



SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA



JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Shift Left Detection and Root Cause Analysis of Synthesis Optimized Registers at RTL Level

Wednesday, June 26, 2024

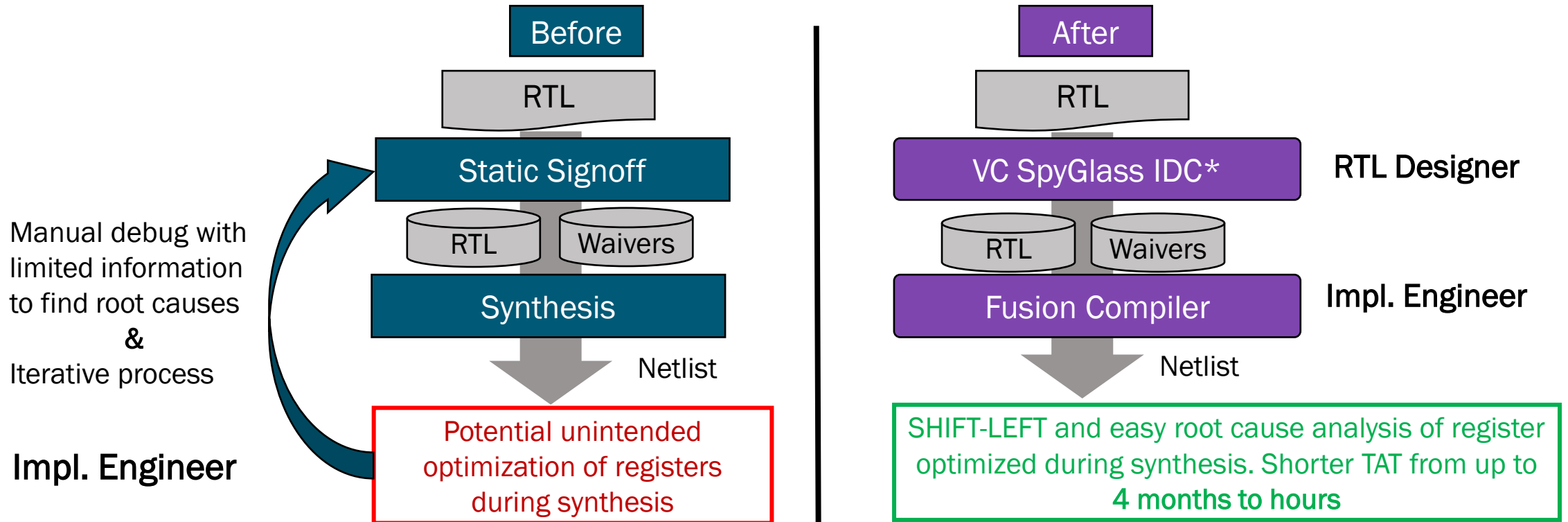


Agenda

- Traditional Flow Challenges and Implemented Workflow
- Implementation Design Checks (IDC) : Flow, Applications & Setup
- Constant and Unloaded Optimized Registers Examples
- Batch mode reports to find the source of optimization
- Comparison with DC/FC output
- Summary

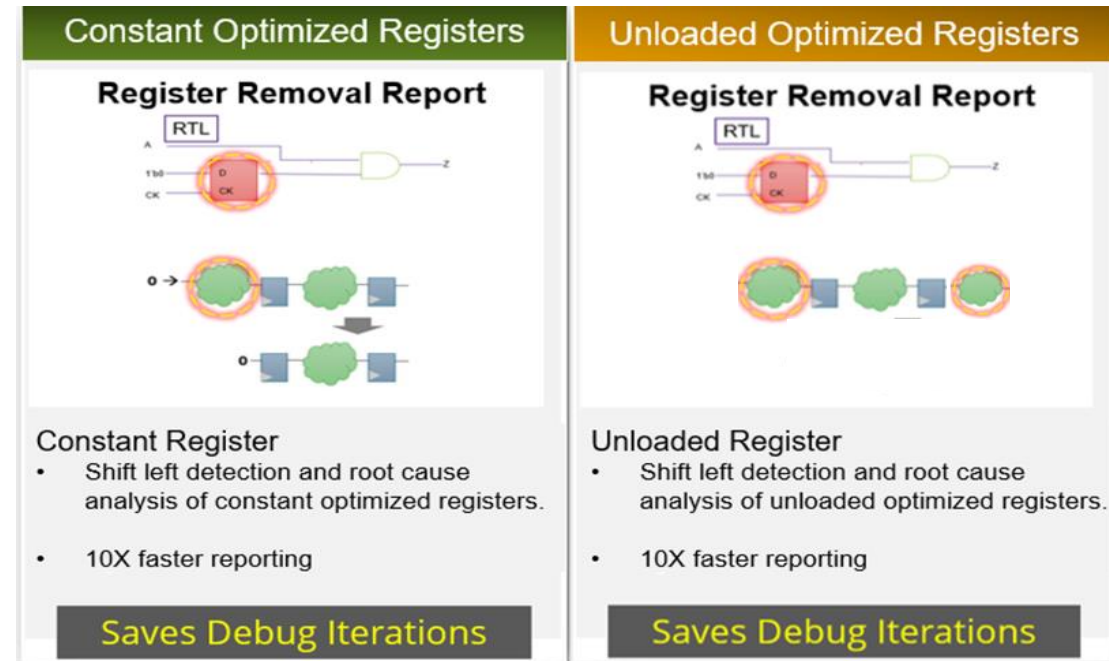
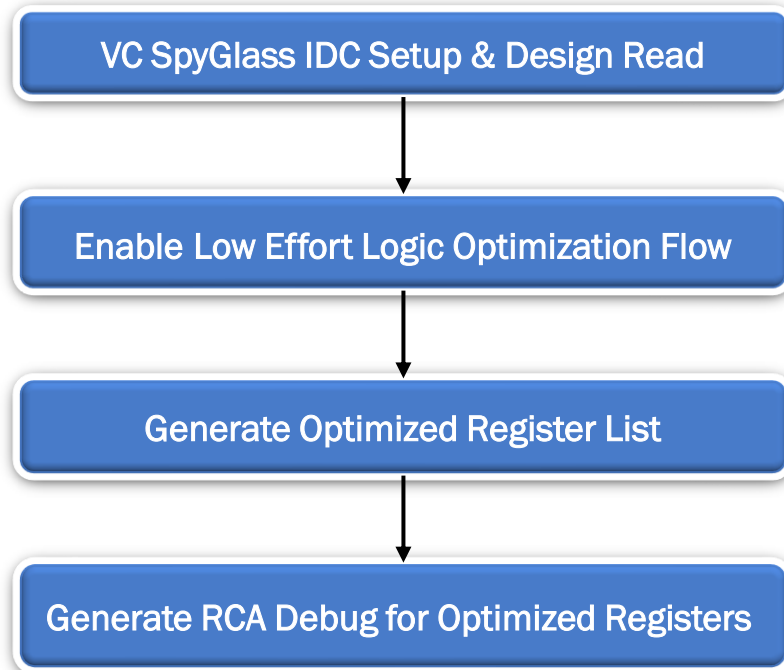
Is Your Design Implementation Ready?

Current Flow Challenges and proposed workflow



Implementation Design Checks (IDC)

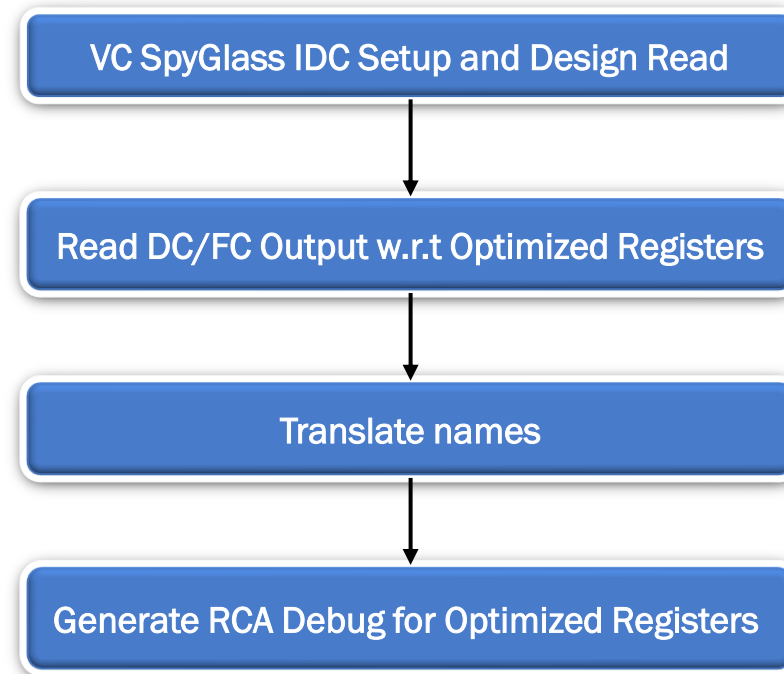
Flow & Applications



1. Fix the un-intended Register Optimizations at RTL level
2. If some optimizations are intended after analyzing the root causes, **pass on the information about analyzed registers from IDC to DC/FC RTR (optimized register) report** to avoid duplicate analysis post synthesis

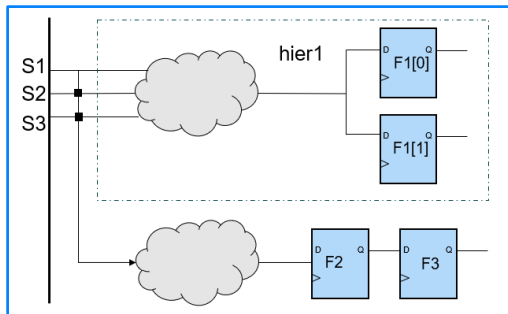
Implementation Design Checks (IDC)

- Users can pass on Optimized Registers Output from DC/FC (report_transformed_registers) directly to VC SpyGlass IDC so that VC SpyGlass can provide root causes for Synthesis Optimized Registers



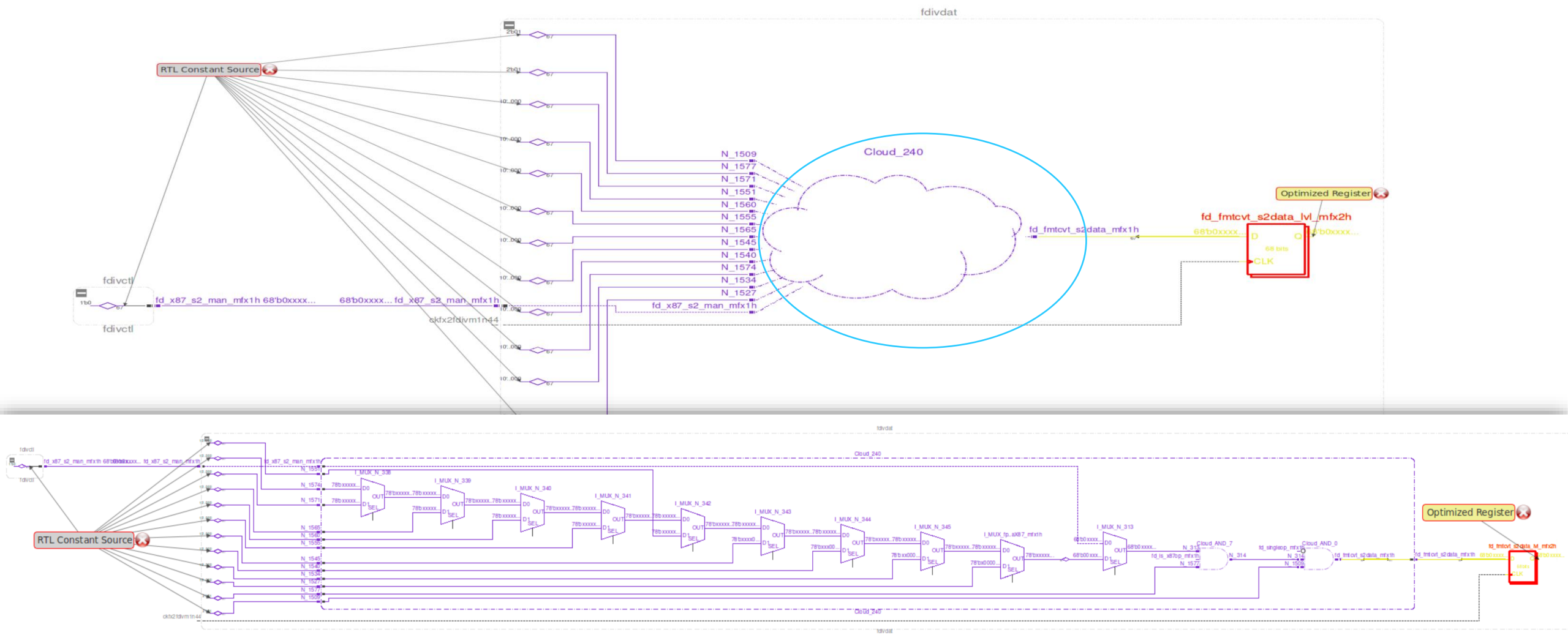
Implementation Design Checks

- Following apps are available
 - **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

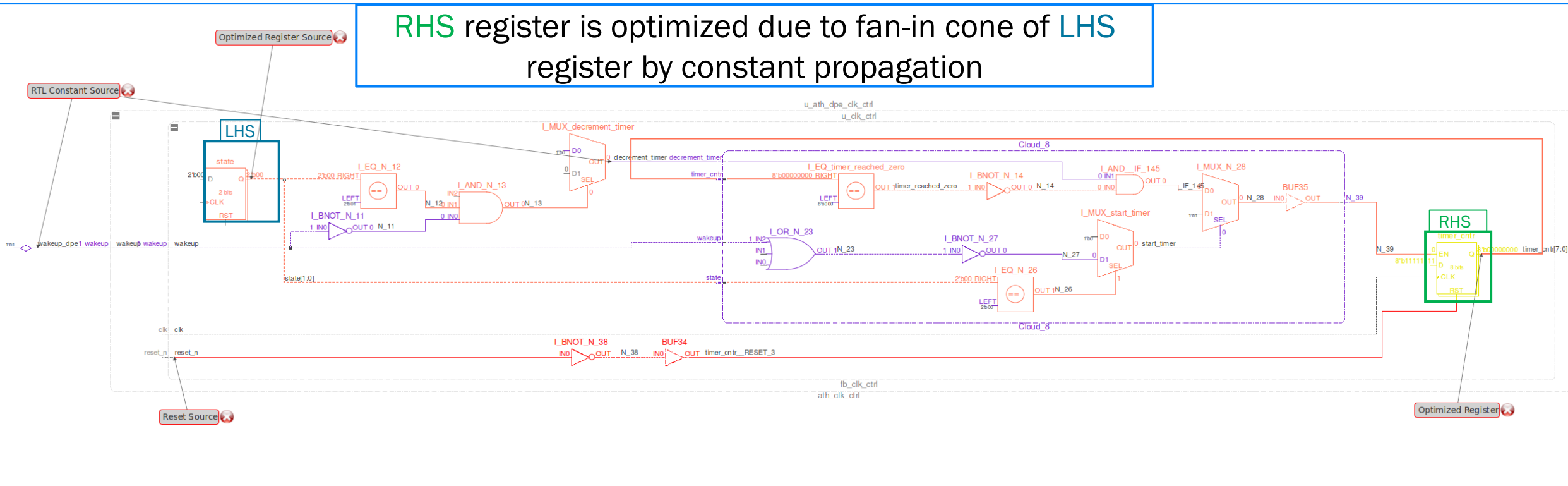


Bus Merging		
Non-Const Sources	Const Sources	Optimized Registers
S1, S2	S3	hier1.F1[1:0] F2 F3

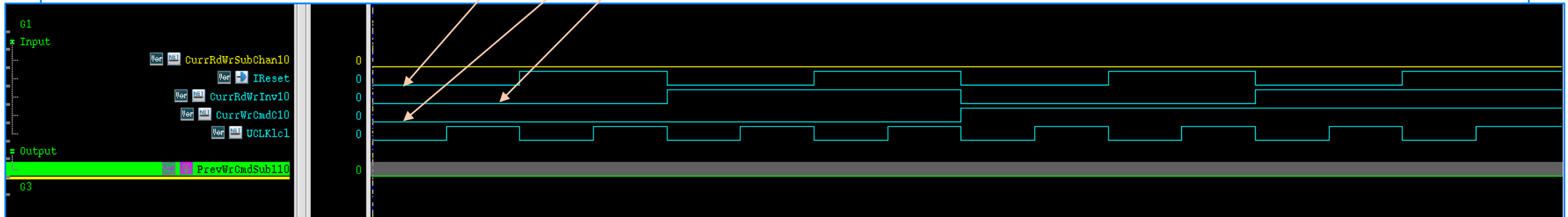
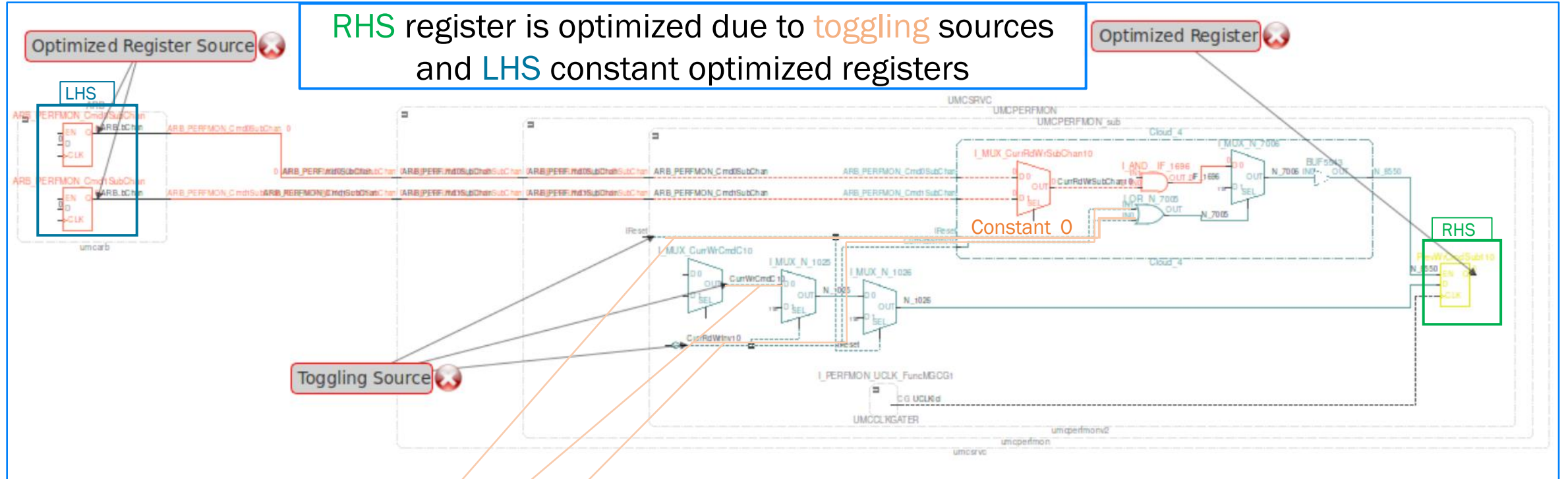
Debugging Optimized Register due to a Constant Source



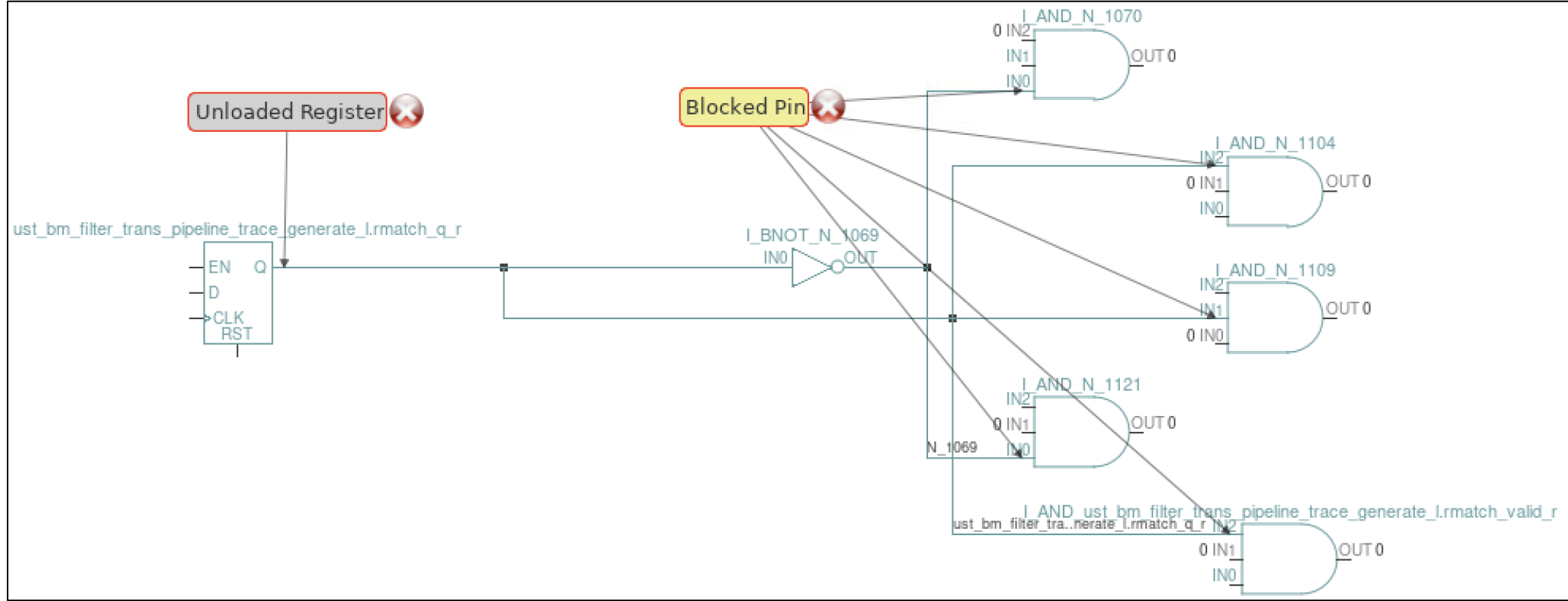
Debugging Optimized Register due to a Another Opt Reg + Const Source



Optimized Register due to Toggling Sources



Optimized Registers due to Blocked Fanout Path



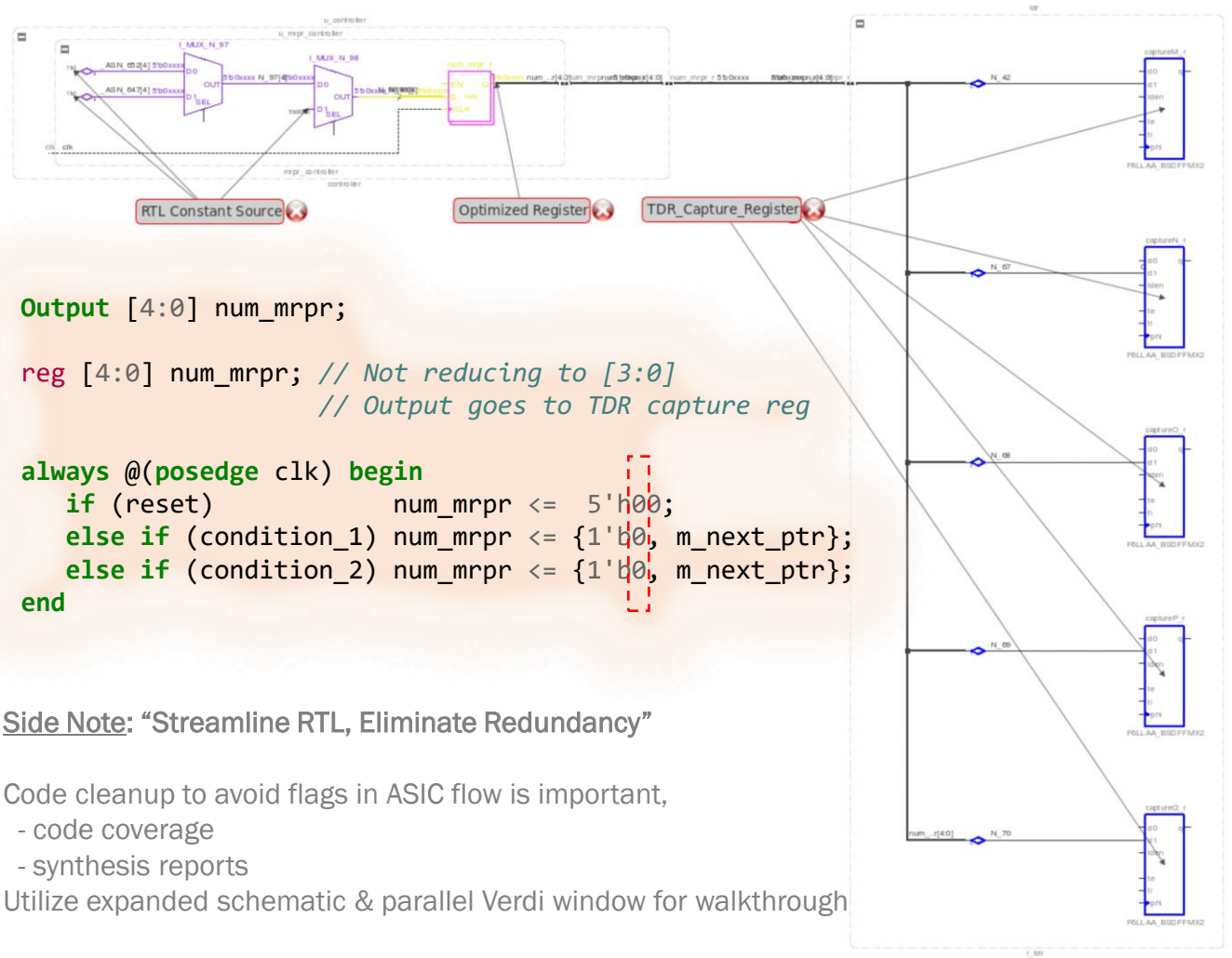
Debugging Optimized Register & Preserve It

Key TDR Register Redundancy & Unintended Optimization Challenge

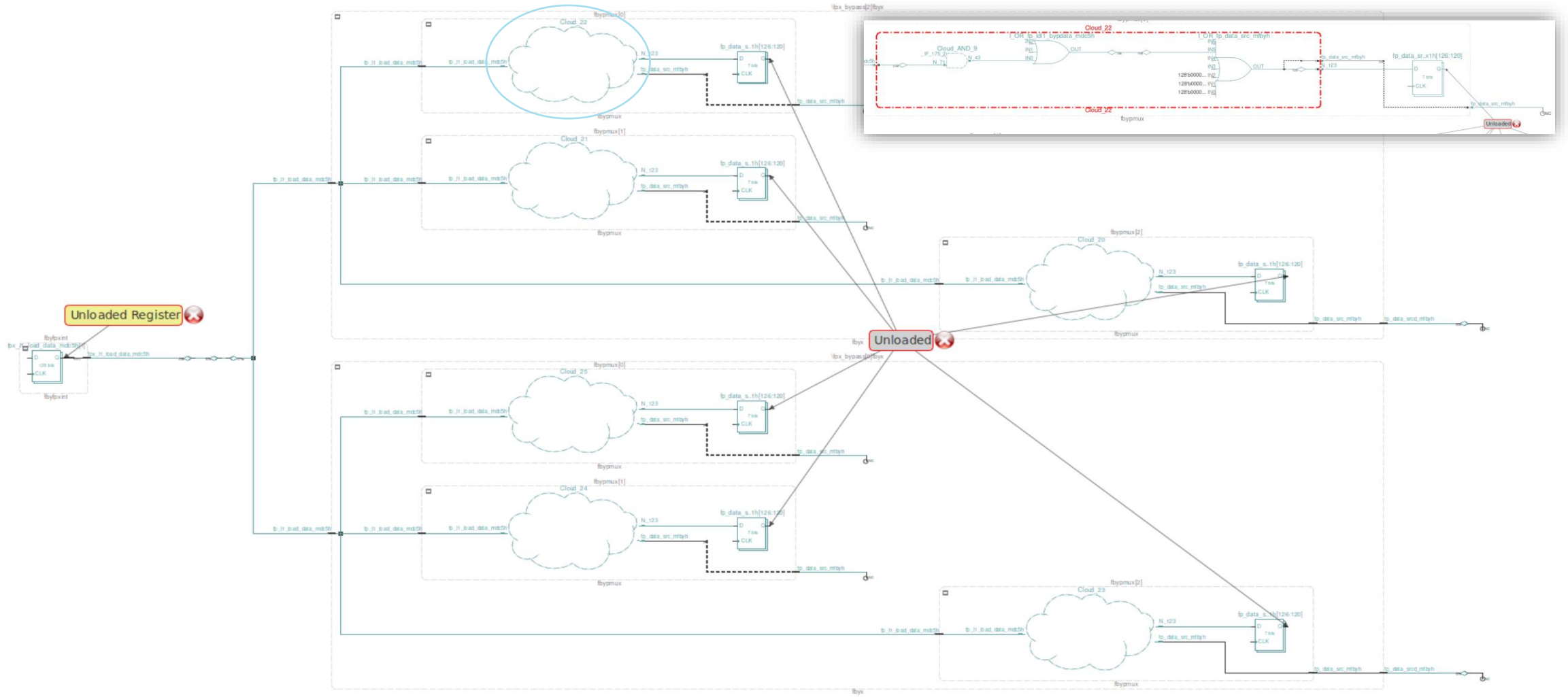
- Tool highlighted removal of Key TDR register from legacy RTL code, MSB bit redundant after RTL modification
- All 5 registers, including the redundant MSB, are part of the TDR chain, hence optimization is unintended which is caught by IDC at RTL level (shift left).
- TDR module is manually constructed and not subject to optimization

Preservation of TDR Module

- Maintain the current TDR chain length for stability
- Implement 'dont_touch' property on the register during synthesis
- Keep RTL unchanged to preserve TDR module integrity



Optimized Register (Chain of Unloaded Regs)



Optimized Registers Report

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				
34				
35				
36	b_cache.flopstage_masked_rdata3.out[529]	Constant thru Opt Reg	a_fetch.ls_rreq_metadata[3].cache_line_number[7]	Design Constant
37	b_cache.flopstage_masked_rdata3_c1.out[529]		b_cache.cache_replacement_pointer[0][7]	Optimized Register
38	b_cache.flopstage_masked_rdata3_c2.out[529]		b_cache.cache_replacement_pointer[1][7]	Optimized Register
39	b_cache.flopstage_masked_rdata3_c3.out[529]		b_cache.cache_replacement_pointer[2][7]	Optimized Register
40	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[0][9]		b_cache.cache_replacement_pointer[3][7]	Optimized Register
41	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[10][9]			
42	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[11][9]			
43	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[1][9]			
44	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[2][9]			
45	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[3][9]			
46	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[4][9]			
47	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[5][9]			
48	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[6][9]			
49	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[7][9]			
50	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[8][9]			
51	genblk2[3].ls_rreq_metadata.fifo.flop_based.data[9][9]			
52	genblk2[3].ls_rreq_metadata.fifo.out[9]			
53				
54	a_fetch_ctrl.a_ctrl2math_fifo.flop_based.data[0][21]	Optimized by Logic	a_fetch_ctrl.a_ctrl2math_fifo.flop_based.wr_ptr_one_hot[0]	NON_CONST
55			a_fetch_ctrl.a_ctrl2math_fifo.in[21]	NON_CONST
56				
57	a_fetch_ctrl.a_ctrl2math_fifo.flop_based.data[1][21]	Optimized by Logic	a_fetch_ctrl.a_ctrl2math_fifo.flop_based.wr_ptr_one_hot[1]	NON_CONST
58			a_fetch_ctrl.a_ctrl2math_fifo.in[21]	NON_CONST
59				

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

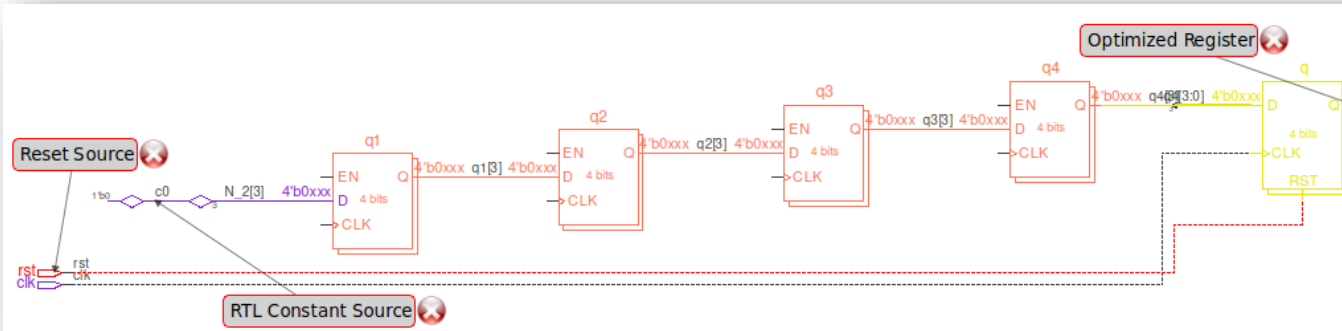
SOURCE_TYPE

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

Optimized Registers Report Contd...

```
11 OPTIMIZED REGS
12   List of optimized registers which have the same set of sources
13
14 OPTIMIZATION TYPE
15   Direct Constant      : When the inferred optimized register is directly driven by a constant (or a propagated constant)
16   Optimized by Logic   : When a combination of inputs driving the register causes it to get optimized
17   Constant thru Opt Reg : If an inferred optimized register is driving another inferred optimized register
18
19 SOURCES
20   List of sources causing the register to get optimized
21
22 SOURCE_TYPE
23   Design Constant      : Constant logic
24   Optimized Register   : Another inferred optimized register
25   NON_CONST           : Non Constant logic
26
27
28 -----
29 OPTIMIZED REGS   OPTIMIZATION TYPE   SOURCES   SOURCE TYPE
30 -----
31 q1[3]            Direct Constant      c0(./test.v:8) Design Constant
32 q2[3]
33 q3[3]
34 q4[3]
35 q[3]
```

For a chain of registers, only the relevant constant source is shown which is responsible for optimizing the whole chain



VC SpyGlass IDC vs DC/FC

DC/FC Constant Register Report

```
1 *****
2 Report : report_transformed_registers
3 Version: T-2022.03-SP5-CS4
4 Date   : Wed Nov 1 13:15:49 2023
5 *****
6
7 Legend:
8 C0p - constant 0 register preserved
9 C1p - constant 1 register preserved
10 C0r - constant 0 register removed
11 C1r - constant 1 register removed
12 ulp - preserved unloaded register
13 ulr - removed unloaded register
14 mrg - merged register
15 srh - shift register head
16 srf - shift register flop
17 mb  - multibit
18 mbd - multibit debanked
19 rep - replicated
20 inv - inverted
21 rtm - retimed
```

```
29 fpa/fxhfnunit_noscan/scmisr_fpa_noscan/scmisr_dout_mnnlh_reg[6] C0r
30 fpa/fxhfnunit_noscan/scmisr_fpa_noscan/scmisr_dout_mnnlh_reg[5] C0r
31 fpa/fxhfnunit_noscan/scmisr_fpa_noscan/scmisr_dout_mnnlh_reg[4] C0r
32 fpa/fxhfnunit_noscan/scmisr_fpa_noscan/scmisr_dout_mnnlh_reg[3] C0r
33 fpa/fxhfnunit_noscan/scmisr_fpa_noscan/scmisr_dout_mnnlh_reg[2] C0r
```

```
3266 fpa/frfp/fpx_rf_data_out_mfbyh_reg[2][0][86] ulr
3267 fpa/frfp/fpx_rf_data_out_mfbyh_reg[0][1][87] ulr
3268 fpa/frfp/fpx_rf_data_out_mfbyh_reg[0][1][86] ulr
3269 fpa/frfp/fpx_rf_data_out_mfbyh_reg[0][1][67] ulr
3270 fpa/frfp/fpx_rf_data_out_mfbyh_reg[0][0][87] ulr
3271 fpa/frfp/fpx_rf_data_out_mfbyh_reg[0][0][86] ulr
```

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

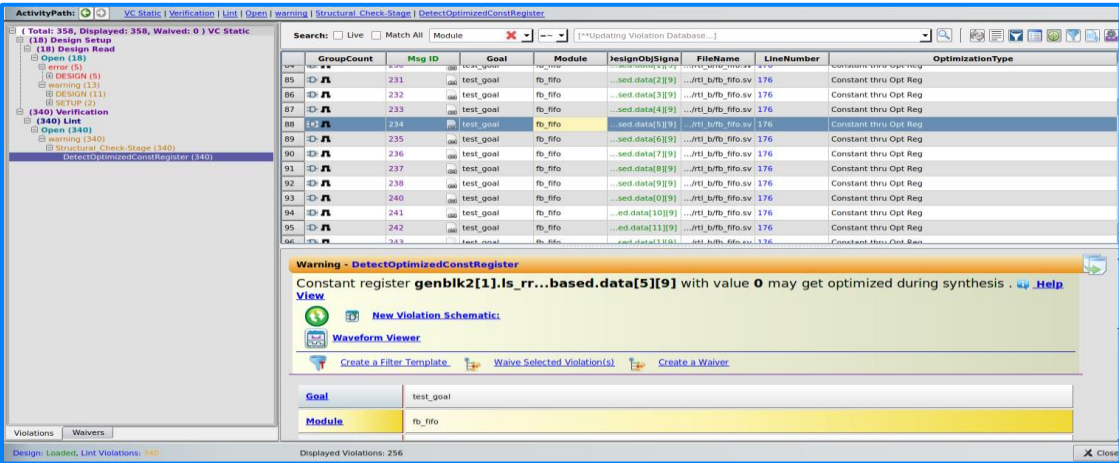
QoR Matching Summary using Comparison script

```
### 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
```

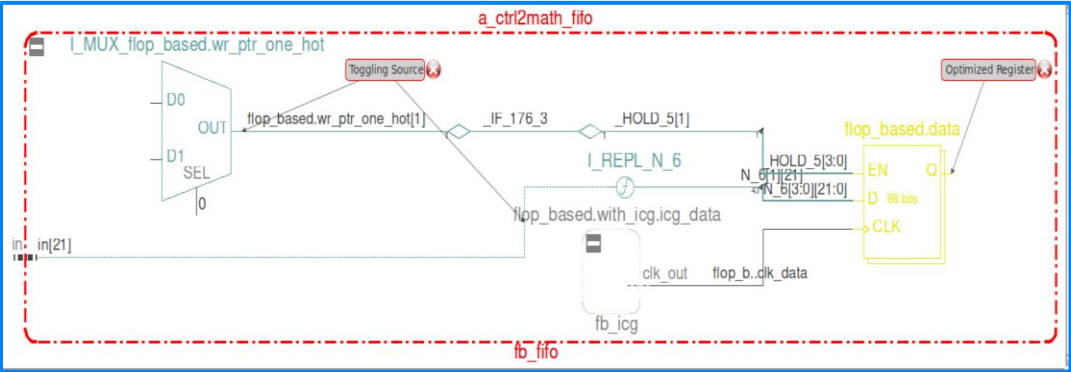
```
### Unused Registers ###
QOR_VC_unused_registers = 51548
QOR_FC_unused_registers = 53230
QOR_unused_registers_matching = 51514
QOR_unused_registers_FC_only = 1716
QOR_unused_registers_VC_only = 34
```


Debug using Verdi

Violation Message reported in Verdi GUI

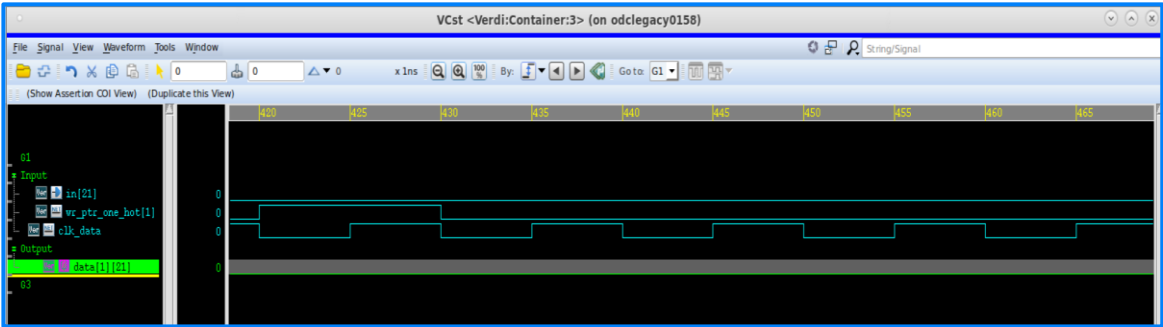


Schematic for the violation (with source and register info)



```
VCst *Src3:ath_pe_dpe.a_fetch_ctrl.a_ctrl2math_fifo.flop_base...p/mtia/athena/main/design/lib_design/common/rtl_b/fb_fifo.sv Line: 176
169
170     genvar ptr;
171     //python declare_separately = 0
172
173     for (ptr = 0; ptr < DEPTH; ptr = ptr + 1) begin : ptr_loop
174         //python reg("den", "WIDTH", "in data[ptr] wr_ptr_one_hot[ptr]", clk_sig="clk_data")
175         always_ff @(posedge clk_data) begin
176             if (wr_ptr_one_hot[ptr]) begin data[ptr] <= in; end
177         end
178     end
179     //python declare_separately = 1
180
181     `endif
182
183     if (OUTPUT_FLOPPED == 1) begin : output_flopped
```

Violation source code (Back-ref)



Waveform for the violation with all non-const signals toggled

Design Results – Optimized Registers

Design	Design Size	Run Time	Optimized Registers
Design1	~5M	40 mins	73,328
Design2	~10M	1 hrs	91,109
Design3	~12M	1.5 hrs	41,170

Summary

- Traditional RTL Linting tools don't report **optimized registers** & **unused 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 & unused registers are identified late in the design cycles with **no root-cause-analysis** (RCA) to debug
- **Shift-left Methodology using 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

Thank You