


# Taking Formal Verification to a Higher-Level with Jasper C

Cadence Jasper C team

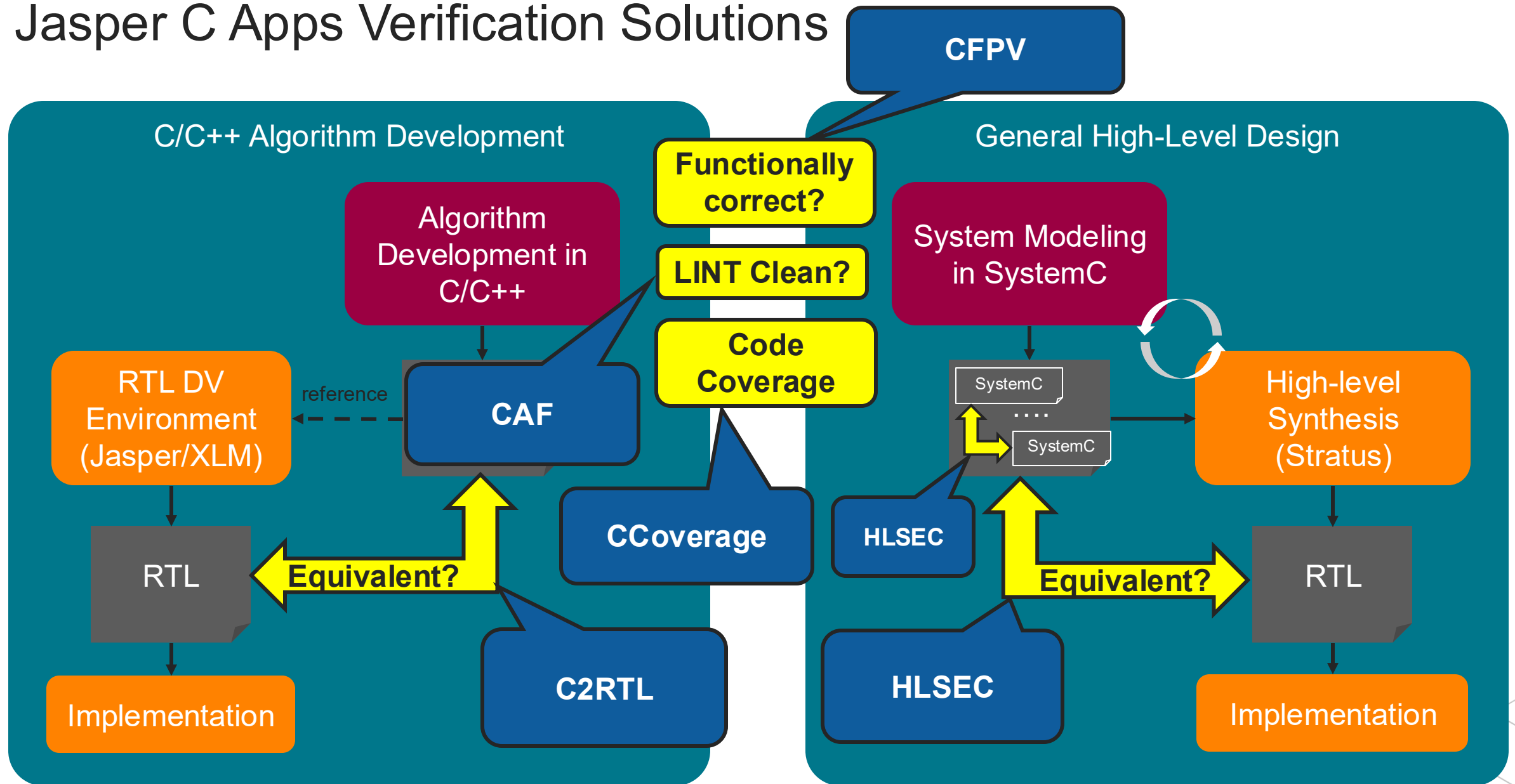
November 18, 2025

 cadence<sup>®</sup>

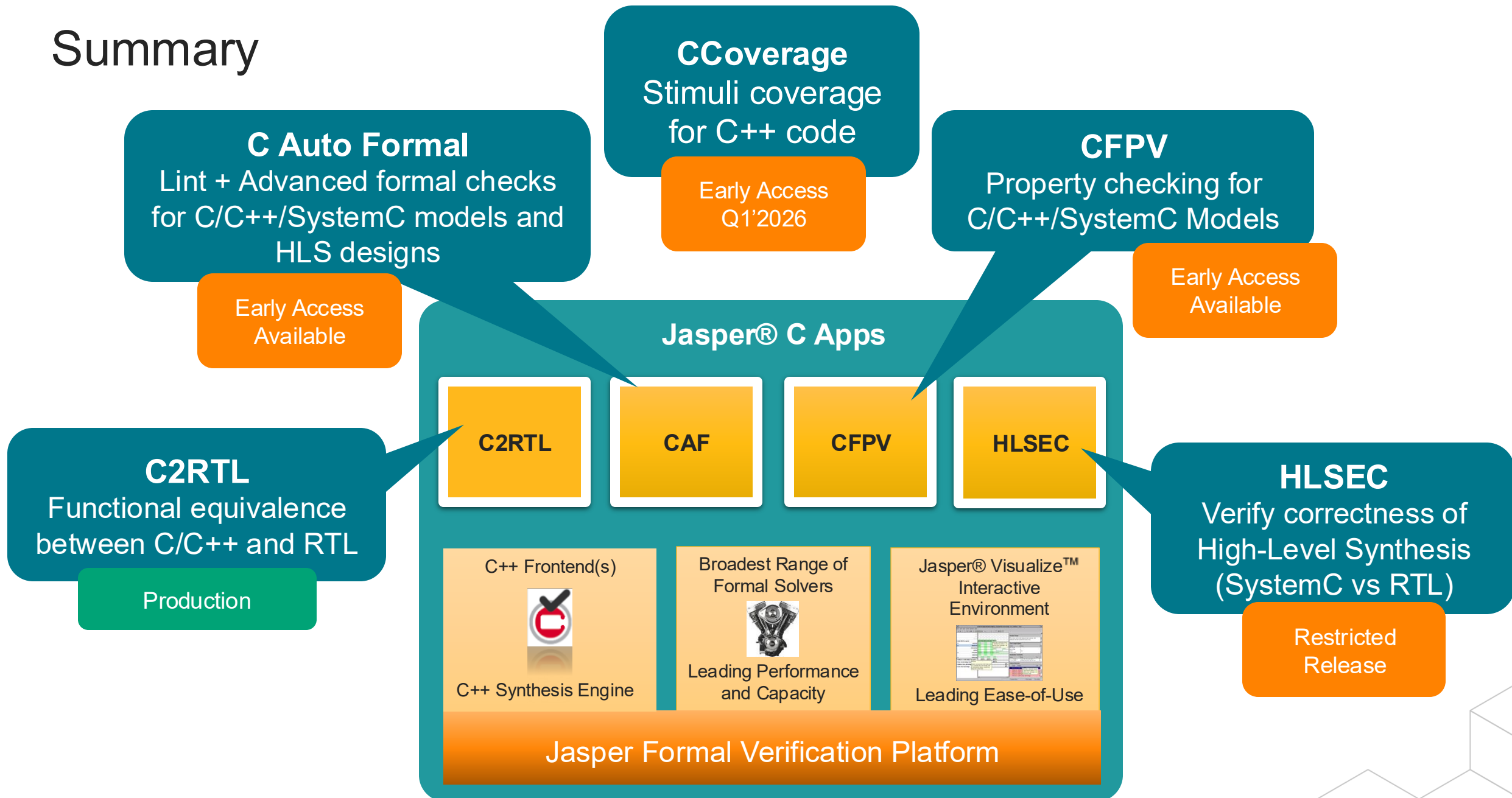
# Agenda

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

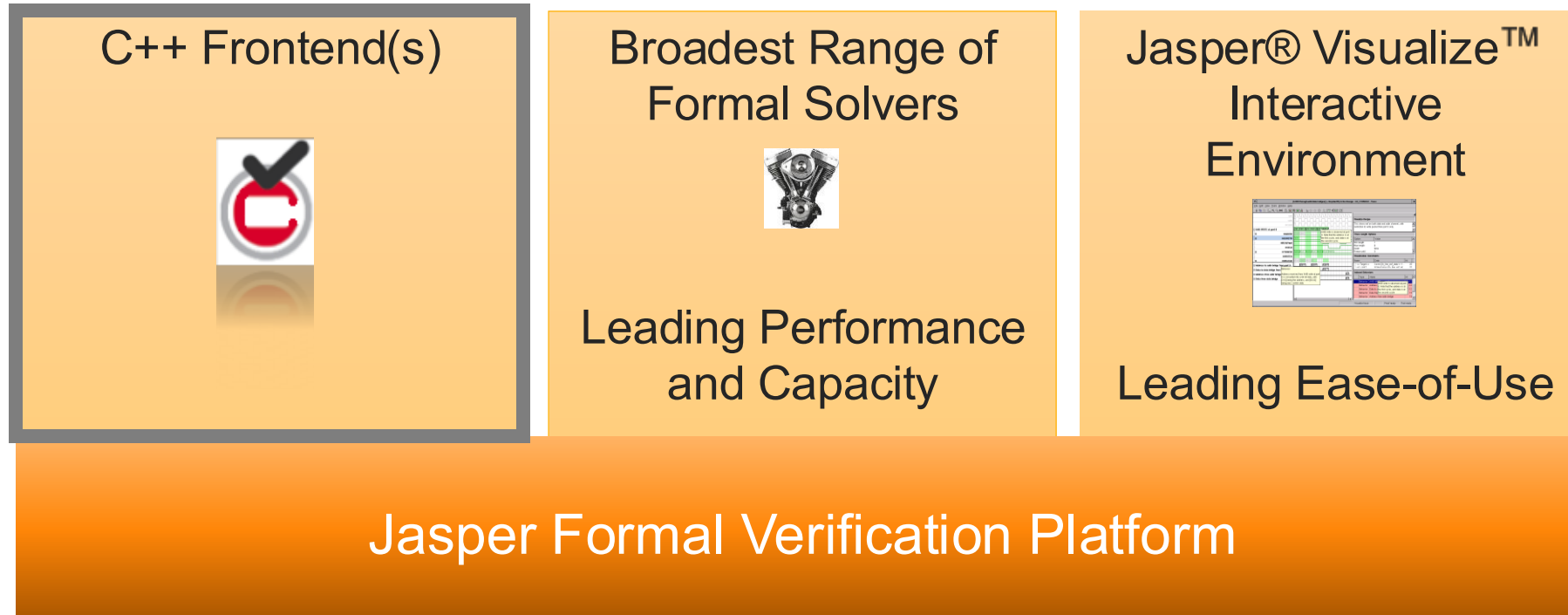
# Jasper C Apps Verification Solutions



# Summary



# Key Components



# 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

Profiler (on fav-qa3)

```
40 void test2() {
41 #undef N
42 #define N 25
43
44 int hash;
45
46 for (int i = 0; i < N; i++) {
47     hash = hash << (i % 32);
48     hash = hash | i;
49     test1();
50 }
```

main.c

Threshold: Sec ▾ a.b ▾ Filter by name

Function Name	Included Time	Self time	Included Memory	Self Memory	# calls
All functions	24.18	0.00	2.48 GB	0 bytes	0
Global scope	0.01	0.01	0 bytes	0 bytes	0
main	24.18	0.01	2.48 GB	0 bytes	1
test2	24.18	0.14	2.48 GB	32.00 kB	1
test1	24.05	24.05	2.48 GB	2.48 GB	25

Filter by name

Function	Inc	Self	# calls	Inc Memory	Self Memory
Internal init	0.01%	0.01%	1	0.00%	0.00%
main	99.99%	0.01%	1	100.00%	0.00%
test2	99.99%	0.55%	1	100.00%	0.01%
test1	99.45%	99.45%	25	100.00%	100.00%

# 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
```

Property Table		
		No filter
	Type	Name
	Assert(c2rtl, ...	main.cpp_ArrayOutOfBoundWrite_1

# Key Components

C/C++/SC Frontend(s)

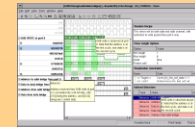


Broadest Range of  
Formal Solvers



Leading Performance  
and Capacity

Jasper® Visualize™  
Interactive  
Environment

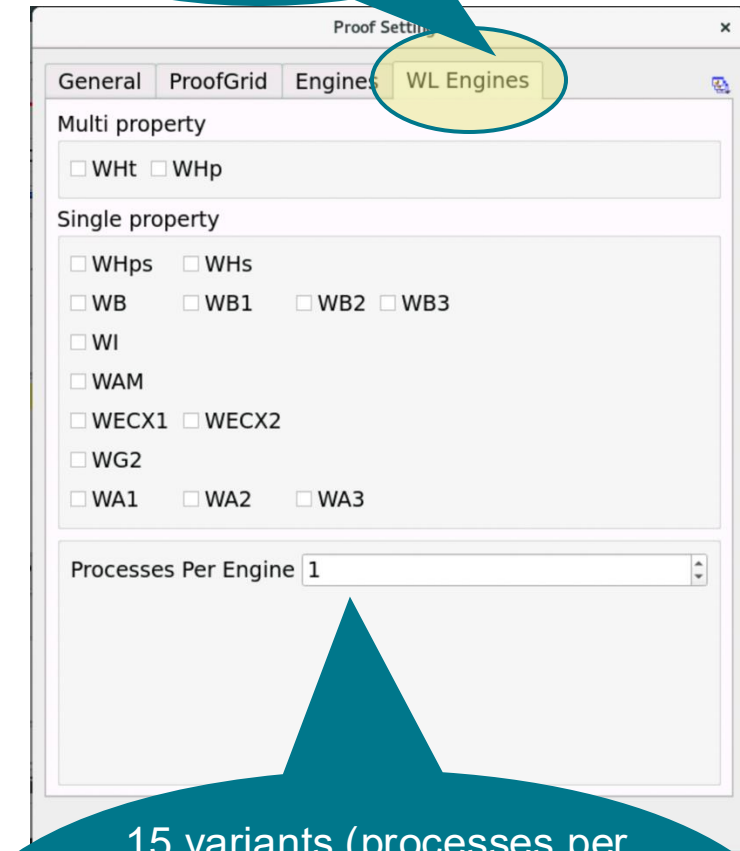


Leading Ease-of-Use

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

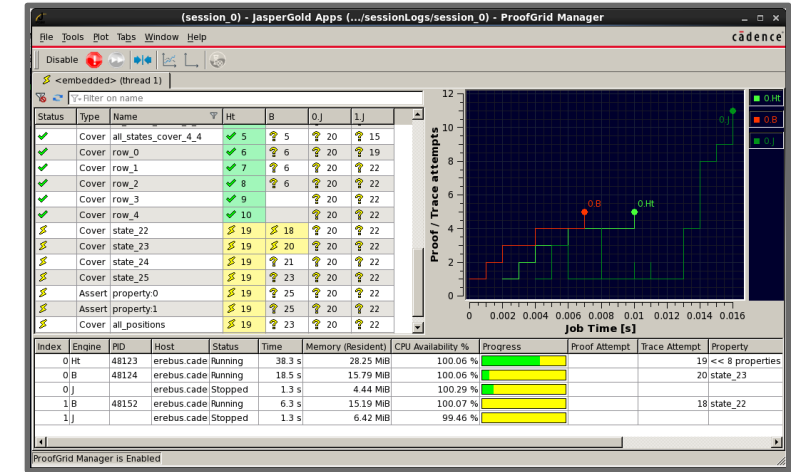


16 Word-level engines

15 variants (processes per engine) each for different hardware architectures and sequential depth optimizations

# 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



Multiple Levels of Assume-guarantee

Name	Type	PS Result	Propagation
All Nodes			
SPFMA	Root	3:0:0	None
asm_gnt	Assume Guarantee	3:0:0	All
mult_proof	imp(guarantee)	1:0:0	All
disconn_mult_inputs	Stopat	1:0:0	All
disconn_mult_inputs.0	imp(stopat)	1:0:0	Only Proven+Un...
adder_proof	imp(cumulative-AG)	1:0:0	All
fma_proof	imp(assume)	1:0:0	All

Stopat Operation

Case split operation

Name	Type	PS Result	Propagation
All Nodes			
SPFMA	Root	1:0:0	None
fma_cases	Case Split	1:0:0	All
x_is_NAN	imp(case split)	1:0:0	All
y_is_NAN	imp(case split)	1:0:0	All
z_is_NAN	imp(case split)	1:0:0	All
DDD	imp(case split)	1:0:0	All
DDN	imp(case split)	1:0:0	All
DND	imp(case split)	1:0:0	All
DNN	imp(case split)	1:0:0	All
NDD	imp(case split)	1:0:0	All
NDN	imp(case split)	1:0:0	All
NND	imp(case split)	1:0:0	All
NNN	imp(case split)	1:0:0	All
fma_cases.c...	imp(case split)	1:0:0	All

# 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)

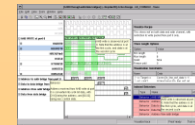


Broadest Range of  
Formal Solvers



Leading Performance  
and Capacity

Jasper® Visualize™  
Interactive  
Environment



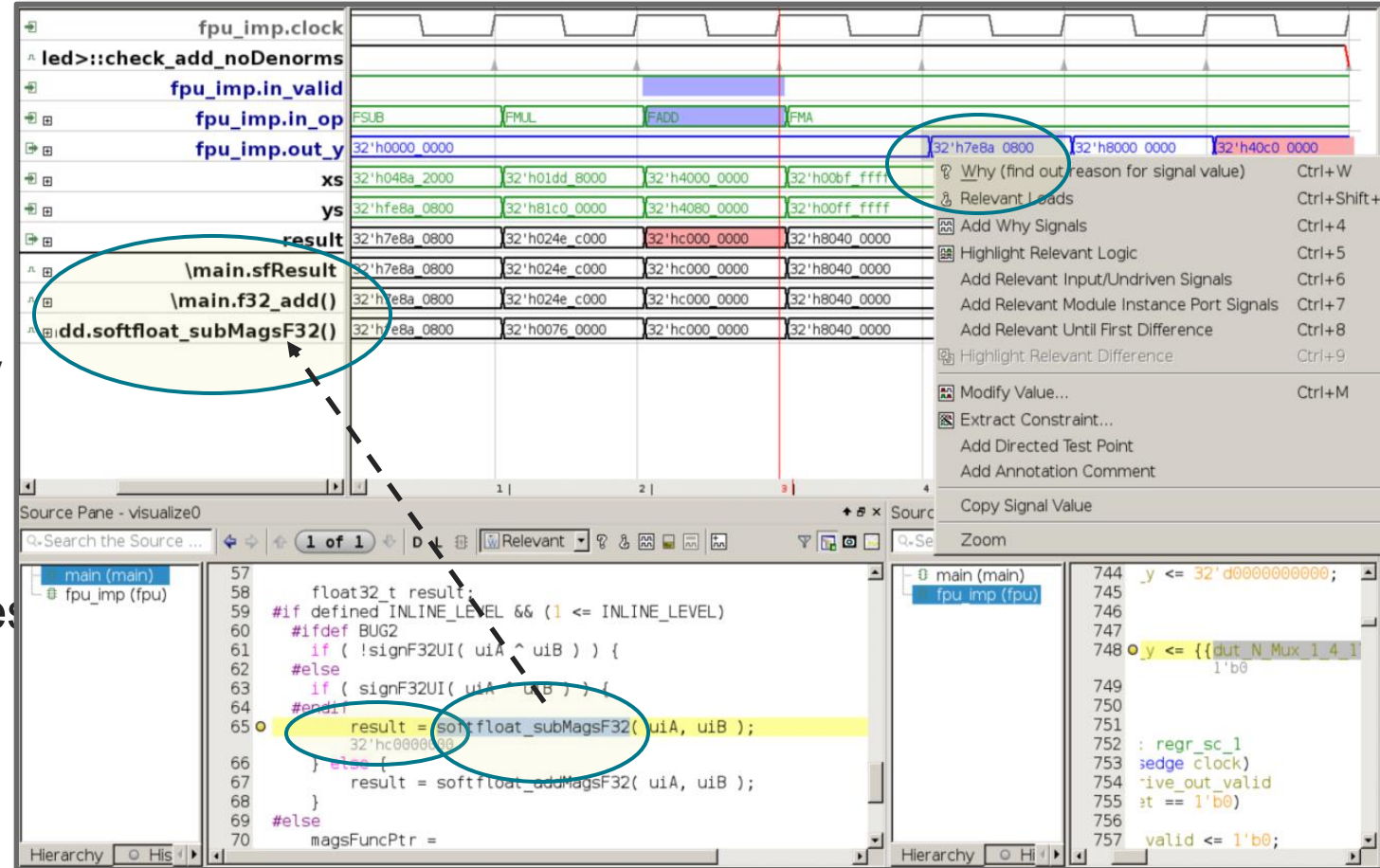
Leading Ease-of-Use

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

The screenshot displays the Cadence Jasper Apps interface for debugging. The main window shows a target configuration for 'tx' with memory addresses. Two 'Open With GDB' dialog boxes are open, showing configuration options like Cycle Number, Path to Gdb, Path to Terminal, Select Mode, and Compilation Command. A third window shows the GDB console output.

**Open With GDB (on vlnx576) Dialog 1:**

- Cycle Number: 1
- Path to Gdb: gdb
- Path to Terminal: xterm
- Select Mode: Use Compile Command
- Compilation Command: Use Existing Executable
- Link SystemC 2.3.3
- Compile SystemC Sources

**Open With GDB (on vlnx576) Dialog 2:**

- Cycle Number: 1
- Path to Gdb: gdb
- Path to Terminal: xterm
- Select Mode: Use Compile Command
- Compilation Command: g++ div\_radixR.cpp -D RADIX=4 -D WIDTH=32
- Link SystemC 2.3.3
- Compile SystemC Sources

**gdb (on vlnx576) Console Output:**

```
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...done.
Trace file loaded for this run: /home/barbara/Examples/division/Rochester-SystemC/jgproject/sessionLogs/session_0/visualize:1_cycle_1.
Input name: x , file path: /home/barbara/Examples/division/Rochester-SystemC/div_radixR.cpp , line: 176
      value:0xf8000000, signal size:32
Input name: y , file path: /home/barbara/Examples/division/Rochester-SystemC/div_radixR.cpp , line: 177
      value:0x00000001, signal size:32
(gdb)
```



# Jasper C2RTL

## Datapath Verification App

# C2RTL App GUI

The screenshot displays the C2RTL App GUI with several callouts highlighting key components:

- Spec (C/C++)**: Points to the 'Spec Design' tab in the Design Hierarchy.
- Imp (RTL)**: Points to the 'Imp Design' tab in the Design Hierarchy.
- Source code**: Points to the C code in the Spec Design tab and the RTL code in the Imp Design tab.
- Properties & verification results**: Points to the 'Property Table' window.

The 'Property Table' window shows the following data:

Type	Name	Bound	Trace	Time
Assume	assume:0	?	0	0
Assert(...)	out[15:0]	WA1 ... Infinite	0	<0

The 'Console' window shows the following output:

```
##2 (mul2_top_imp.out[15:0] == $past(out[15:0], 2))
```

The 'Signal Mapping' window shows the following status:

Signal Mapping	Property Table
Total: 2	Filtered: 2
Selected: 1	Validity: 1:0:0:1
Run: 1:0:0:1	

The 'Console' window also shows the following status:

Console	Lint Messages	Warnings / Errors	Proof Messages
- unknown	: 0	- error	: 0

The 'Console' window also shows the following status:

Console	Lint Messages	Warnings / Errors	Proof Messages
- unknown	: 0	- error	: 0

# 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)



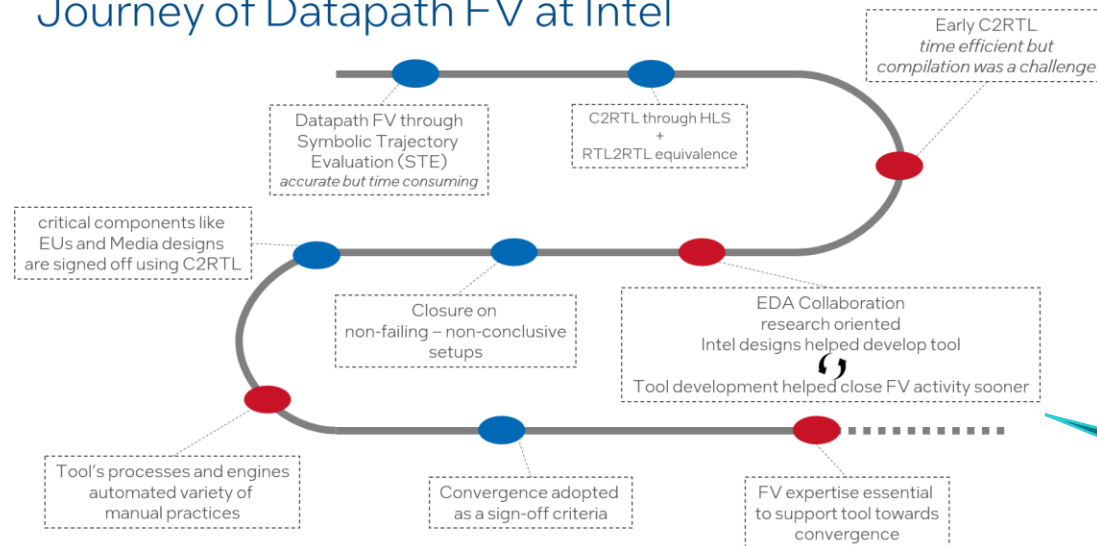
# Jasper C2RTL: Essential for Datapath Verification

## PROOF CONVERGENCE TIME - HIGHLIGHTS

### Conclusion

### Results

## Journey of Datapath FV at Intel



Formal Verification Central Tech Office (FVCTO)

intel.

Intel @ JUG 2021

NXP @ CDNLive India 2022  
Exhaustive formal datapath verification of RISC-V-based Floating Point Units

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

ST @ CDNLive EMEA 2023  
Techniques to check equivalence between C model from Matlab vs RTL for Signal Processing IPs

TI @ CDNLive India 2023  
C2RTL gives ~40x productivity & exhaustive verification on Floating Point and Trigonometric operations

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

Intel @ JUG India 2024  
Software formal for Firmware

AheadComputing @ JUG 2025  
RISC-V based 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

The screenshot displays the HLSEC App GUI, which is used for hardware-logic equivalence checking. The interface is divided into several panes:

- Design Hierarchy:** Shows the hierarchy of the design, with 'Spec Design' and 'Imp Design' tabs. A callout bubble points to 'Spec (SystemC)' in the Spec Design pane.
- Source code:** The main area displays the source code for the design. A callout bubble points to 'Source code' in the Spec Design pane, and another points to 'Source code' in the Imp Design pane.
- Imp (RTL):** A callout bubble points to 'Imp (RTL)' in the Imp Design pane.
- Mapping and verification results:** A callout bubble points to 'Mapping and verification results' in the 'Signal Mapping' pane.

The 'Signal Mapping' pane shows a table of mappings between Spec and Imp signals:

Status	ID	Spec Signal	Imp Signal	Mapping
✓	0	d_out1[31:0]	top.d_out1[...]	Primary O
✗	1	d_out2[31:0]	top.d_out2[...]	Primary O
●	2	clock	top.clock	Primary Ir
●	3	reset	top.reset	Primary Ir
●	4	d_in[31:0]	top.d_in[31:0]	Primary Ir

The bottom of the GUI shows the 'Console' pane with lint messages, warnings/errors, and proof messages. The status bar indicates 'No proofs running' and 'Console input ready'.



# HLSEC App Status

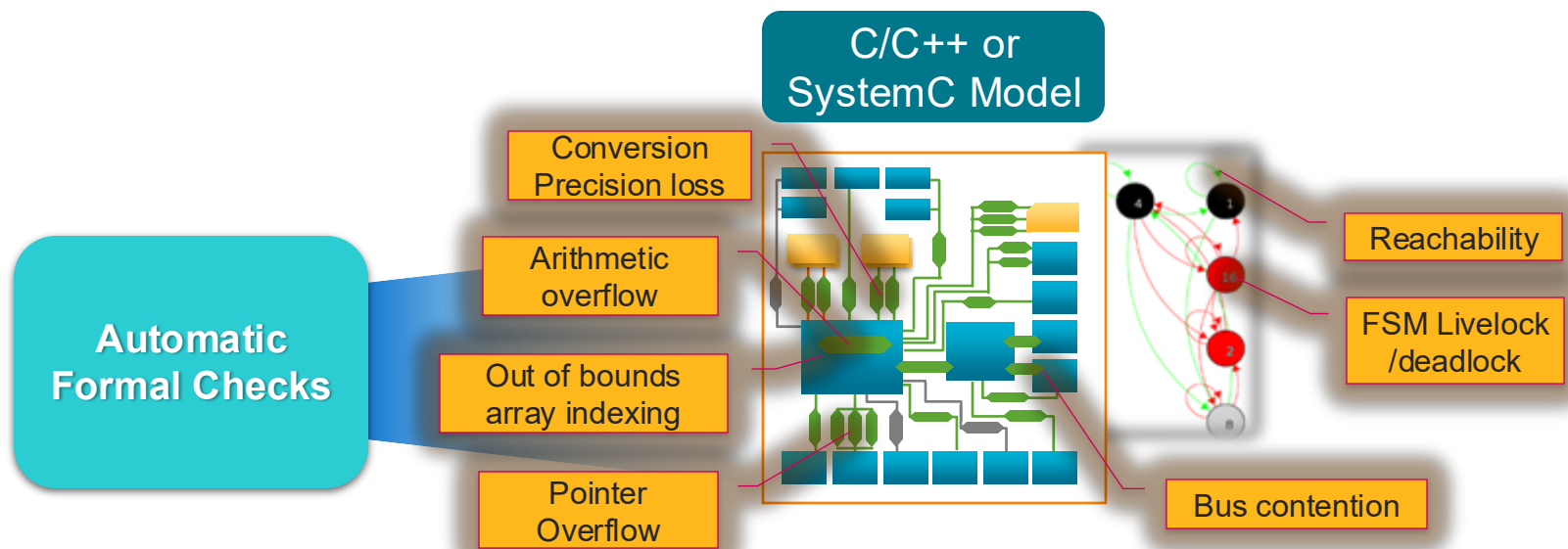
- Working with selected customers
  - Jasper® C HLSEC App flow is working well in all engagements
  - Excellent proof capacity and performance already demonstrated on multiple designs
- Frontend:
  - No gaps in ongoing engagements
  - Goal: Consume all SystemC code that Stratus can compile
  - R&D is 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

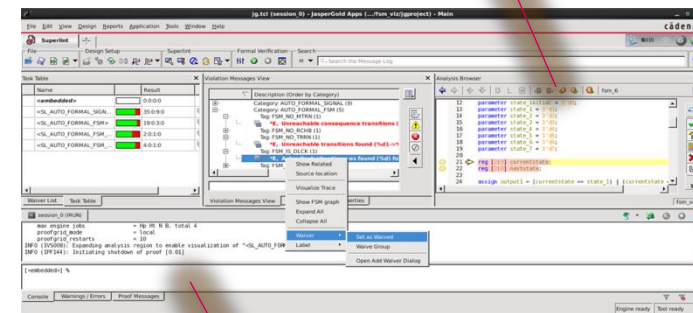
# Jasper CAF Overview



Enabled by true formal technology

Best-in-class debug

Jasper® Visualize™ Interactive Debug Environment



Low-noise violation and waiver handling

**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

The screenshot displays the CAF App GUI with several key components and callouts:

- Groups:** A callout points to the **Task Tree** on the left, which lists tasks like `<embedded>`, `<CAF_AUTO_...`, and `<CAF_AUTO_...` with their respective results.
- Checks:** A callout points to the **Automatic Formal Properties** table in the center. This table lists properties with columns for Tag, Type, Name, Engine, and Boolean value. The table shows several `HLS_CV_OVFL` assertions and one `Assume` property.
- Source code:** A callout points to the **Analysis Browser** on the right, which displays the source code for `HLS_CV_OVFL`. The code includes comments and function calls like `c.write(a.read() << b.read());`.
- Waives:** A callout points to the **Waiver List** at the bottom right, which shows `<No waiver data>`.

The bottom of the GUI features a **Console** tab with **Warnings / Errors** and **Proof Messages**, and a status bar indicating **No proofs running** and **Console input ready**.

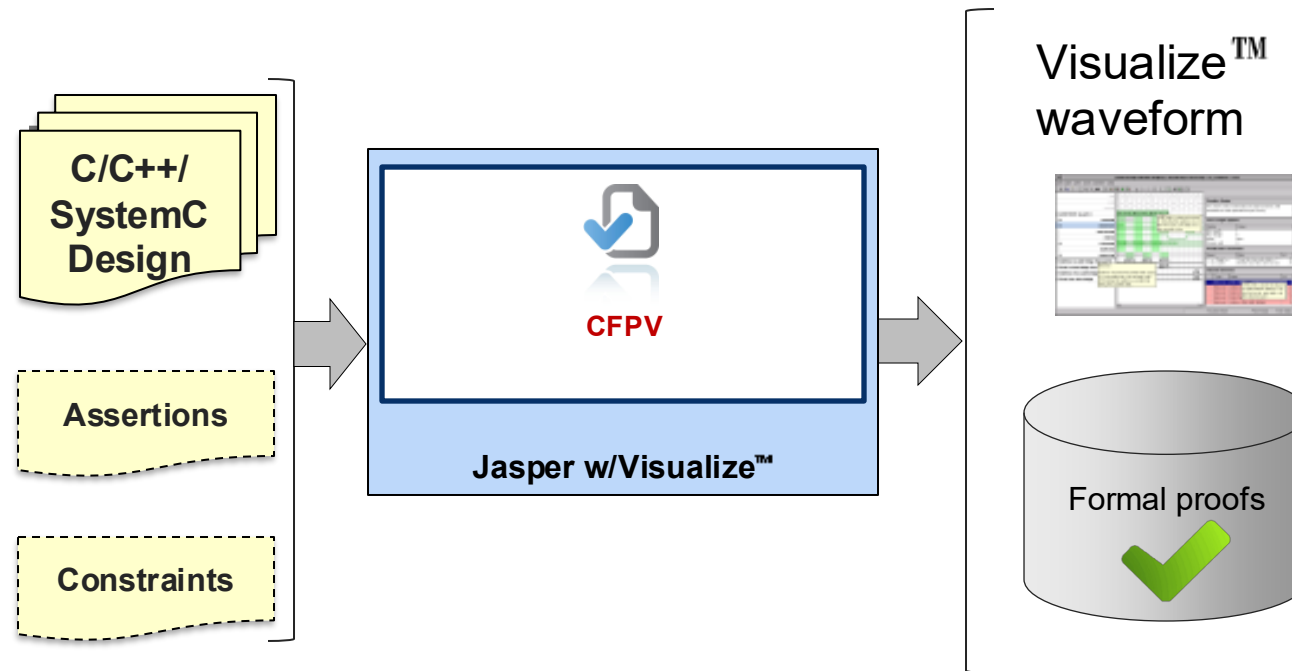


# Jasper CFPV and CCoverage

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

The screenshot shows the Cadence Jasper App GUI interface. The main window displays a C code snippet for a function `main()` that uses `JASPER_C` macros for input and output. The interface includes a menu bar (File, Edit, View, Design, Application, Window, Help), a toolbar, and several panels:

- Design Hierarchy:** Shows the module structure for `main`.
- Code Editor:** Displays the C code snippet.
- Property Table:** A table listing properties and their verification status.
- Console:** Shows the execution log.

Four callouts highlight specific features:

- Call stack:** Points to the `div_radixR` function call in the code.
- C/C++/SystemC:** Points to the `JASPER_C` macro usage in the code.
- Proof Structure:** Points to the `Proof Structure` tab in the bottom panel.
- Properties:** Points to the **Property Table** panel.

Type	Name	Engine	Bound	Target Bou	Traces	Time
Assume	main.check_q\$13:assume	?		N/A	0	0.0
Assume	main.check_r\$13:assume	?		N/A	0	0.0
Assume	main.check_q\$14:assume	?		N/A	0	0.0
Assume	main.check_r\$14:assume	?		N/A	0	0.0
Assert(...)	final_q	PRE	Infinite	N/A	0	0.0
Assert(...)	final_r	Hp (1)	Infinite	N/A	0	<0.1
Assume	main.check_q:assume	?		N/A	0	0.0
Assume	main.check_r:assume	?		N/A	0	0.0
Assume	main.check_q\$1:assume	?		N/A	0	0.0
Assume	main.check_r\$1:assume	?		N/A	0	0.0

q \* y + r == x

Total: 602 Filtered: 602 Selected: 1

Console: 2025-11-07 22:15:09: Completed  
2025-11-07 22:15:09: Completed  
2025-11-07 22:15:09: Completed  
-I- All node runs are complete.

[TOP] %

Console Lint Messages Warnings / Errors Proof Messages

No proofs running Console input ready



# CCoverage

The screenshot displays the Cadence Jasper tool interface. The top menu bar includes File, Edit, View, Design, Application, Window, and Help. The main window is divided into several sections:

- Coverage Analysis:** Shows Formal Statement Local Coverage at 2882/2893 (99.62%) and Stimuli Statement Local Coverage at 2882/2893 (99.62%).
- Models:** Includes Branch, Statement, Expression, Toggle, FSM, Covergroup, and Property.
- Scope:** Set to Local.
- Current Instance:** Set to main.

The main display area shows a table of coverage results for the Stimuli Statement. The table has columns for Expression, Form, Stimuli, Check, COI, Description, and Source Location. The results show that all stimuli are covered (100% coverage).

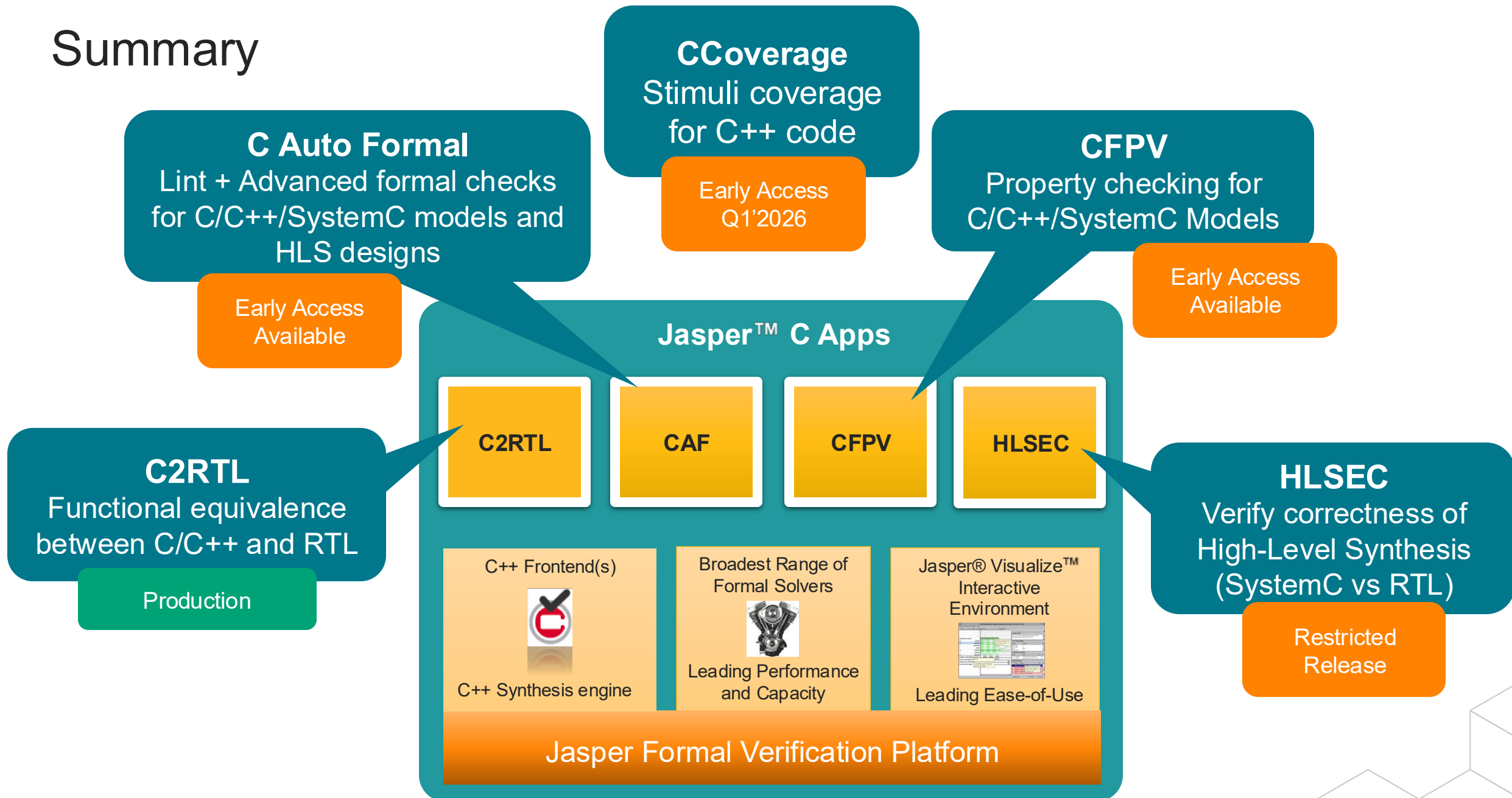
Below the table, a code snippet is displayed, showing a C/C++/SystemC implementation of a loop. The code is highlighted in yellow, indicating it is the selected task.

**Stimuli Coverage**

**C/C++/SystemC**

The bottom status bar indicates: No proofs running, Console input ready, No error in console.

# Summary





# cādence®

© 2025 Cadence Design Systems, Inc. All rights reserved worldwide. Cadence, the Cadence logo, and the other Cadence marks found at <https://www.cadence.com/go/trademarks> are trademarks or registered trademarks of Cadence Design Systems, Inc. Accellera and SystemC are trademarks of Accellera Systems Initiative Inc. All Arm products are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All MIPI specifications are registered trademarks or service marks owned by MIPI Alliance. All PCI-SIG specifications are registered trademarks or trademarks of PCI-SIG. All other trademarks are the property of their respective owners.

