Achieve Faster Root Cause Analysis of Synthesis-Optimized Registers early at RTL

Harish Aepala - Meta Platforms
Anshul Bansal - Meta Platforms
Suresh Babu Barla - Synopsys Inc
Bhavana Mallikarjunaiah - Synopsys Inc
Himanshu Kathuria - Synopsys Inc

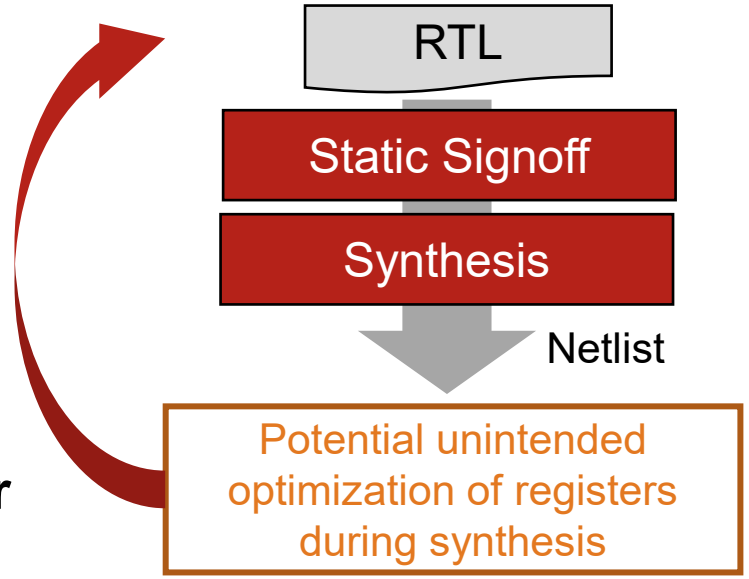
Agenda

- Introduction
- Implementation Design Checks (IDC) Early Analysis
  - Generate Optimized Register List by enabling Low Effort Implementation Flow
- Implementation Design Checks (IDC) Sign-off Analysis
  - Feed Optimized Registers List from High Effort Implementation Flow
- Results
- Summary

Design Readiness for Implementation

Traditional Flow Challenges

Traditional



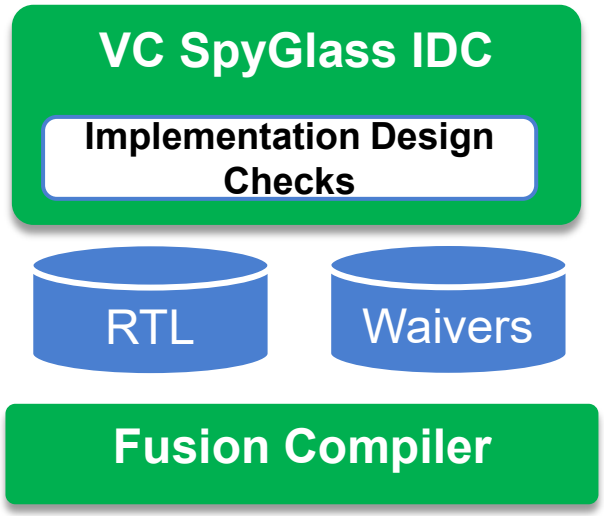
Manual debug with limited information to find root causes
&
Iterative process

Impl. Engineer

Finding root-cause for **100k-500k** range of registers obvious
Higher TAT months for root cause analysis

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

Implemented



RTL Designer

Impl. Engineer

SHIFT-LEFT and easy root cause analysis of register optimized during synthesis. Shorter TAT from months to days

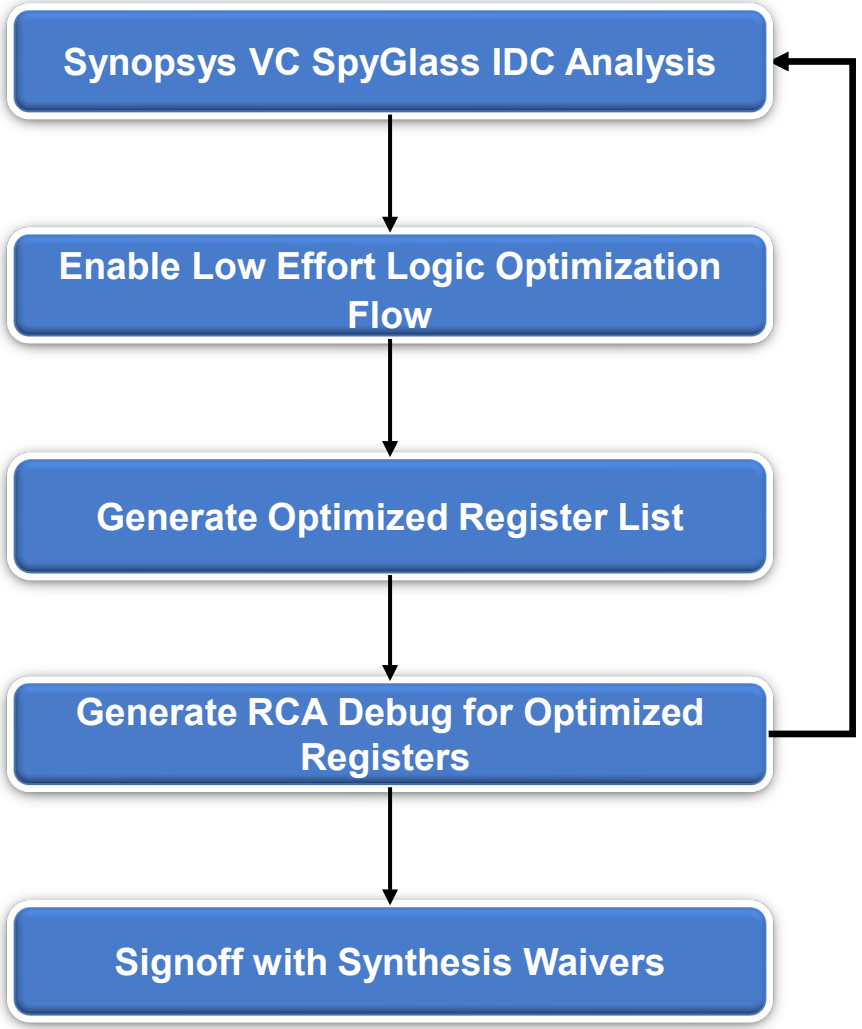
No iteration due to unintentional register optimizations due to constant propagation or unloaded registers

Implementation Design Checks (IDC) Early Analysis

Generate Optimized Register List by Enabling Low Effort Implementation Flow

Implementation Design Checks (IDC) Early Analysis

Generate Optimized Register List by enabling Low Effort Implementation Flow



- Create Synopsys VC SpyGlass IDC Setup
- Enable low effort implementation tool flow
- Generated optimized register list due to constant propagation and unused register output
- Debug via tool generated root cause for optimized registers
- Fix RTL if needed, RTL Signoff along with Synthesis waivers

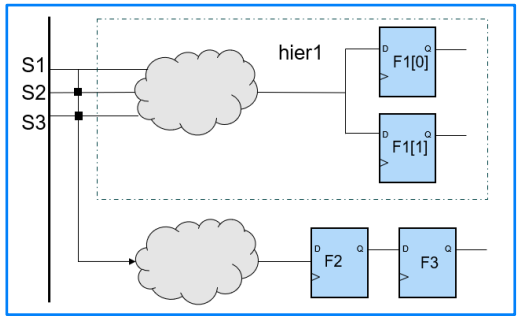
Implementation Design Checks Overview

- Following apps are developed in collaboration
  - ***DetectOptimizedConstRegister*** – Detects constant registers that get optimized during synthesis
  - ***DetectOptimizedUnUsedRegister*** – Detects unloaded registers that get optimized during synthesis
- Report mechanism
  - Report generated based on uniquified constant and non-constant sources
  - Optimized Registers can be bus merged through an option

Run script for IDC Early Analysis

```
1  # Enabling platform fpr running VC Spyglass Lint
2  set_app_var enable_lint true
3
4  # Enabling Implementation Aware flow
5  set_app_var lint_impl_mode true
6
7  # Configuring tags and parameters
8  configure_lint_tag -enable -tag "DetectOptimizedConstRegister" -goal test_goal
9  configure_lint_tag -enable -tag "DetectOptimizedUnUsedRegister" -goal test_goal
10 lint_disable_bus_merge -tag DetectOptimizedConstRegister
11 lint_disable_bus_merge -tag DetectOptimizedUnUsedRegister
12 configure_lint_setup -goal test_goal
13
14 # Design read
15 analyze -format sverilog { -f <filelist> }
16 elaborate <top>
17
18
19 # Running checks in VC Lint
20 check_lint
21
22 # Report Generation
23 report_violations
24 report_constant_reg_srcs -file <file name>
25
26 exit
```

Bus merging can be enabled /disabled using options

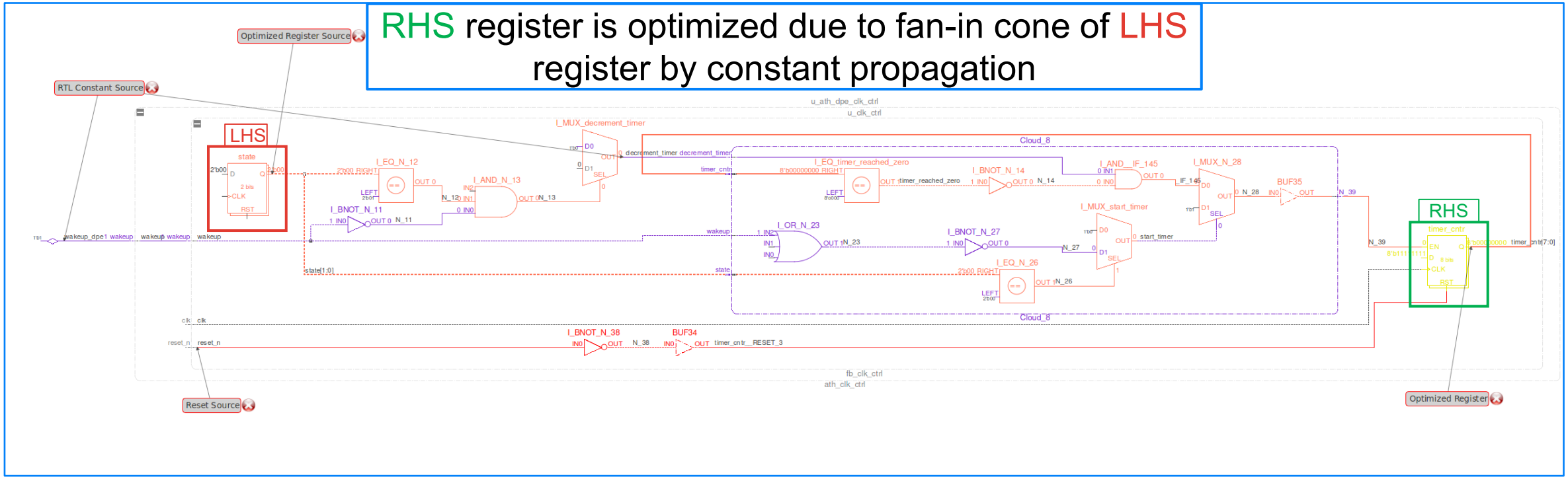


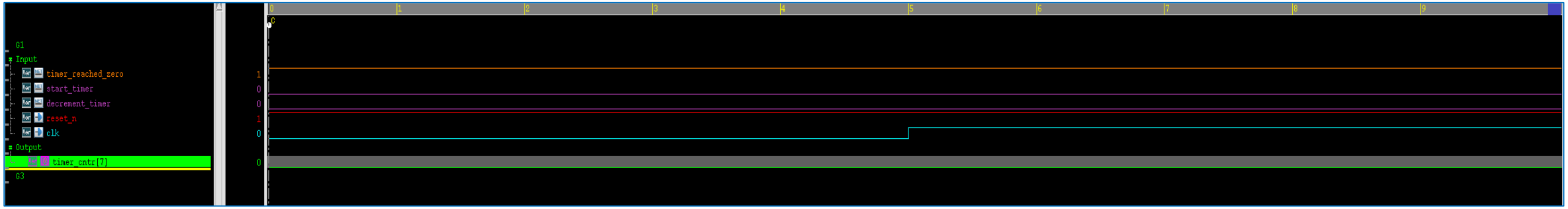
Bus Merging

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

Optimized Register Due to a Constant Source

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

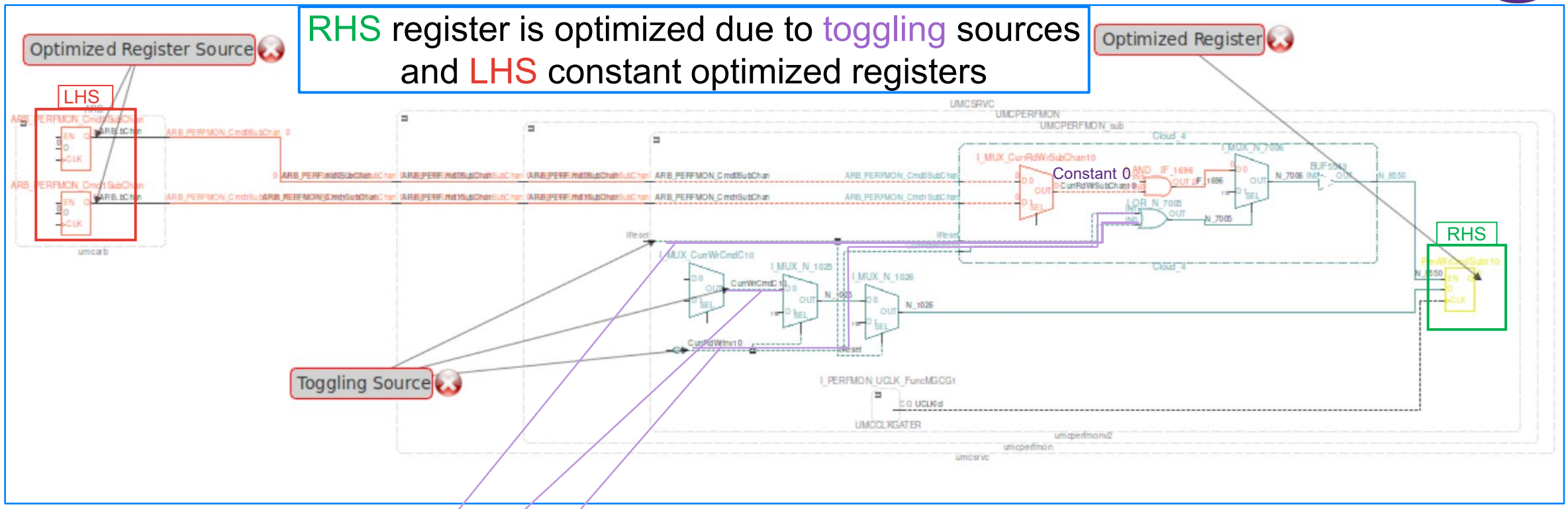


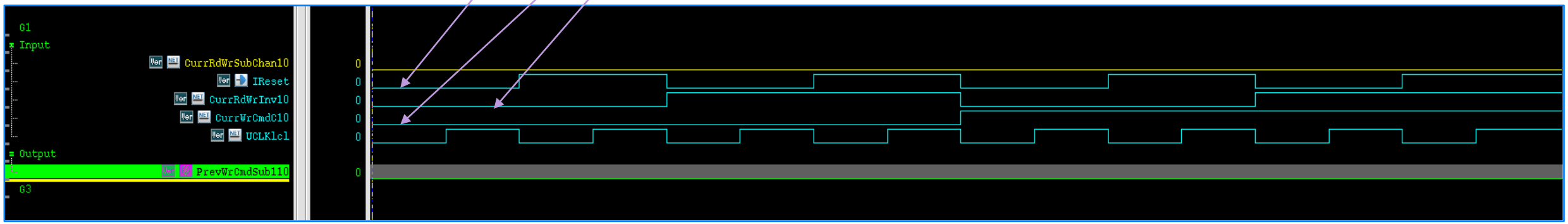


Waveform Witness

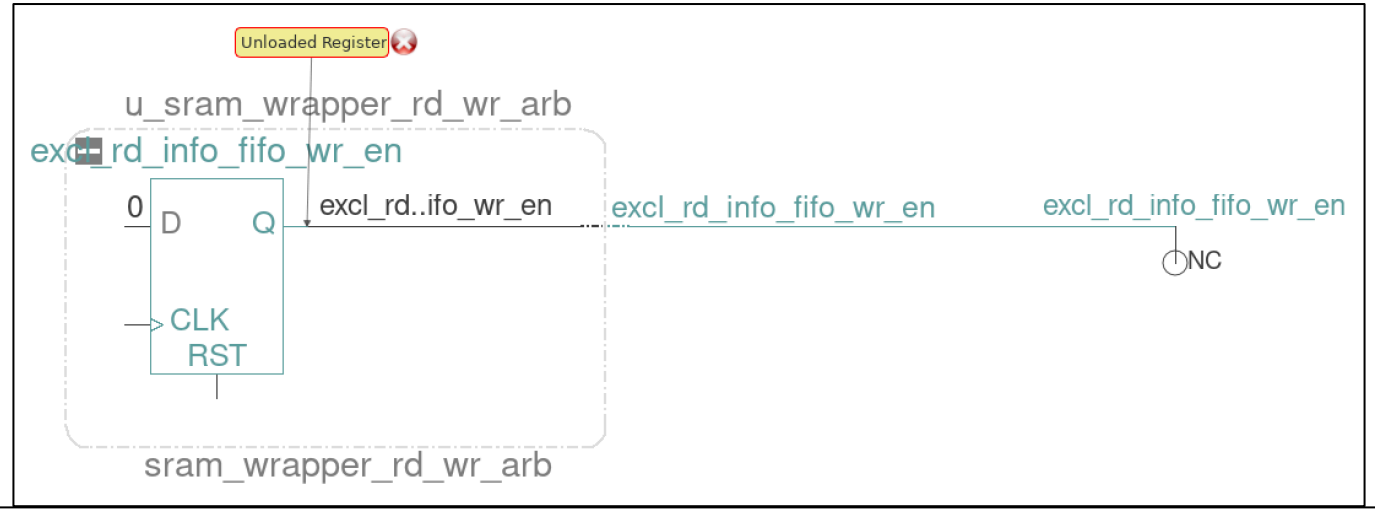
Optimized Register Due to Toggling Sources

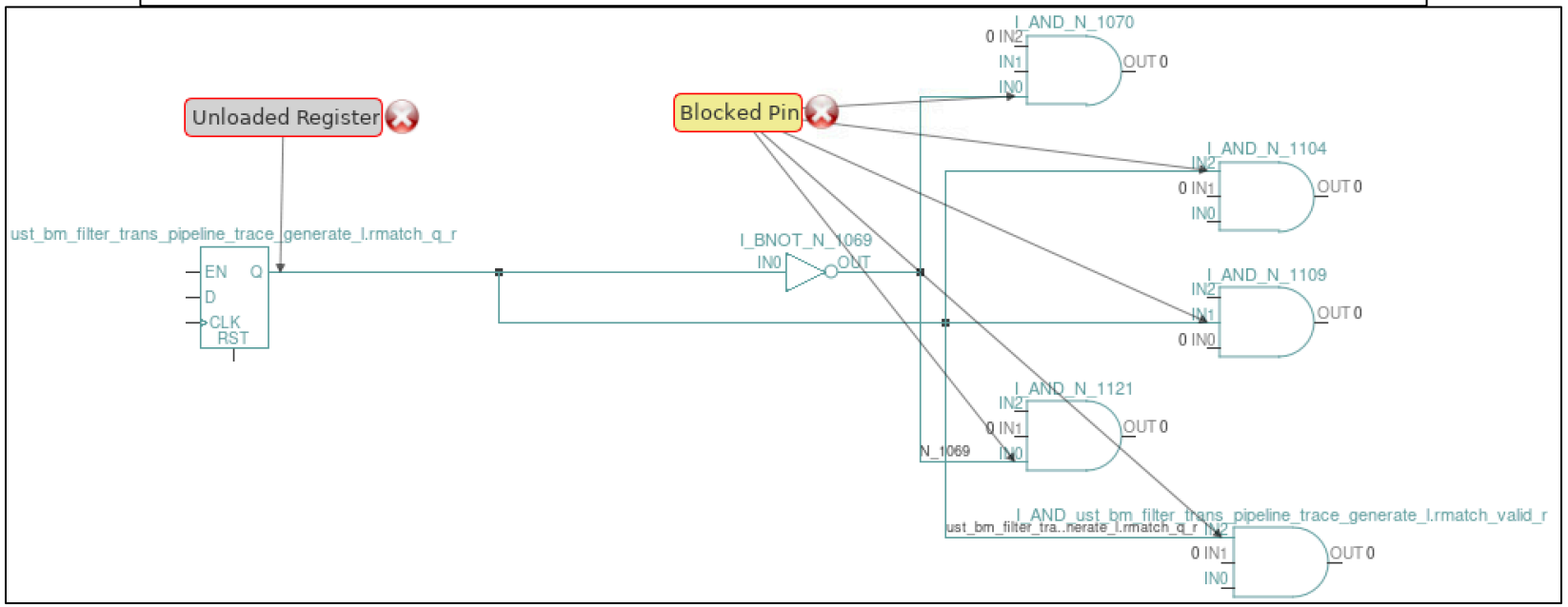
RHS register is optimized due to toggling sources and LHS constant optimized registers





Optimized Registers Due to Unused Registers





Optimized Registers Report

Constant Register Sources and Optimization Type

```
28 ------------------------------------------------------------------------------------------------------------------------------------------------------
29 OPTIMIZED REGS                                          OPTIMIZATION TYPE      SOURCES                                                  SOURCE TYPE
30 ------------------------------------------------------------------------------------------------------------------------------------------------------
31 a_cache.tags[0].single_use                              Direct Constant        a_fetch.cache_wtag.single_use                            Design Constant
32 a_cache.tags[1].single_use
33
```

```
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
```

FC Optimized register list – Summary

Constant 0 , Constant 1 and Unloaded

```
 85 a_cache/tags_reg_0__single_use                    C0r
 86 a_cache/tags_reg_1__single_use                    C0r
```

```
163 b_cache/flopstage_masked_rdata3/out_reg_529_      C0r
164 b_cache/flopstage_masked_rdata3_c1/out_reg_529_   C0r
165 b_cache/flopstage_masked_rdata3_c2/out_reg_529_   C0r
166 b_cache/flopstage_masked_rdata3_c3/out_reg_529_   C0r
```

```
 33 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_7__9_ C0r
 34 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_6__9_ C0r
 35 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_5__9_ C0r
 36 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_4__9_ C0r
 37 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_3__9_ C0r
 38 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_2__9_ C0r
 39 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_1__9_ C0r
 40 genblk2_0__ls_rreq_metadata/fifo/flop_based_data_reg_0__9_ C0r
 41 genblk2_0__ls_rreq_metadata/fifo/out_reg_9_       C0r
 42 genblk2_1__ls_rreq_metadata/fifo/flop_based_data_reg_11__9_ C0r
 43 genblk2_1__ls_rreq_metadata/fifo/flop_based_data_reg_10__9_ C0r
 44 genblk2_1__ls_rreq_metadata/fifo/flop_based_data_reg_9__9_ C0r
 45 genblk2_1__ls_rreq_metadata/fifo/flop_based_data_reg_8__9_ C0r
```

Register	Count
Constant Registers Deleted	200
Constant Registers Preserved	0
Unloaded Registers Deleted	1636
Unloaded Registers Preserved	0
Shift Registers	0
Merged	0
Multibit	0
Inverted	0
Replicated	0
Retimed	0

QoR Analysis of Synopsys VC SpyGlass IDC vs Synopsys Fusion Compiler™ Flows

- **Shift-left** flow : QoR of Synopsys VC SpyGlass IDC vs Synopsys FC :
  - Constant register – **99.99%**
  - Unloaded register – **96.77%**

- In **shift-left** flow (Low effort implementation flow), Synopsys VC SpyGlass IDC maybe unable to generate RCA for some registers (< 5%) optimized in Synopsys FC

- In **sign-off** flow (High effort implementation flow), Synopsys VC SpyGlass IDC would generate RCA for those ~5% registers along with correlation reports

QoR Summary

```
### Const Registers ###
QOR_VC_const_registers = 121780
QOR_FC_const_registers = 121447
QOR_const_registers_matching = 121446
QOR_const_registers_FC_only = 1
QOR_const_registers_VC_only = 334
```

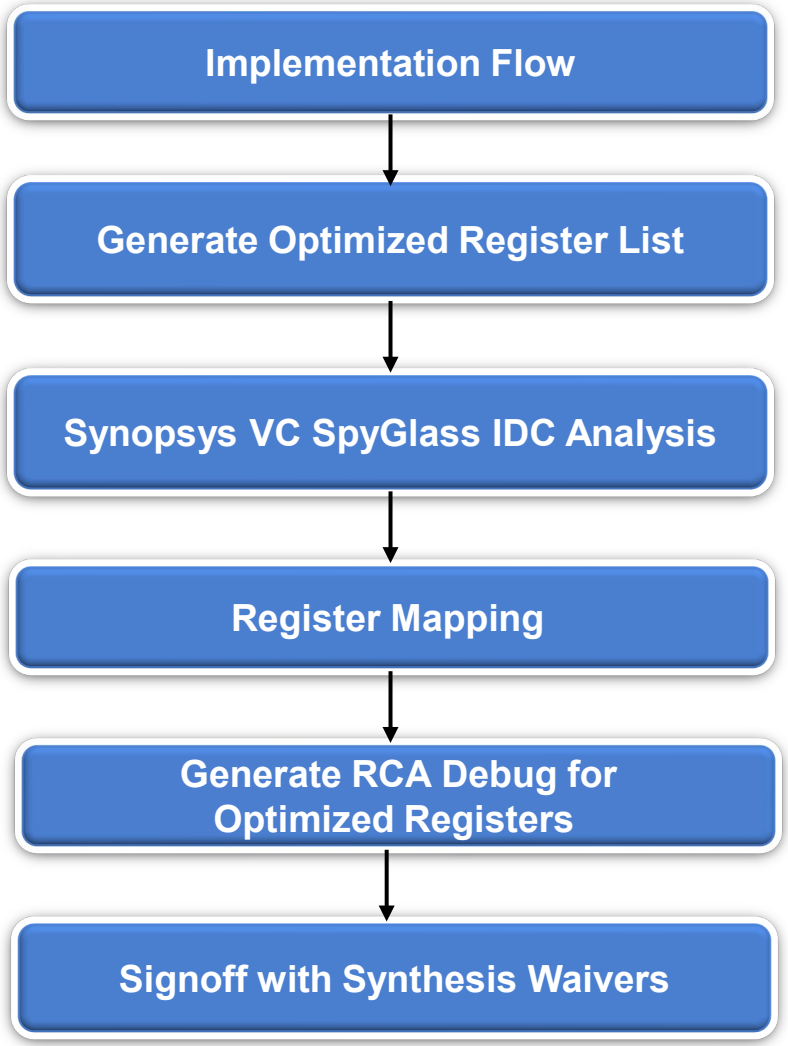
```
### Unused Registers ###
QOR_VC_unused_registers = 51548
QOR_FC_unused_registers = 53230
QOR_unused_registers_matching = 51514
QOR_unused_registers_FC_only = 1716
QOR_unused_registers_VC_only = 34
```

Implementation Design Checks Sign-off Flow

Feed Optimized Registers List from High Effort Implementation Flow

Implementation Design Checks Sign-off Flow

Feed Optimized Registers List from High Effort Implementation Flow



- Run Implementation Flow
- Feed optimized register list from actual implementation flow
- Run Synopsys VC SpyGlass IDC Analysis
- Map registers from Implementation flow to RTL
- Debug via tool generated RCA for optimized registers
- Fix RTL if needed, Sign-off along with synthesis waivers

IDC Signoff Flow Scripts and Reports

Run Script for IDC Sign-off Flow

```
# Enabling platform fpr running VC Spyglass Lint
set_app_var enable_lint true

# Enabling Implementation Aware flow
set_app_var lint_impl_mode true

# Providing constant register file from RTLA or Fusion Compiler as input
comfigure_lint_impl_setup -registerFile <path to constant register csv>

# Configuring tags and parameters
configure_lint_tag -enable -tag "DetectOptimizedConstRegister" -goal test_goal
configure_lint_tag -enable -tag "DetectOptimizedUnUsedRegister" -goal test_goal
lint_disable_bus_merge -tag DetectOptimizedConstRegister
lint_disable_bus_merge -tag DetectOptimizedUnUsedRegister
configure_lint_setup -goal test_goal

# Design read
analyze -format sverilog { -f <filelist> }
elaborate <top>


# Running checks in VC Lint
check_lint

# Report Generation
report_violations
report_constant_reg_srcs -file <file name>

exit
```

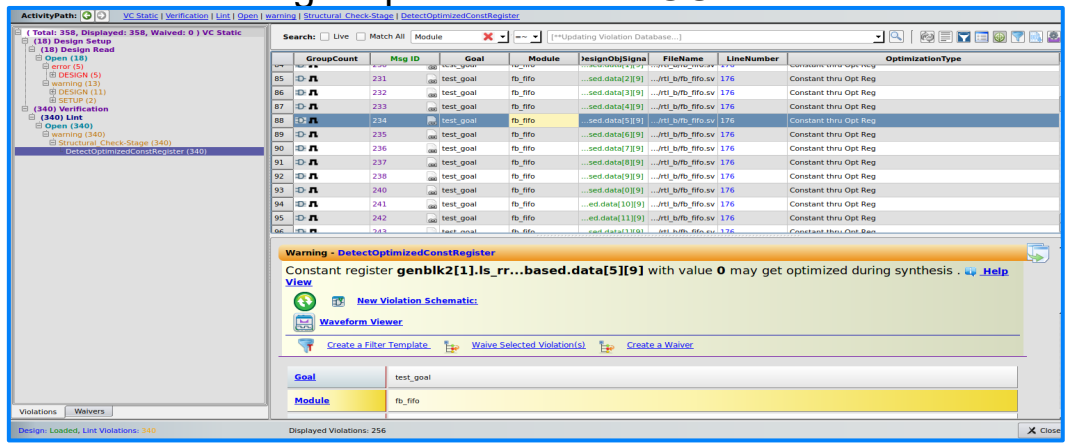
Correlation reports

```
Correlation Mapping Statistic
-----------------------------

Total Registers:              340
Correlated Registers:         340 (100%)
Uncorrelated Registers:         0 (0%)
```

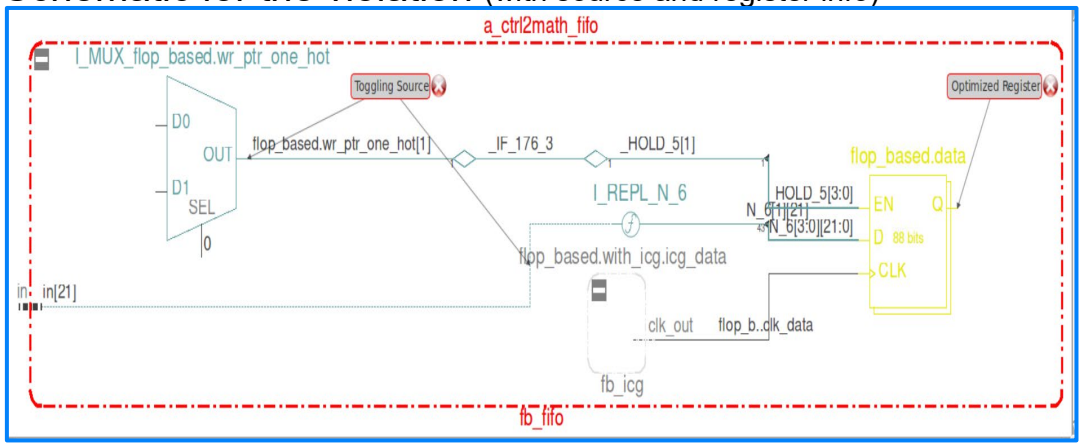
```
Correlated Mapping
------------------

Netlist register => RTL register

a_cache/tags_reg[0][single_use] => a_cache.tags[0].single_use
a_cache/tags_reg[1][single_use] => a_cache.tags[1].single_use
a_fetch_ctrl/a_ctrl2math_fifo/flop_based.data_reg[0][21] => a_fetch_ctrl.a_ctrl2math_fifo.flop_based.data[0][21]
a_fetch_ctrl/a_ctrl2math_fifo/flop_based.data_reg[1][21] => a_fetch_ctrl.a_ctrl2math_fifo.flop_based.data[1][21]
a_fetch_ctrl/a_ctrl2math_fifo/flop_based.data_reg[2][21] => a_fetch_ctrl.a_ctrl2math_fifo.flop_based.data[2][21]
a_fetch_ctrl/a_ctrl2math_fifo/flop_based.data_reg[3][21] => a_fetch_ctrl.a_ctrl2math_fifo.flop_based.data[3][21]
a_fetch_ctrl/a_ctrl2math_fifo/out_reg[21] => a_fetch_ctrl.a_ctrl2math_fifo.out[21]
b_cache/cache_replacement_pointer_reg[0][7] => b_cache.cache_replacement_pointer[0][7]
b_cache/cache_replacement_pointer_reg[1][7] => b_cache.cache_replacement_pointer[1][7]
b_cache/cache_replacement_pointer_reg[2][7] => b_cache.cache_replacement_pointer[2][7]
b_cache/cache_replacement_pointer_reg[3][7] => b_cache.cache_replacement_pointer[3][7]
```

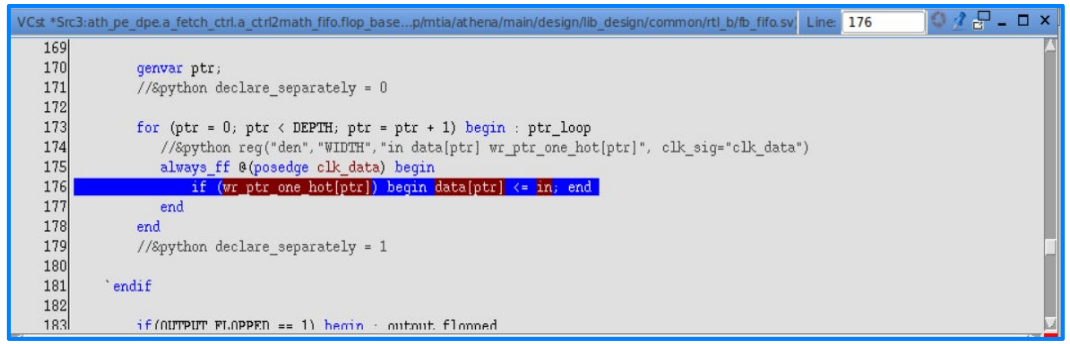
Debug Using Synopsys Verdi®

Violation Message reported in Verdi GUI

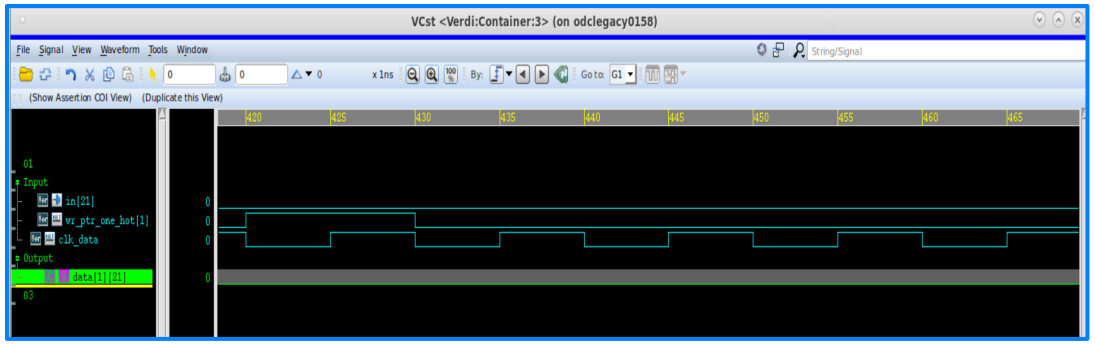


Schematic for the violation (with source and register info)





Violation source code



Waveform for the violation with all non-const signals toggled

Design Results

Design Results – Optimized Registers

Design	Design Size	Run Time (Hrs)	Optimized Registers
Design1	~13M	3.5hrs	173328
Design2	~12M	5.5hrs	251170
Design3	~24M	7hrs	172577

Summary

- Traditional RTL Linting tools don’t report optimized registers aligned with synthesis tools
- **Multiple iterations** between RTL Designers & Implementation Engineers to verify the correctness of register optimizations done by Synthesis tools
  - Optimized registers are identified late in the design cycles with no root-cause-analysis (RCA) to debug
- **Shift-left Methodology using Synopsys VC SpyGlass IDC** to accurately identify optimized registers at the RTL stage with RCA debug capabilities
- User can find and fix the RTL leading to **unintended optimization** using incremental debug aided by waveform and schematic
- Next steps - Improve debugging and usability

THANK YOU

YOUR
INNOVATION
YOUR
COMMUNITY