App GUI



HLSEC App Status

- Working with selected customers
  - Jasper® C HLSEC App flow is working well in all engagements
  - Excellent proof capacity and performance already demonstrated on multiple designs
- Frontend:
  - No gaps in ongoing engagements
  - Goal: Consume all SystemC code that Stratus can compile
  - R&D is constantly working on closing gaps against the Stratus™ High-Level Synthesis (HLS) frontend
- Proof strategies
  - Tuning of existing Jasper proof strategies (from SEC and C2RTL datapath) already showing excellent results on cycle-accurate problems as well as non-cycle-accurate ones
  - Work is ongoing for enhancing proof capacity using hints on internal equivalence from Stratus HLS flow
- Results/Benchmark:
  - Besides testcases from customers, we are working on ~7000+ testcases from Stratus HLS regressions to identify and close the frontend gaps.
- Debug
  - Native debug through SystemC code in Jasper® Visualize™ Interactive Debug Environment is working

Jasper CAF

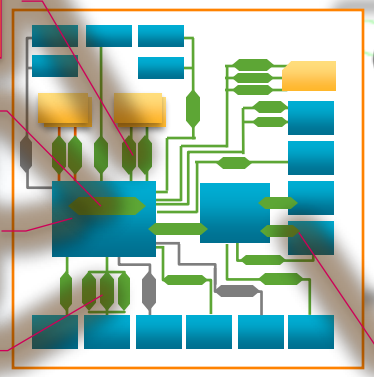
Auto Formal for C/C++/SystemC designs

Jasper CAF Overview

Automatic Formal Checks

- Conversion Precision loss
- Arithmetic overflow
- Out of bounds array indexing
- Pointer Overflow

C/C++ or SystemC Model

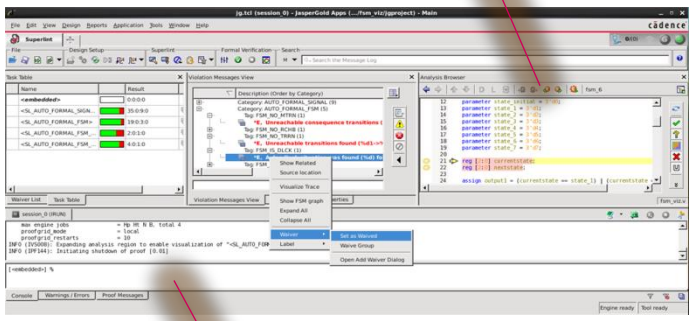


- Reachability
- FSM Livelock /deadlock
- Bus contention

Enabled by true formal technology

- Best-in-class debug
- Low-noise violation and waiver handling

Jasper® Visualize™ Interactive Debug Environment

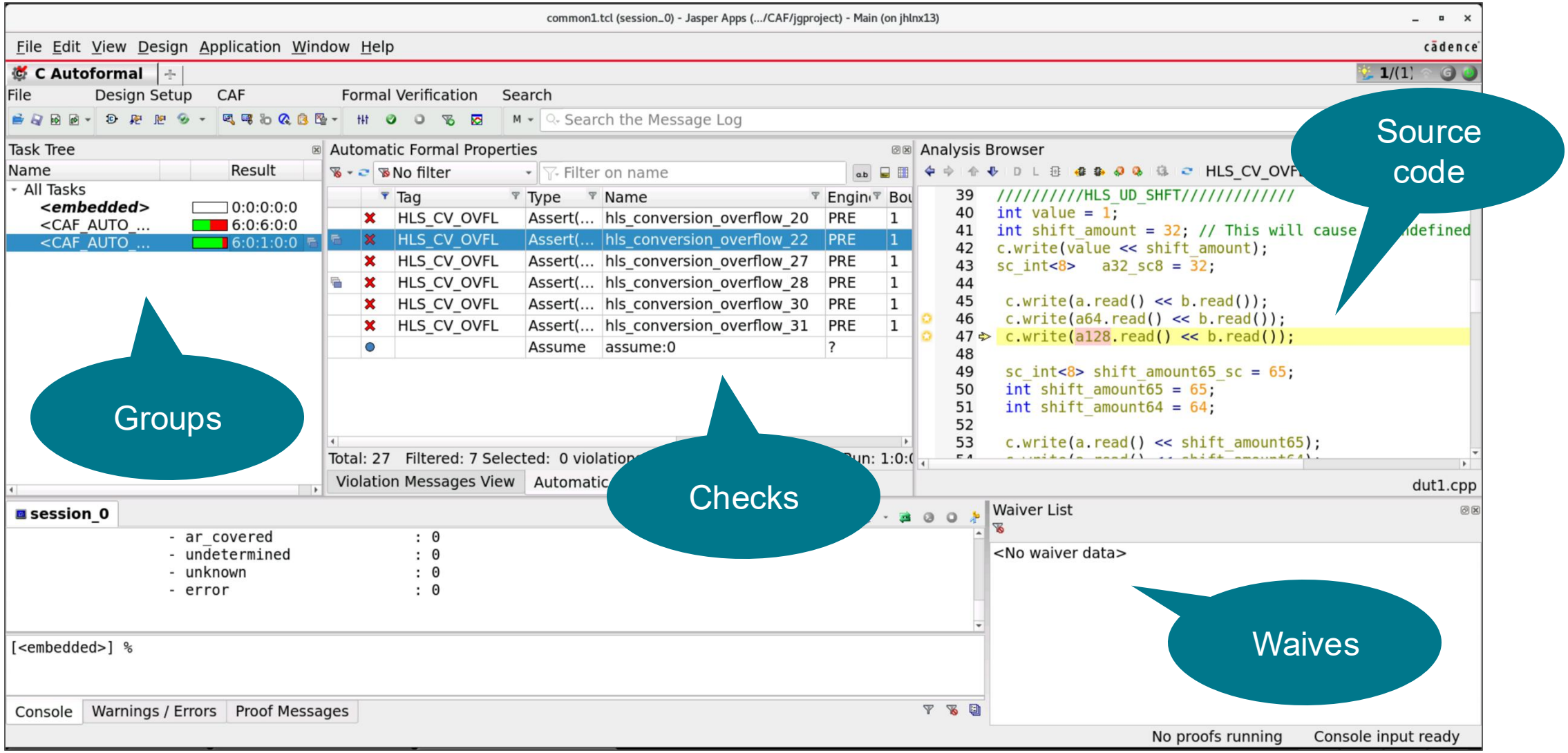


Auto functional checks, violation debug and waiver handling based on best-in-class formal analysis

Key Features

- Brings RTL-level auto formal checks to high-level design domain
  - Left shift in verification productivity
- Comprehensive SystemC model signoff
  - Signoff against a set of static lint and auto-formal checks
  - All checks are ‘automatic’ and the user just inputs SystemC model + constraints
- Features borrowed from industry-leading Jasper® Superlint App
  - Violation view (for non-formal users)
  - Formal property view (for formal-savvy users)
  - Persistent and highly portable waiver mechanism
  - Auto grouping for efficient analysis
  - Observability enabled debug

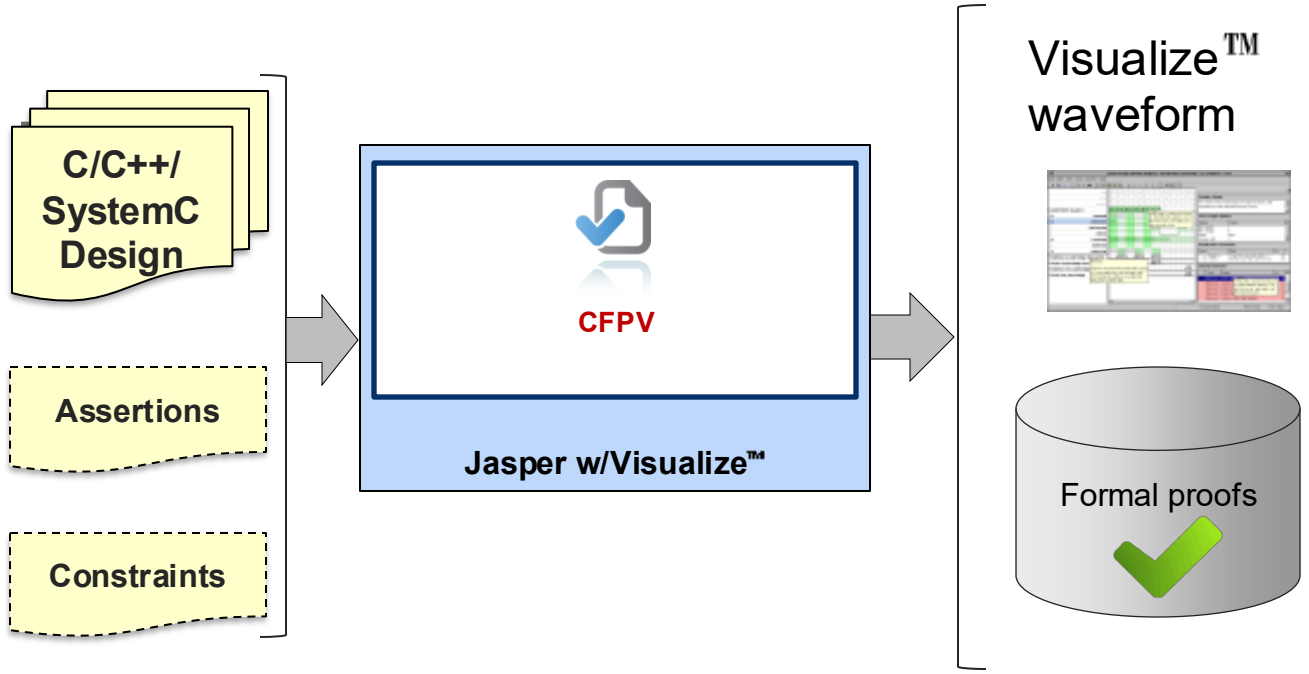
CAF App GUI



Jasper CFPV and CCoverage

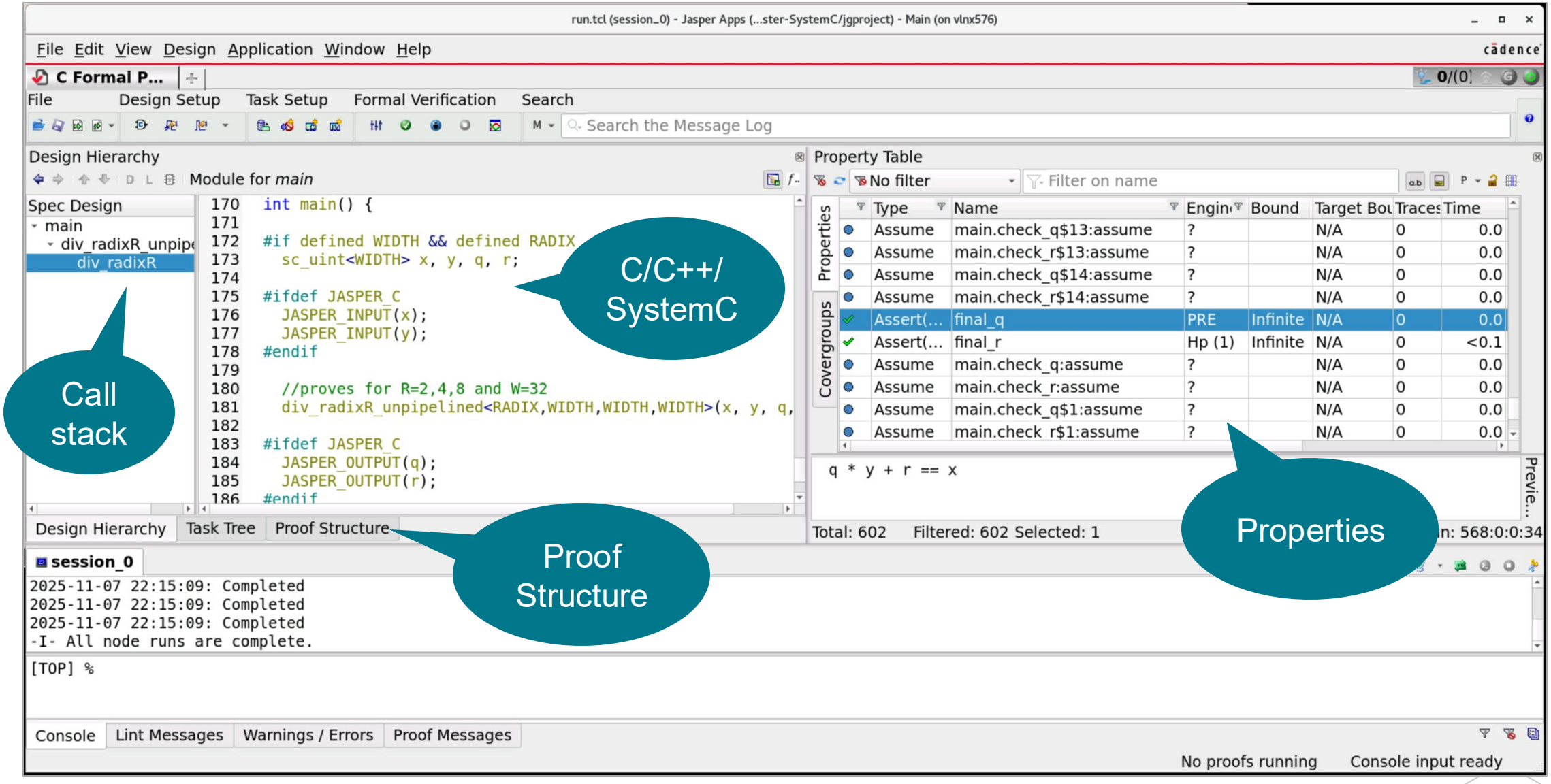
Formal property verification for C/C++/SystemC designs

CFPV App Flow

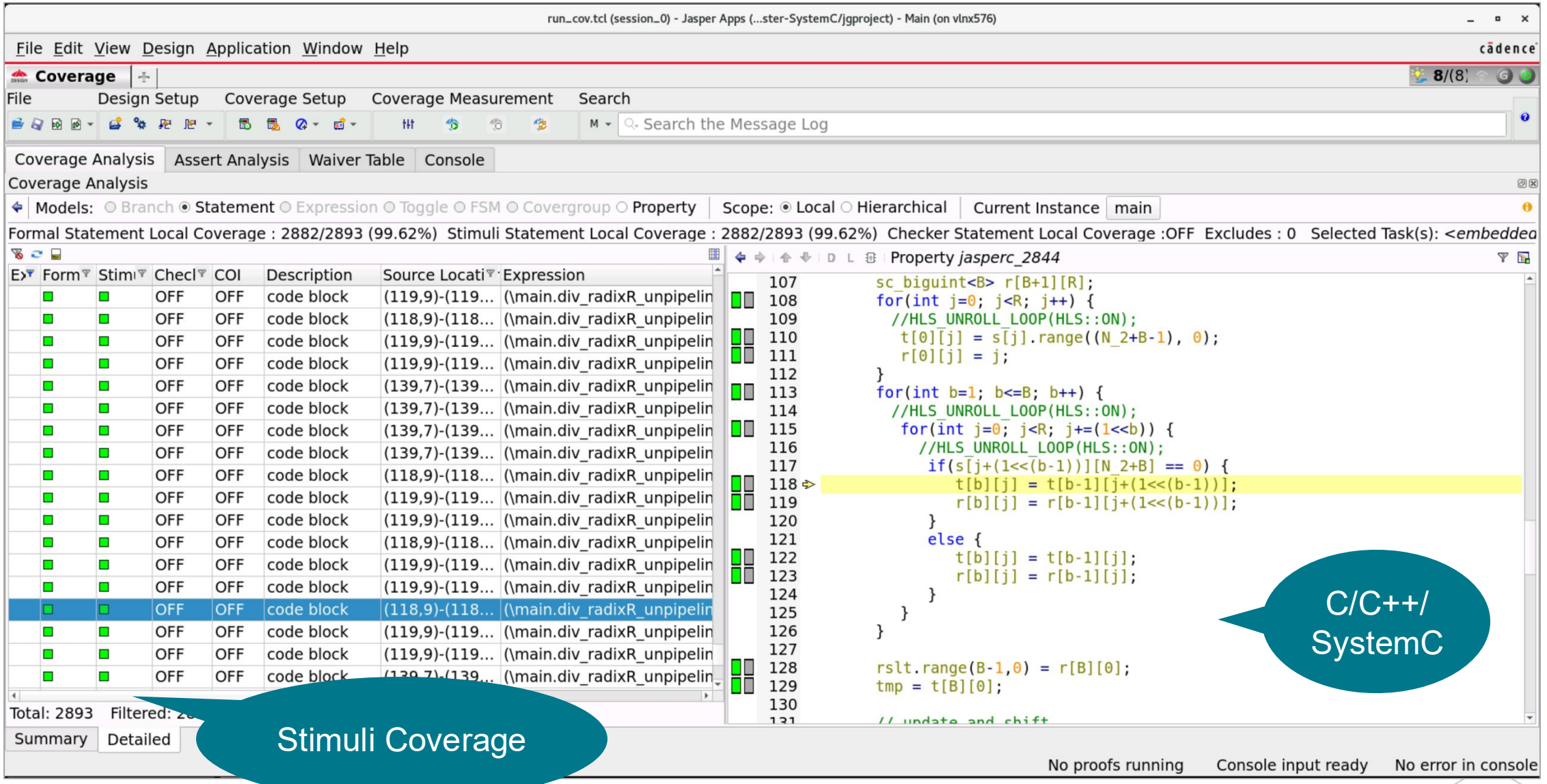


- CFPV App for **exploration**:
  - Jasper® Visualize™ Interactive Debug Environment is used without properties to exercise interesting design behavior
- CFPV App for **verification**:
  - Used as a “classic formal” tool, with assertions and constraints written in SVA
  - Metric-driven sign-off methodology (based on Coverage) on the roadmap for next year.

CFPV App GUI



CCoverage



Summary

CCoverage
Stimuli coverage for C++ code

Early Access Q1'2026

C Auto Formal
Lint + Advanced formal checks for C/C++/SystemC models and HLS designs

Early Access Available

CFPV
Property checking for C/C++/SystemC Models

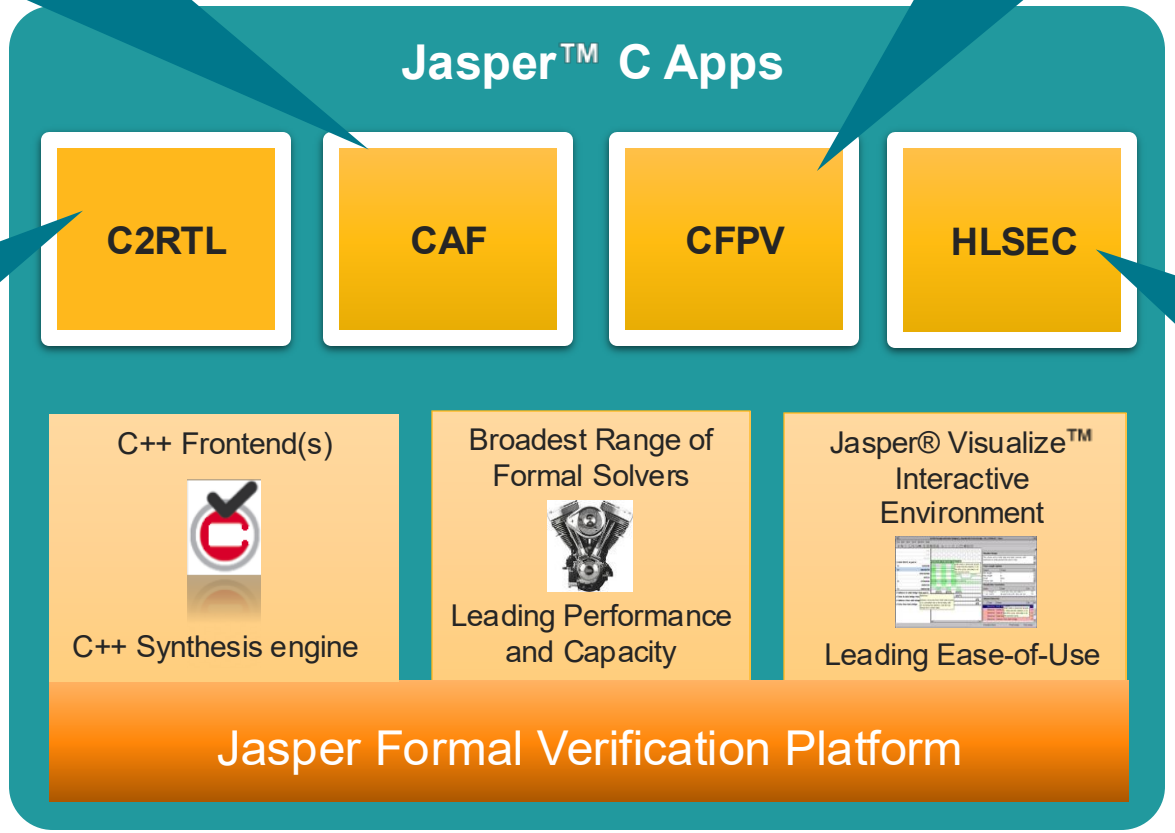
Early Access Available

C2RTL
Functional equivalence between C/C++ and RTL

Production

HLSEC
Verify correctness of High-Level Synthesis (SystemC vs RTL)

Restricted Release



This slide contains forward-looking statements regarding Cadence's business or products. Actual results may differ materially from the information presented here.

cadence