Achieve First-pass Silicon with Efficient RTL to Gate Static Signoff Methodology Using VC SpyGlass

Rimpy Chugh, Sr Staff Product Manager

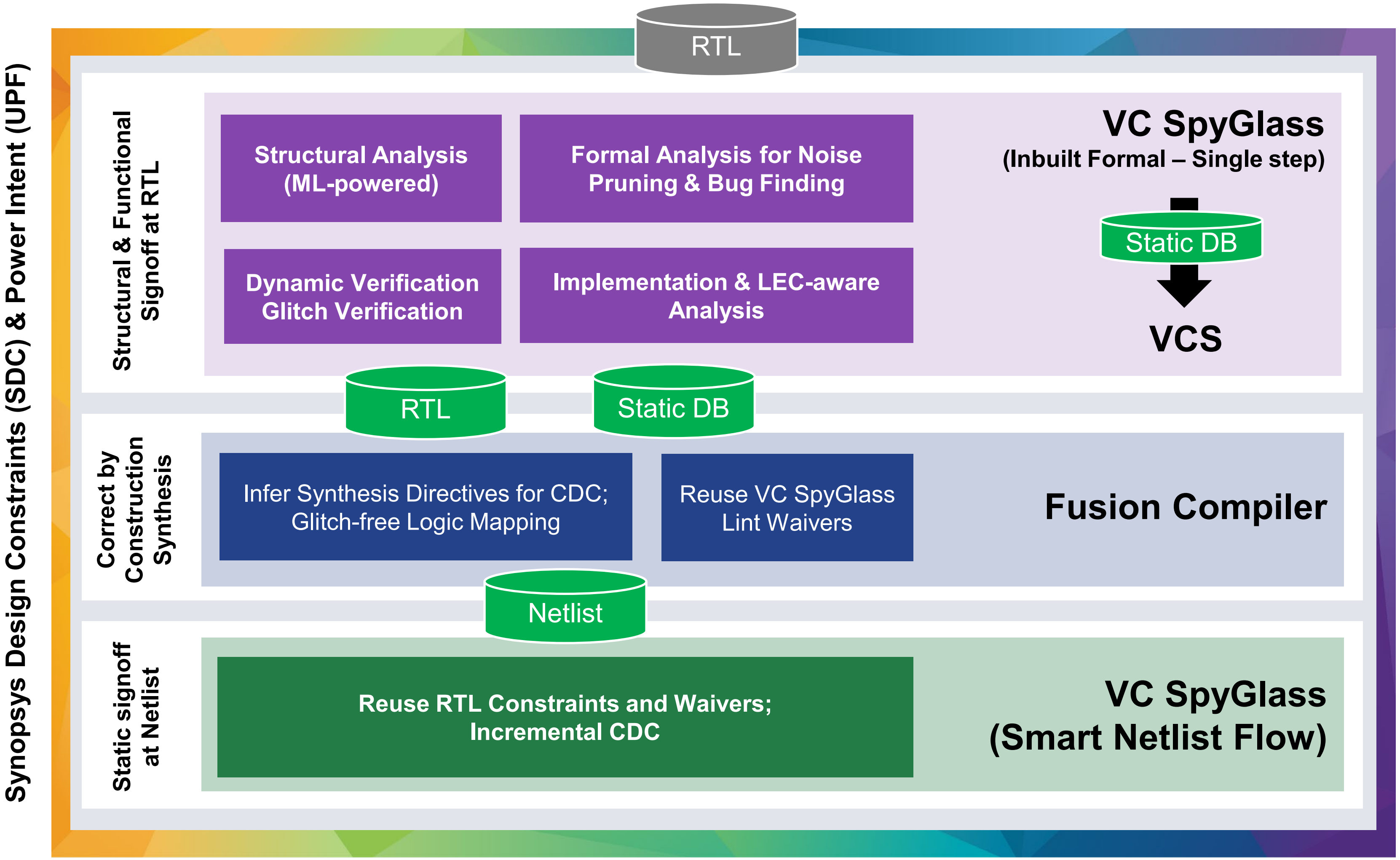
Synopsys

Agenda

- Achieve Static Signoff Across RTL2Gate Flow
  - Find Critical Bugs using Comprehensive Glitch Verification
- Is Your Design Implementation Ready?
- Ensure Prequalified CDC Paths Remain Intact During Synthesis
  - Static-aware Synthesis
  - Smart Netlist Flow
- Gain 100% Confidence on CDC Assumptions & Protocols
  - Seamless Functional CDC Signoff Using Inbuilt Formal and Waveform Replay Technology
- Correct by Construction Design Development Beyond Linting
  - Formality-aware Linting
  - Power Linting
  - Testbench Linting using Euclide

Achieve Static Signoff Across RTL2Gate Flow

Ensuring correct-by-construction design for static bugs till later design cycles



Synopsys Design Constraints (SDC) & Power Intent (UPF)
RTL
Structural & Functional Signoff at RTL
Structural Analysis (ML-powered)
Formal Analysis for Noise Pruning & Bug Finding
Dynamic Verification Glitch Verification
Implementation & LEC-aware Analysis
VC SpyGlass
(Inbuilt Formal – Single step)
Static DB
VCS
RTL
Static DB
Correct by Construction Synthesis
Infer Synthesis Directives for CDC; Glitch-free Logic Mapping
Reuse VC SpyGlass Lint Waivers
Fusion Compiler
Netlist
Static signoff at Netlist
Reuse RTL Constraints and Waivers; Incremental CDC
VC SpyGlass
(Smart Netlist Flow)

Highly Accurate & Productive signoff

Catch functional bugs early at RTL; Gain best possible productivity

Static Aware Synthesis

Ensure pre-verified CDC/RDC/MCP paths remain intact during synthesis

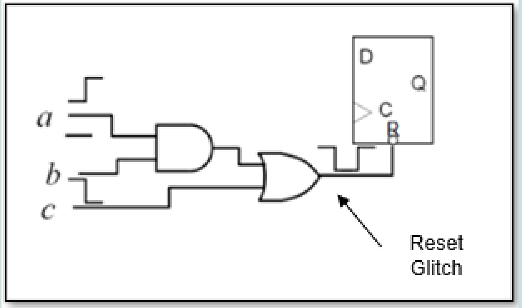
Incremental Gate-level Verification

Ensure no new Static bugs introduced at netlist requiring fixes

Find Critical Bugs Using Comprehensive Glitch Verification

Asynchronous Paths

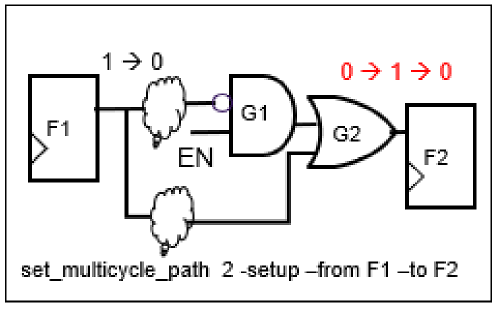
(CDC, Clock, RDC, Reset)



Caught by current static solutions

Synchronous Paths

(MCP, FP, Max-delay)

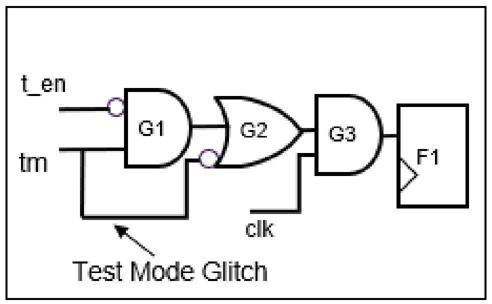


```
set_multicycle_path  2 -setup –from F1 –to F2
```

Not caught by STA or CDC tools

Test (DFT) Paths

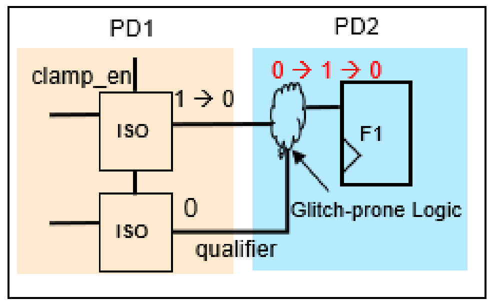
(Mode Transition, Clock Merge)



Mode transition glitches not caught by STA or CDC tools due to constraints

Special Glitch Paths

(Power Clamp, A2D, D2A)



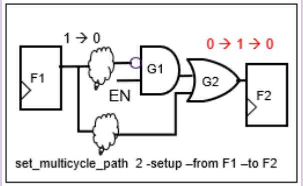
Specific logic might cause functional glitch issues

Comprehensive Glitch Solution Accelerated Productivity from Days to 1.5 hours

Application	GPU application 2.6B instances

Challenges	Results with VC SpyGlass
• Internal solution runtime takes days/weeks	• Significantly better runtime compared to internal solution
• Painful & inefficient post-processing analysis demanding significant bandwidth	• Ability to avoid manual post-processing without impacting QoR

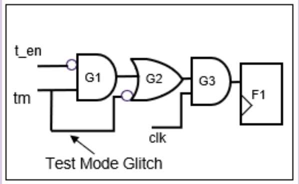
Synchronous Paths (MCP, FP, Max-delay)



set_multicycle_path 2 -setup –from F1 –to F2

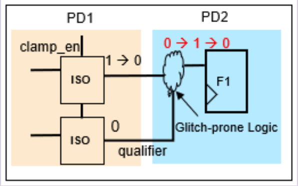
STA and CDC tools miss these glitches on these paths

Test (DFT) Paths (Mode Transition, Clock Merge)



STA and CDC tools will not catch mode transition glitches due to constraints

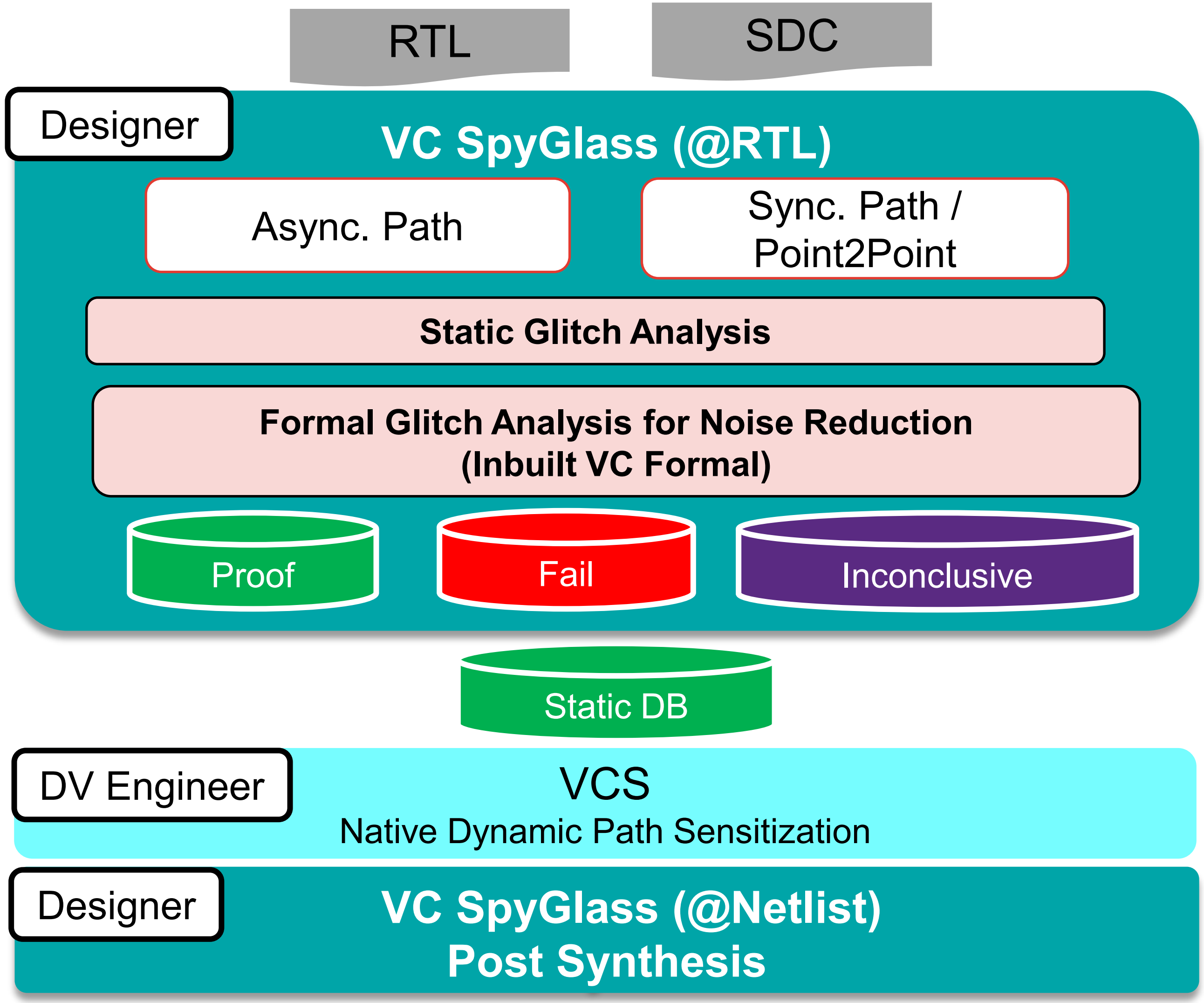
Special Glitch Paths (Power Clamp, A2D, D2A)



Specific logic might have functional glitch issues missed by traditional flows

Comprehensive Glitch Verification Flow Incorporated in Netlist Signoff Checklist

End-2-End Glitch Verification Methodology



Expand from traditional VC SG structural analysis user @RTL

1. Expand RTL structural analysis from async. paths to all other paths in the design

2. Achieve noise reduction by enabling inbuilt formal for CDC/Reset/MCP/P2P

3. Verify remaining paths using dynamic VCS simulation
   - Native dynamic path sensitization-based simulation enables faster TAT and provides easy analysis leveraging glitch-specific coverage

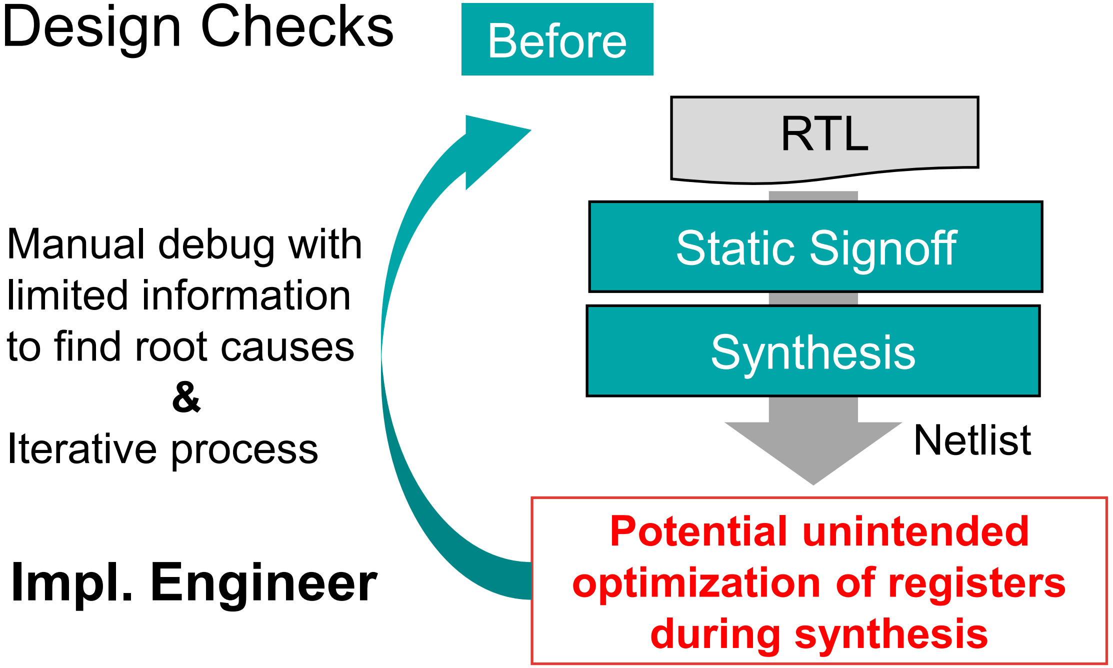
4. Structural analysis using VC SpyGlass @Netlist

Is your Design Implementation Ready?

Avoid unintended bugs due to synthesis optimizations with Implementation Design Checks

Is Your Design Implementation Ready?

Current Flow Challenges and SHIFT-LEFT with Implementation Design Checks



Before

Manual debug with limited information to find root causes

&

Iterative process

Impl. Engineer

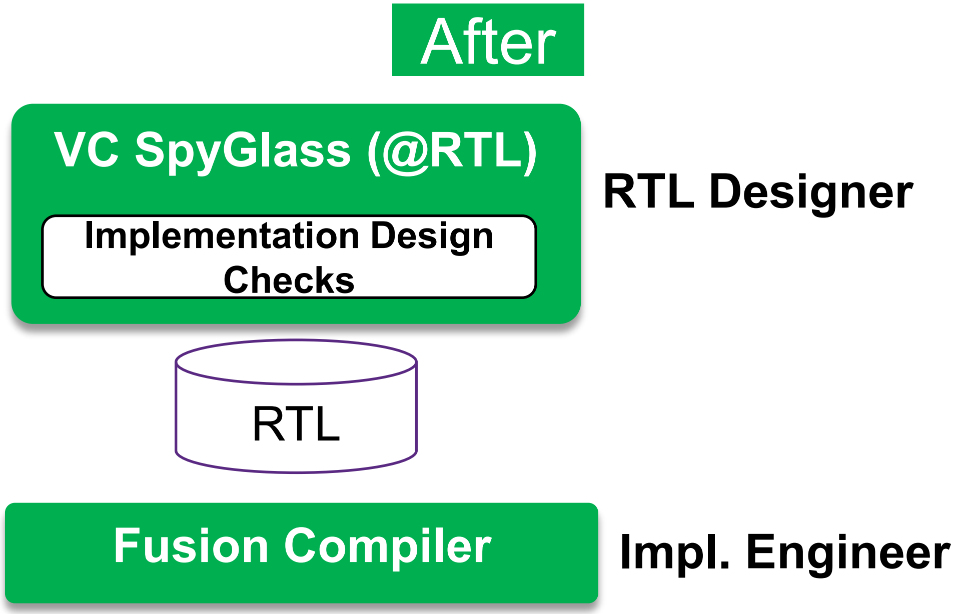
RTL

Static Signoff

Synthesis

Netlist

Potential unintended optimization of registers during synthesis



After

VC SpyGlass (@RTL)

Implementation Design Checks

RTL Designer

RTL

Fusion Compiler

Impl. Engineer

Finding root-cause for ~70% of registers obvious.
Higher TAT (2-3 months) for root cause analysis of pending 30%

Lack of mechanisms to determine logic congestion issues before RTL handoff

Multiple iterations between RTL & Synthesis teams
Ex. huge difference in area from one RTL drop to another

SHIFT-LEFT and easy root cause analysis of constant register optimized during synthesis

SHIFT-LEFT and easy design fixes of logic congestion issues as part of RTL signoff

No iteration due to unintentional register optimizations or logic congestion; Predictable overall design cycle

Implementation Design Checks (IDC): Case Studies

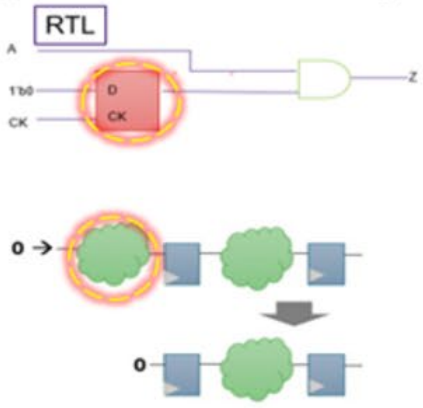
Infrastructure Technology Leader, US	Leading Processor Supplier, US	Leading Technology Company, US	Smart EV Maker, China
Unexpected optimization due to registers incorrectly tied to zero	Manual & iterative high debug TAT for constant registers optimized during synthesis	Manual & high debug TAT of 3-4 months on 8B+ design (32 tiles)	Unexpected optimization of 72k registers were optimized during synthesis
⬇	⬇	⬇	⬇
Quoted “Exceptional proficiency in identifying stuck at fault registers/flops early in the RTL coding phase”	Decided to use IDC for weekly regression runs for every design	Hard to find complex root-causes were detected within a week earlier in design cycle	Caught real RTL bug where a config register was unexpectedly optimized per design intent

Implementation Design Checks (IDC)

Applications

Constant Optimized Registers

Register Removal Report



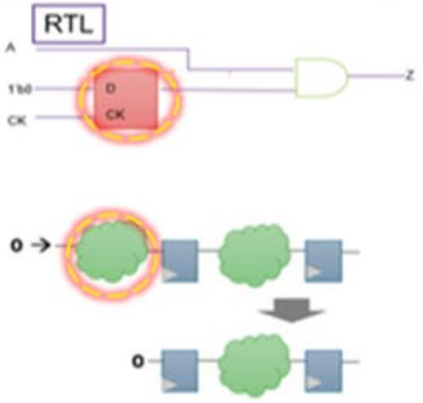
Constant Register

- Shift left detection and root cause analysis of constant optimized registers.
- 10X faster reporting

Saves Debug Iterations

Unloaded Optimized Registers

Register Removal Report



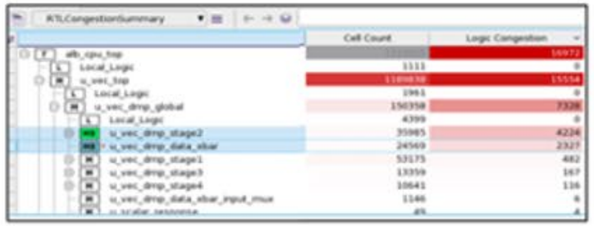
Unloaded Register

- Shift left detection and root cause analysis of unloaded optimized registers.
- 10X faster reporting

Saves Debug Iterations

Logical Congestion

Congestion Report



	Cell Count	Logic Congestion
alb_cpu_top	[illegible]	16972
Local_Logic	1111	0
u_vec_top	1189838	15554
Local_Logic	1961	0
u_vec_dmp_global	150358	7328
Local_Logic	4399	0
u_vec_dmp_stage2	35985	4224
u_vec_dmp_data_xbar	24569	2327
u_vec_dmp_stage1	53175	482
u_vec_dmp_stage3	13359	167
u_vec_dmp_stage4	10641	116
u_vec_dmp_data_xbar_input_mux	1146	6
u_scalar_response	45	4

```
// 8bit read
module read_8 ( regfile , raddr , out ) ;

  input [7:0] regfile [255:0] ;
  input [7:0] raddr ;
  output [7:0] out ;

  assign out = regfile[raddr] ;

endmodule
```

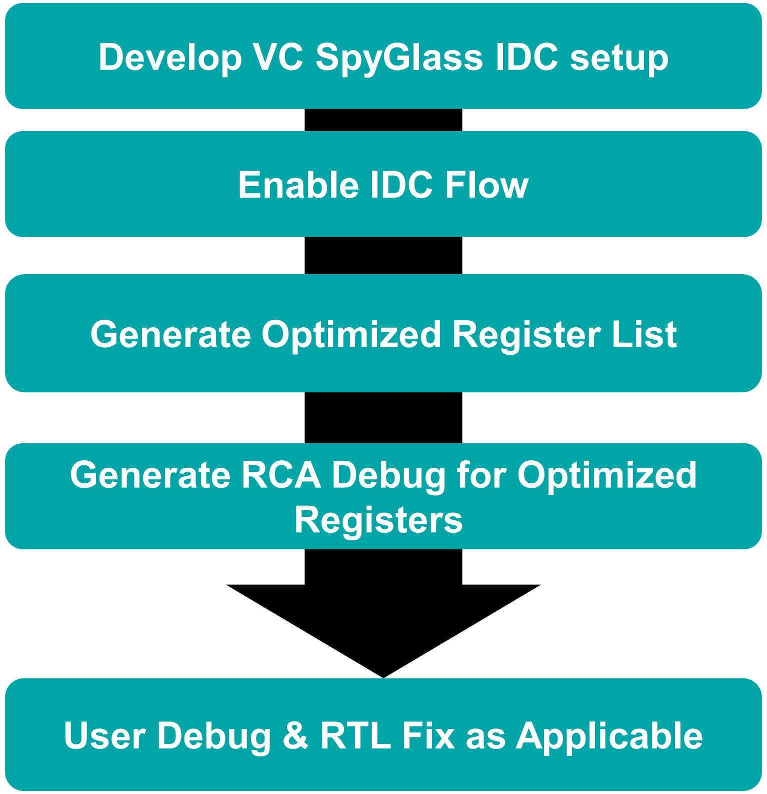
Big MUX

Identify Problematic RTL

- Select Ops
- X-bars

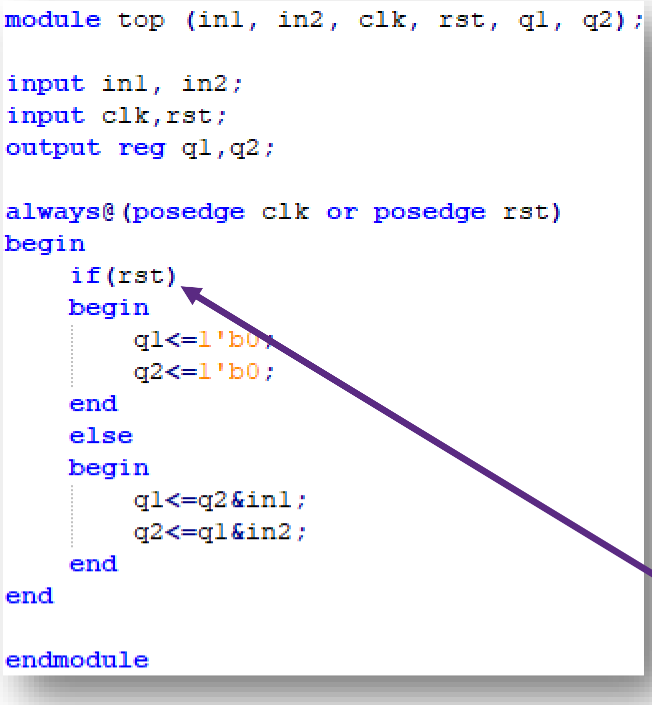
Improve P&R Success

Implementation Design Checks (IDC) Flow

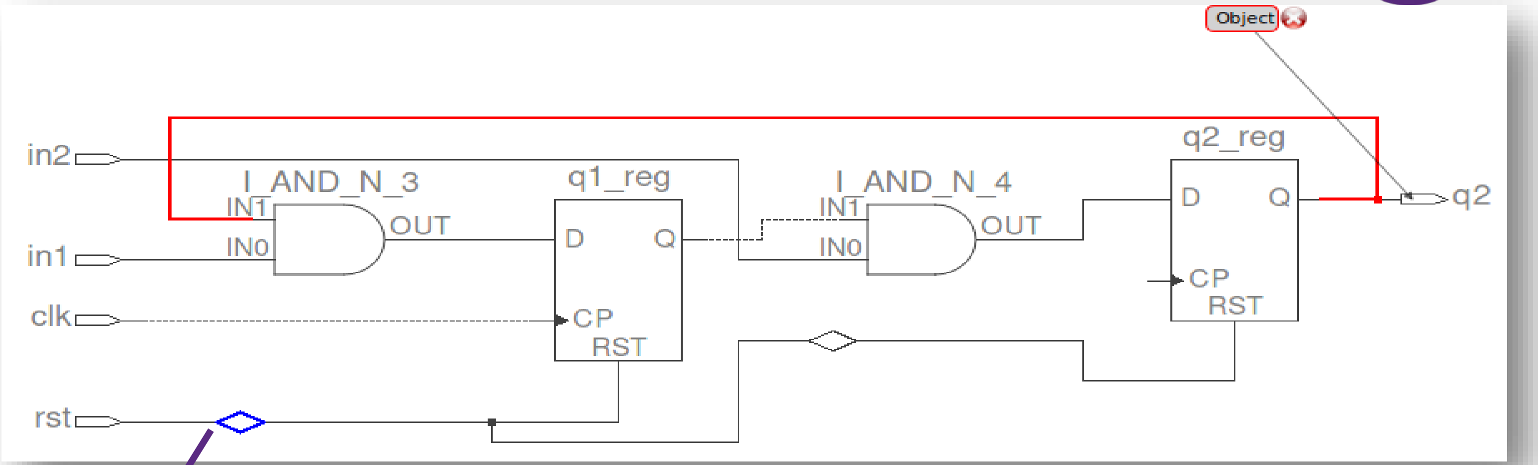

Develop VC SpyGlass IDC setup
Enable IDC Flow
Generate Optimized Register List
Generate RCA Debug for Optimized Registers
User Debug & RTL Fix as Applicable

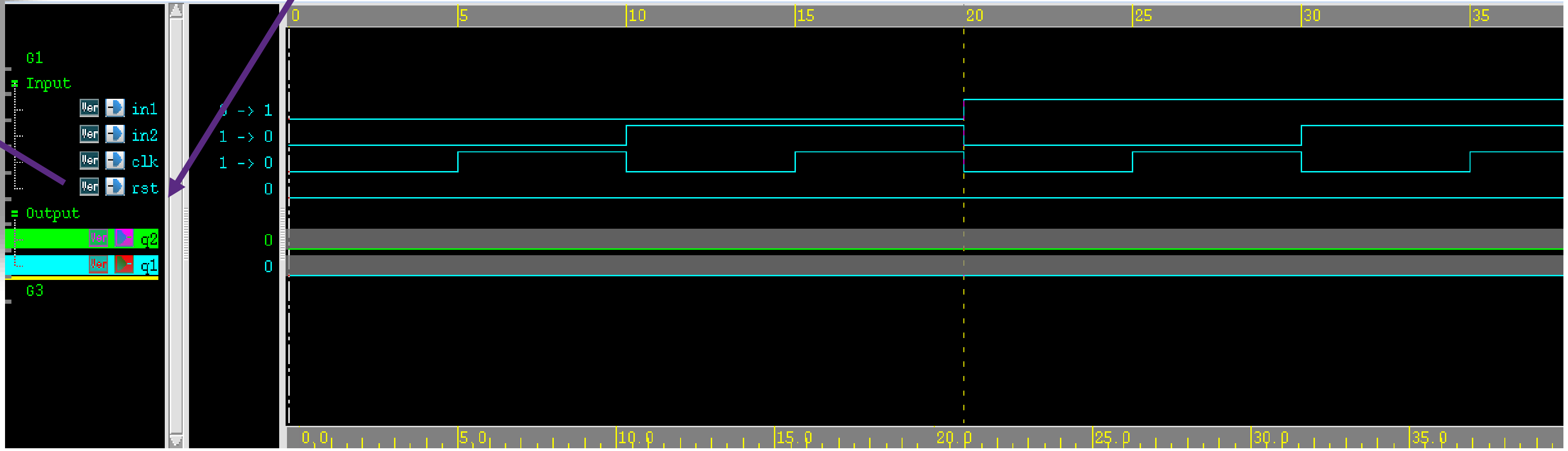
- Develop VC SpyGlass IDC setup
  - Reuse existing VC SpyGlass setup
- Enable push-button IDC flow
- Identify optimized registers due to constant propagation and unused register output
- Easy determination of root cause for optimized registers
- Fix RTL or add waivers for synthesis where applicable

Finding Root Cause for Constant Optimized Registers (CR)

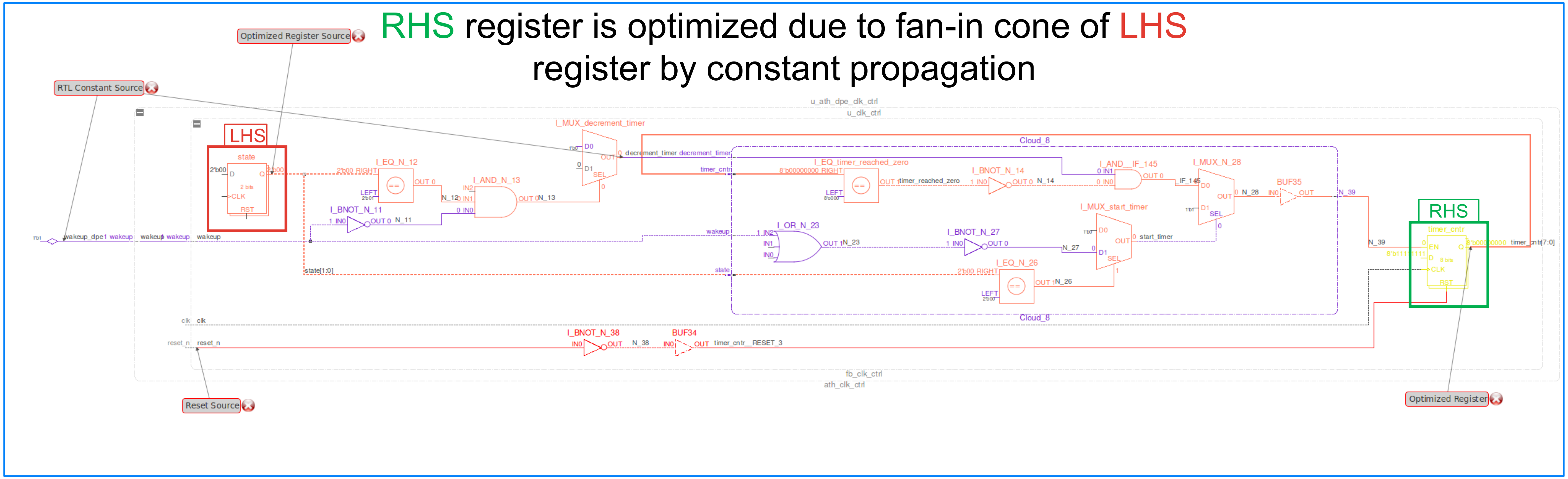


```
module top (in1, in2, clk, rst, q1, q2);

input in1, in2;
input clk,rst;
output reg q1,q2;

always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        q1<=1'b0;
        q2<=1'b0;
    end
    else
    begin
        q1<=q2&in1;
        q2<=q1&in2;
    end
end

endmodule
```

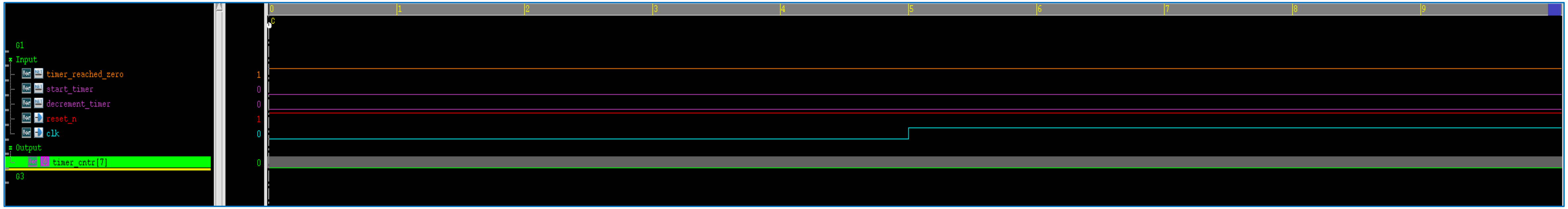




Register Optimized due to a Constant Source



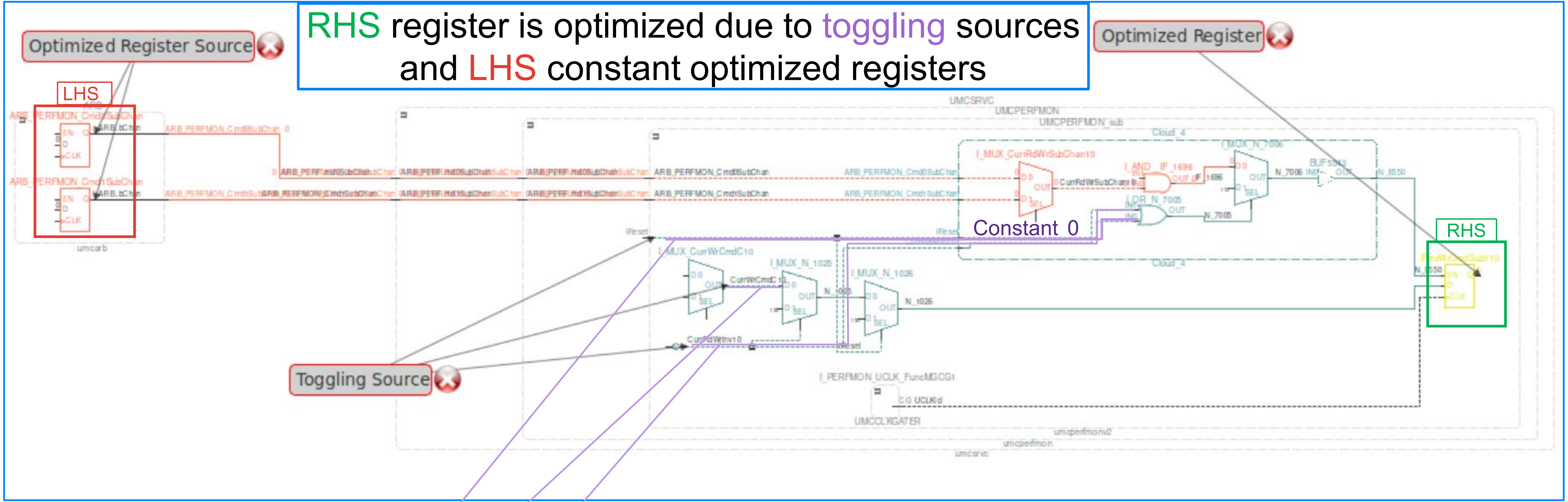
RHS register is optimized due to fan-in cone of LHS register by constant propagation



```
G1
Input
  timer_reached_zero   1
  start_timer          0
  decrement_timer      0
  reset_n              1
  clk                  0
Output
  timer_cntr[7]        0
G3
```

Waveform Witness

Register Optimized due to Toggling Sources



RHS register is optimized due to toggling sources and LHS constant optimized registers

Optimized Register Source

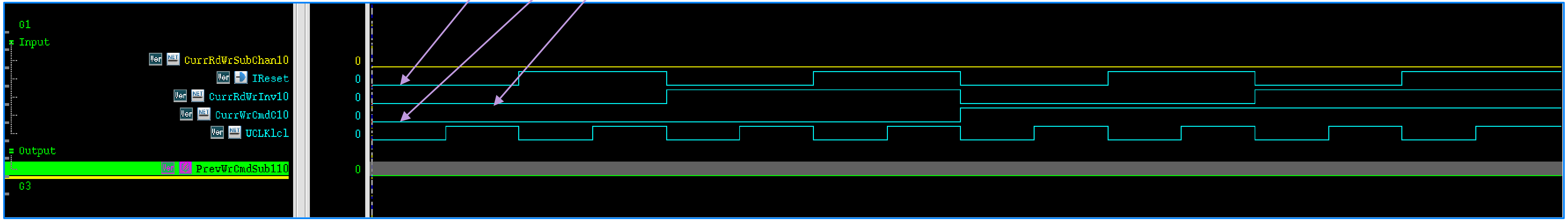
LHS

Optimized Register

Constant 0

RHS

Toggling Source



```
G1
Input
        CurrRdWrSubChan10   0
                   IReset   0
            CurrRdWrInv10   0
             CurrWrCmdC10   0
                  UCLKlcl   0
Output
          PrevWrCmdSub110   0
G3
```

Finding Root Cause of Unloaded Optimized Registers (ULR)

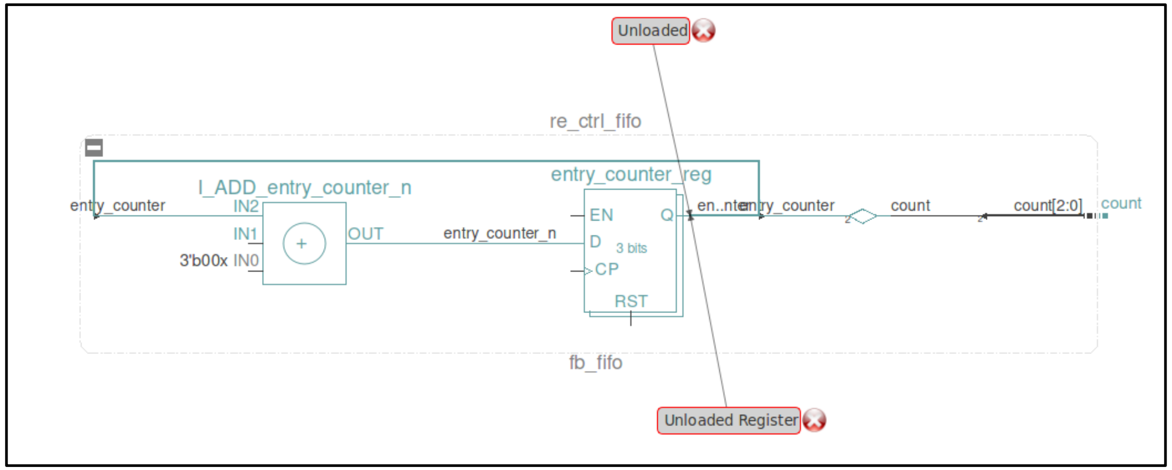
- Synthesis tools optimizes several registers because they're unloaded (ULR).

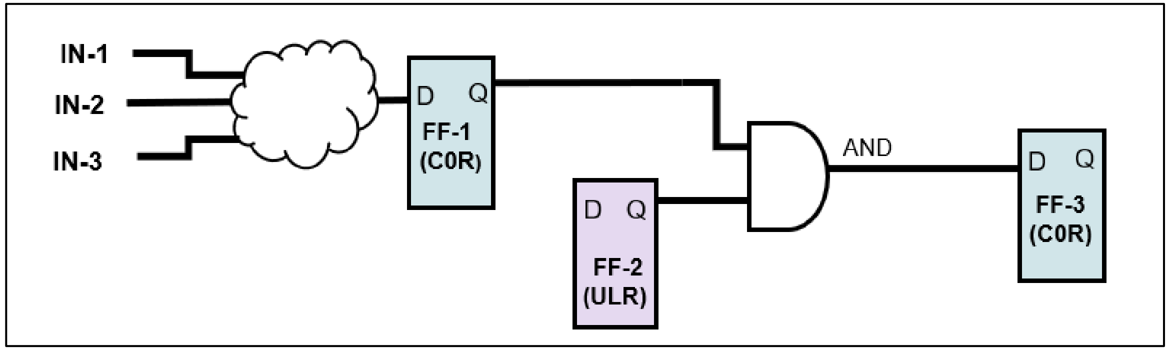
- VC SpyGlass provides detection and pinpointing of the root cause for optimization using following tag :

```
configure_lint_tag -enable -tag "DetectOptimizedUnUsedRegister" -goal <>
```

- For bit-blasted reporting of registers, disable bus-merging:

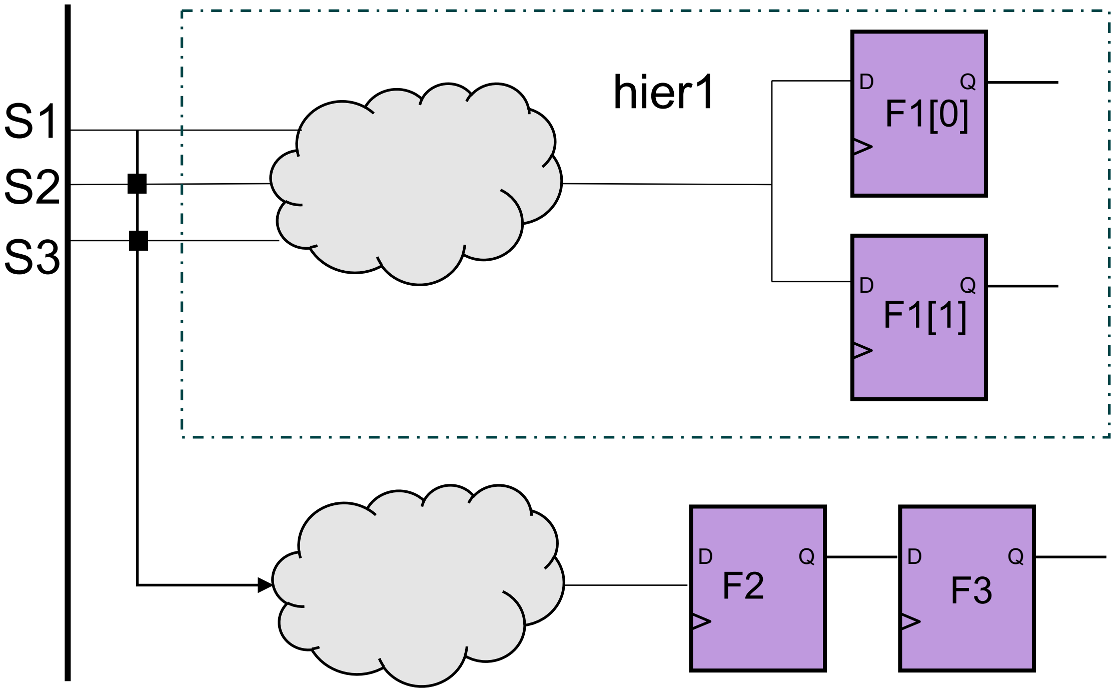
```
lint_disable_bus_merge -tag DetectOptimizedUnUsedRegister
```





Easy Debug via Precise Reporting

- Easy identification of constant & non-constant sources
- Precise reports for multi-bit optimized registers
  - Bus merged when specified
- Reporting for single optimized register from single hierarchy when multiple instances present



Non-Const Sources	Const Sources	Optimized Registers
S1, S2	S3	hier1.F1[0], hier2.F1[0] hier1.F1[1], hier2.F1[0] F2 F3

Bus Merging

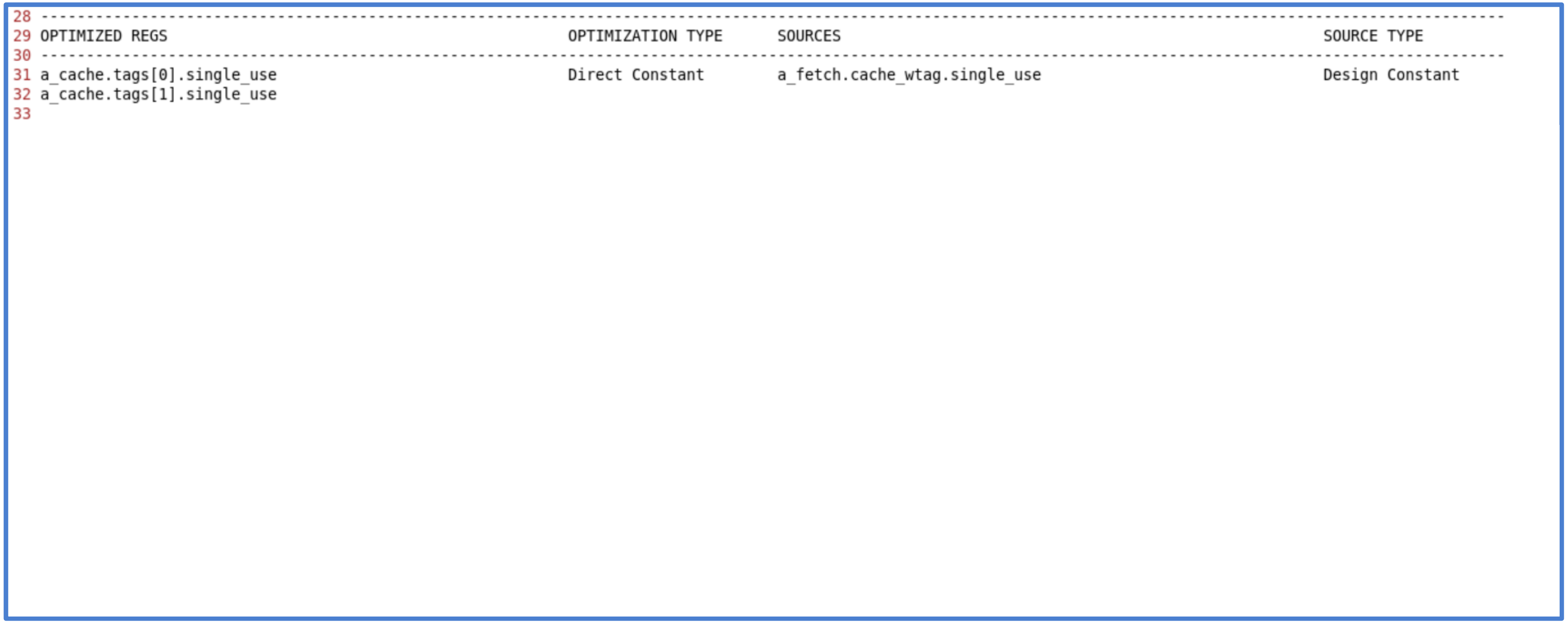
Non-Const Sources	Const Sources	Optimized Registers
S1, S2	S3	hier1.F1[1:0] hier2.F2[1:0] F2 F3

Report from one Hierarchy

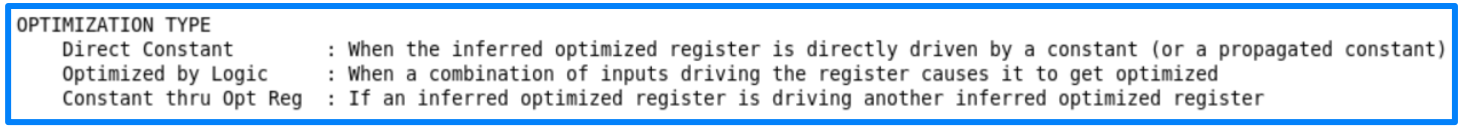
Non-Const Sources	Const Sources	Optimized Registers
S1, S2	S3	hier1.F1[1:0] F2 F3

Optimized Registers Report

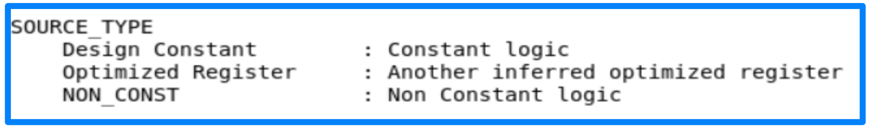
Constant Register sources and Optimization type



```
---------------------------------------------------------------------------------------------------------------------------------------------------------------
OPTIMIZED REGS                                         OPTIMIZATION TYPE      SOURCES                                                     SOURCE TYPE
---------------------------------------------------------------------------------------------------------------------------------------------------------------
a_cache.tags[0].single_use                             Direct Constant        a_fetch.cache_wtag.single_use                               Design Constant
a_cache.tags[1].single_use
```



```
OPTIMIZATION TYPE
    Direct Constant        : When the inferred optimized register is directly driven by a constant (or a propagated constant)
    Optimized by Logic     : When a combination of inputs driving the register causes it to get optimized
    Constant thru Opt Reg  : If an inferred optimized register is driving another inferred optimized register
```



```
SOURCE_TYPE
    Design Constant          : Constant logic
    Optimized Register       : Another inferred optimized register
    NON_CONST                : Non Constant logic
```

Ensure Prequalified CDC Paths Remain Intact During Synthesis

Static Aware Synthesis

Synthesis is Unaware of CDC/RDC Paths

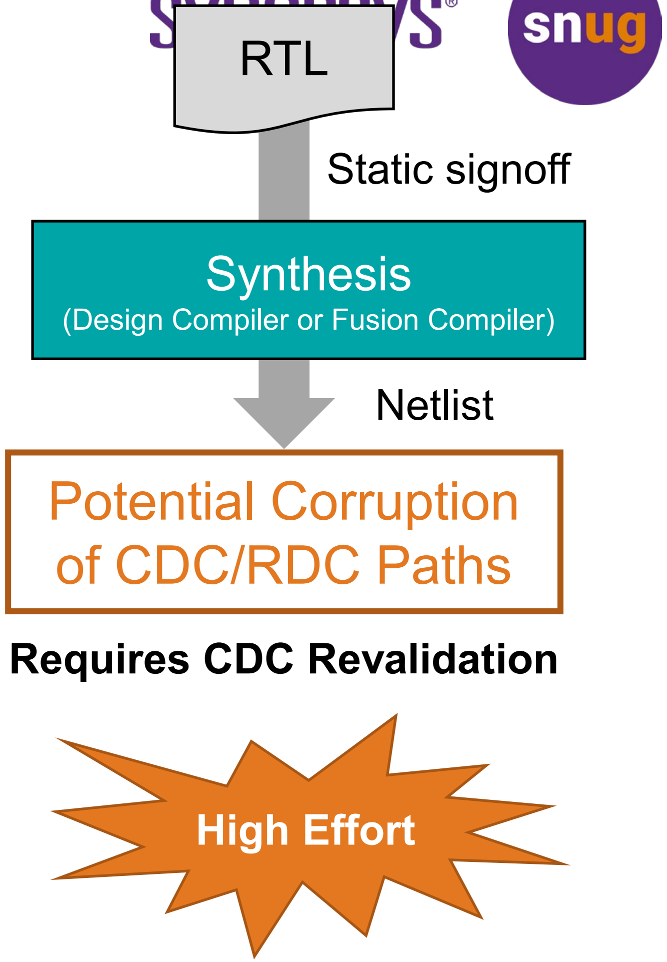
Current Flow Challenges

Error-prone methods used to protect CDC paths during Synthesis using RTL pragmas, manual synthesis directives

Possible corruption of CDC Paths during Synthesis leading to silicon-respin

An additional effort at Gate-level for CDC re-verification

Multiple iterations between RTL & Synthesis teams



RTL

Static signoff

Synthesis

(Design Compiler or Fusion Compiler)

Netlist

Potential Corruption of CDC/RDC Paths

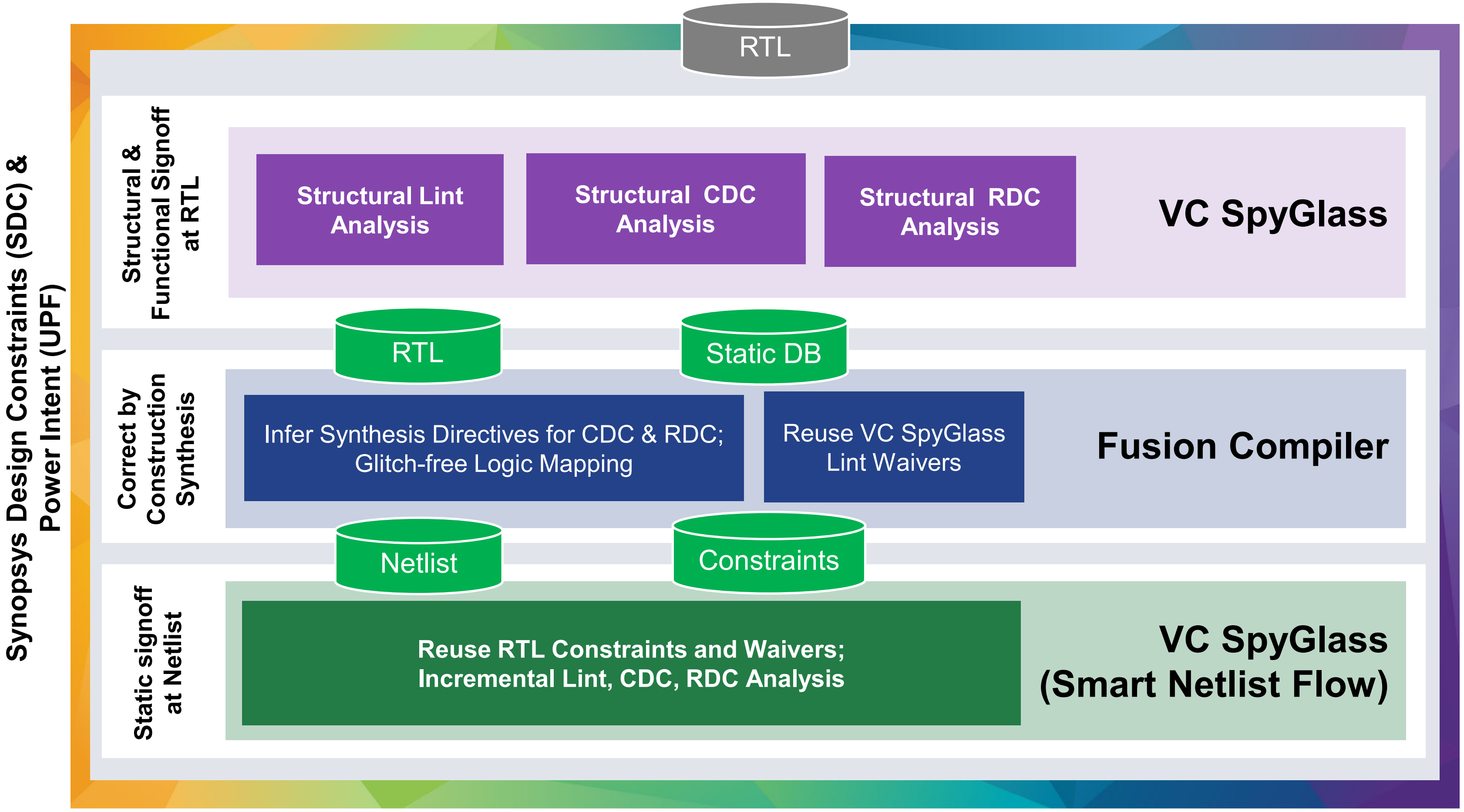
Requires CDC Revalidation

High Effort

Synthesis Need to be Guided to Avoid Disruption of Prequalified CDC/RDC Paths

Synthesis-Aware Static Signoff

Correct by construction synthesis


RTL
Synopsys Design Constraints (SDC) & Power Intent (UPF)
Structural & Functional Signoff at RTL
Structural Lint Analysis
Structural CDC Analysis
Structural RDC Analysis
VC SpyGlass
RTL
Static DB
Correct by Construction Synthesis
Infer Synthesis Directives for CDC & RDC; Glitch-free Logic Mapping
Reuse VC SpyGlass Lint Waivers
Fusion Compiler
Netlist
Constraints
Static signoff at Netlist
Reuse RTL Constraints and Waivers; Incremental Lint, CDC, RDC Analysis
VC SpyGlass (Smart Netlist Flow)

Highlights

✓Preserves CDC signed-off logic at synthesis level enabling correct-by-construction netlist

✓Reduces CDC validation effort at gate-level

✓Reuse of Lint waivers to suppress redundant messages during synthesis avoiding duplicated effort

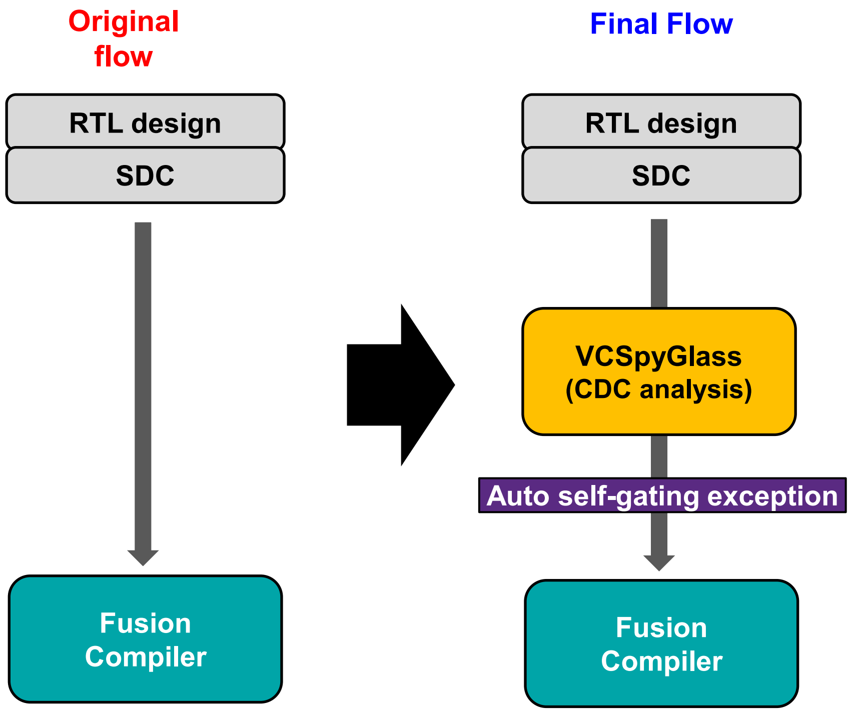
Static-Aware Synthesis Delivers Improved PPA

- ~40% Dynamic power, - 8% Leakage power reduction & ensures NO disruption of pre-verified CDCs

- Objective
  - Reduce total power for taped-out project.
  - Enhance 100% exhaustive CDC analysis ratio without manual workaround

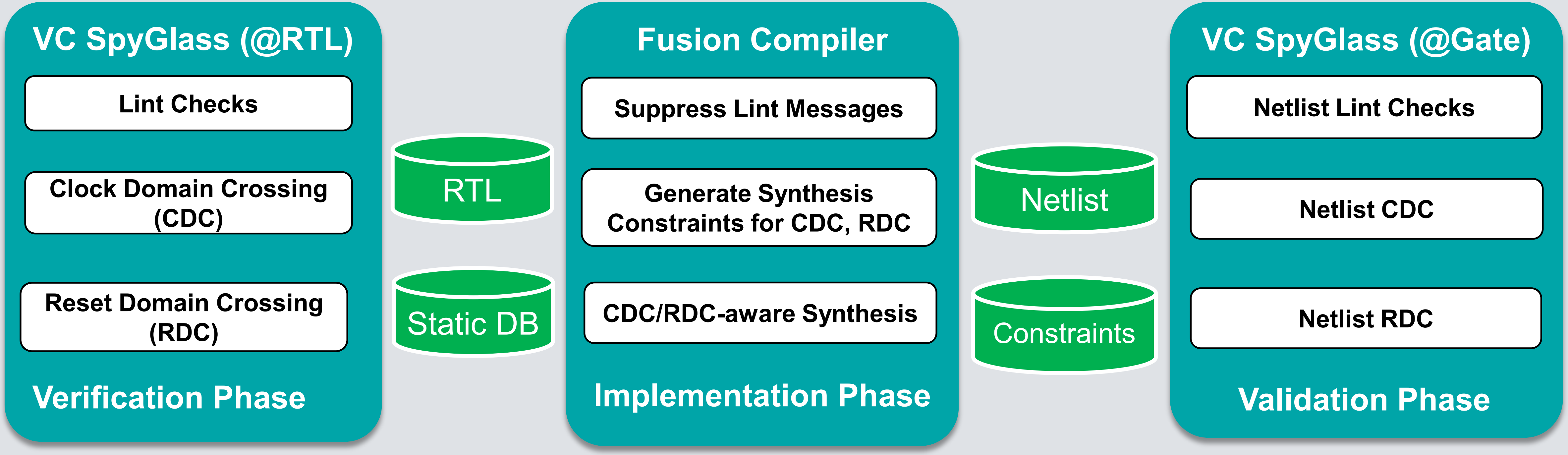
- Background:
  - Concerned of dynamic power reduction by modifying RTL design.
  - Wanted self-gating feature to improve total power reduction in FC.
  - Requested VC SpyGlass to make FC aware 100% CDC paths for self-gating exception

Static-Aware Synthesis



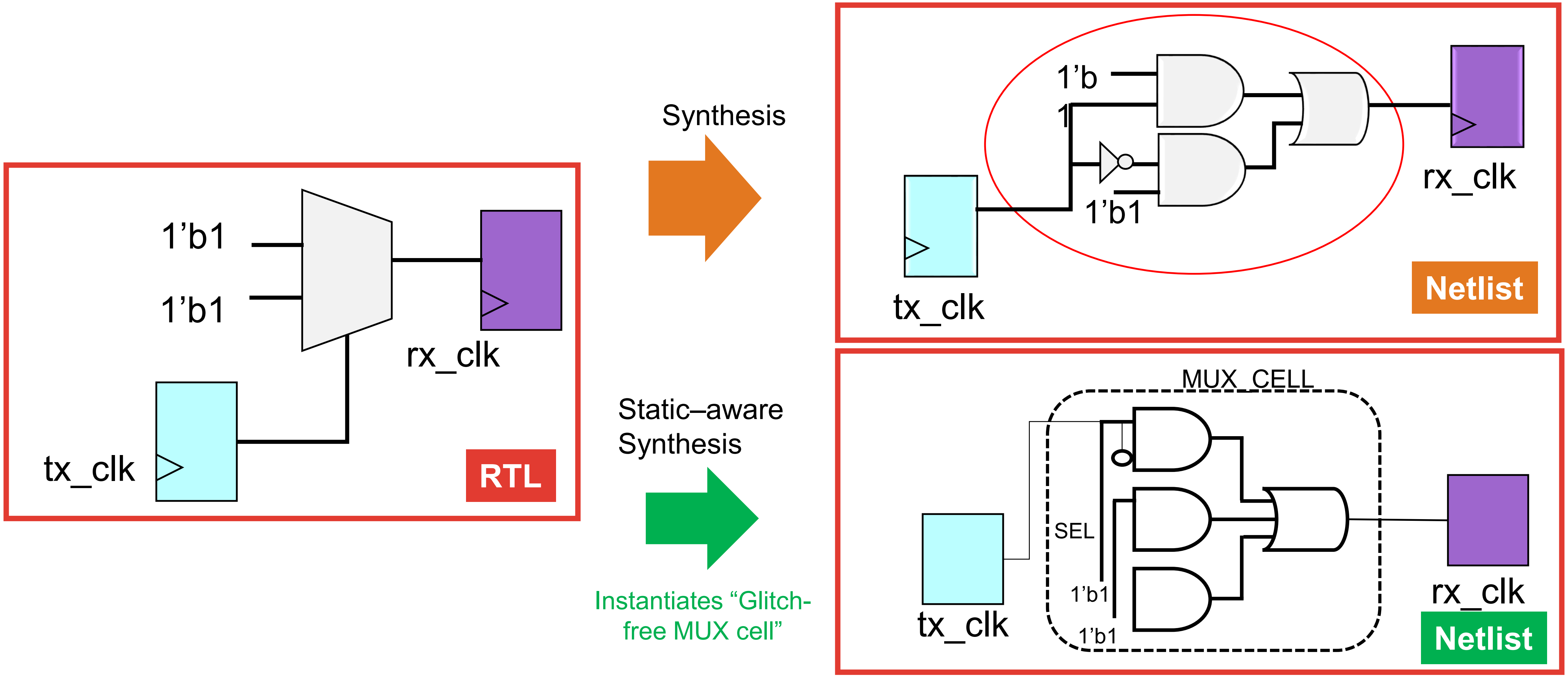
Static-Aware Synthesis Flow Overview

Ensure bug-free netlist transformation during backend stages & new logic introduction



- ✓ Preserves CDC signed-off CDC logic at synthesis level ensuring correct-by-construction netlist
- ✓ Reduces the CDC validation effort at gate-level

Static-aware Synthesis Flow Ensures Instantiation of Glitch-Free Mux'es



Synthesis

Static–aware Synthesis

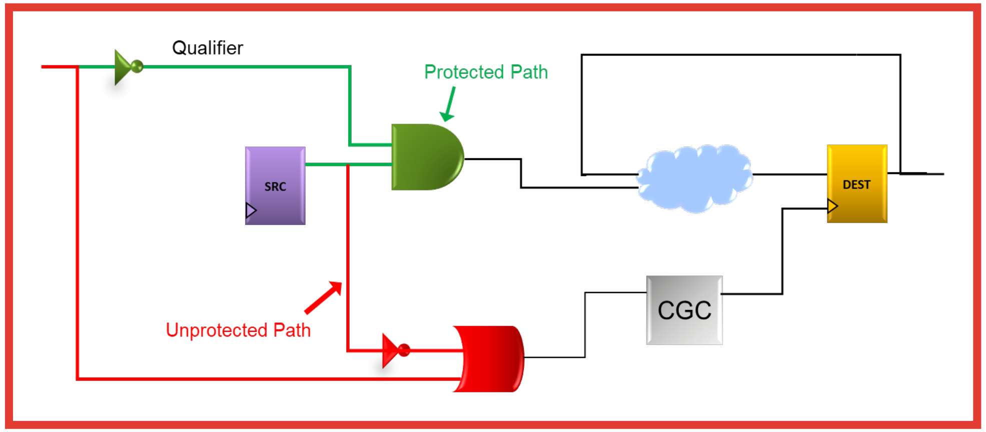
Instantiates "Glitch-free MUX cell"

Avoid Transformation of Pre-verified CDC Paths to Unsafe

Asynchronous signal moves to Clock Gating path

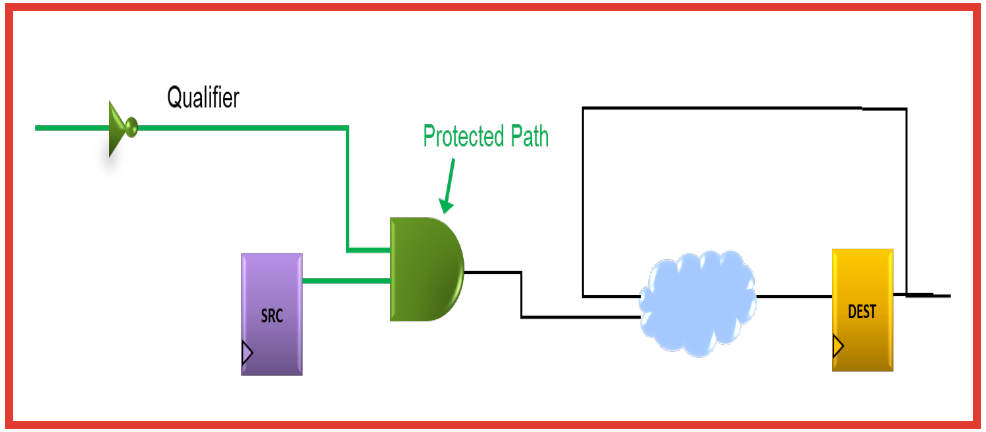
Regular Synthesis Flow

- Prequalified CDC path transforms into unsafe CDC path during synthesis
- Unprotected path introduced without blocking signal during synthesis



Static-aware Synthesis Flow

- Ensures no optimization in clock gating for asynchronous sources
- Avoids translation of pre-qualified CDC paths to unprotected CDC path

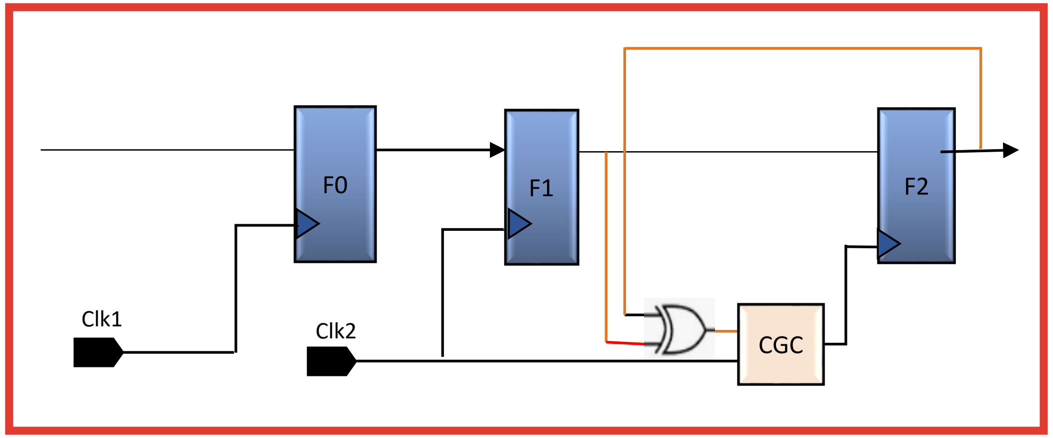


Ensure No Insertion of Clock Gating to Synchronized Paths

Unexpected self clock-gate insertion introduces new bugs

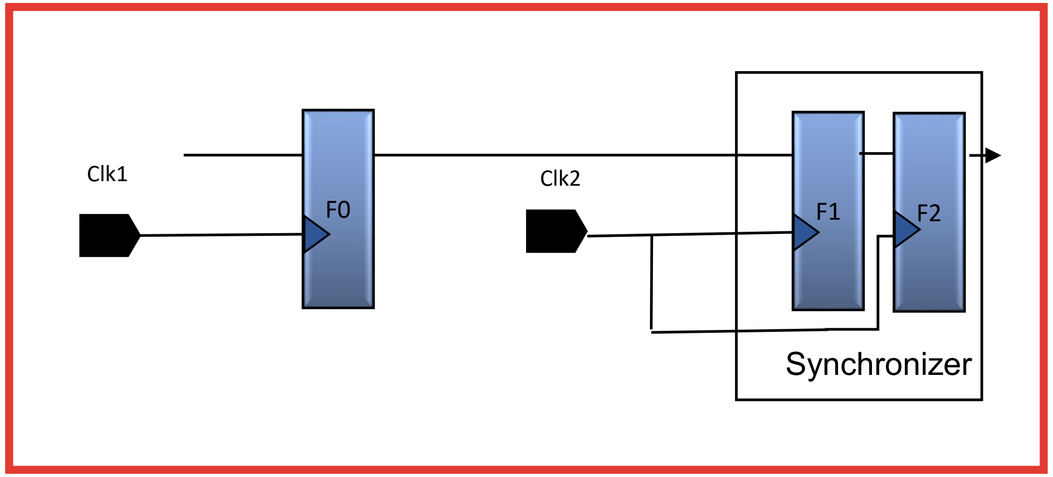
Regular Synthesis Flow

New multi-paths get introduced between destination & synchronizer

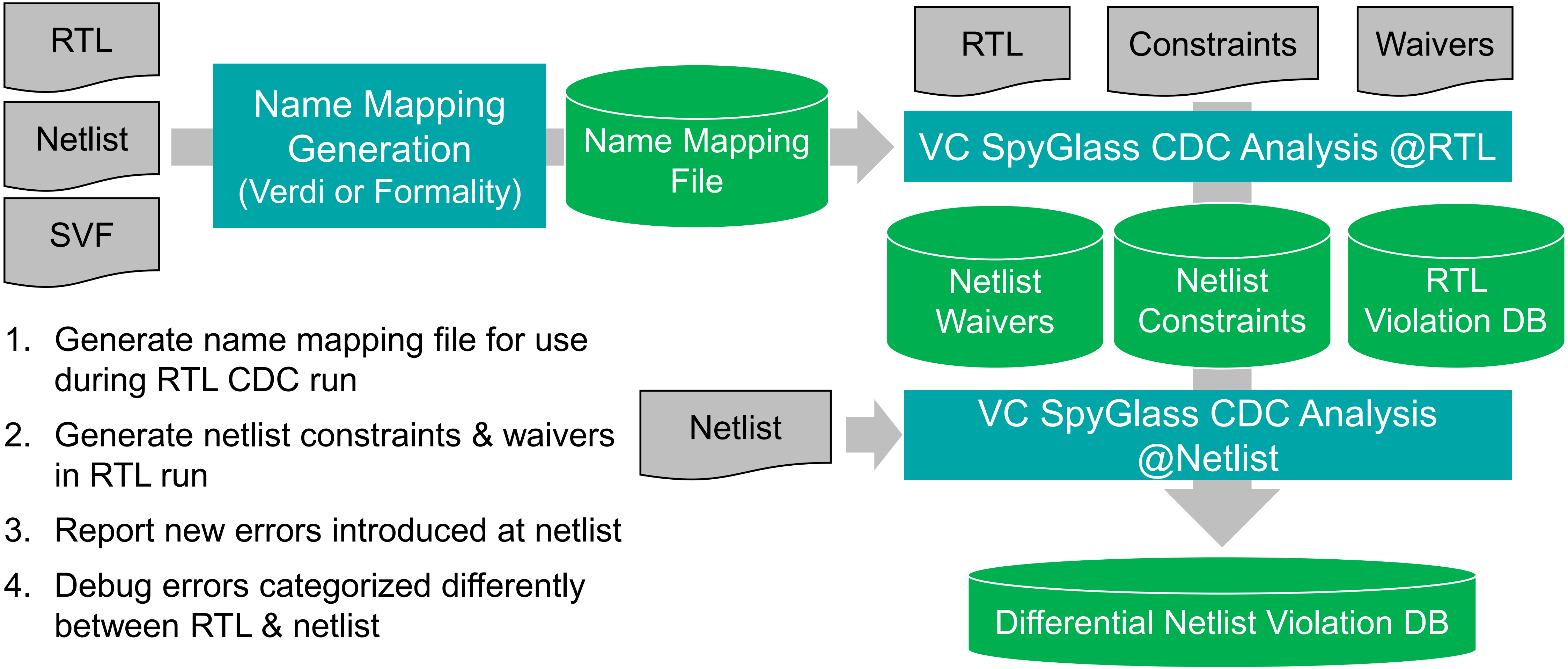


Static-aware Synthesis Flow

Ensure no CGC insertion happens within synchronizer



Accelerated Static Signoff Closure Using Smart Netlist Flow


RTL
Netlist
SVF
Name Mapping Generation (Verdi or Formality)
Name Mapping File
RTL
Constraints
Waivers
VC SpyGlass CDC Analysis @RTL
Netlist Waivers
Netlist Constraints
RTL Violation DB
Netlist
VC SpyGlass CDC Analysis @Netlist
Differential Netlist Violation DB

1. Generate name mapping file for use during RTL CDC run
2. Generate netlist constraints & waivers in RTL run
3. Report new errors introduced at netlist
4. Debug errors categorized differently between RTL & netlist

Gain 100% Confidence on CDC Assumptions & Protocols

Leverage Inbuilt formal and waveform replay engines within VC SpyGlass

Need for Functional CDC Signoff

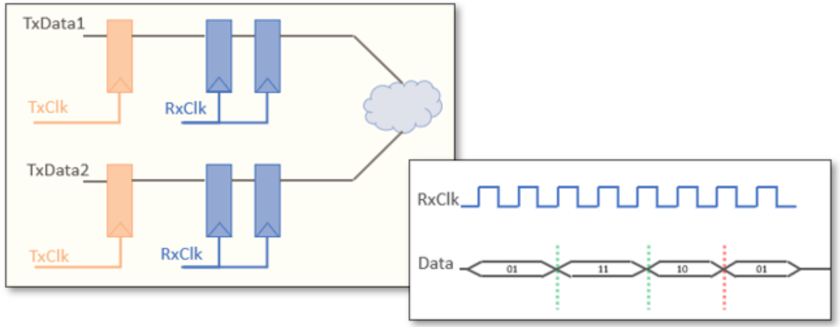
CDC constraints and protocols require validation

CDC Constraints

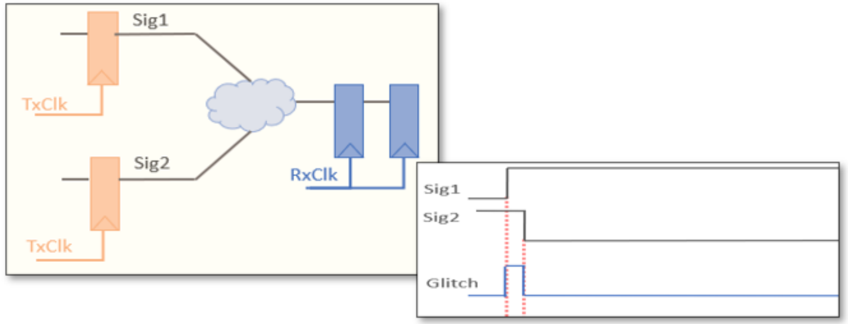
Constraint Name
create_clock create_generated_clock
create_reset
create_static
set_case_analysis
configure_cdc_convergence -ignore_among_signals
set_input_delay
set_cdc_ignore_path

CDC Protocols

Convergence: Verify that sources follow Gray Encoding protocol

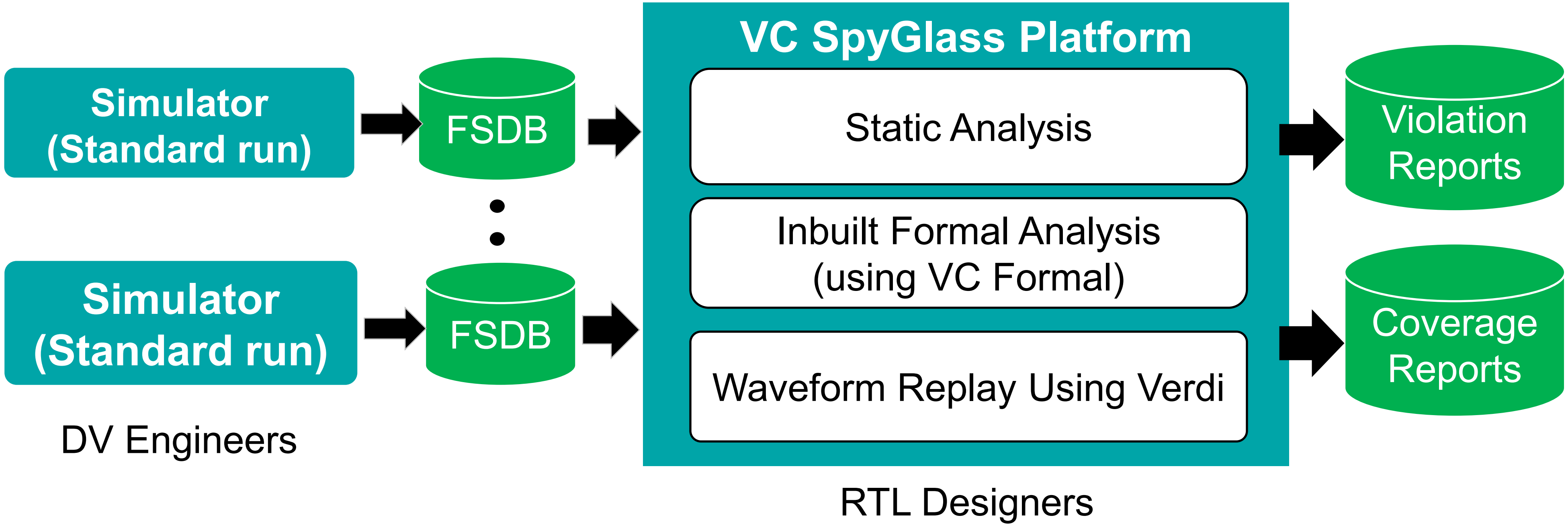


Multi-source Glitch: Multiple source converging at comb-gate can Glitch



Shift-Left & Catch Functional Bugs Seamlessly

Designers ensures complete functional CDC signoff upfront without relying on DV



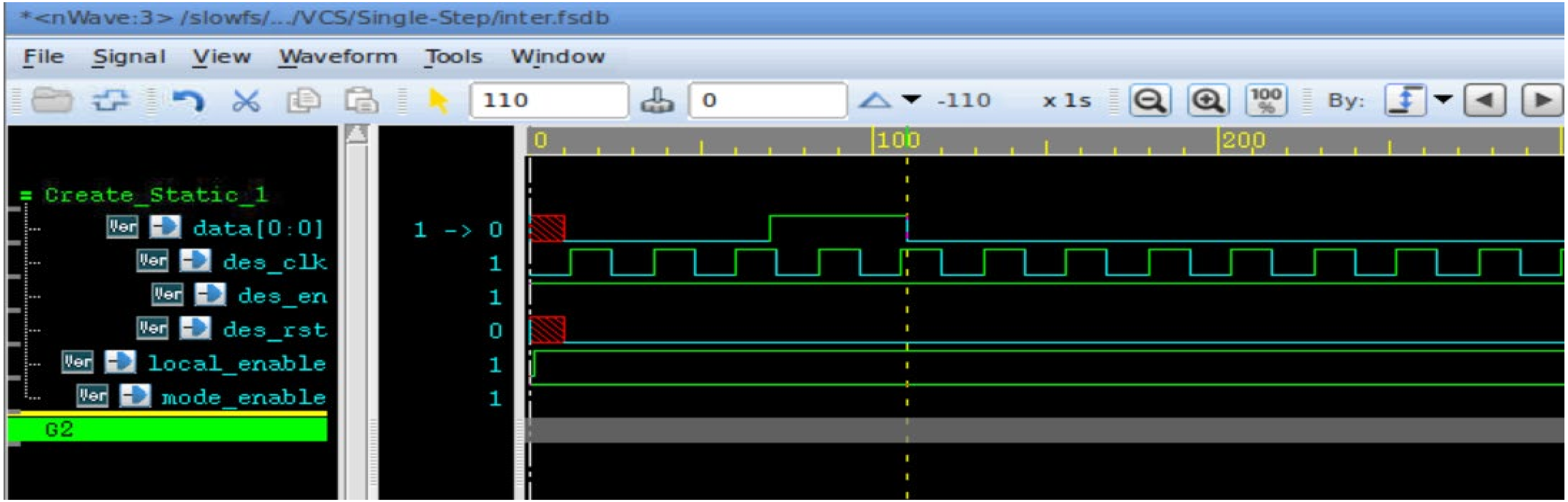
- Inbuilt formal & dynamic analysis with combined results reporting
- Enables parallel development of static & simulation environments
- No need to simulate assertions with DUT for each test

Design Scenario: Validation of Create_static Constraint

Simulator Message:-

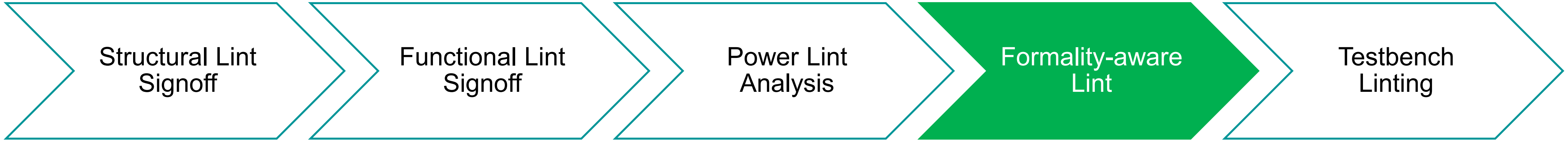
"/global/apps/vcstatic_2022.06-SP2/auxx/cdc/static_db/VCS/assumptions_definitions.sdb", 454: testbench_top.inst.i_Assumption_mod_chip_top.Create_Static_1.ADVCDC_DETECT_QS_TOGGLE: started at 110s failed at 156s

Create_static signal is toggling



Detect RTL Structures that Cause Long Formality Runtimes (Inconclusives)

VC SpyGlass Formality-aware Lint



Structural Lint Signoff

Functional Lint Signoff

Power Lint Analysis

Formality-aware Lint

Testbench Linting

Could be adopted in any order after Structural Lint closure

Is Design Functional Equivalence Verifiable?

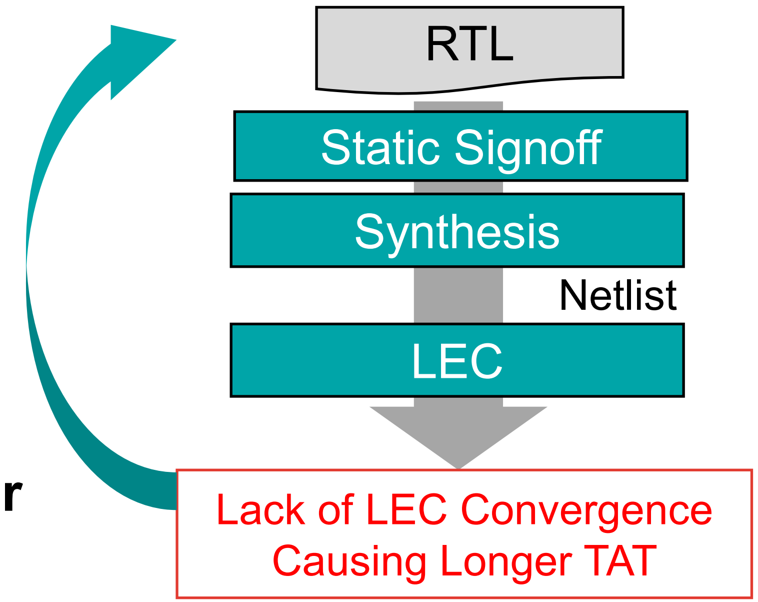
Challenge – Design is synthesizable but not verifiable

Before

RTL Designer

Iterative & hard process

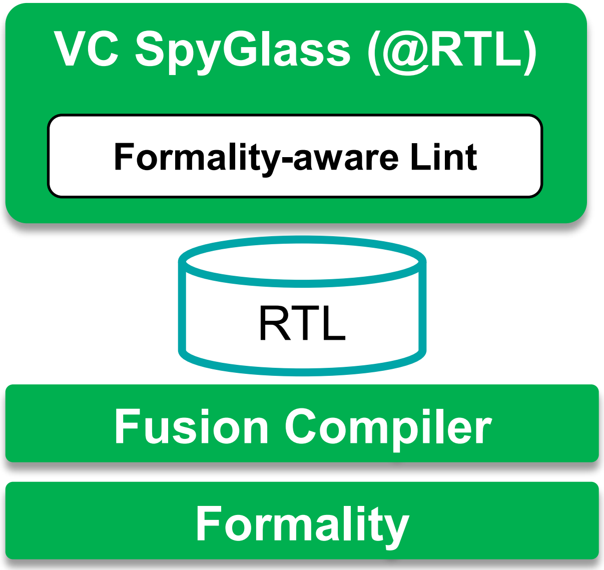
Impl. Engineer



Some RTL styles stress Equivalence Checking;
Resolving these RTL styles a.k.a *Hard Verification Points* may require re-synthesis

Detecting these points later during verification is TOO LATE
Huge runtime during logical equivalence checking run

After



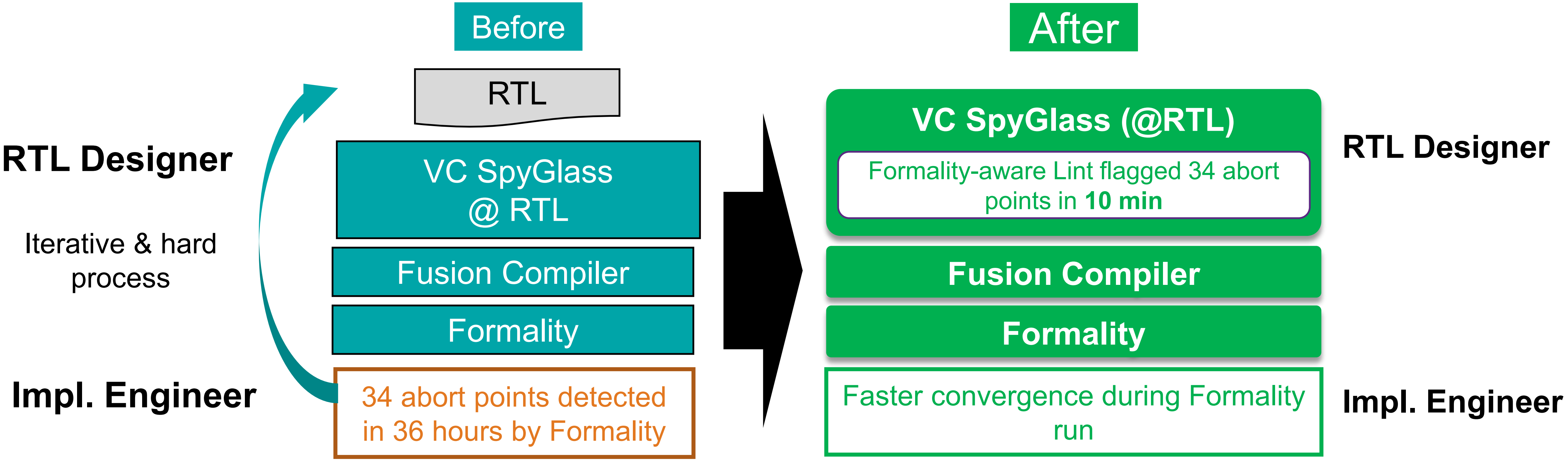
RTL Designer

Impl. Engineer

SHIFT-LEFT by early warning to designers about hard verification points and shorten LEC TAT

No iteration by early detection of hard-to-verify points; Designers get to ensure the design is verifiable upfront
May require re-coding RTL or update synthesis setup constraints

Abort Points Detected Using VC SG in Min. vs. Formality Run Later in Days

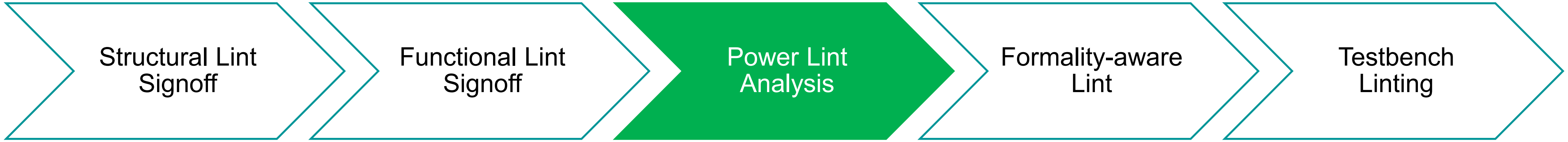


VC SpyGlass Highlighted 34 Abort Points in 10 min Earlier at RTL vs. 36 hrs Formality Run After Implementation

Formality Aware Lint Incorporated within RTL Signoff Checklist in Regression Usage Mode

Identify Opportunities to Reduce Power using Power-aware Lint

VC SpyGlass Power-aware Lint



Structural Lint Signoff

Functional Lint Signoff

Power Lint Analysis

Formality-aware Lint

Testbench Linting

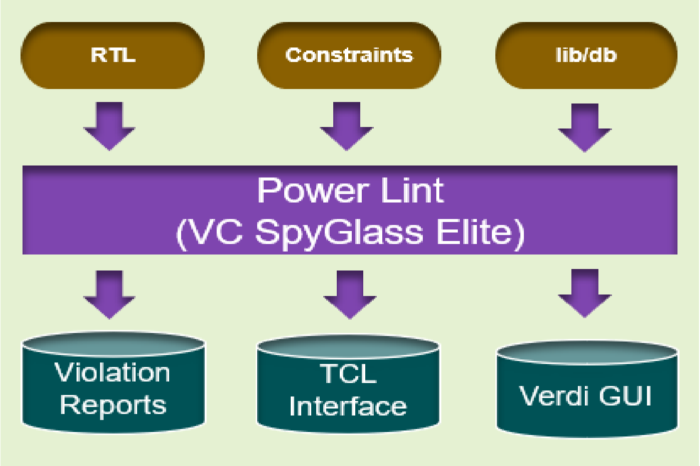
Could be adopted in any order after Structural Lint

Early Power Reduction Leveraging Power-aware Lint

Problem Statement

- EARLY guidance at RTL to accurately pinpoint coding styles resulting in high power consumption. Ex:-
  - Missing clock gating structures
  - Inefficient/Redundant clock gating
  - Structural inefficiency for clock gating
  - Connectivity/Optimization of clock gating logic
- Enables best design practices to avoid power-hungry logic
  - Use of SRAM instead of 2-3k bit flop arrays
  - Redundant Flop arrays
  - Unintended bus toggles

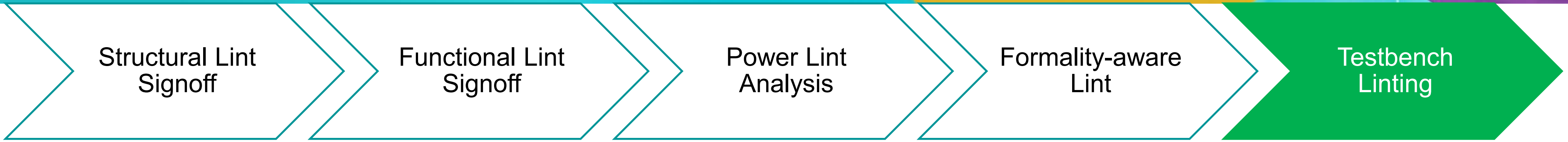
Solution: Early Power Linting



- Expand Regular Lint run with Power Lint (with or without SDC)
- 27 rules enabled via Power Lint with 2023.03-SP2-3
- Applicable for both IP & SoC level RTL signoff

Testbench Consuming More Simulation Time, Need to Improve TB Quality

Euclide TB Linting – Batch Mode



Could be adopted in any order after Structural Lint

Need for Testbench Linting

Testbench code is different than RTL

- Verification testbenches contain Classes
- Synthesizable RTL Lint handled excellently by VC SpyGlass
- No testbench linting or correct-by-construction checking for Class based code or UVM
- Engineers run into many iterative testbench issues at compile, simulation run or debug

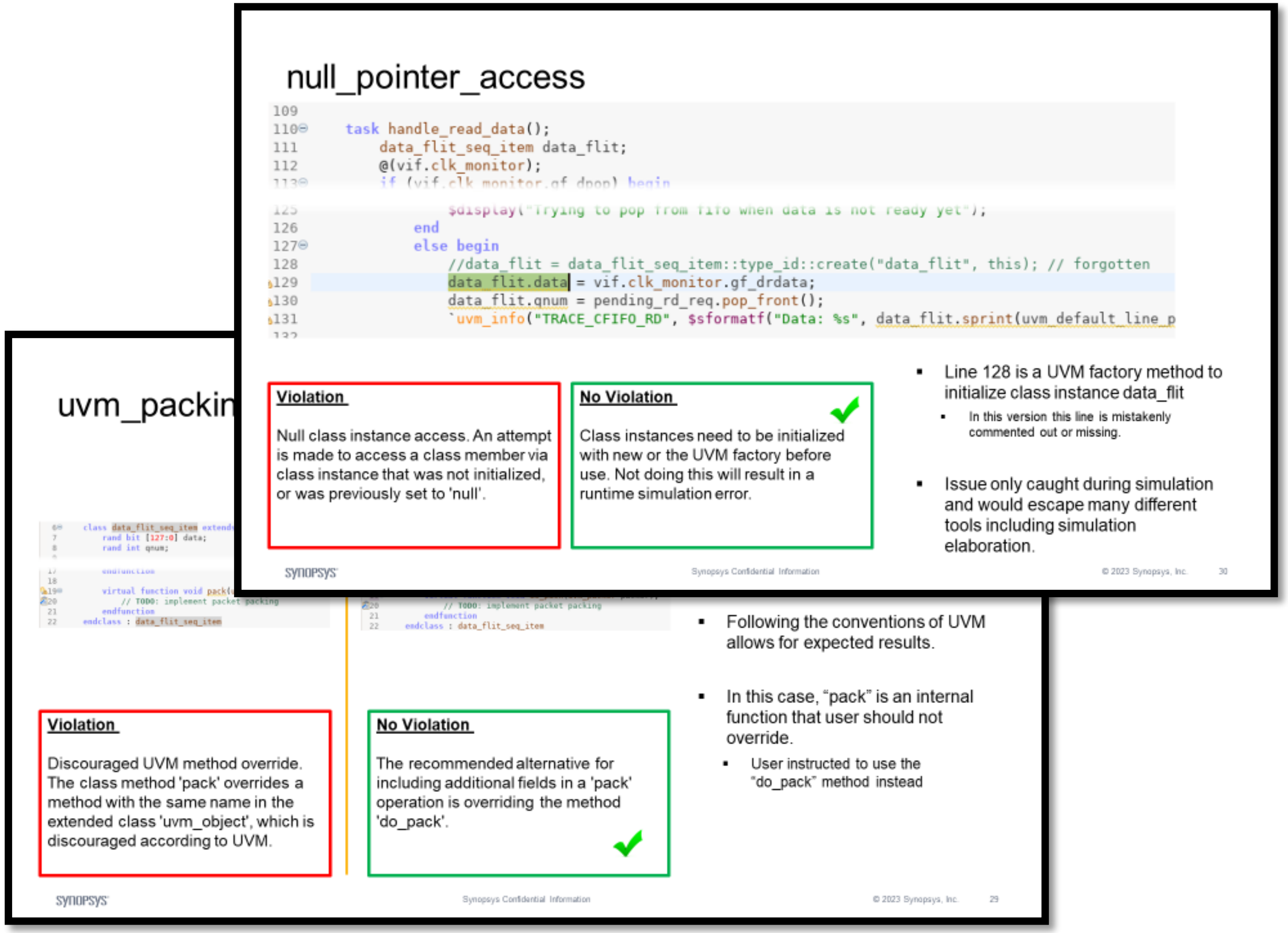
Synopsys Euclide Enables Testbench Linting During System Verilog and UVM Code Development

Testbench Linting Using Euclide

Testbench Linting in Batch mode applicable for Verification Engineers (or VCS users)

- Similar to RTL Linting for Designers, Verification Engineers also need early guidance on Testbench quality

- Engineers get early reports for addressing TB/UVM issues without waiting for entire design to compile



null_pointer_access

```
109
110    task handle_read_data();
111        data_flit_seq_item data_flit;
112        @(vif.clk_monitor);
113        if (vif.clk_monitor.gf_dpop) begin
125                $display("Trying to pop from fifo when data is not ready yet");
126            end
127            else begin
128                //data_flit = data_flit_seq_item::type_id::create("data_flit", this); // forgotten
129                data_flit.data = vif.clk_monitor.gf_drdata;
130                data_flit.qnum = pending_rd_req.pop_front();
131                `uvm_info("TRACE_CFIFO_RD", $sformatf("Data: %s", data_flit.sprint(uvm_default_line_p
132
```

Violation

Null class instance access. An attempt is made to access a class member via class instance that was not initialized, or was previously set to 'null'.

No Violation ✓

Class instances need to be initialized with new or the UVM factory before use. Not doing this will result in a runtime simulation error.

- Line 128 is a UVM factory method to initialize class instance data_flit
  - In this version this line is mistakenly commented out or missing.

- Issue only caught during simulation and would escape many different tools including simulation elaboration.

SYNOPSYS Synopsys Confidential Information © 2023 Synopsys, Inc. 30

uvm_packin

```
6    class data_flit_seq_item extend
7        rand bit [127:0] data;
8        rand int qnum;
17       endfunction
18
19       virtual function void pack(
20           // TODO: implement packet packing
21       endfunction
22   endclass : data_flit_seq_item
```

```
20           // TODO: implement packet packing
21       endfunction
22   endclass : data_flit_seq_item
```

Violation

Discouraged UVM method override. The class method 'pack' overrides a method with the same name in the extended class 'uvm_object', which is discouraged according to UVM.

No Violation ✓

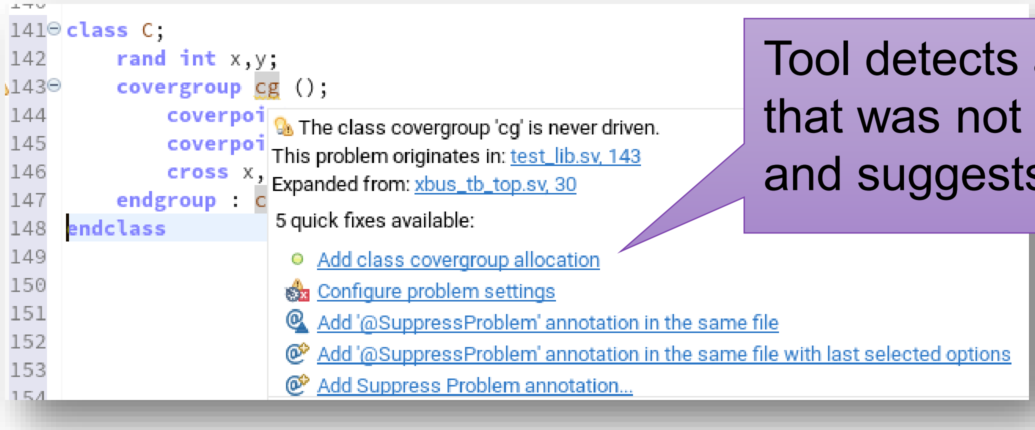
The recommended alternative for including additional fields in a 'pack' operation is overriding the method 'do_pack'.

- Following the conventions of UVM allows for expected results.

- In this case, "pack" is an internal function that user should not override.
  - User instructed to use the "do_pack" method instead

SYNOPSYS Synopsys Confidential Information © 2023 Synopsys, Inc. 29

Testbench Checks Examples



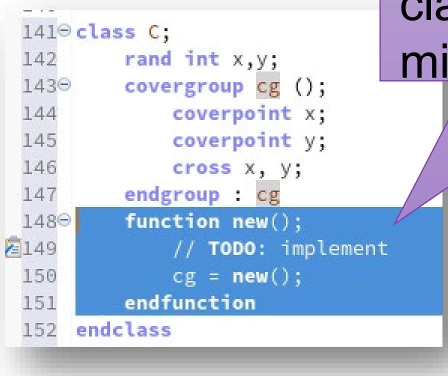
```
class C;
    rand int x,y;
    covergroup cg ();
        coverpoi
        coverpoi
        cross x,
    endgroup : c
endclass
```

The class covergroup 'cg' is never driven.
This problem originates in: test_lib.sv, 143
Expanded from: xbus_tb_top.sv, 30

5 quick fixes available:

- Add class covergroup allocation
- Configure problem settings
- Add '@SuppressProblem' annotation in the same file
- Add '@SuppressProblem' annotation in the same file with last selected options
- Add Suppress Problem annotation...

Tool detects a covergroup that was not constructed and suggests a quick fix

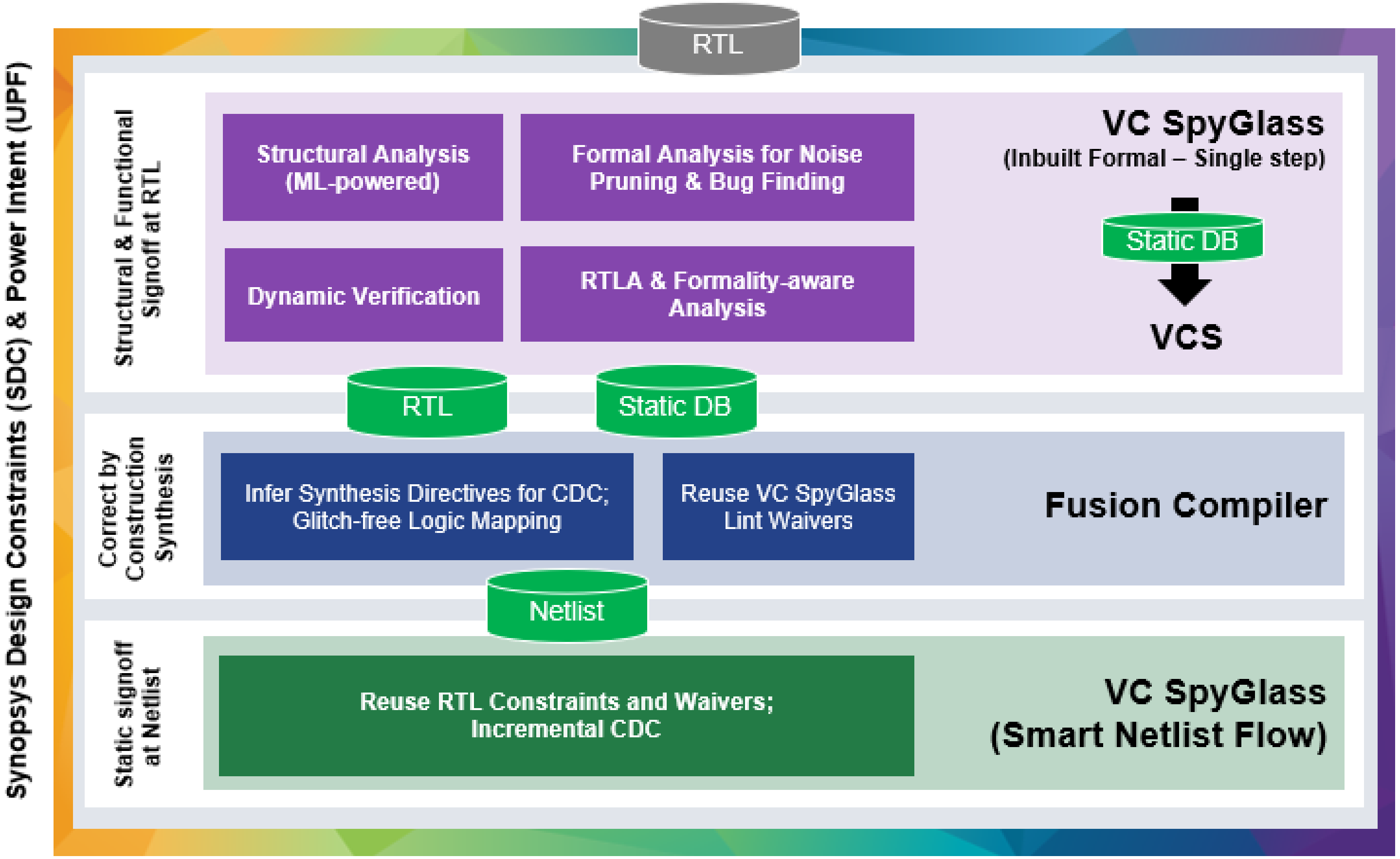


```
class C;
    rand int x,y;
    covergroup cg ();
        coverpoint x;
        coverpoint y;
        cross x, y;
    endgroup : cg
    function new();
        // TODO: implement
        cg = new();
    endfunction
endclass
```

Quick fix constructs the covergroup and also generates the class constructor if it is missing

Key Takeaways

- Glitch Verification & Functional CDC signoff is critical to avoid silicon failure
- Implementation Design Checks enable easy root cause of synthesis optimized registers
- Formality-aware Lint, Power Linting enable Advanced SHIFT-LEFT
- Static-aware synthesis followed by Smart netlist flow ENABLES end-2-end static signoff


Synopsys Design Constraints (SDC) & Power Intent (UPF)
RTL
Structural & Functional Signoff at RTL
Structural Analysis (ML-powered)
Formal Analysis for Noise Pruning & Bug Finding
Dynamic Verification
RTLA & Formality-aware Analysis
VC SpyGlass
(Inbuilt Formal – Single step)
Static DB
VCS
RTL
Static DB
Correct by Construction Synthesis
Infer Synthesis Directives for CDC; Glitch-free Logic Mapping
Reuse VC SpyGlass Lint Waivers
Fusion Compiler
Netlist
Static signoff at Netlist
Reuse RTL Constraints and Waivers; Incremental CDC
VC SpyGlass
(Smart Netlist Flow)

THANK YOU

YOUR
INNOVATION
YOUR
COMMUNITY