Don’t Let CDC Kill Your Chip!

April 2014

Shaker Sarwary

Note: The information in this presentation may contain forward-looking statements regarding product development. This development is subject to change at the sole discretion of Atrenta. Atrenta will have no liability for the delay or failure to deliver any features included in this presentation.

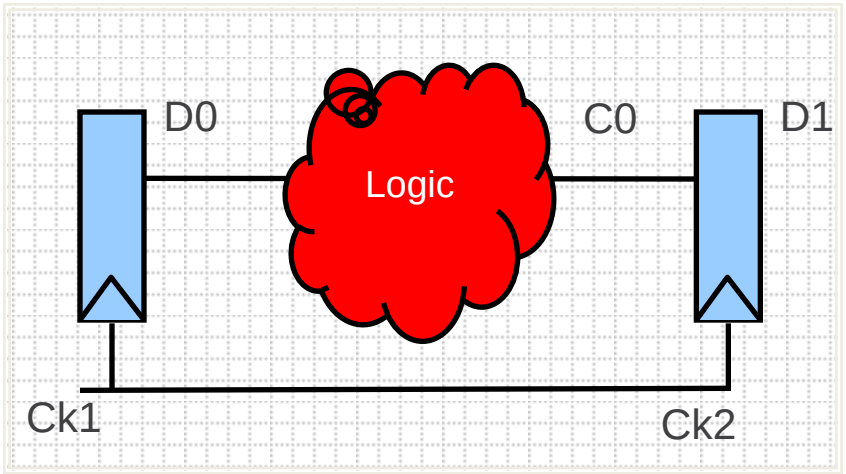
Agenda

- **Meta-stability and System Behavior**
- **Safe Data Transfer Across Clock Domains**
- **Killer Bugs in CDC**
- **Good and Bad Design Practices**
- **Atrenta's SpyGlass CDC Solution**
- **CDC Verification Challenges and Trends**

Clocks Usage Exploding

- **Heterogeneous applications like wireless, networking, video, graphics have *hundreds of clocks* with asynchronous crossings**
- **Power domains drive additional clocks explosion**
- **Traditional EDA tools do not analyze asynchronous interfaces (STA, Functional verification, …)**

Principle of Synchronous Design



Input to combination logic changes upon clock edge

Combo logic computation → transient glitches

Computation settles before next edge of clock

D1 captures a stable data honoring setup/hold time

But what if D1 is clocked by another asynchronous clock?

Synchronous Design Verification

SPYGLASS® FROM ATRENTA

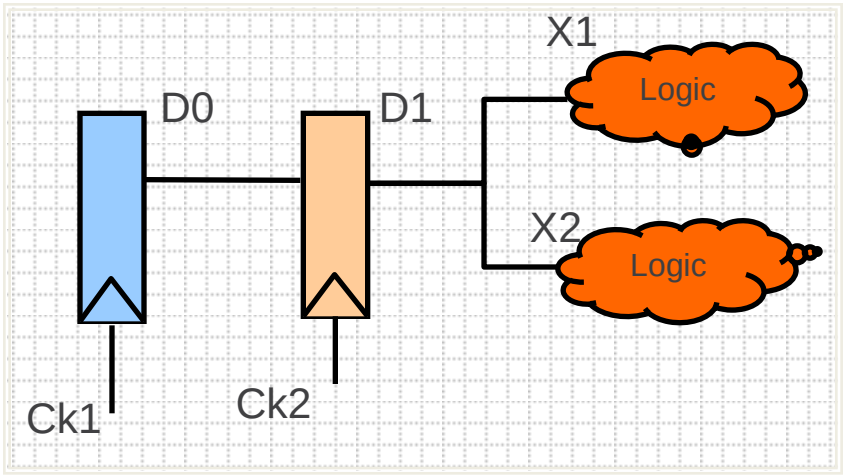
Functional Verification

- Simulation or formal verification
- Assumption: Cycle based, event-driven
- Effective in detecting logical bugs

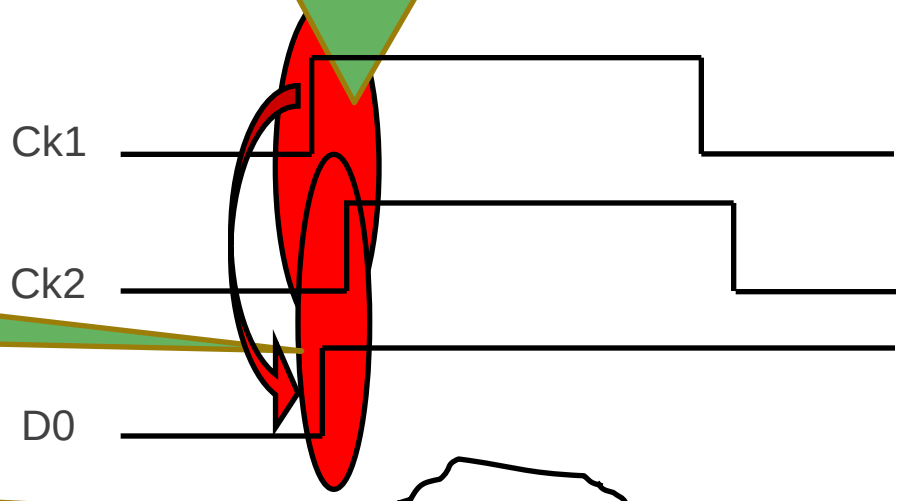
Static Timing Verification

- Timing model and load information required
- Focus on synchronous paths (`set_false_path` defined on asynchronous paths)
- Effective in detecting timing violations on synchronous paths

Meta-stability - Cause and Effect

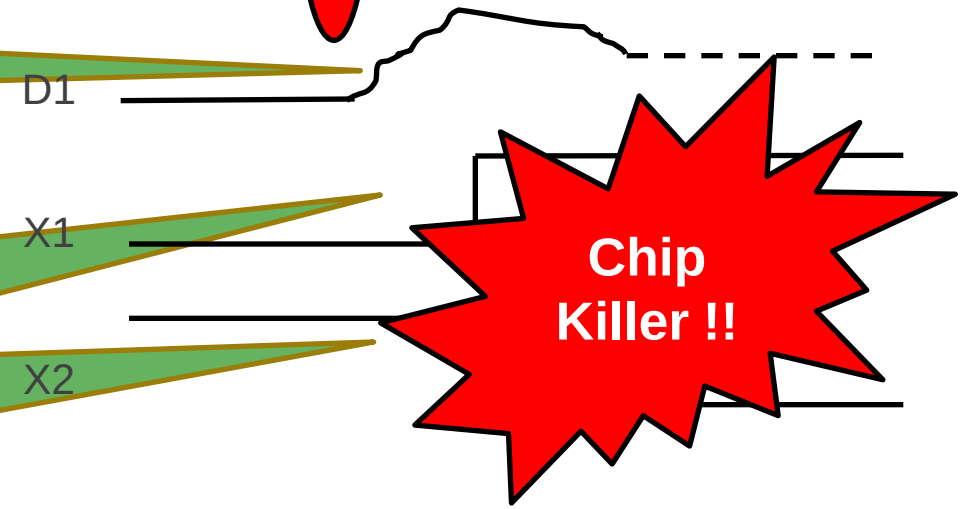


Asynchronous clocks – No phase relation



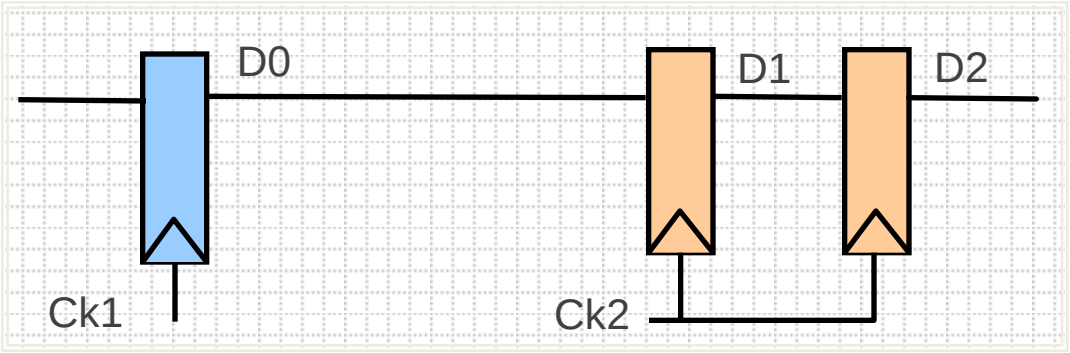
Unpredictable setup/hold time violation

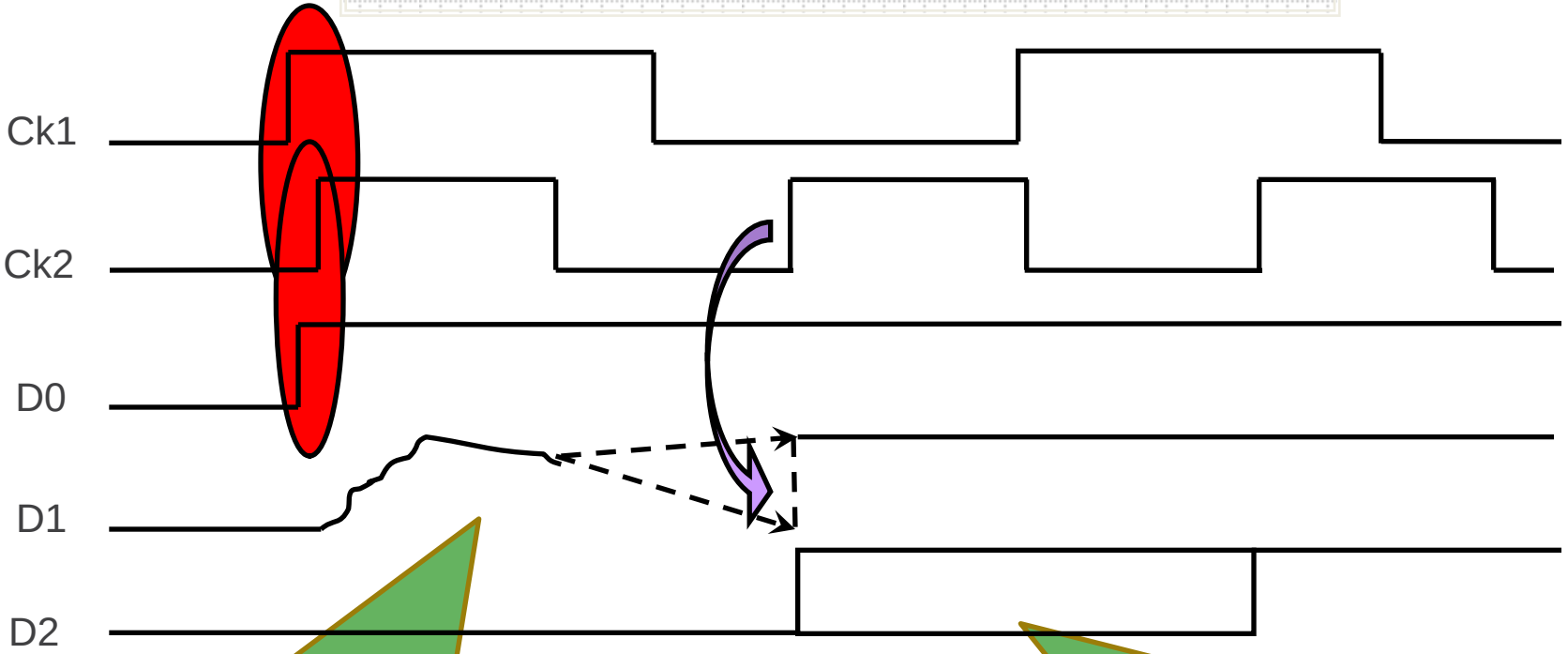
Meta-stability caused by setup/hold time violation



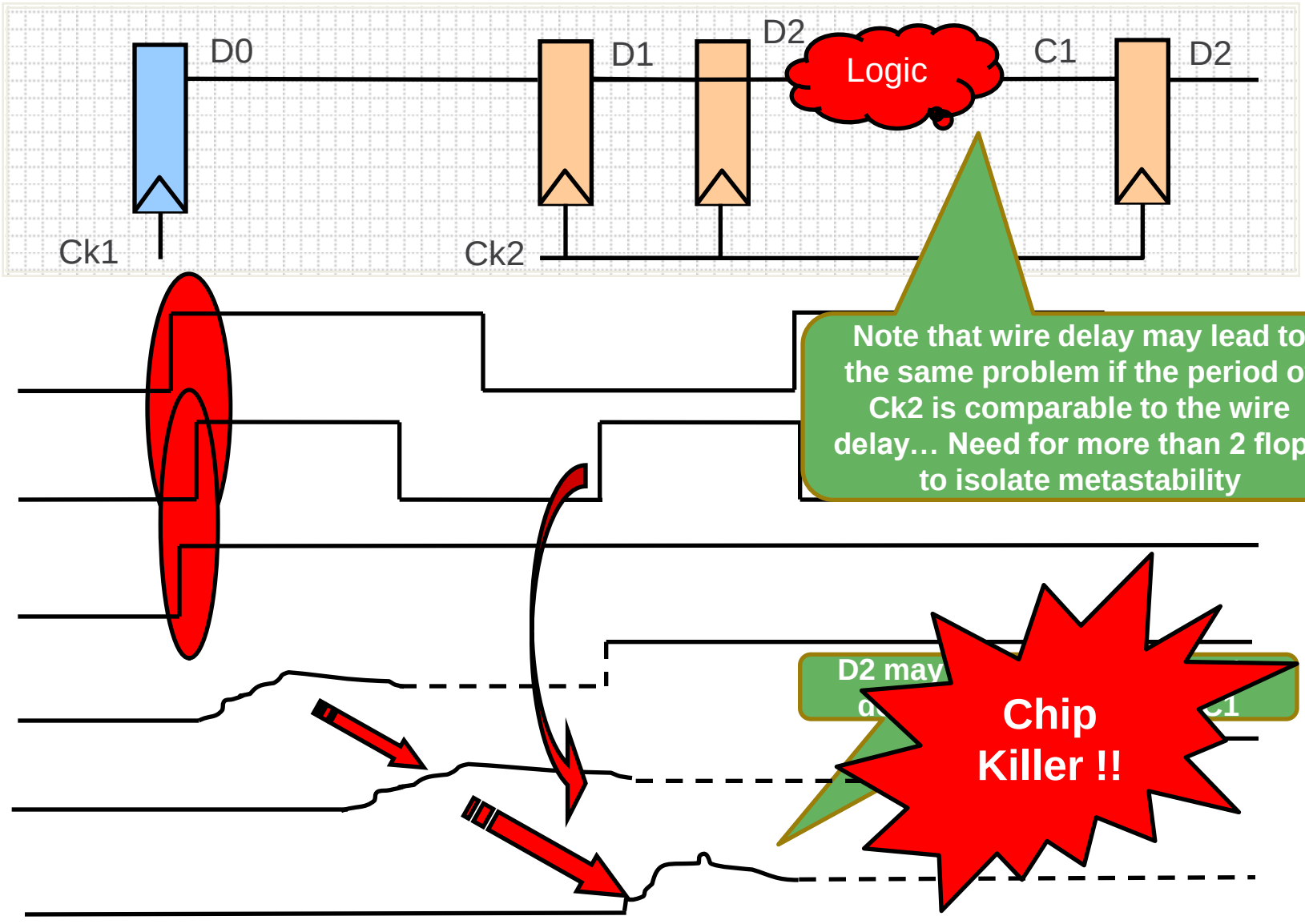
Based on the load of combo logic meta-stability at D1 can be interpreted as 0, 1 at X1, X2…

Synchronization and Isolating Meta-stability

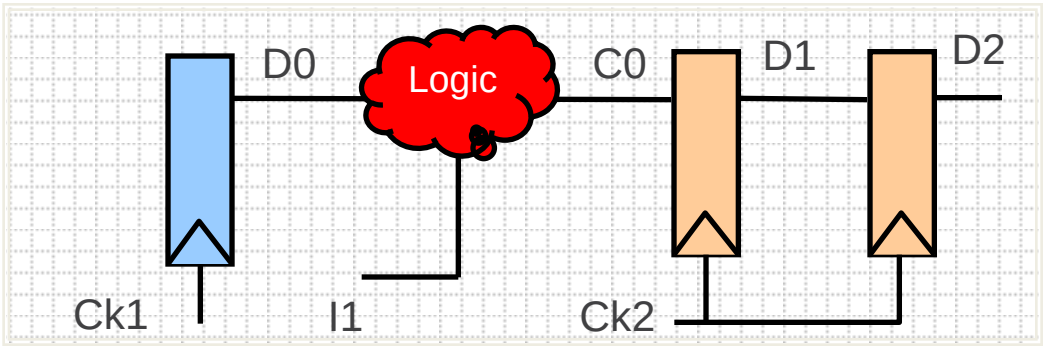




Effect of Delay Between Synchronizer flops



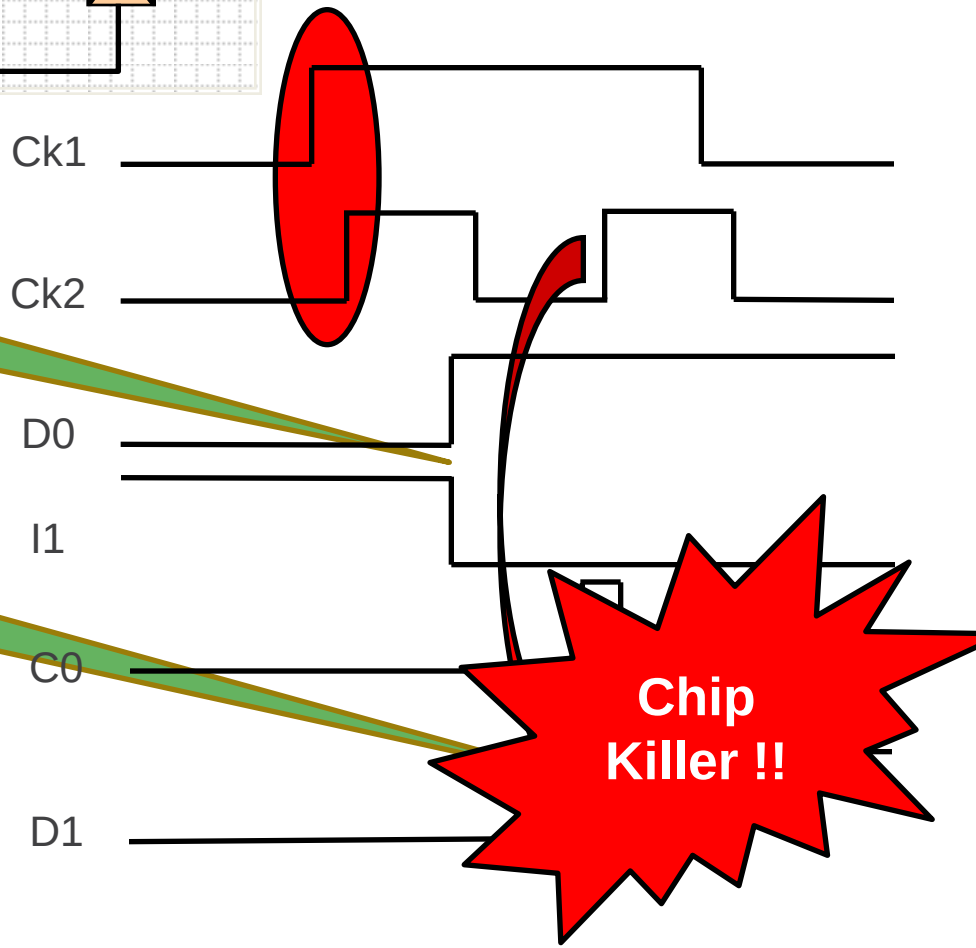
Effect of Combo Logic Before Multi-flop Synchronizer



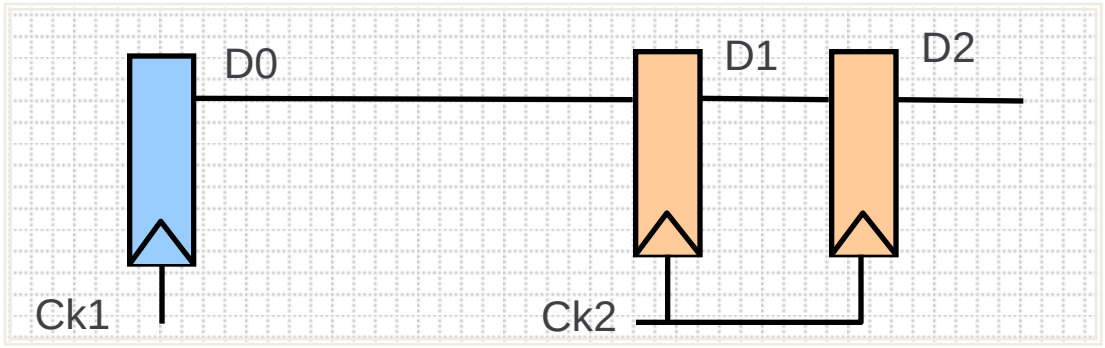
When D0/I1 change the combo logic can glitch at any time independently from Ck2

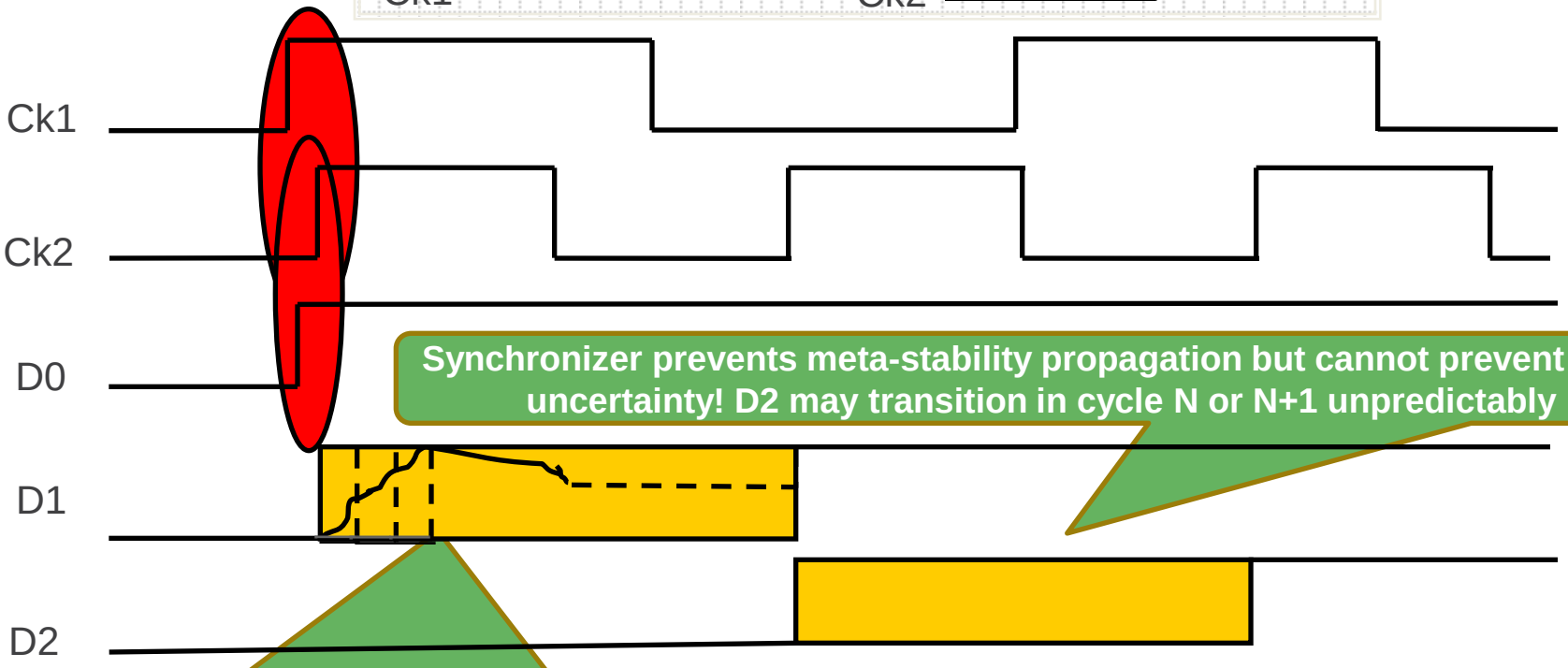
D1 may now see a glitch at its input as a pulse!!

Risk of chip failure due to glitch on CDC; The risk increases with Ck2 frequency



Meta-stability, Synchronization, and Cycle Uncertainty

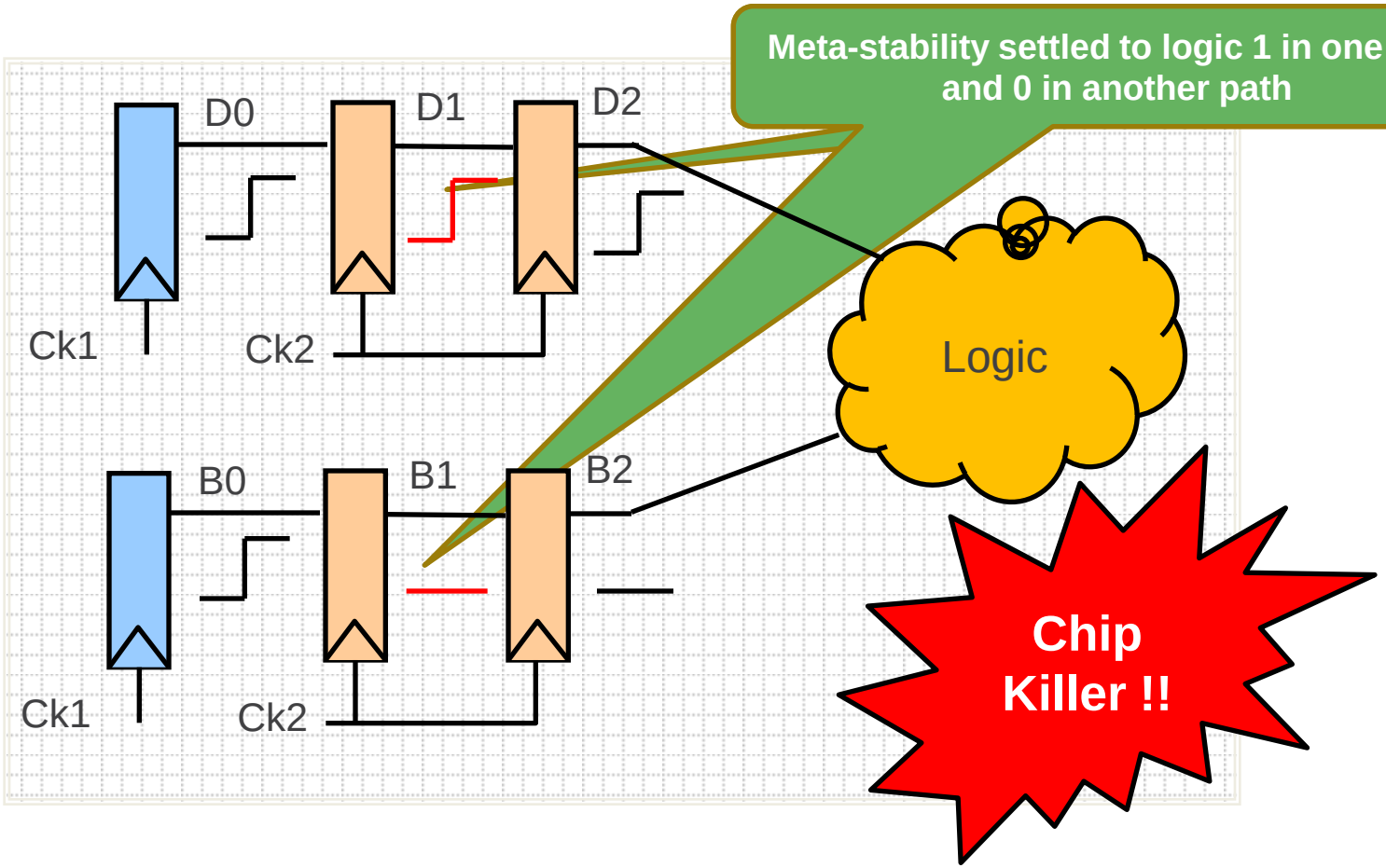




Synchronizer prevents meta-stability propagation but cannot prevent cycle uncertainty! D2 may transition in cycle N or N+1 unpredictably

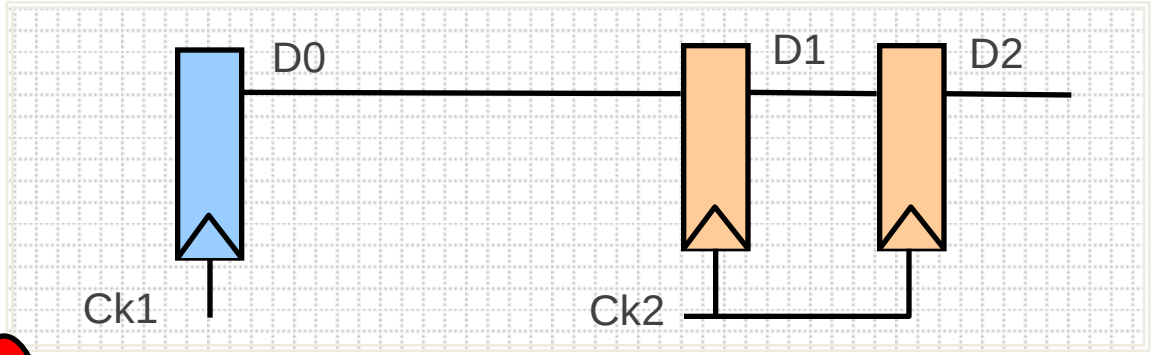
Ck2 may come on time to capture D0 or not => transition on D0 may or may not be captured in current cycle into D1

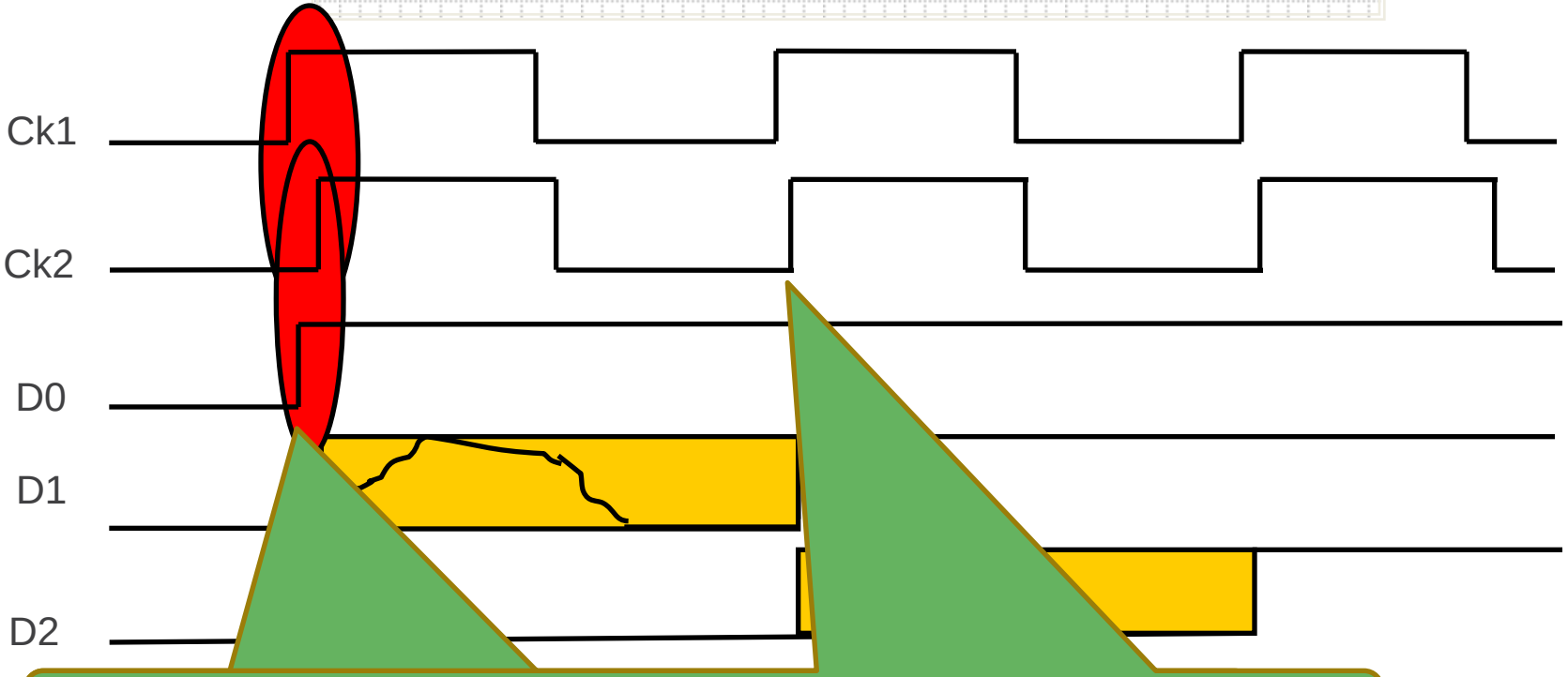
Cycle Uncertainty and Data Coherency (Convergence Problem)



Cycle uncertainty corrupts coherency of correlated signals; D0/B0 transition "00 → 11" is perceived as a transition "00 → 10"

Data Hold Requirement and Safety Margin





Because D0 may not be captured at the first active edge of Ck2;
D0 should be stretched to cover at least 2 active edges of Ck2

What Happens if a CDC Bug Slips into Silicon?

SPYGLASS® FROM ATRENTA

- **How do you catch CDC bugs in silicon?**
  - The nature of CDC problem is random, and unpredictable
- **If a CDC bug is observed, how is it debugged?**
  - Initial reaction is often being in denial as CDC bugs cannot be reproduced easily
  - Significant cost and time is involved before the CDC bug is isolated
- **How is a CDC bug typically addressed once the chip is manufactured?**
  - Low cost options are:
    - Can a software change fix the bug?
    - Can a slower clock rate fix the problem?

Prevent CDC bugs in silicon and avoid painful re-spins. Catch them early at RTL!

Pop Quiz

- **It is safe to have combinational logic between double flops of a synchronizer**
  a) True, because the signal has already crossed the clock domain
  b) False, because the combinational logic may glitch
  c) False, because combinational logic will extend the meta-stability

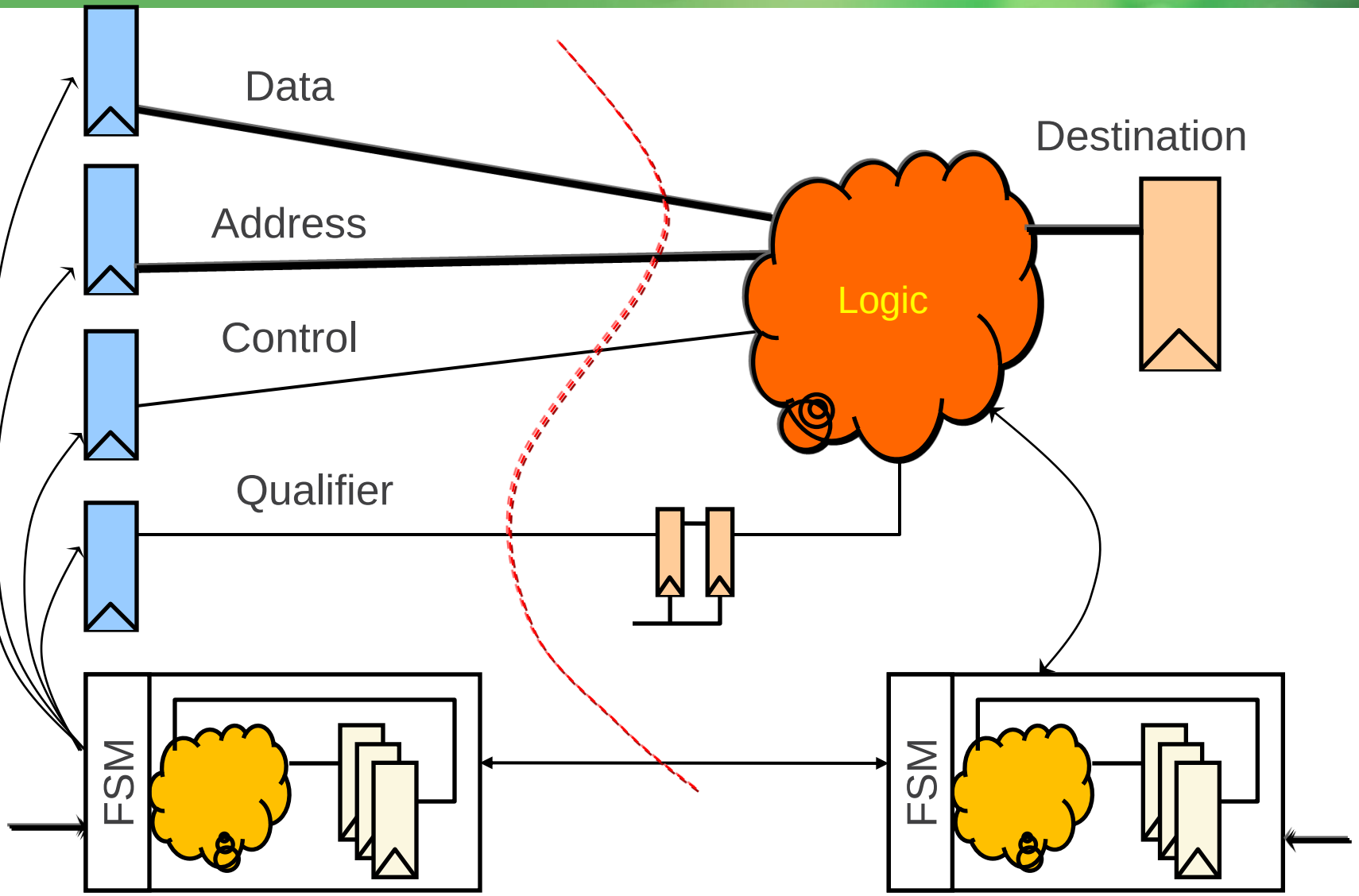
- **Paths set as false-path for static timing analysis cannot cause chip failure**
  a) True, because these paths are not functional
  b) False, because these paths may cause meta-stability
  c) False, because these paths may contain functional bugs

Agenda

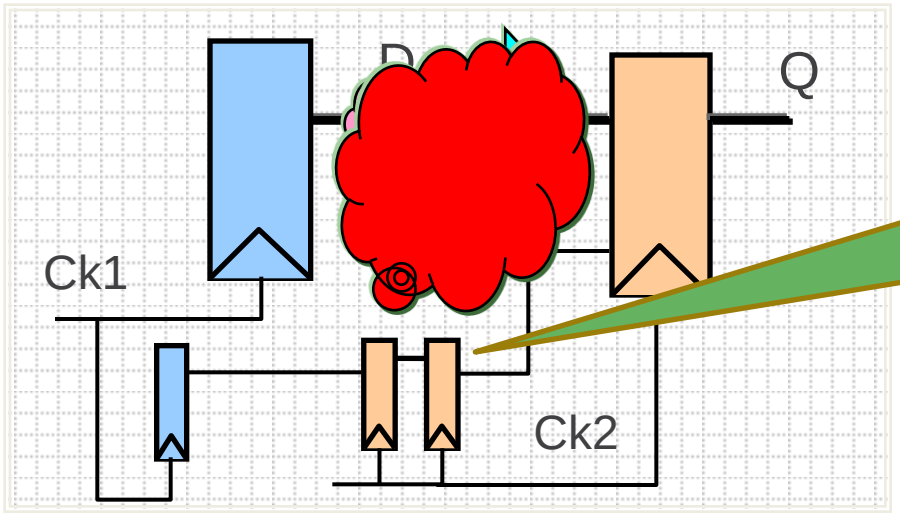
SPYGLASS® FROM ATRENTA

- Meta-stability and System Behavior
- **Safe Data Transfer Across Clock Domains**
- Killer Bugs in CDC
- Good and Bad Design Practices
- Atrenta's SpyGlass CDC Solution
- CDC Verification Challenges and Trends

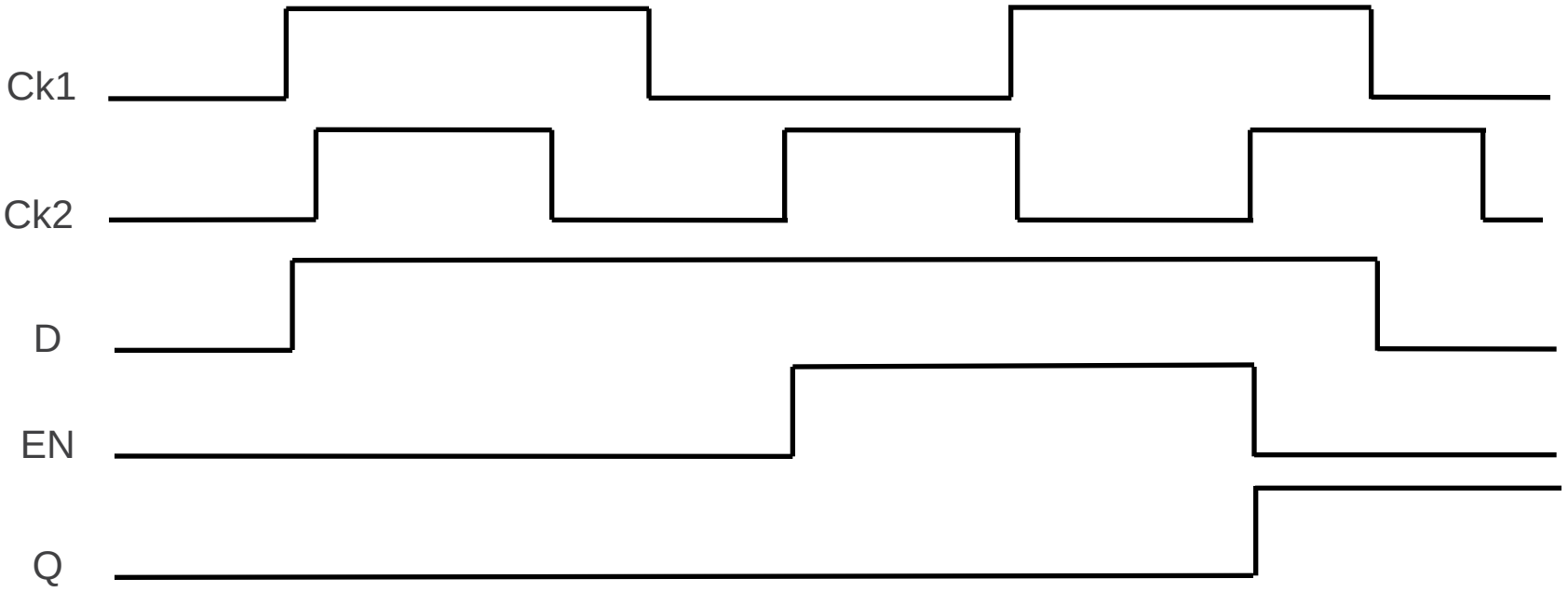
Example of Data Transfer Across Clock Domains



Data Transfer Across Clock Domains



Data transfer relies on a "qualifier" signal synchronized in destination domain for safe data transfer

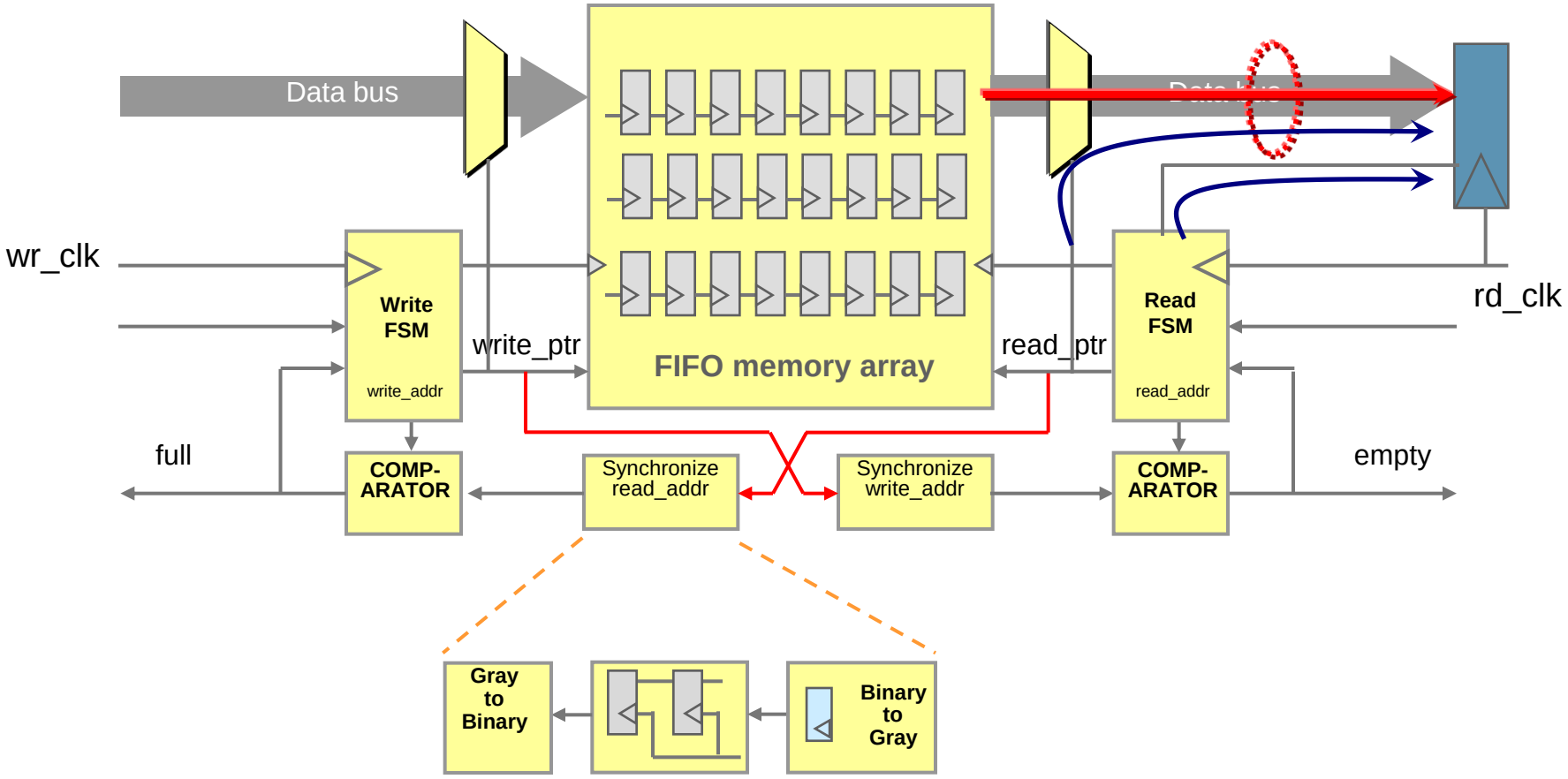


What is Really Making a Data Synchronized?

SPYGLASS® FROM ATRENTA

- **Concept**
  - There must be a separate channel from launch to capture to communicate data readiness…
  - Is there any other way? (remember data line is a multi-bit signal that cannot be gray encoded)
- **How to design such channel**
  - Protocols including FIFOs and Handshakes
  - Where is the channel in a standard FIFO?
  - Where is that channel in an Empty-less FIFO?
  - Where is that channel in a handshake?
- **How to verify data synchronization?**
  - Identify the channel and ensure its functional intent is conform to data synchronization
  - How can you achieve this structurally?
  - How can you achieve this functionally?

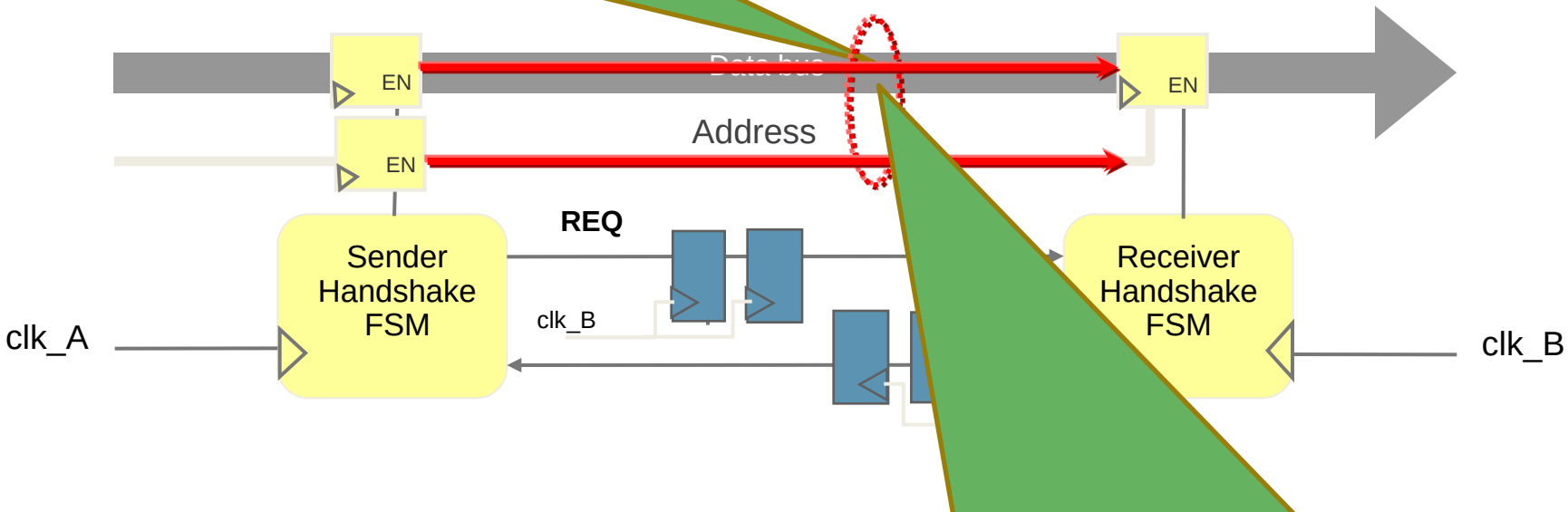
FIFO Synchronization



Can a FIFO without Empty Flag be safe?
What types of Glitches can impact FIFO functionality?
Is recognizing structure of a FIFO a good indication of synchronization?
Is the implication true? “No overflow/underflow → no metasbility”? And glitch?

Handshake Synchronization

Data/Address crossing clock domain; Are the crossings synchronized?



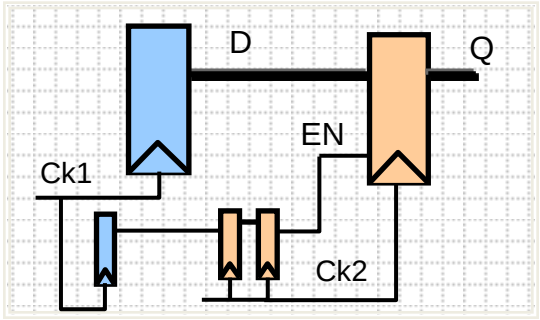
Data will cross the clock domain safely if handshake protocol is properly designed; request and acknowledge are synchronized, data is stable when request is asserted

What difference 2 or 4 phase handshakes make in terms of CDC?

Formal CDC Verification

SPYGLASS® FROM ATRENTA

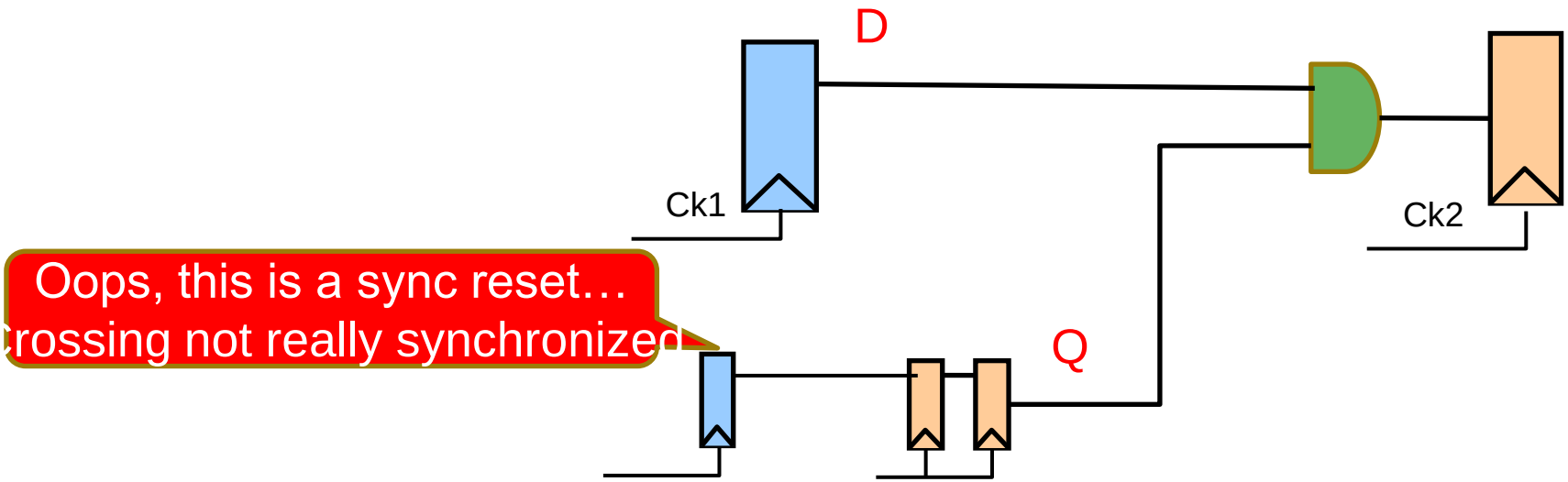
- **Why Formal verification is needed?**
  - Well-accepted structure will not stop metastability if D changes when E is high
  - Many signals in the enable path may have multi-flop sync, not all of them are designed to prevent metastability
  - Only functional verification can ensure correctness



- **MUST-Verify**
  - Data hold check for data crossings
  - Width extender for control crossings
  - Exclusivity of correlated signals
- **NICE to run**
  - FIFO overflow/underflow checks
  - Handshake protocol check
- **What if you do not run functional verification**
  - Structural CDC verification typically finds most of CDC issues, however
  - You may still have some apparently properly built synchronizers that can cause metastability – e.g. qualifier not filling its duty of halting asynchronous data transfer

Example of data-hold check

- **Properly synchronized crossing…**

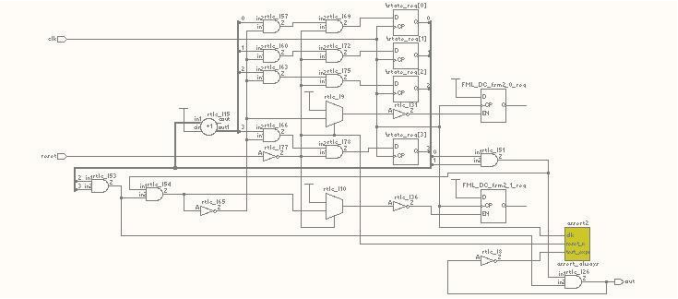


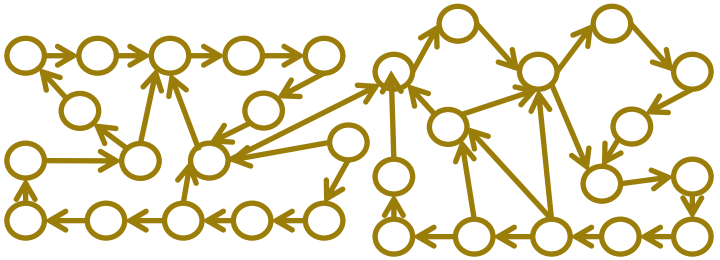
- **Ensure that if Q is asserted then D will not toggle!**
  - A synchronous reset, or any other signal not intended for synchronization will not be able to fulfill “qualifier” functional requirement

How Formal Verification Works

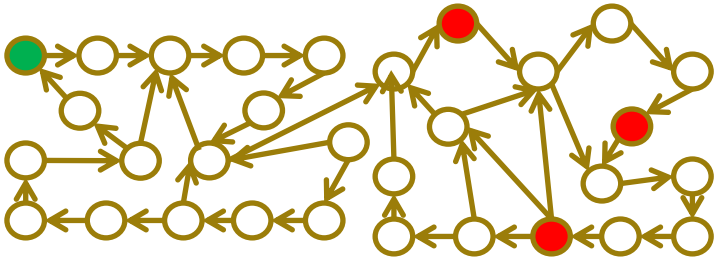
SPYGLASS® FROM ATRENTA

- **View this circuit as a BIG Finite-State Machine**
  - Example: If a design has 1000 flops, it is a BIG FSM with **2^{1000} =10^{301} states**





- **Visit states of the machine in search of a bug**
  - Start from an initial state – colored GREEN
  - Search for a bug state – colored RED
    - If path is found, there is a BUG, report **FAIL**
    - If no path is found, no bug found, report **PASS**
    - If time/memory resources are UP, report **Partially-Proved**



How Formal Verification Works

SPYGLASS® FROM ATRENTA

- **Searching for a path between GREEN and RED states is extremely resource-intensive operation (time and memory)**

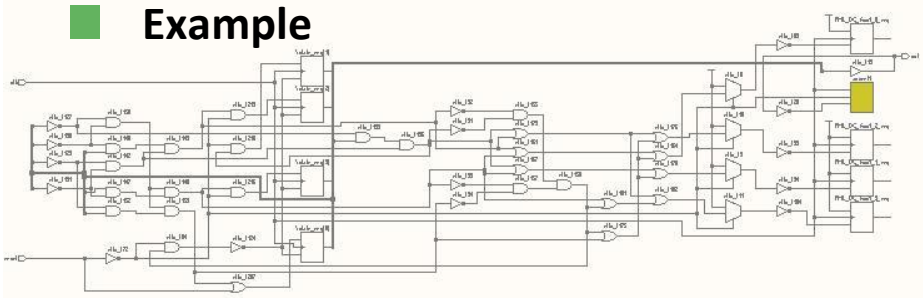
- **Witness mode Formal Engines**
  - Exercise the design in search of a bug – this can cover exhaustively all states up to a certain depth
  - Often, combined with design knowledge can help designer to decide on correctness of a design
  - Cannot provide a proof of correctness as sequential depth of circuit is not known

- **Proof mode Formal Engines**
  - Applies mathematical approaches to prove correctness of a property regardless of sequential depth
  - E.g. proof by induction, or full design reachability analysis
  - Proof is often hard to achieve

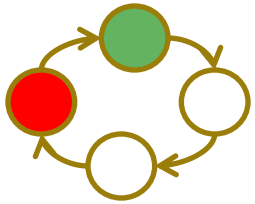
Exhaustive Verification Complexity

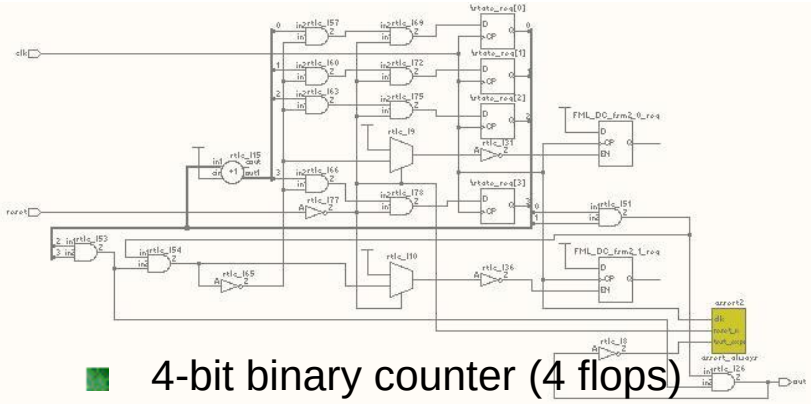
SPYGLASS® FROM ATRENTA

- **Can't we pick the right engine for a given problem & avoid partially proved results?**
  - We don't know what engine works best for a given property
  - We do some guesses based on the flop count, type of the property, but these are guesses
- **Example**

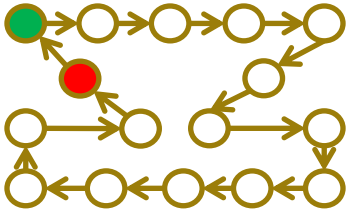


- 4-bit one-hot counter (4-flops)
  - ***4 cycles*** to cover all states





- 4-bit binary counter (4 flops)
  - 2^4 = ***16 cycles*** to cover all states



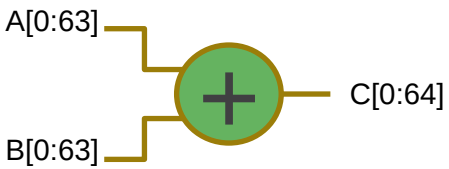
- A bug in the last cycle of the counters would be detected within 3 cycles in one-hot counter but within 15 cycles with binary counter (15 cycles vs. 65535 cycles for 16 bit counters)

→ 2 circuits with same apparent complexity (number of flops, gate count, …) may have completely different functional complexity

What Is Formal Verification

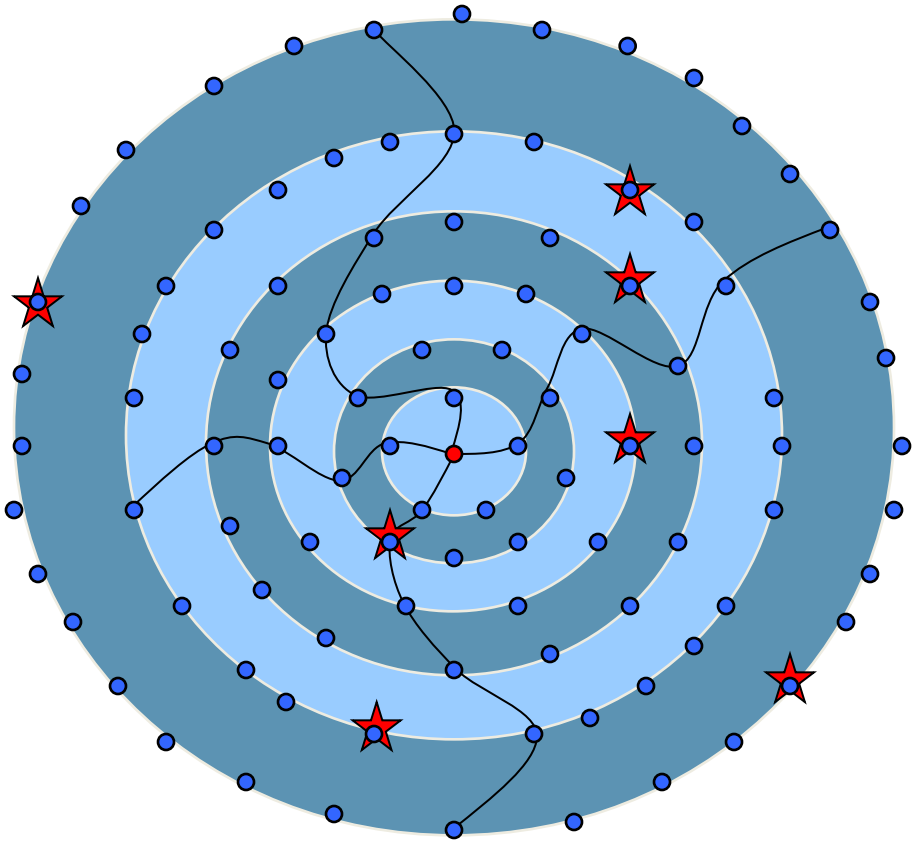
SPYGLASS® FROM ATRENTA

- **Simulation: the traditional approach to find bugs**
  - Write test cases to test particular scenarios that designer/verification engineer has in mind
  - Can only target a small portion of all the possible scenarios since the possible number is extremely large
  - Example: Exhaustively testing a 64-bit adder
    - We have 2^{128} =3.4x10^{38} possible inputs
    - If we simulate each input in 1ns, simulation will take **3.4x10^{29} sec**
    - Age of universe = **4.4x10^{17} sec** !!



- **Formal verification: tackling the limitations of simulation**
  - Systematically searches all possible scenarios WITHOUT explicit user testbenches
  - Mathematically proves or disproves the absence of a bug
  - “No free lunch”: cost of verification is high: takes too much time or too much memory!

Formal Verification is Exhaustive



Simulation

- Depends on number and quality of test vectors
- Uncovers some bugs

Formal Verification

- No testbench vectors needed
- Few cycles of formal depth is equivalent to millions of simulation cycles
- Uncover corner case bugs

How to Verify Clock Domain Crossings?

- **Traditional verification tools target synchronous designs**
  - Functional simulation does not always consider asynchronous nature of domain crossings
  - Static timing analysis does not analyze asynchronous paths

- **CDC verification must target specific problems caused by asynchronous clocks and ensure:**
  1. Signals crossing clock domains are properly **synchronized**
  2. No glitches on Control crossings
  3. Correlated signals crossing clock domains are **Gray encoded**
  4. Signals crossing fast clock domains are **stretched to arrive in time** for target clock with acceptable margin
  5. **Enable control** is implemented so that there is no glitch propagating asynchronously to target clock

Pop Quiz

- **Multi-flop synchronizers can be used to synchronize multi-bit data crossing clock domains**
  a) True
  b) False
  c) True only if no combo logic exists on the crossing path

- **Glitches on synchronized data paths are always safe**
  a) True, since the paths are synchronized
  b) False, glitches are never safe on crossings
  c) False, if glitches from source are not blocked by a qualifier

Agenda

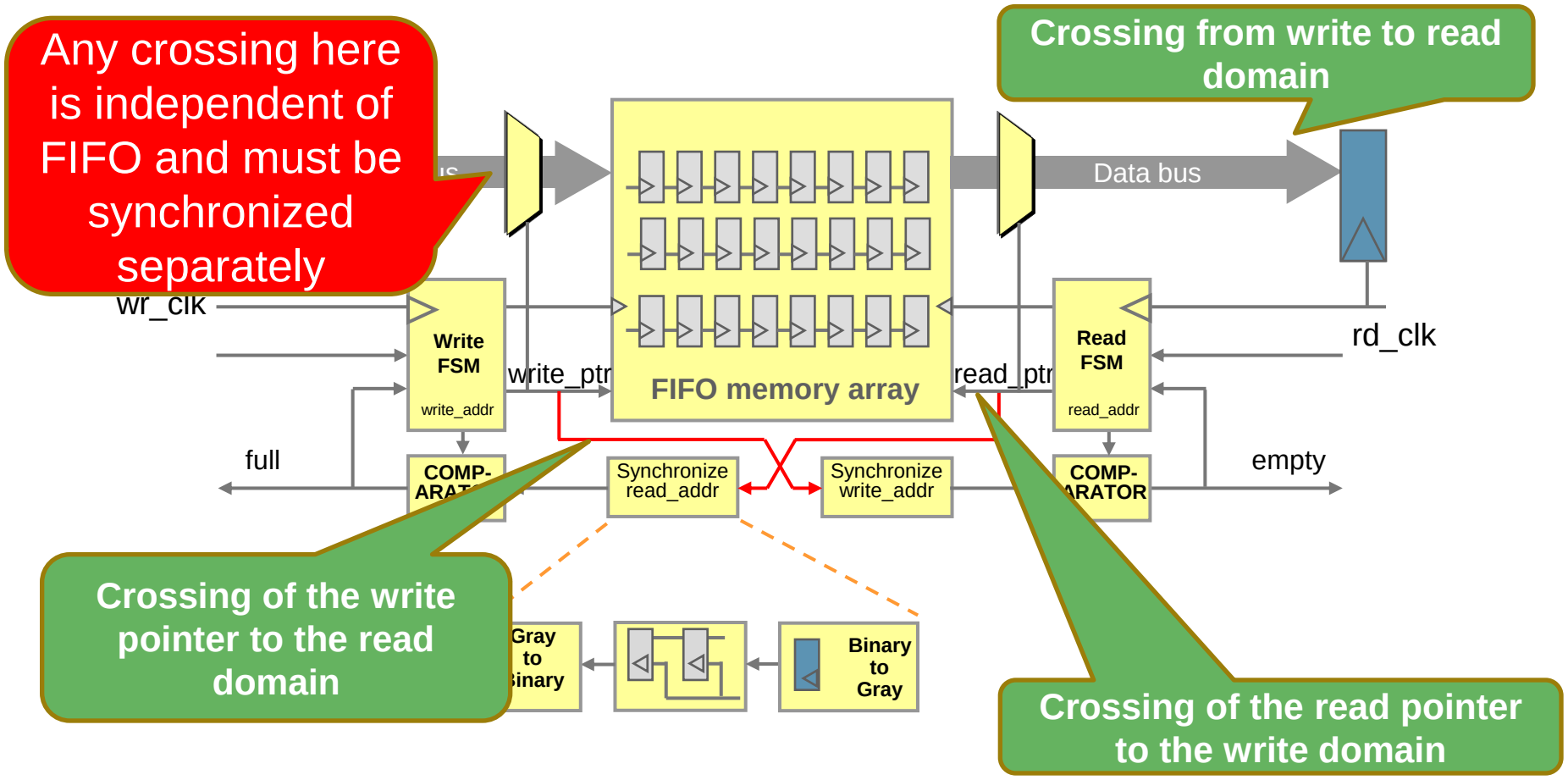
- Meta-stability and System Behavior
- Safe Data Transfer Across Clock Domains
- **Killer Bugs in CDC**
- Good and Bad Design Practices
- Atrenta's SpyGlass CDC Solution
- CDC Verification Challenges and Trends

Killer Design Bug - Example 1

FIFO recognition and verification…

SPYGLASS® FROM ATRENTA

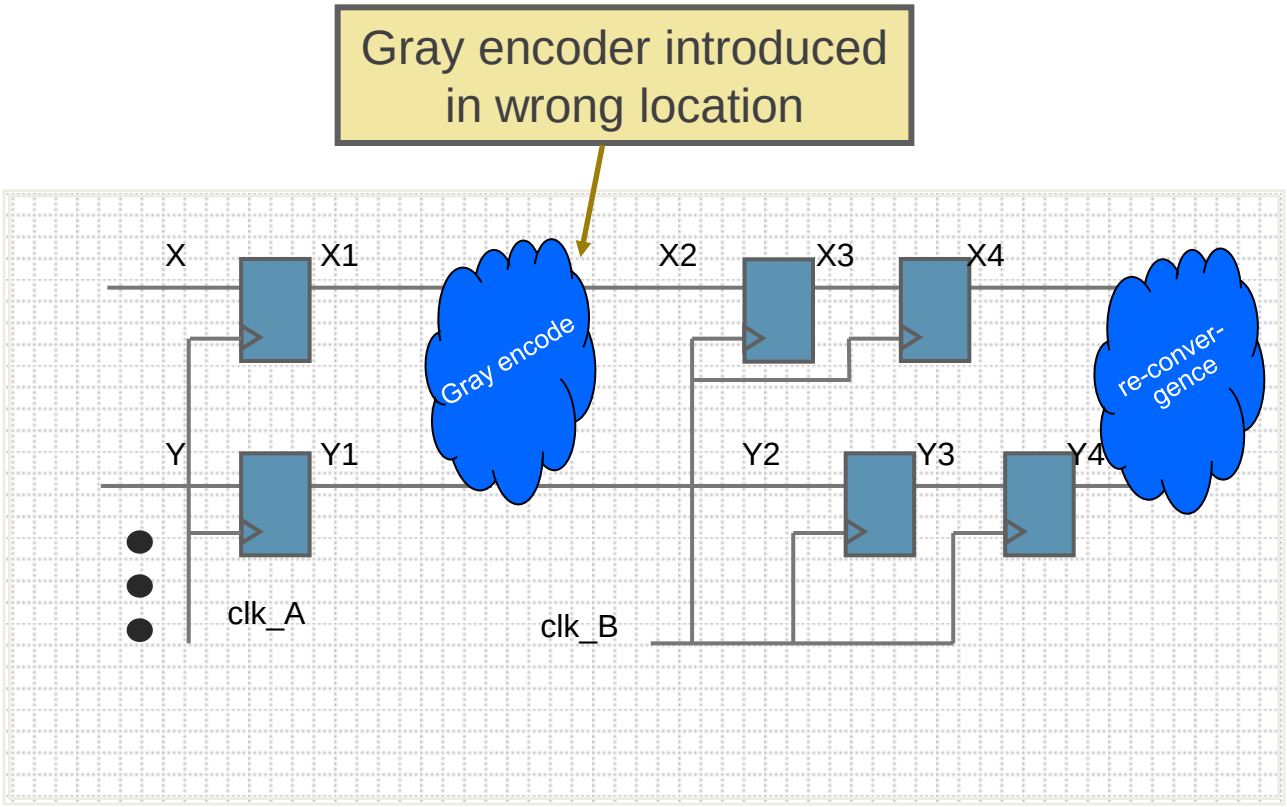
- **Synchronization issue from a source flop to a memory overlooked by a designer**
  - Bug was missed despite CDC verification by a third party tool, due to lack of FIFO analysis



Killer Design Bug - Example 2

Gray encoding check…

- **Gray encoder introduced in wrong location causing data integrity failure**
- **SpyGlass-CDC identified Gray encoding check and reported the issue**



Killer Design Bug - Example 3

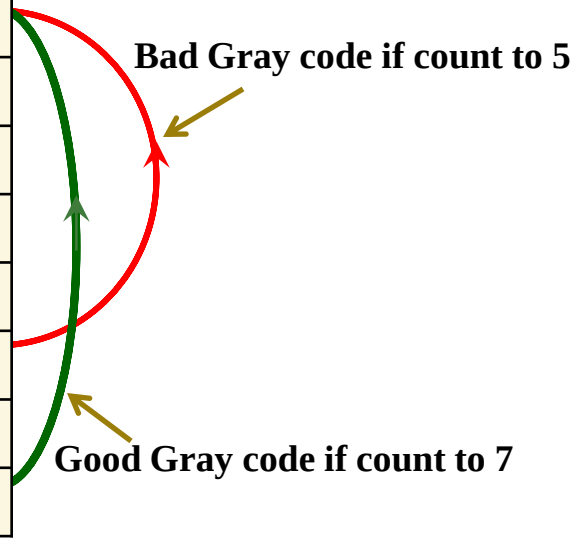
Gray encoding check…

SPYGLASS® FROM ATRENTA

- **Gray counter introduced, but did not match the binary counter pattern!**

FIFO designed to use only 6 locations

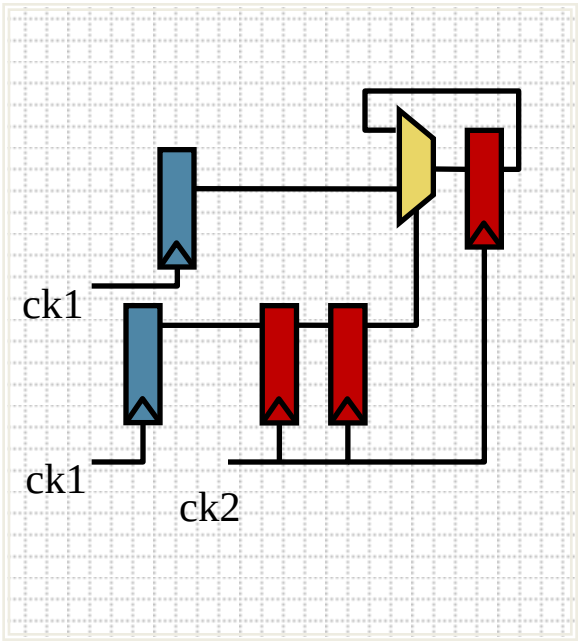
	Binary count	Gray count
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

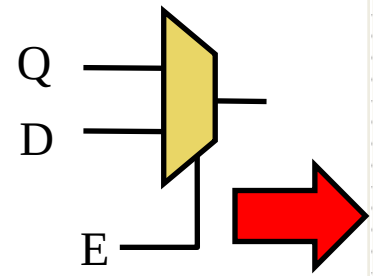


Killer Design Bug - Example 5

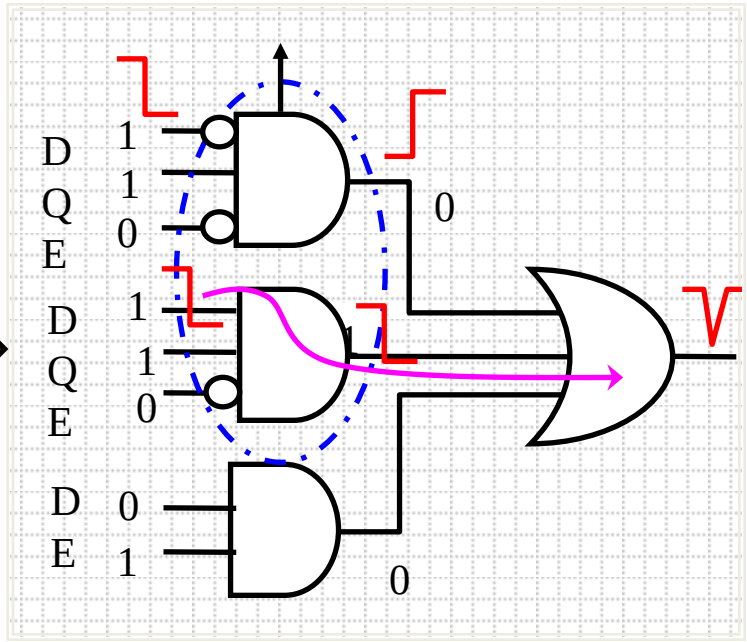
Flawless MUX transformed by synthesis…

- **Implemented correct synchronizer with some logic on the path**
- **Synthesis/optimization tools implemented a glitchy MUX causing design to fail on system**



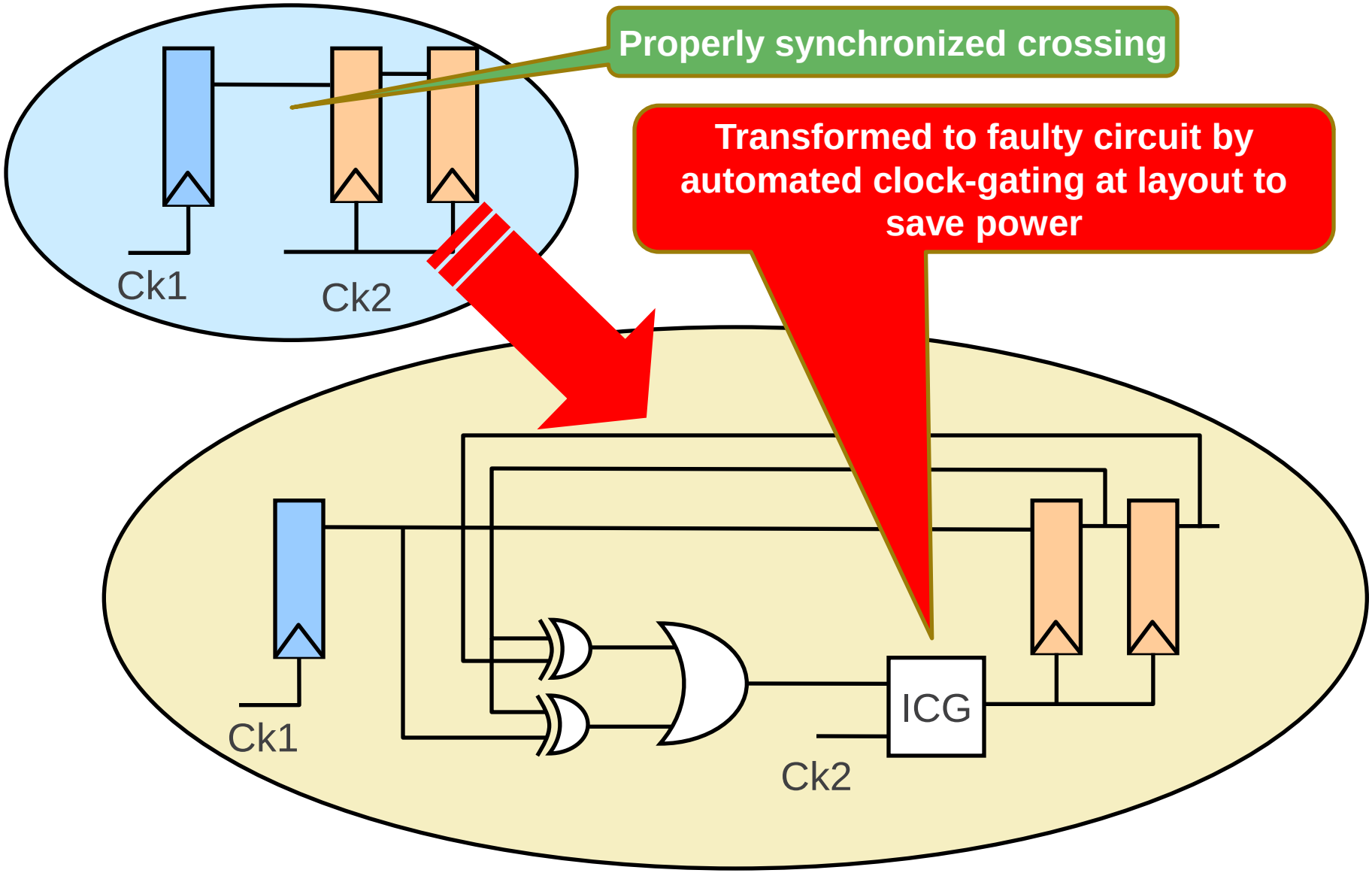


Glitch-free MUX



Killer Design Bug - Example 6

Clock-gating effects synchronization…



Pop Quiz

- **Reading from an empty FIFO can cause metastability**
  a) True
  b) False
  c) False, but causes functional failure due to read of uninitialized memory

- **Synthesis tools cannot break synchronizers**
  a) True, because synthesis tools preserve functionality of the design
  b) False, because they can transform non-glitchy logic into glitchy logic
  c) False, because retiming can bring combinational logic between synchronizer flops

Agenda

- Meta-stability and System Behavior
- Safe Data Transfer Across Clock Domains
- Killer Bugs in CDC
- **Good and Bad Design Practices**
- Atrenta's SpyGlass CDC Solution
- CDC Verification Challenges and Trends

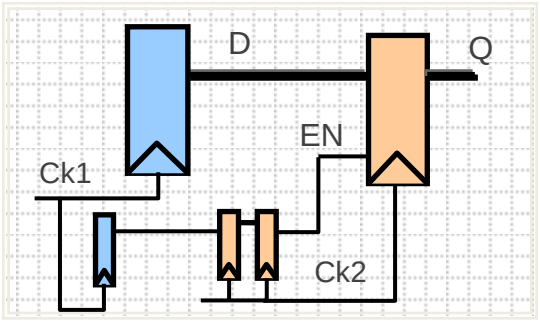
Design Guidelines for Control Synchronization

SPYGLASS® FROM ATRENTA

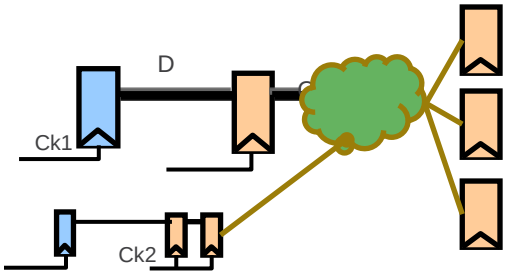
- **Build simple synchronizers and enforce usage across projects/designs**
  - E.g. simple module f2sync, f3sync, for 2 flop and 3 flop synchronizers
  - Can use SpyGlass CDC to enforce usage of specific synchronizers based on clocks or clocks frequencies
- **Never add combinational logic on the crossing**
  - You can always bring logic in either source domain or destination domain, avoid logic on the crossing
  - SpyGlass enforces this and do not waive such issues
- **Never add combinational logic between multi-flop synchronizers**
  - Combo logic defeats the purpose of synchronizer by extending metastability
  - If you enforce usage of sync cells you can prevent this issue
- **Document/comment your synchronization intents**
  - Name your synchronizers consistently
  - Name your static signals with a same naming convention across designs
  - Mark which control signal is intended for which data crossings/protocol

Design Guidelines for Data Synchronization

- **KEEP IT SIMPLE (KIS) - It will save your chip!**
  - Keep your control close to the destination data
  - Push combination logic out of the crossing



- **Avoid computation on a data crossing**
  - You never know what synthesis will do to your logic
  - Can cause glitch
  - Do all your computation prior to the crossing, or push it to destination domain
- **Use simple enable logic**
  - Enabled flop, or recirculation MUX should be all what you need
- **Avoid use of multiple control logic for a same data crossing**
  - You can again combine all the logic at the source and create a simple control logic
- **Do not delay synchronization**
  - An FPGA company fixed 3 bugs in a same crossing because of delayed synchronizers
  - You are deliberately creating metastability
    - Someone else can touch logic after crossing



Design Guidelines for Data Synchronization

SPYGLASS® FROM ATRENTA

- **Chose and adopt common synchronizers**
  - Chose FIFOs for faster throughput but avoid customization if possible(1)
    - Always use empty flag to gate reading from the FIFO memory
    - Try to not mix data and control in the memory
  - If latency is not an issue you can use handshake
    - Again go for a standard full-req/ack handshake
    - Do not go for pulse generation if you do not have too (if you want low latency go for FIFO)

- **Avoid special synchronizers**
  - Source synchronous designs – must obey to many rules to work properly
  - Special synchronizers to deal with special frequency ratios

(1) Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons – Clifford E. Cummings

Global Convergences

SPYGLASS® FROM ATRENTA

- **Avoid convergences if possible**
  - Can the computation causing convergence be pulled in the source domain?
- **Document your convergences**
  - Being aware of convergences means you have probably taken care of them
  - Someone else will not make mistake while changing design
- **Document why you think convergences are OK – two possible sources**
  - Sources of convergences are not toggling together
  - A protocol will stop propagation of all sources but one
- **Document synchronizers reaching boundary of your blocks**
  - Convergences may lead to coherency issues

Agenda

SPYGLASS® FROM ATRENTA

- Meta-stability and System Behavior
- Safe Data Transfer Across Clock Domains
- Killer Bugs in CDC
- Good and Bad Design Practices
- **Atrenta's SpyGlass CDC Solution**
- CDC Verification Challenges and Trends

Highlight of Major Features

SPYGLASS® FROM ATRENTA

Most Comprehensive CDC Verification Solution

- In depth analysis and classification of synchronization, convergences, glitches, …
- Verifies glitches on clock and resets, assumption validation flow, power-aware CDC verification, delta-delay, and over 100 different checks
- Convergence found at any depth without user intervention

Advanced Synchronization Verification Techniques

- Low noise – detects static candidates, supports user defined qualifiers, …
- Easier debug – provides root cause of a failure and show in schematic and RTL

Ease of Setup

- Comprehensive SDC support, lib support, constraint generation from block or top

Integrated Structural and Functional CDC Verification Flow

- Just turn on the functional verification built-in the flow – no need for user testbench
- Assertion generation and assumption validation capabilities

Performance

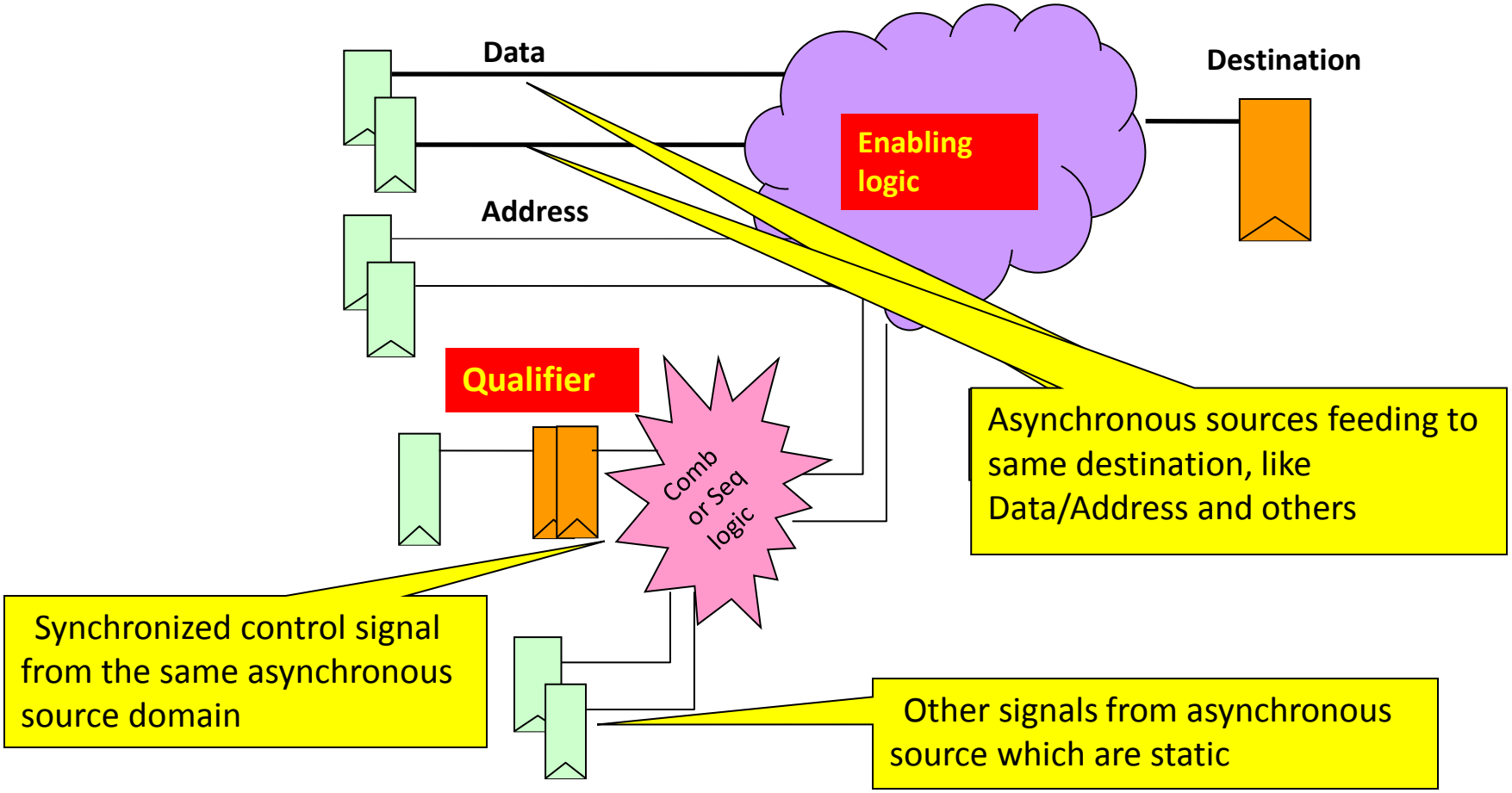
- Very efficient algorithms tested at 150+ customers

Full RTL platform

- You can verify CDC, Lint, Constraints, DFT, … from a same environment, just choose and run

SpyGlass CDC Protocol-independent CDC Verification (Ac_sync)

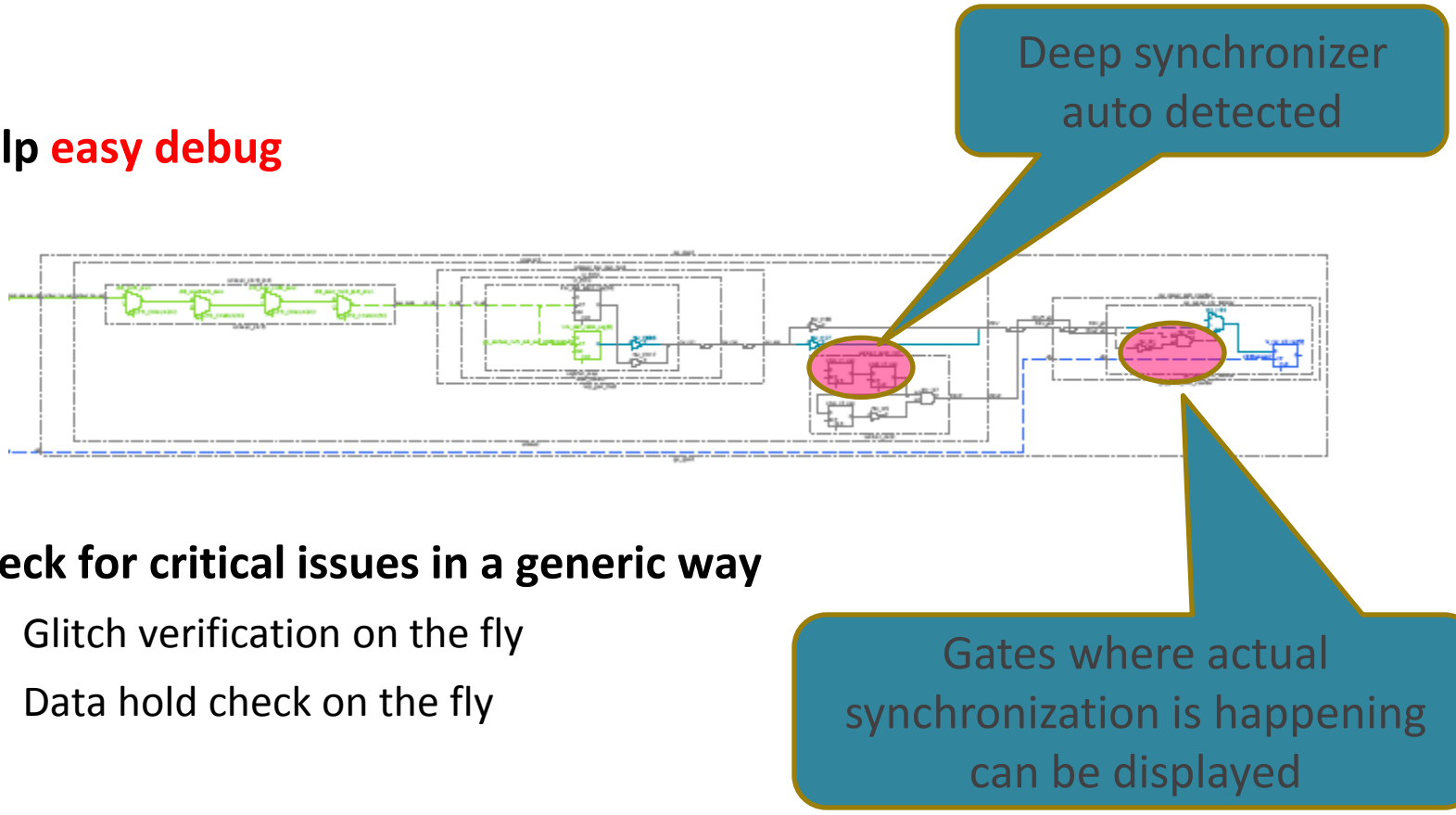
- A global analysis that considers data, address, control lines
- Identifies generic synchronization elements as opposed to rigid structures



Benefits of Protocol-independent Approach

SPYGLASS® FROM ATRENTA

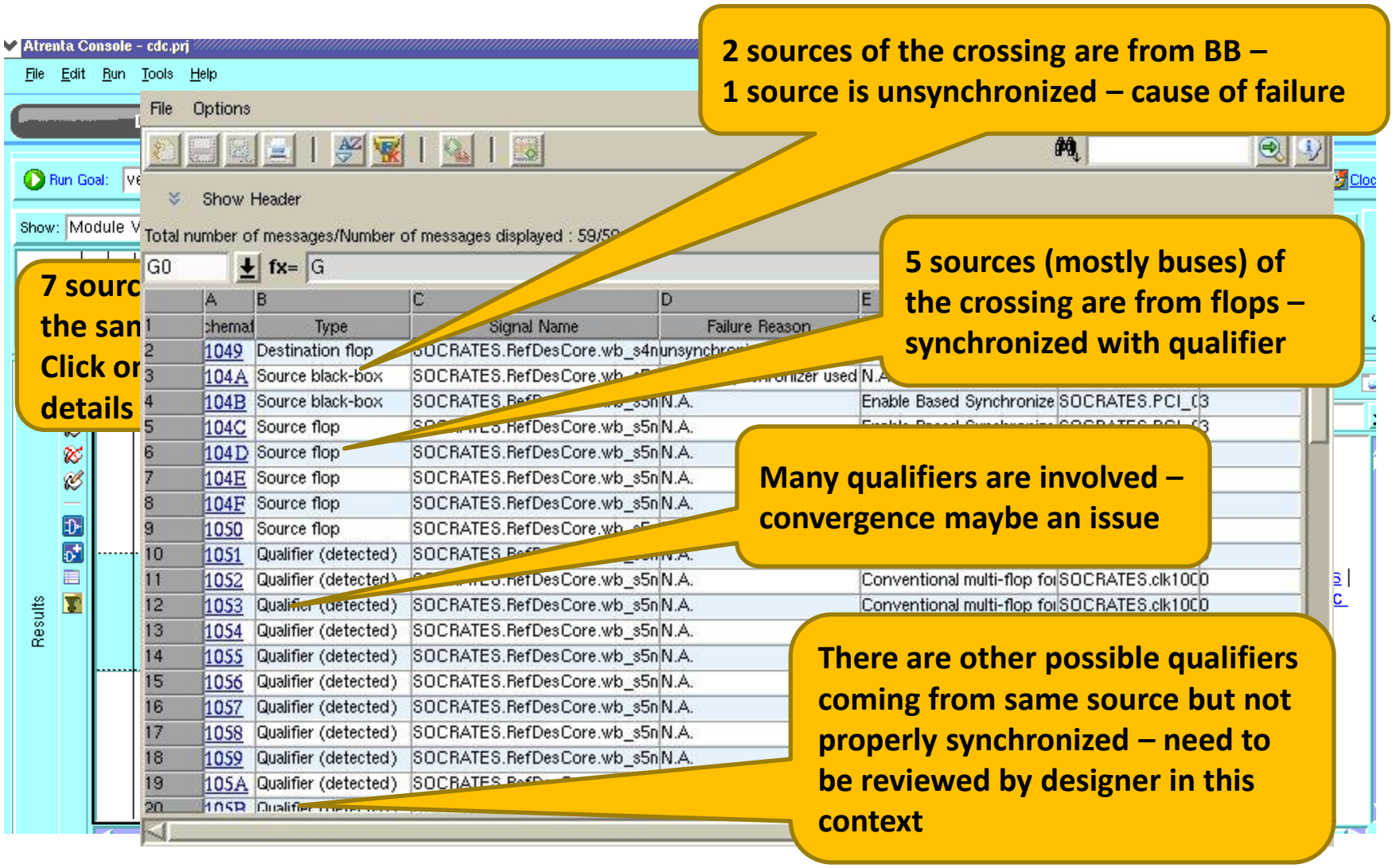
- **Reduce False violations**
  - Identifies custom FIFO/handshake synchronizers
  - More immune to design structure → similar result on netlist/RTL
- **Help easy debug**



- **Check for critical issues in a generic way**
  - Glitch verification on the fly
  - Data hold check on the fly

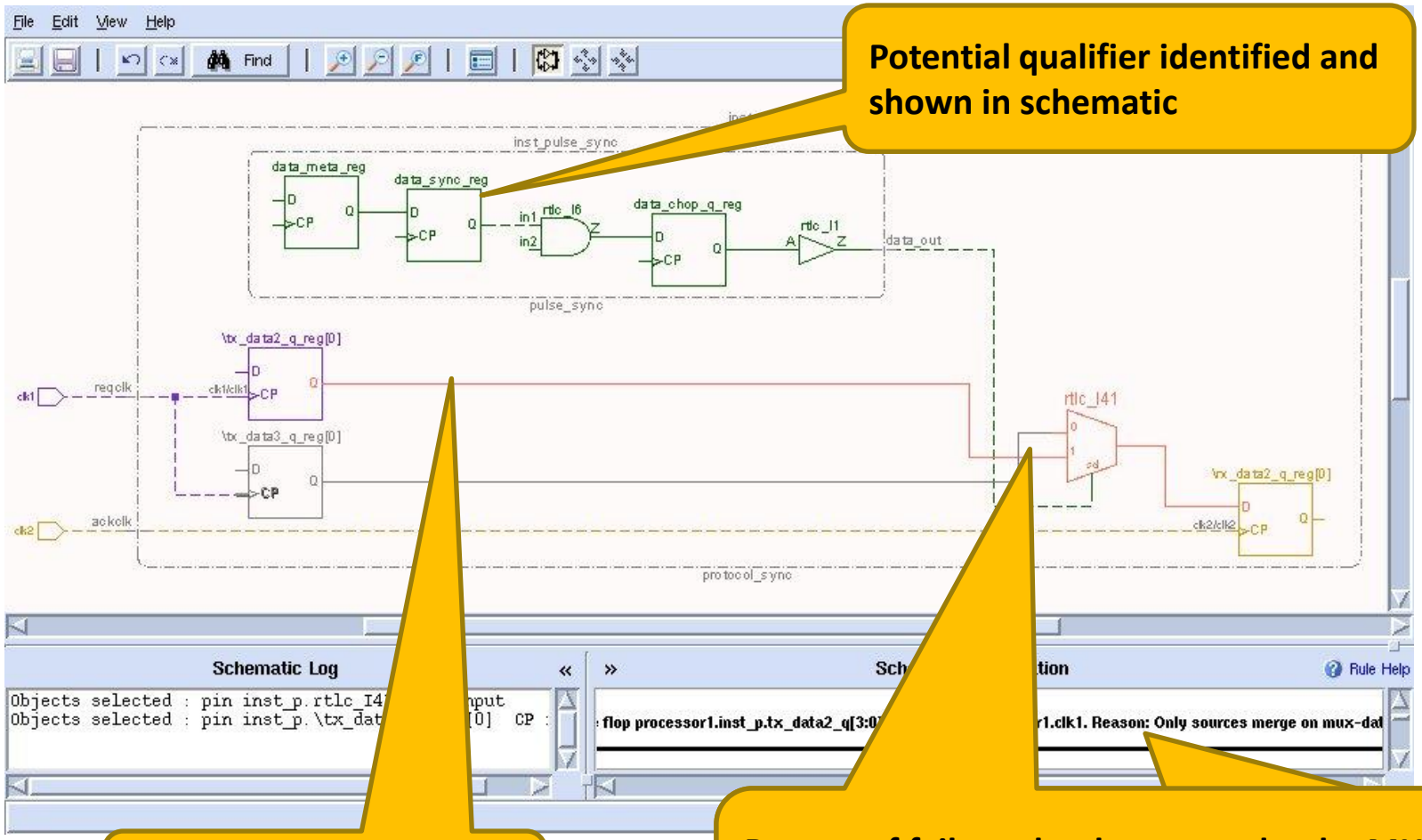
Easy To Explore CDC Results

SPYGLASS FROM ATRENTA



	A	B	C	D	E
1	chemat	Type	Signal Name	Failure Reason	
2	1049	Destination flop	SOCRATES.RefDesCore.wb_s4n	unsynchroni	
3	104A	Source black-box	SOCRATES.RefDesCore.wb_s5n	ronizer used	N.A.
4	104B	Source black-box	SOCRATES.RefDesCore.wb_s5n	N.A.	Enable Based Synchronize SOCRATES.PCI_C3
5	104C	Source flop	SOCRATES.RefDesCore.wb_s5n	N.A.	
6	104D	Source flop	SOCRATES.RefDesCore.wb_s5n	N.A.	
7	104E	Source flop	SOCRATES.RefDesCore.wb_s5n	N.A.	
8	104F	Source flop	SOCRATES.RefDesCore.wb_s5n	N.A.	
9	1050	Source flop	SOCRATES.RefDesCore.wb_s5n		
10	1051	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
11	1052	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	Conventional multi-flop fo SOCRATES.clk10C0
12	1053	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	Conventional multi-flop fo SOCRATES.clk10C0
13	1054	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
14	1055	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
15	1056	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
16	1057	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
17	1058	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
18	1059	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
19	105A	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	
20	105B	Qualifier (detected)	SOCRATES.RefDesCore.wb_s5n	N.A.	

Detailed CDC Failure Analysis

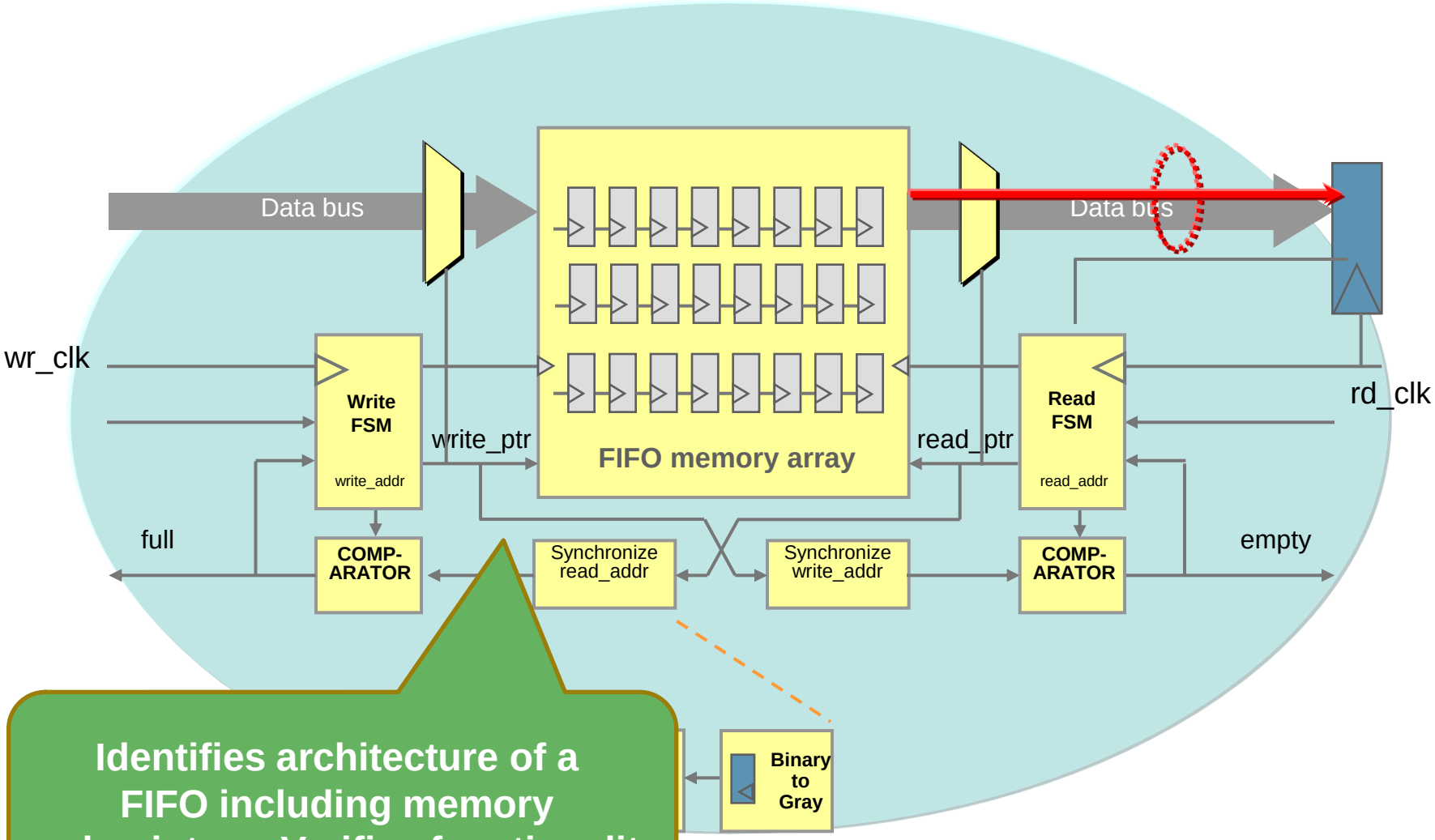


Potential qualifier identified and shown in schematic

Unsynchronized crossing shown in red

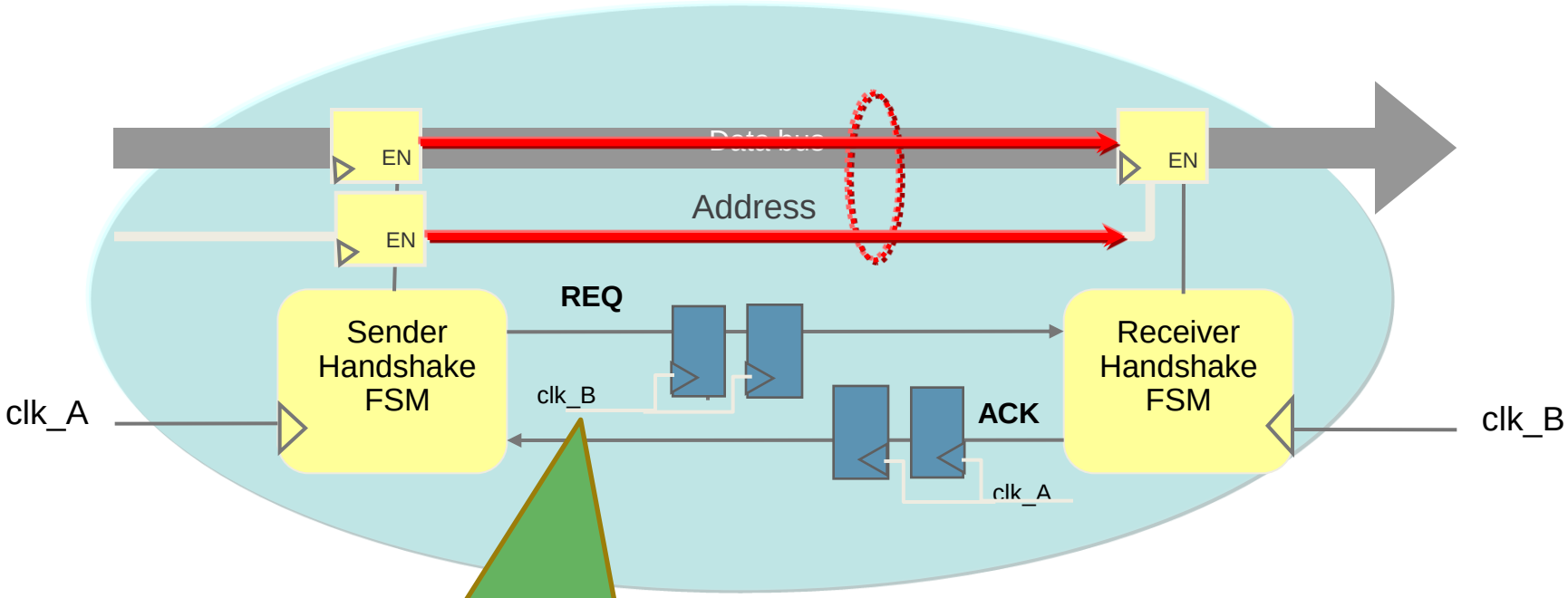
Reason of failure clearly reported – the MUX will select an asynchronous source regardless of the "potential qualifier value"!

Automatic FIFO Recognition & Verification



Identifies architecture of a FIFO including memory and pointers; Verifies functionality

Automatic Handshake Protocol Verification



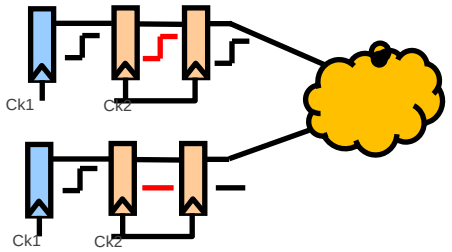
Identifies complex data transfer protocols; Verifies functionality

Data Coherency Across Clock Domains

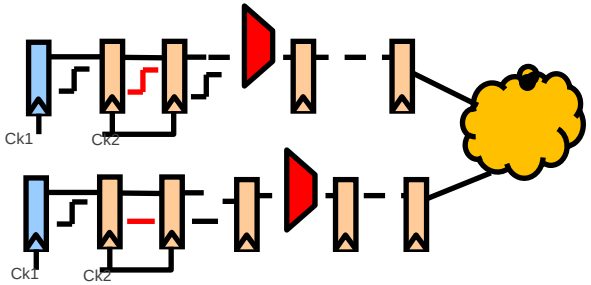
SPYGLASS® FROM ATRENTA

- **Designers take different approaches in avoiding coherency issues, so should verification solutions**

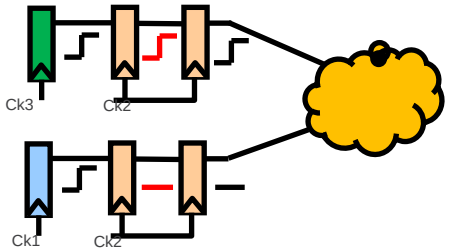
1. **Combinational convergence – Ac_conv02**
   - Designed so that only one source change at a time
   - Gray counter, static signals, gray state machine…
   - Verify that two sources do not change simultaneously



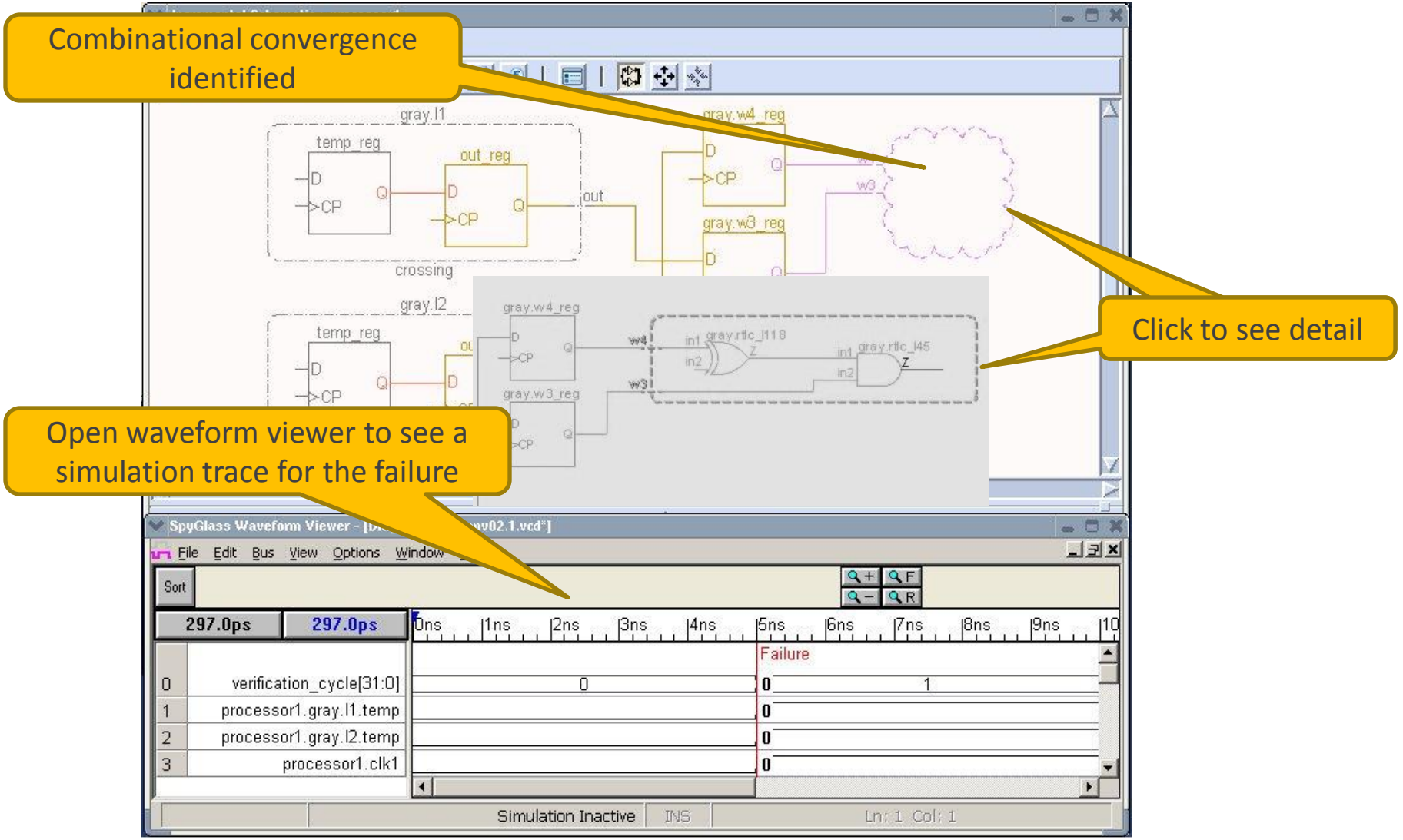
2. **Sequential convergence – Ac_conv01**
   - Protocol is designed to avoid coherency problems
   - Typically a control logic ensures one path active only
   - Designers must review for correctness



3. **Convergence from different domains – Ac_conv03**
   - Typically avoided
   - If present, one source is static
   - Designers must review for correctness

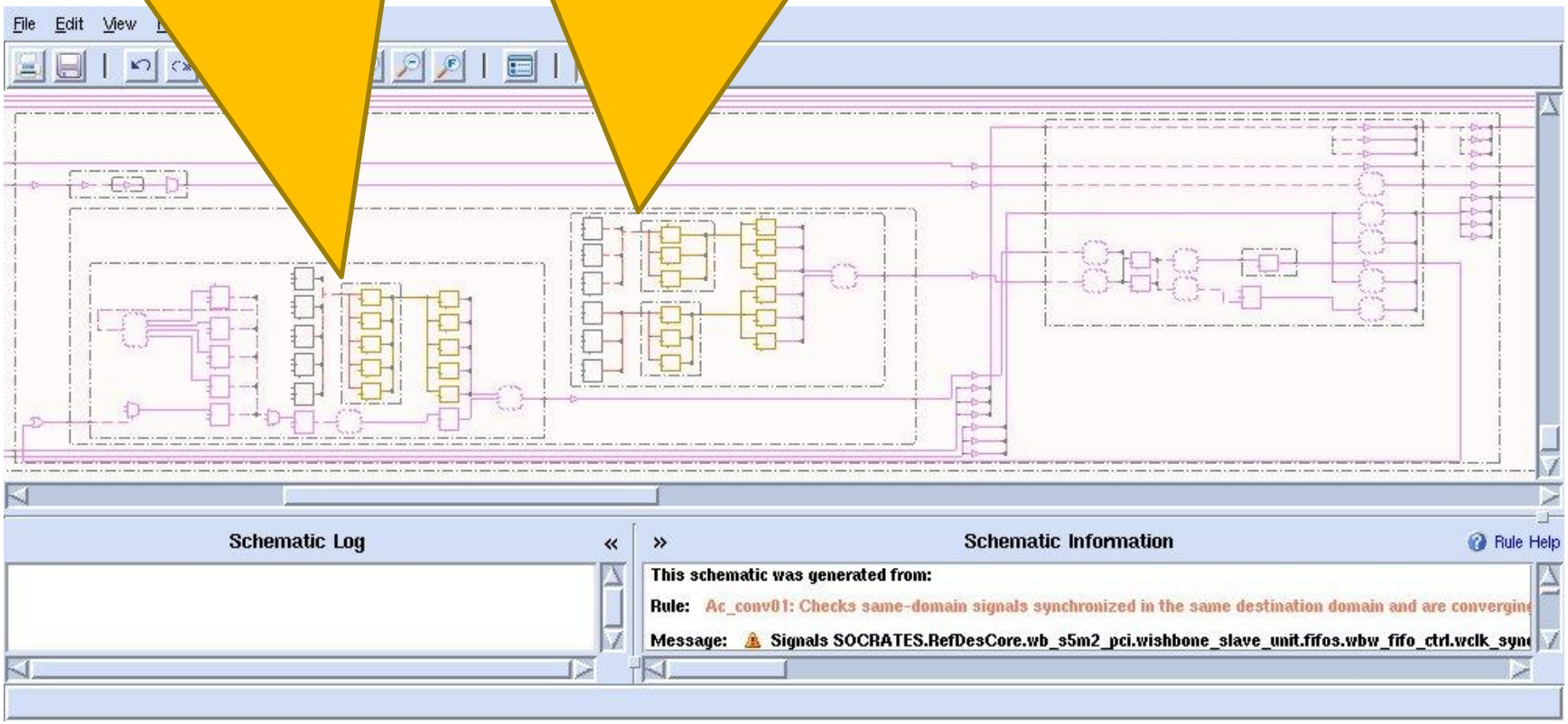


Combinational Convergence



Sequential Convergence

Two different set of synchronized signals converging after layers of sequential and combination logic

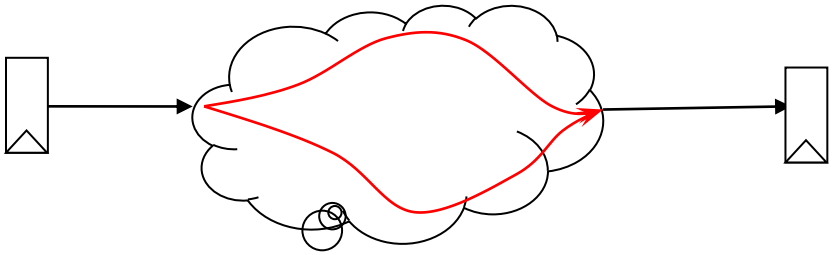


- **Sequential convergences often created by IP integration can be unknown to designers → can cause chip failure**
- **SpyGlass CDC identifies all sequential convergences at any depth**

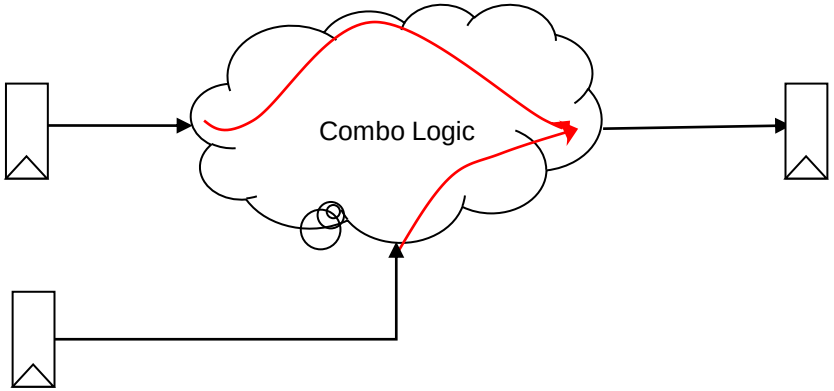
Static analysis of glitches

SPYGLASS FROM ATRENTA

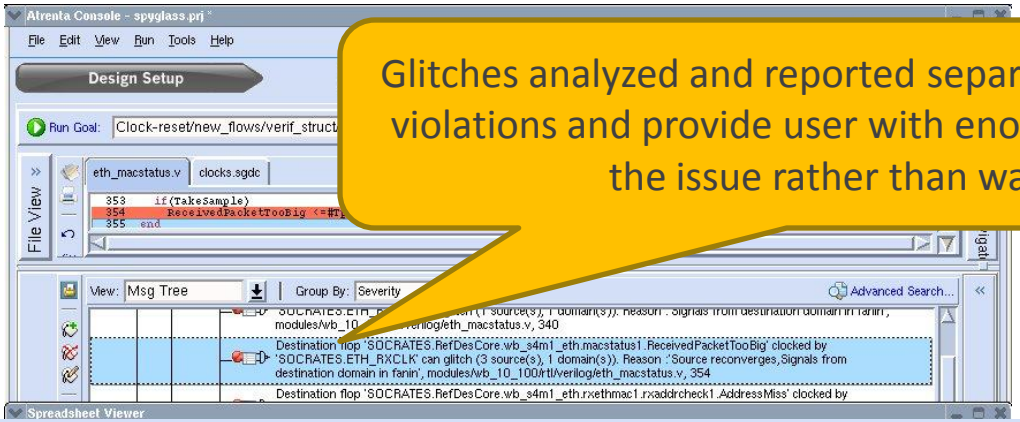
- **Single source glitch**
  - Cause: reconvergence
  - Solution: Remove reconvergence from design, or make it unate reconvergence



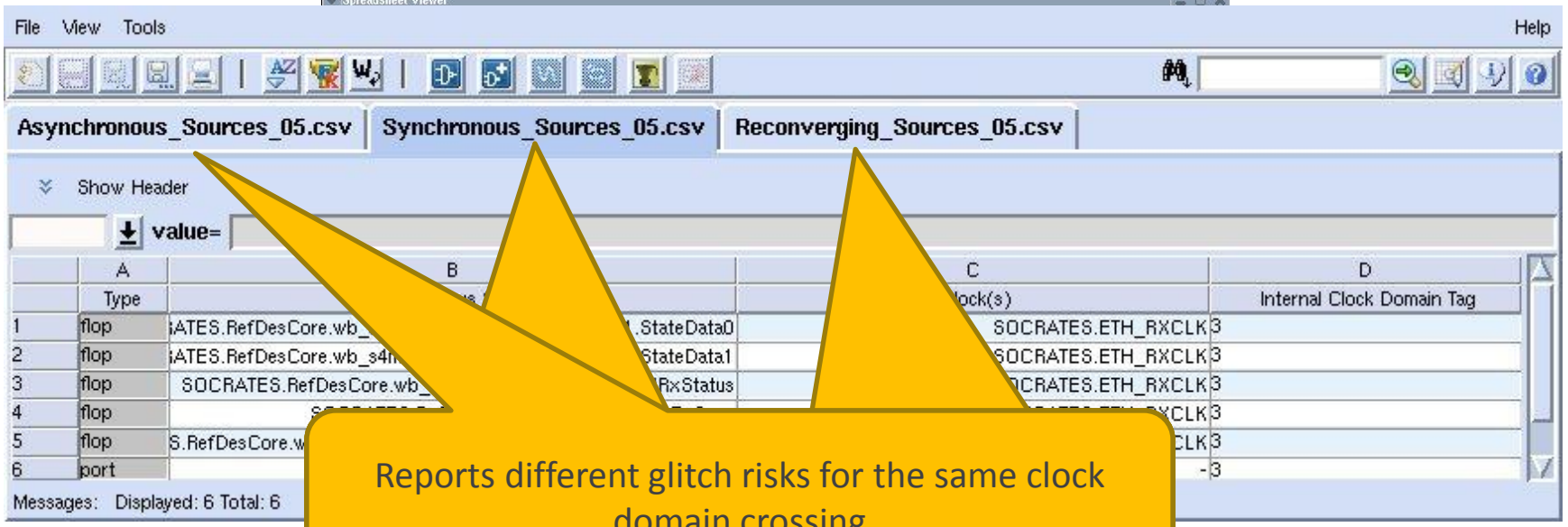
- **Multi-source reconvergence**
  - Cause: two sources toggling at the same time
  - Solution: Make sure one source toggle at a time
    - Some sources may be Quasi-statics
    - Sources are exclusive/gray-coded with adequate timing



Glitch Verification Using SpyGlass CDC

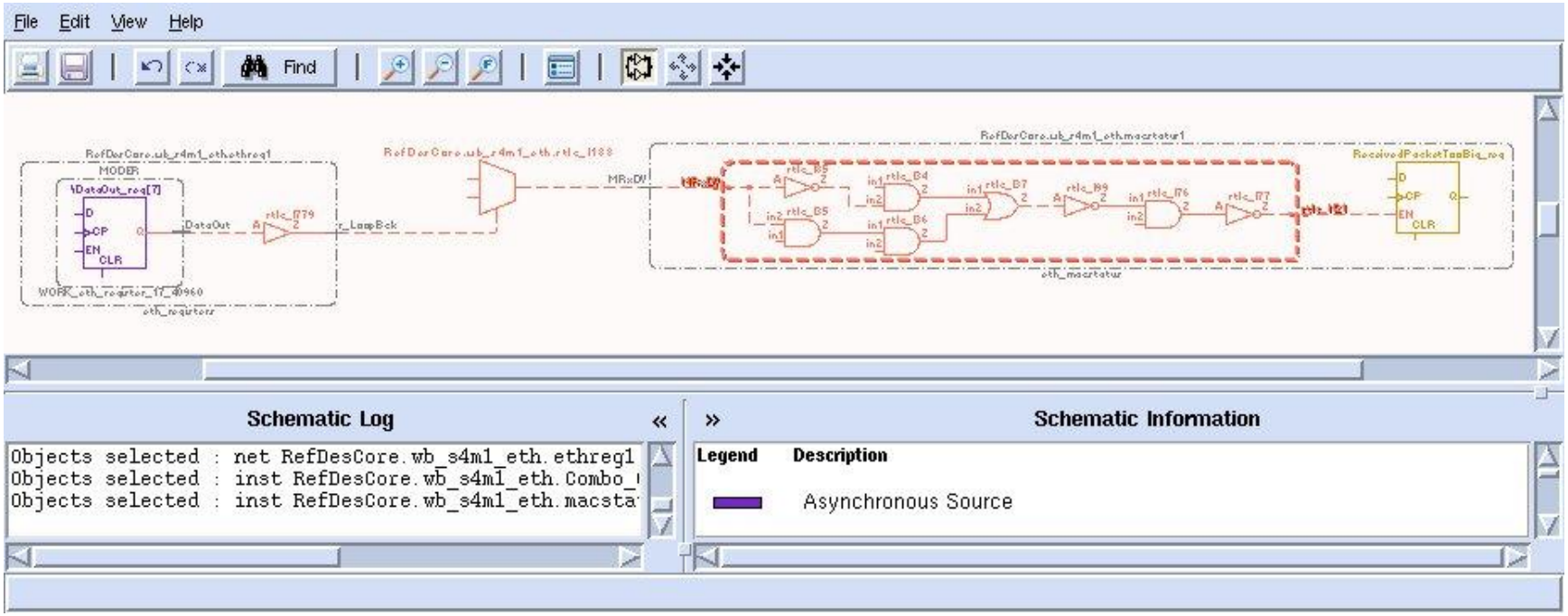


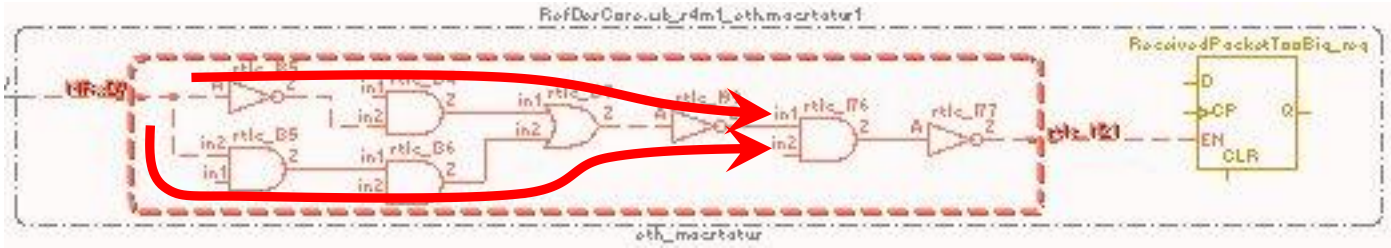
Glitches analyzed and reported separately to reduce false violations and provide user with enough data to address the issue rather than waiving it



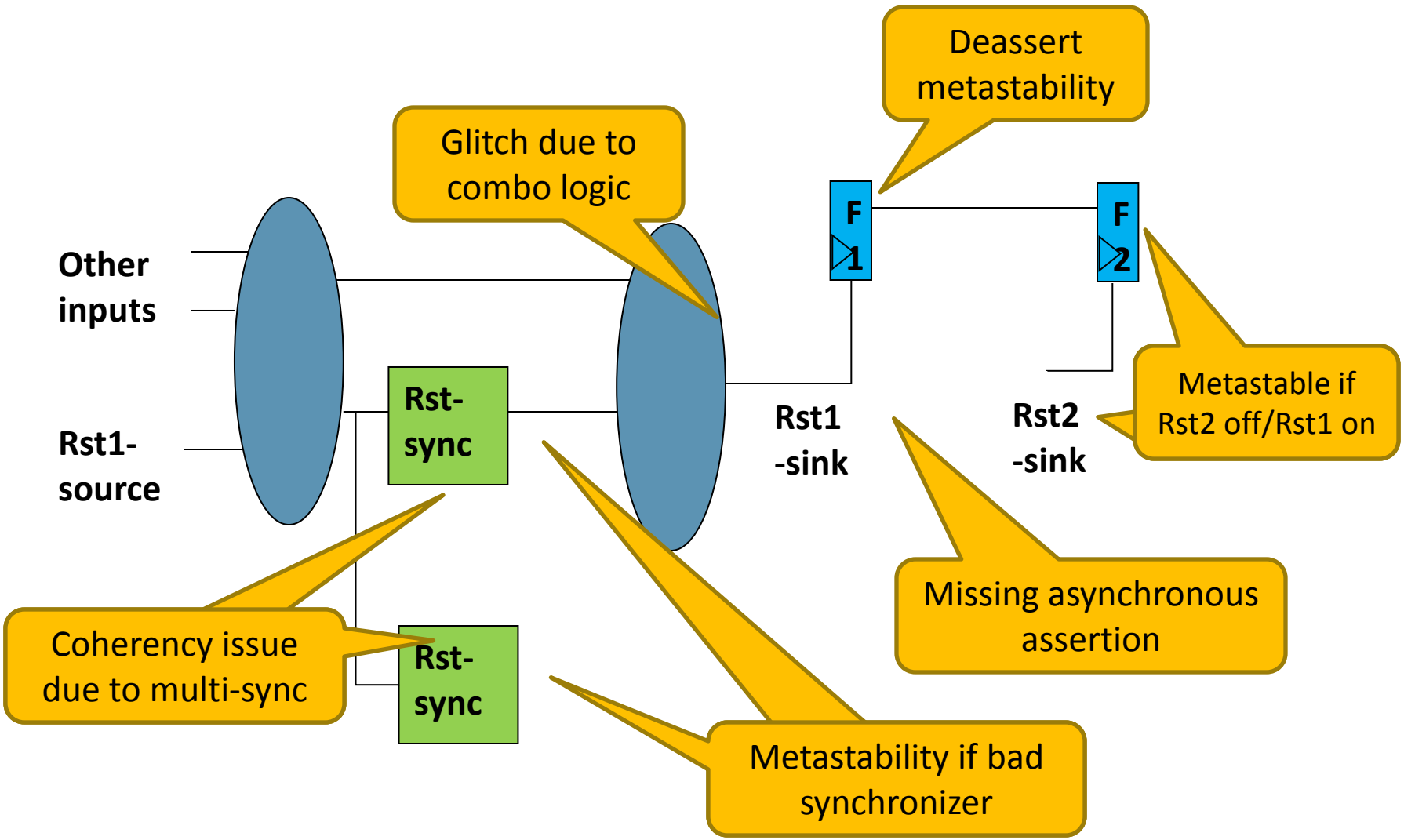
Reports different glitch risks for the same clock domain crossing

SpyGlass CDC Reports Glitches Due To Single and Multi-source Re-convergence

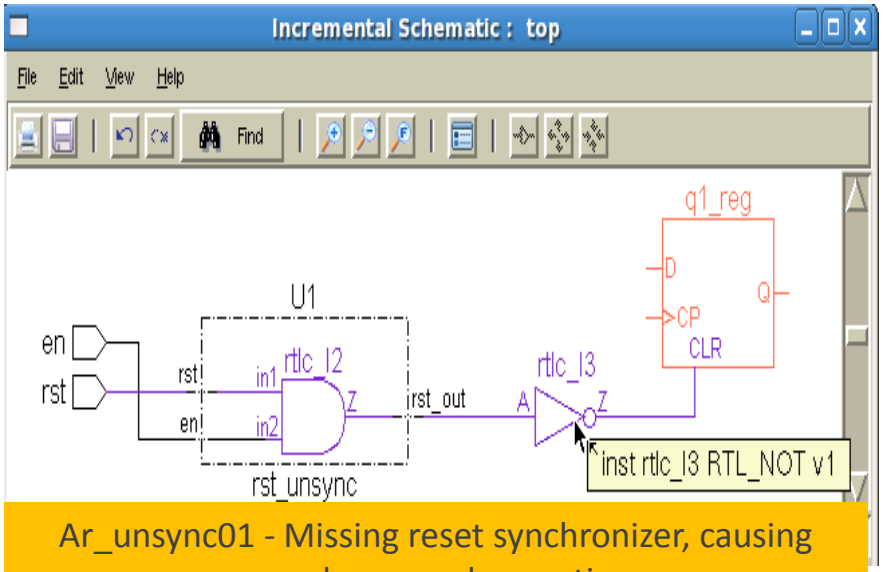




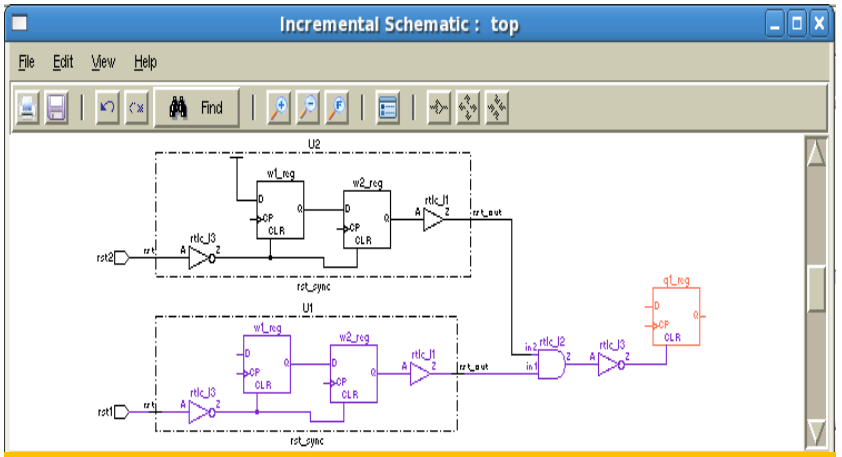
Reset Issues verified by SpyGlass



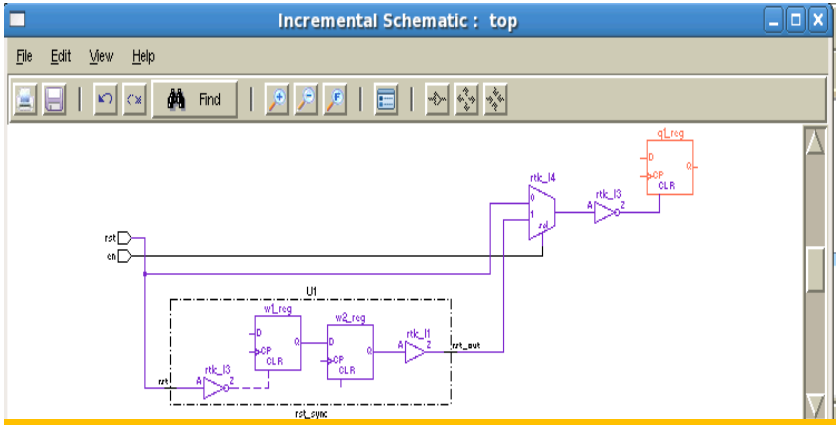
SpyGlass® CDC Reset Checks



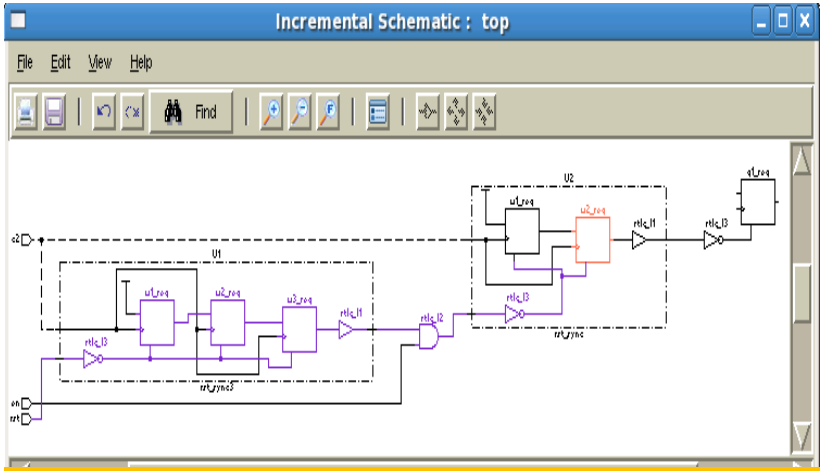
Ar_unsync01 - Missing reset synchronizer, causing asynchronous deassertion



Ar_sync01 - Reset is properly synchronized and deassert synchronously, avoid false violation



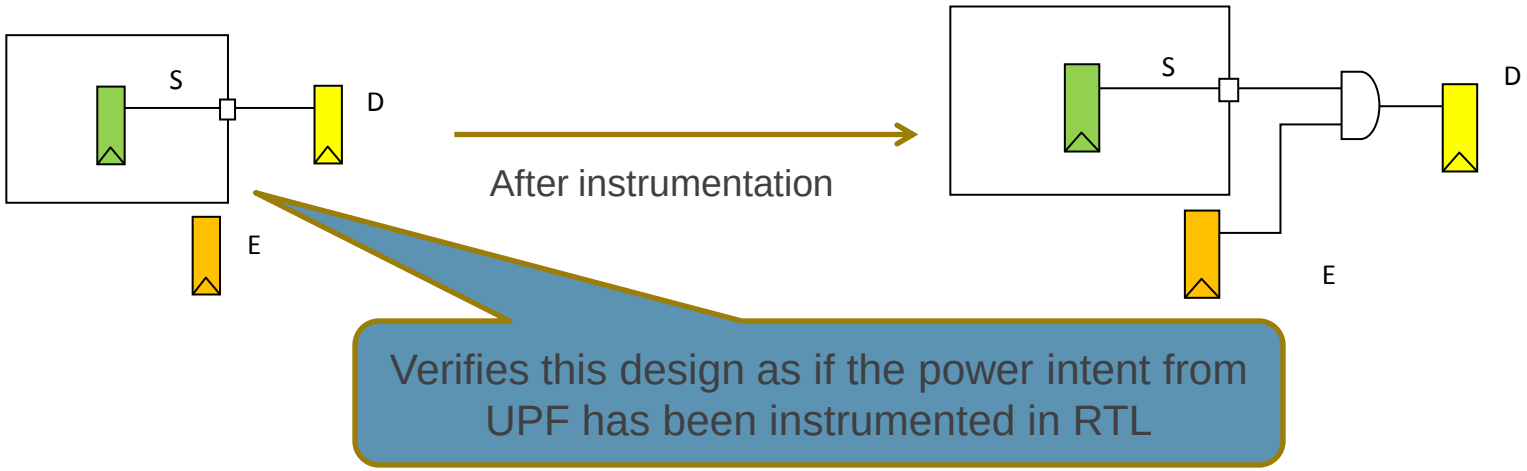
Ar_asyncdeassert01 - Reset properly synchronized but may not deassert synchronously!



Ar_syncdeassert - Complex reset synchronous, avoid false violation

Power-aware CDC Verification

- **Added support to verify non-instrumented RTL where power intent is captured in UPF**
  - Enables early verification of CDC issues around isolation logic at RTL level
  - Performs synchronization check on isolation logic before the logic is introduced in the design
  - The Ac_psync01/Ac_punsync01, Ac_psetup01, and Ac_upfsetup01 rules report the synchronized and unsynchronized crossings as well as setup issues



Agenda

SPYGLASS® FROM ATRENTA

- Meta-stability and System Behavior
- Safe Data Transfer Across Clock Domains
- Killer Bugs in CDC
- Good and Bad Design Practices
- Atrenta's SpyGlass CDC Solution
- CDC Verification Challenges and Trends

Challenges in Effective CDC Bugs

- **Evolving design and synchronization styles**
  - NoC, Source synchronous, …
  - False errors generated by tools prevent effective verification closure
  - Each CDC issue reported should end in a design fix
- **CDC verification is not always performed by the designer**
  - RCA of issues reported are as important as the verification itself
  - Disposing a CDC issue reported by tools should take seconds/minute
- **Design size and complexity are growing**
  - Tools must handle 100+M gates design, RTL as well as netlist
  - Reasonable run time/memory is key to successful CDC verification closure
- **Constraints and CDC are no longer disjoint problems**
  - Consider correct constraint and design creation simultaneously
- **Functional verification challenges**
  - Concluding formal verification (Partial proof issue)
  - Hybrid CDC verification, coverage driven…

Spectrum of CDC Verification Solutions at SoC

SPYGLASS® FROM ATRENTA

Flat Verification

- Concept - Verifying SoC as another block
- Pros
  - Simple Methodology - less effort for setup and Verification
  - Accurate results
- Cons
  - Not scalable, time and number of violations
  - Doesn't always match design flow where blocks and SoC verified are developed by different teams

Preferred use model depending on:
- Tool capacity
- Readiness of design/constraints
- Number of clocks/crossings

Hierarchical Bottom Up Verification

- Concept – Verifying Blocks and leveraging analysis at SoC
- Pros
  - Scalable solution covering virtually any size design
  - Match design flows where blocks are developed and verified b
- Cons
  - Requires iterations - the issues found in block and SoC level need to circulate through the bottom up flow
  - No functional checks

Use model: Distributed
- Block dev/verif happens first
- Disciplined designers
- Very large SoCs

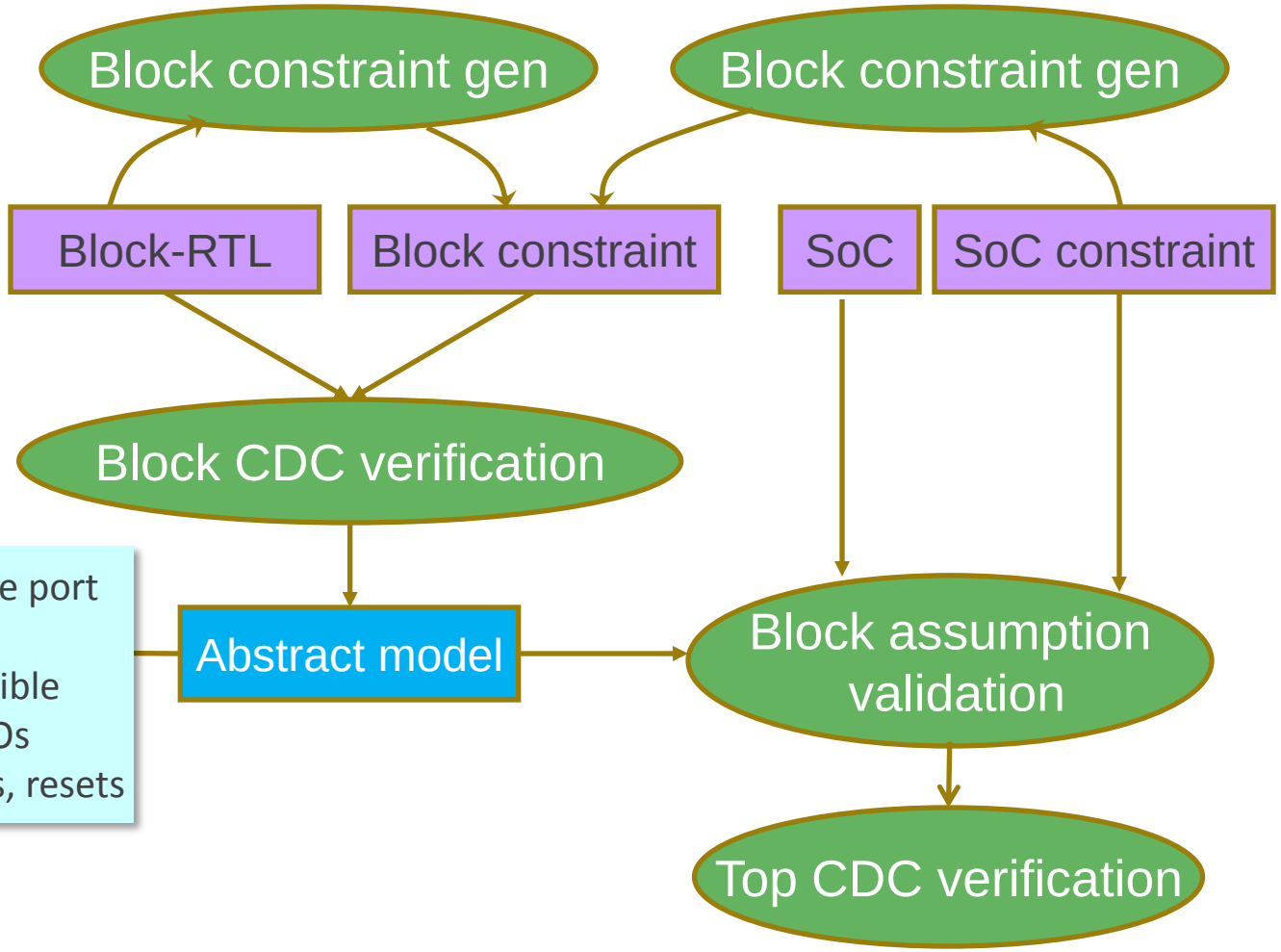
Hierarchical Top Down Flow

- Concept – Often referred to top down constraint mig
- Pros
  - Helps SoC integrator who "do not care" about the block
- Cons
  - High risk of missing bugs, unless use top constraint to perform full block verification

Use model: Centralized
- Blocks are not yet verified and constraints not available
- SoC integrator fully responsible for CDC

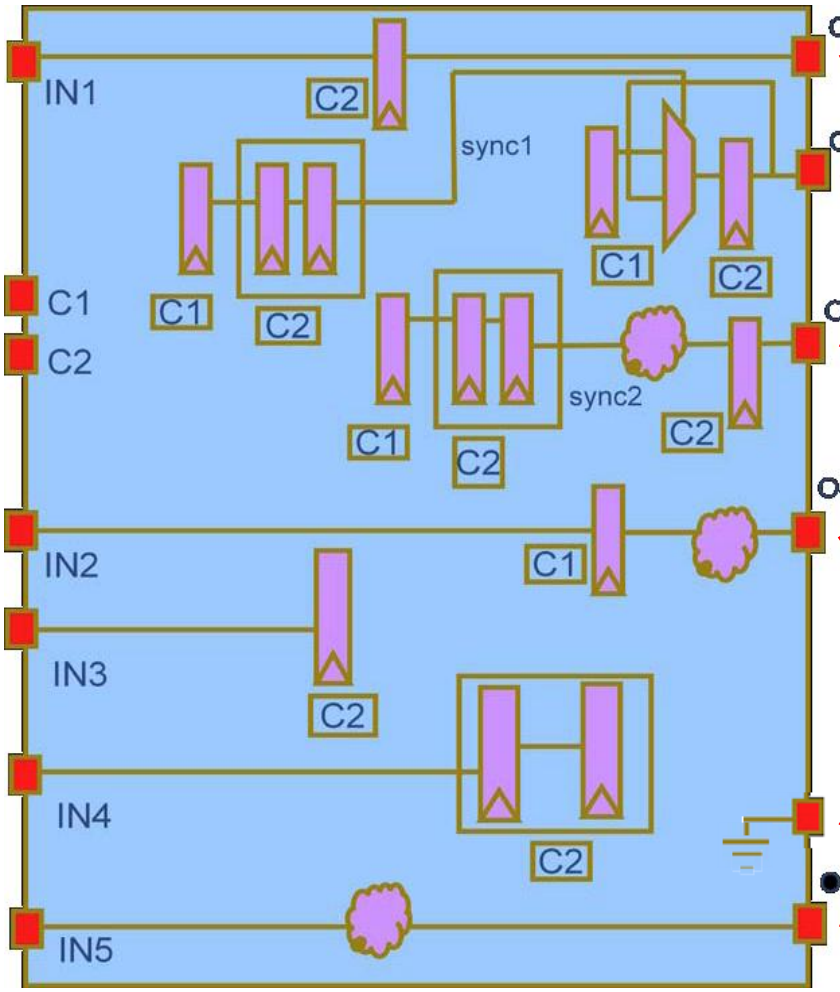
Bottom-up CDC Verification Methodology

SPYGLASS FROM ATRENTA



Fast & Efficient Chip Level CDC Signoff

Example Block Abstraction




```
abstract_port -module <> –ports O1
              –clock C2
              -inputs IN1
```

```
abstract_port –module <> –ports O2
              –clock C2
              –sync inactive  –from C1 –to C2
              –names “<>.sync1”
```

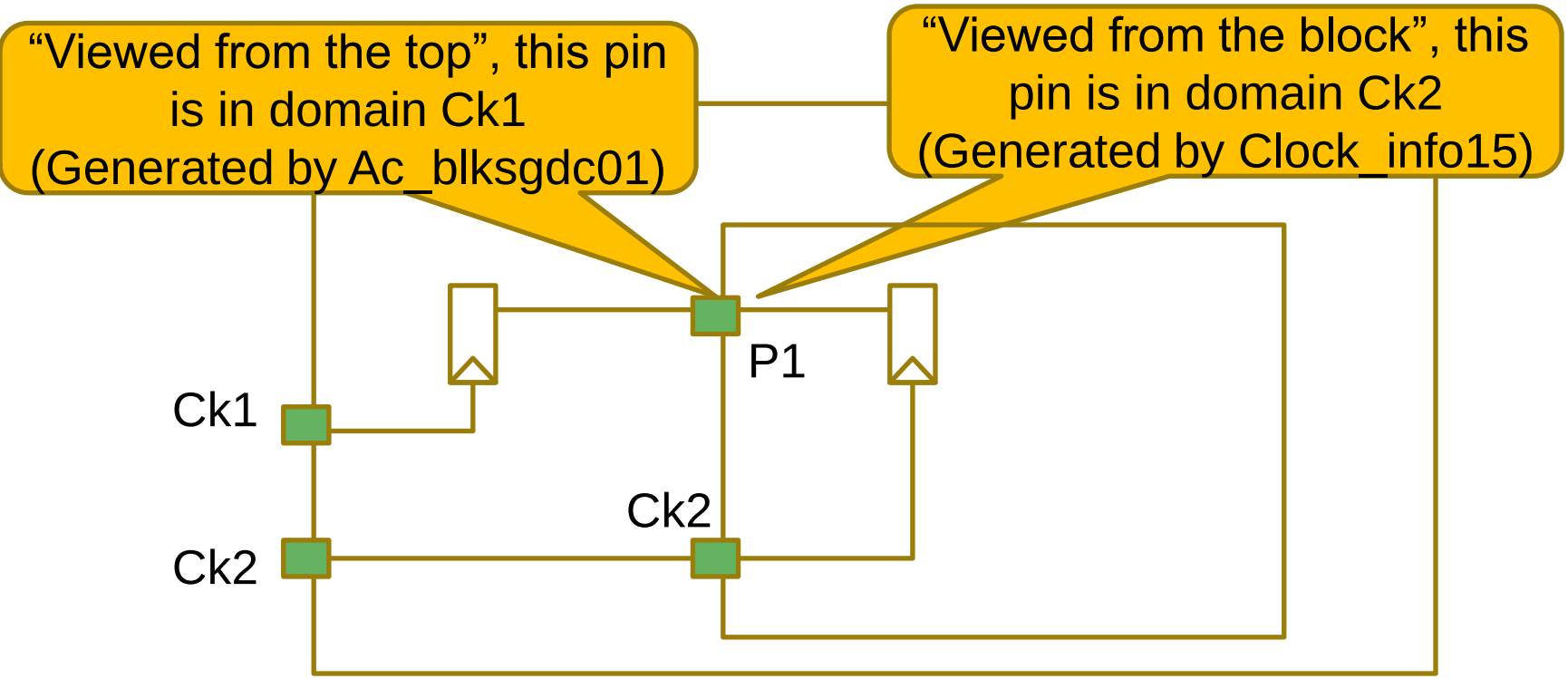
```
abstract_port –module <> –ports O3
              –clock C2
              –sync active –from C1 –to C2
              –seq yes
              –names “<>.sync2”
```

```
abstract_port –module <> –ports O4
              –clock C1
              -inputs IN2
              –combo yes
```

```
set_case_anaysis  -name <> -value 0
```

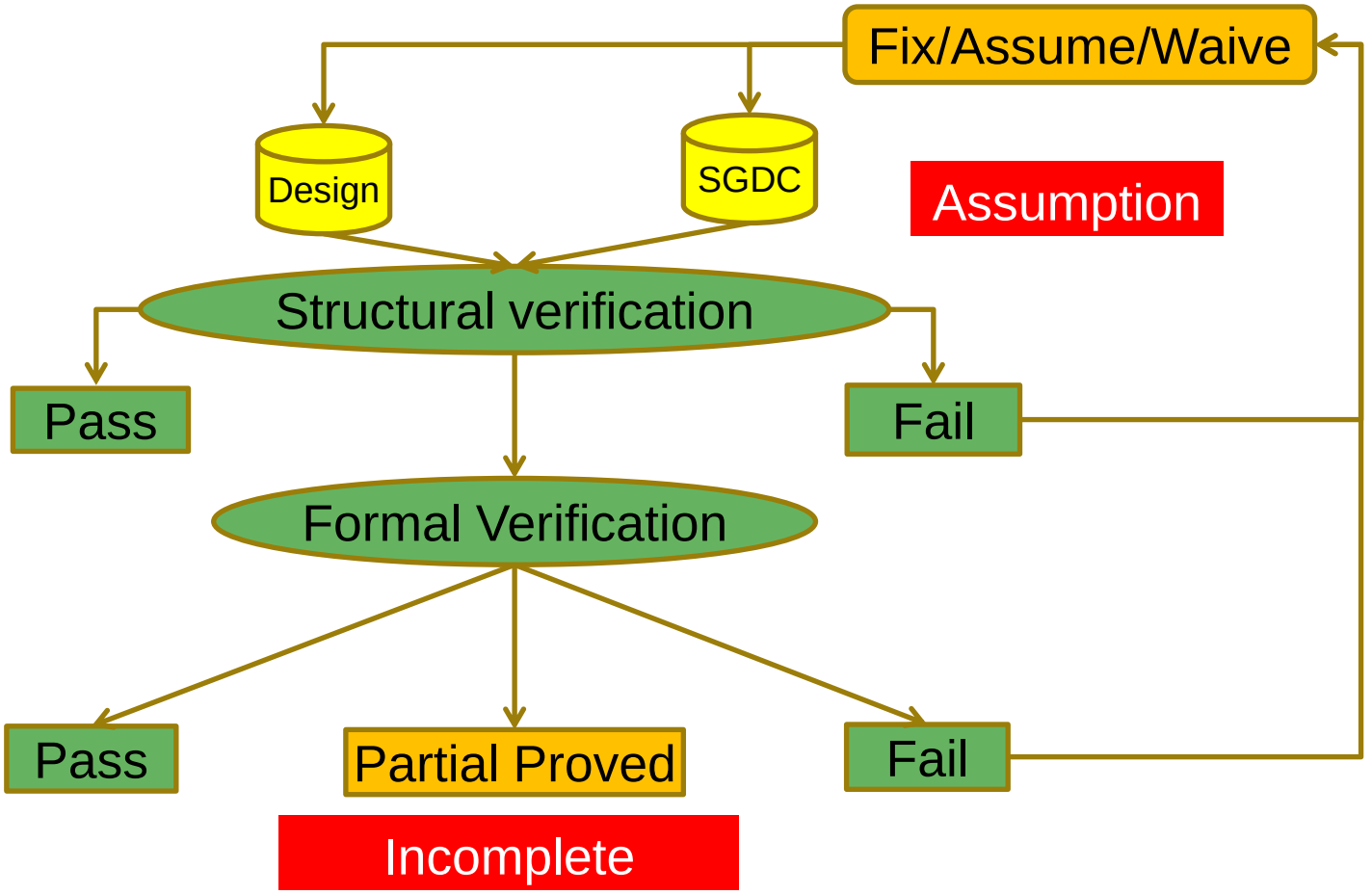
```
assume_path -name <> -input IN5 -output O5
```

Top-Down/Bottom up - "top view" vs. "block view"



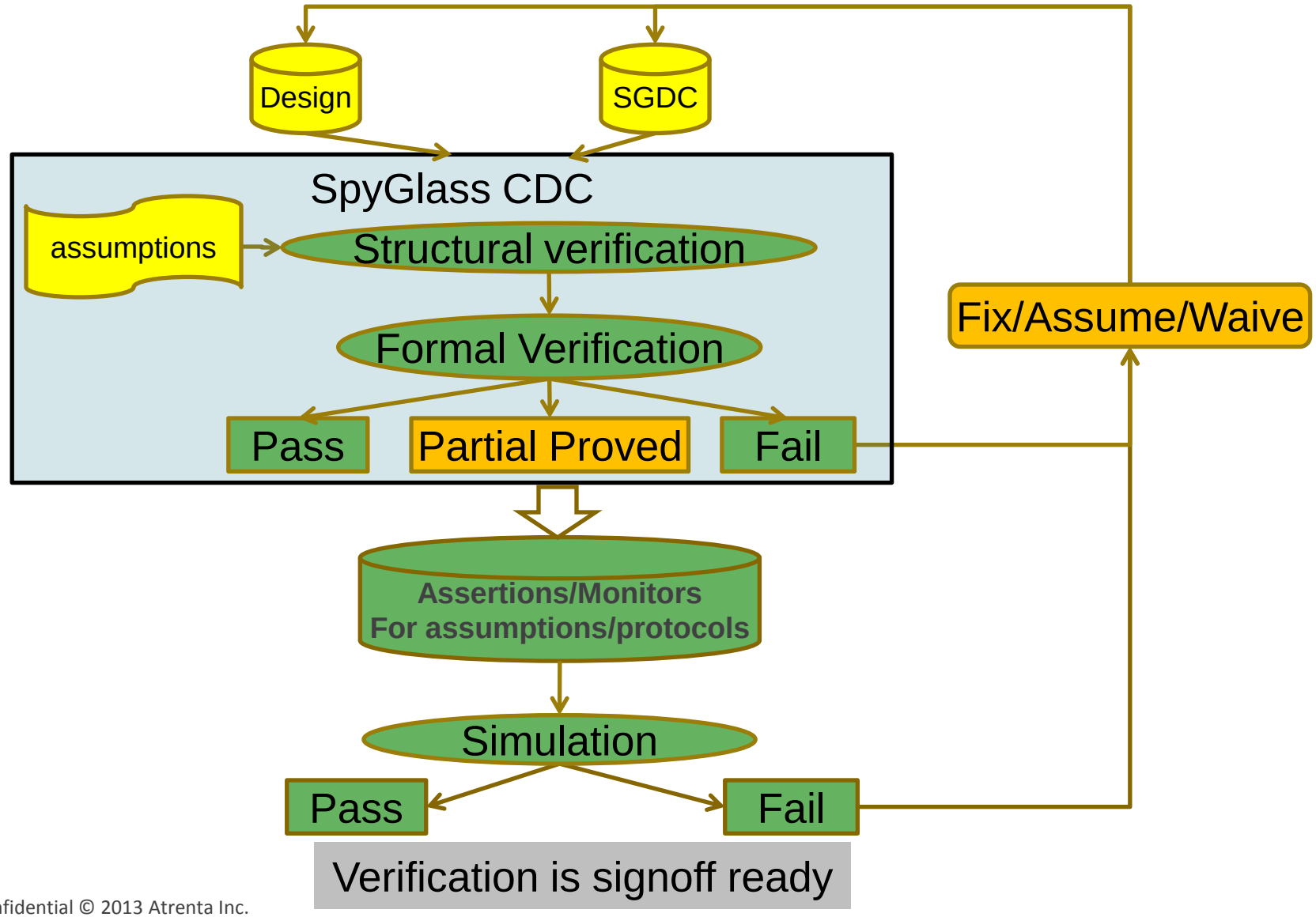
- The SoC verification should use "block view" to verify the SoC containing the block
- The block owner should capture top level assumption, "top view", when he does block verification
- Hierarchical CDC verification consists of matching top and block views

Typical CDC Verification Flow

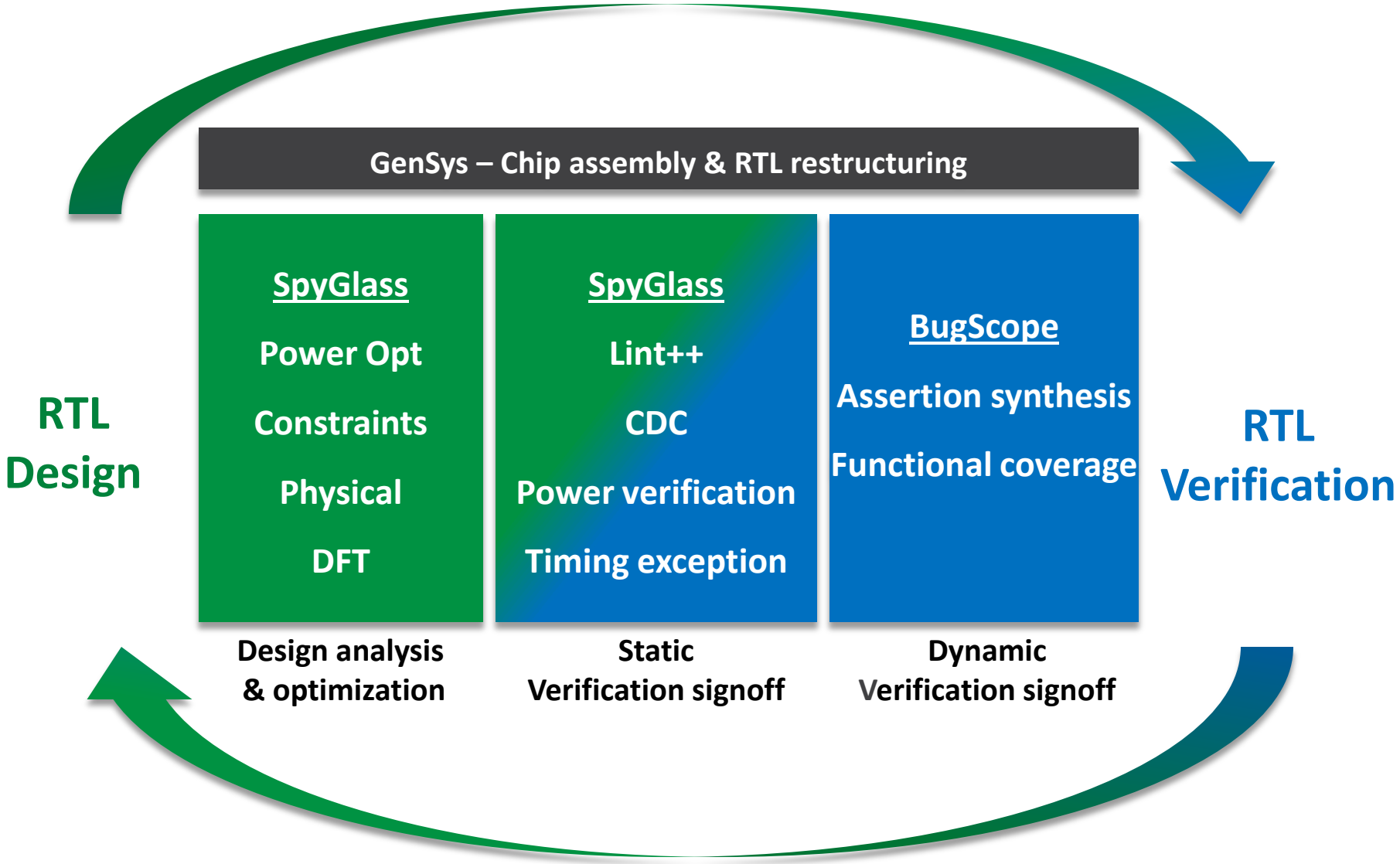


- Structural verification is often closed with <u>assumptions</u> such as “waiver”, “this signal is static and does not need synchronizer” – serious risk to silicon
  - Important to verify “Partially Proved” properties and Assumptions

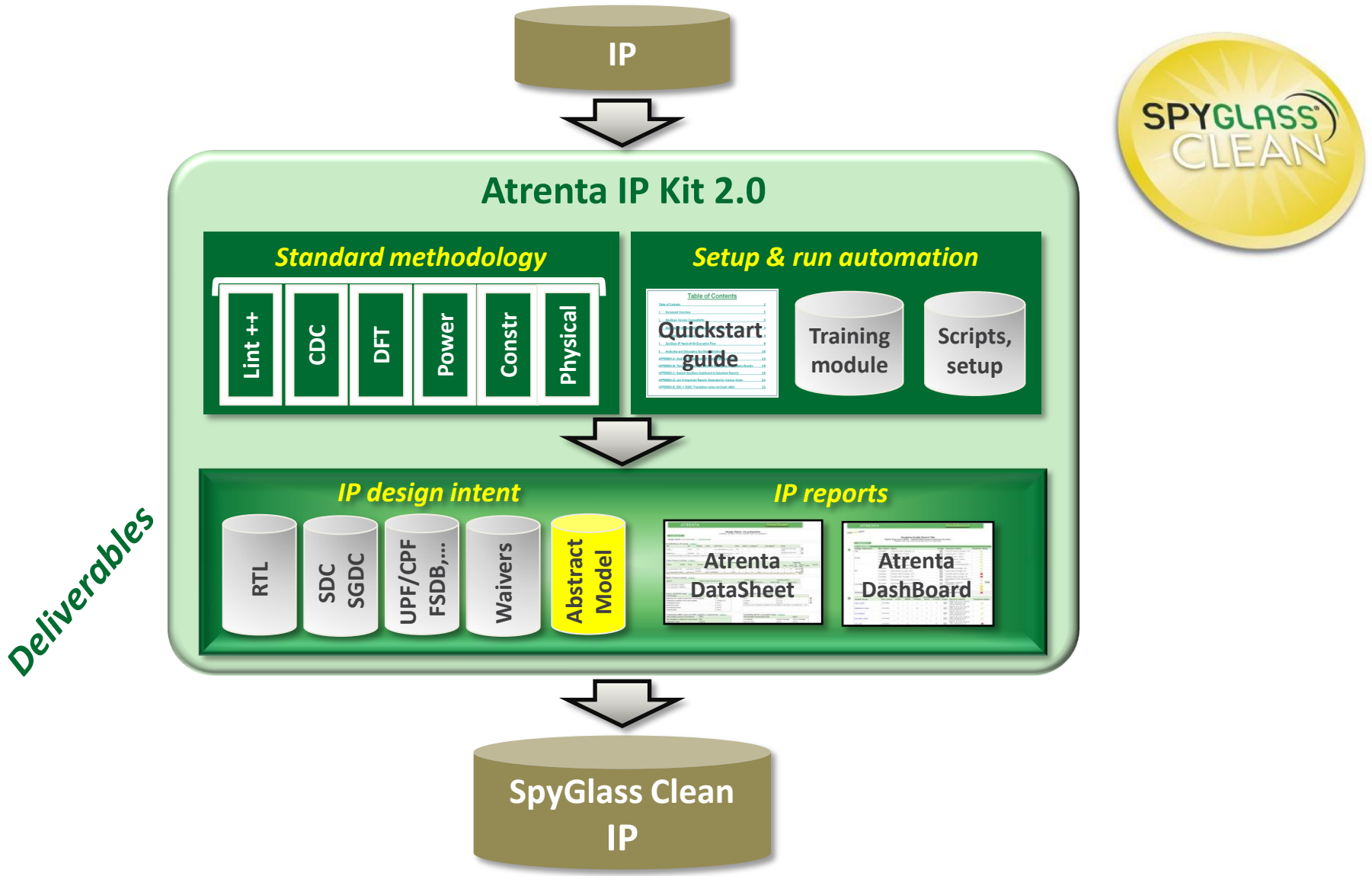
Hybrid CDC Verification Flow with SpyGlassCDC



Atrenta Spans RTL Design & Verification



Atrenta IP Kit – Enabling RTL IP Signoff



IP DashBoard

SPYGLASS FROM ATRENTA

Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build
/u/arbind/work_spyglass/Atrenta_Consolidated_DB/RefDesign2.0_GW2.0_20130325/leon3mp

Summary | Quality Goals | Design Objectives | wb_subsystem

Module: wb_subsystem

Show All | Pass | Fail

Quality Goals	Run Time	Run Status	Unresolved fatal	Unresolved error	Unresolved warning	Waived error	Waived warning	Success Criteria	Status
lint_rtl		Completed	0	3	44	0	5	Fatal =0, Error =0, Warning <500, Waived-Error =0	Fail
clock_reset_integrity		Completed	0	0	1	0	34	Fatal =0, Error =0, Warning <500, Waived-Error =0	Pass
sdc_audit		Completed	0	3	4	0	25	Fatal =0, Error =0, Warning <500, Waived-Error =0	Fail
sdc_check		Completed	0	3	1	0	25	Fatal =0, Error =0, Warning <500, Waived-Error =0	Fail
sdc_exception_struct									
sdc_redundancy									
cdc_verify									
dft_scan_ready									
dft_best_practice									
dft_dsm_best_practice									
power_gated_clock									
power_audit									

IP Quality

SPYGLASS FROM ATRENTA

Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build
/u/arbind/work_spyglass/Atrenta_Consolidated_DB/RefDesign2.0_GW2.0_20130325/leon3mp

Summary | Quality Goals | Design Objectives | wb_subsystem

Module: wb_subsystem

Show All | Pass | Fail

Quality Goals	Run Time	Run Status	Unresolved fatal	Unresolved error	Unresolved warning	Waived error	Waived warning	Success Criteria	Status
Summary		**Completed**	**0**	**28**	**446**	**1**	**553**	**Failed goals = 5**	**Fail**

Show All | Pass | Fail

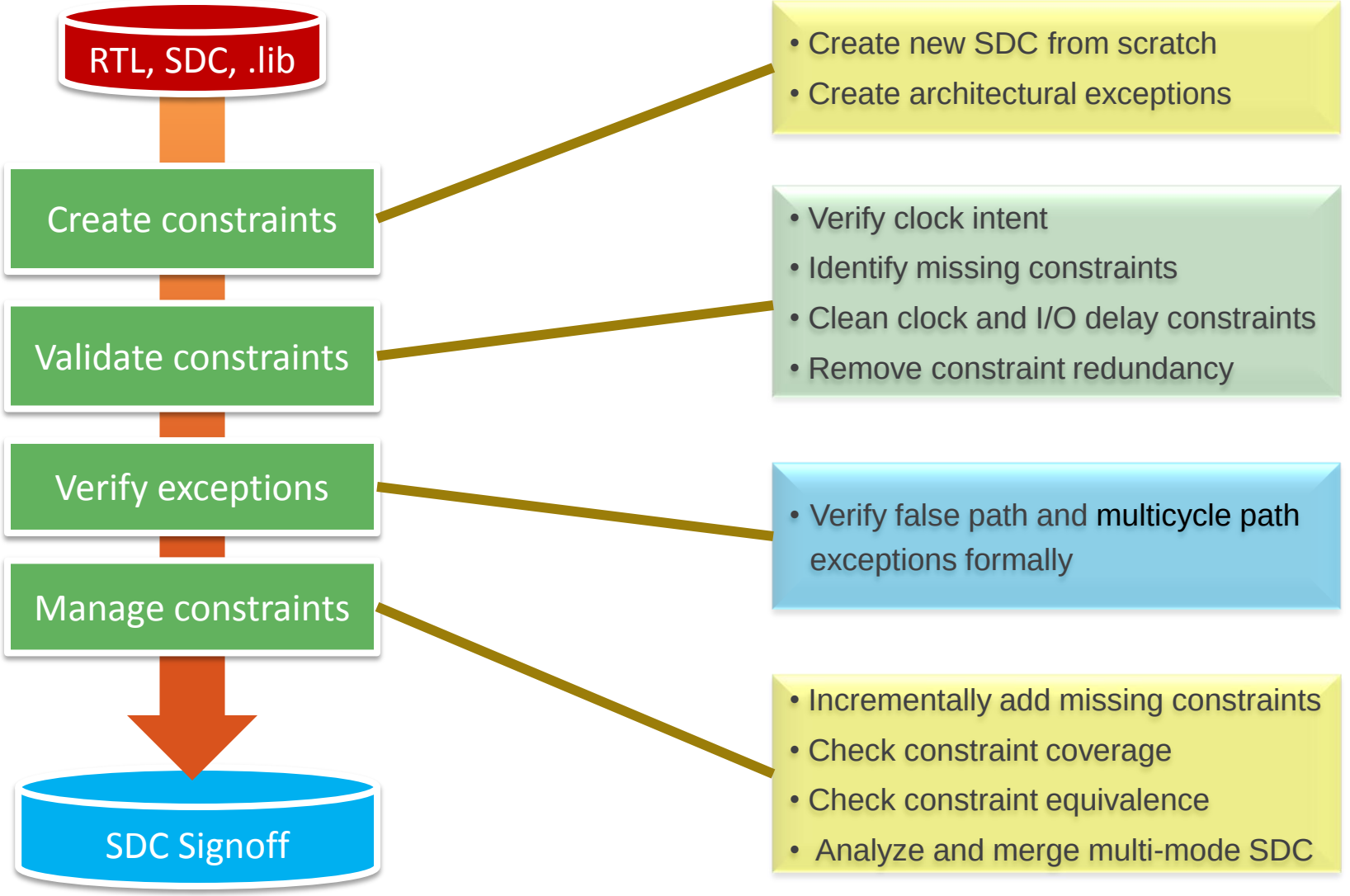
Category	Design Objective	Success Criteria	Status
DFT	Stuck-at fault coverage = 99.0%	Stuck-at fault coverage > 95	Pass
	Stuck-at test coverage = 99.1%	Stuck-at test coverage > 95	Pass
	Transition fault coverage = 70.8%	Transition fault coverage > 80	Fail
	Transition test coverage = 70.8%	Transition test coverage > 80	Fail
	Percentage of scannable flops (2 Goals/Scenarios)		Pass
	dft_scan_ready = 97.0%	Percentage of scannable flops > 95	Pass
	dft_dsm_best_practice = 97.0%	Percentage of scannable flops > 95	Pass
CDC	Unsynchronized crossings = 11	Unsynchronized crossings = 0	Fail
	Synchronization coverage = 21% (3/14)	Synchronization coverage = 100	Fail
	Failed properties = 0% (0/2)	Failed properties = 0%	Pass
Constraints	Percentage of ports constrained = 100.00	Percentage of ports constrained = 100	Pass
	Percentage of registers constrained = 97.46	Percentage of registers constrained > 90	Pass
Power	Internal power = NA	Internal power < 2.25mW	No Data
	Switching power = NA	Switching power < 5mW	No Data

IP Specs

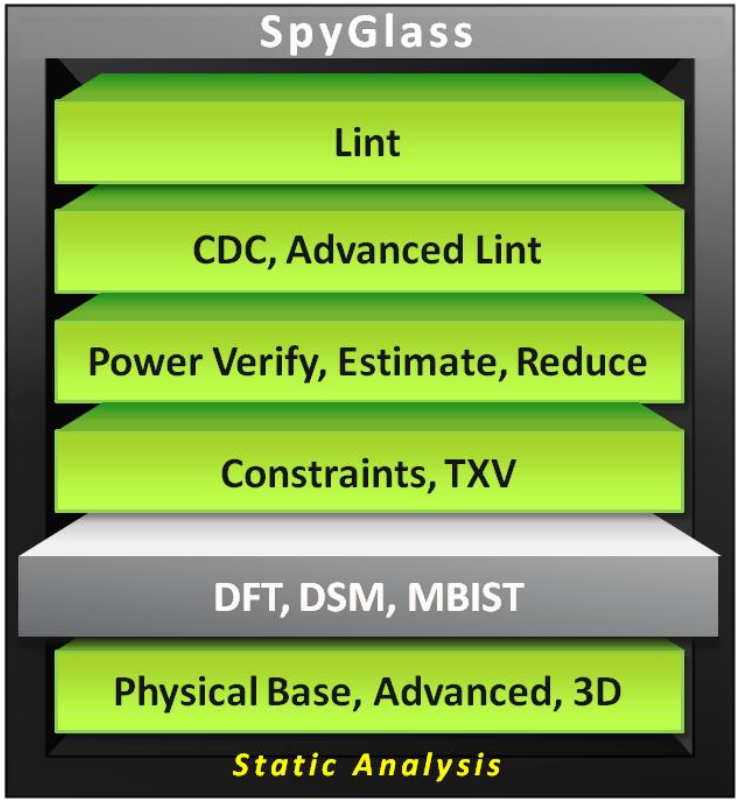
SpyGlass Advanced Lint

- **Analyze FSM's in the design**
  - Identify deadlocked states
  - Identify unreachable states
  - Analyze FSM complexity to gauge verification complexity
- **Analyze dead-code in the design**
  - Helps to determine if the dead-code is unintentional
- **Analyze initialization of Flip-flops**
  - Uninitialized flops lead to unknown state
    - Issues at RTL/Gate level simulation
- **Full case/Parallel/unique/priority case analysis**
- **Bus contention checks**
- **Synchronous FIFO Check**
- **Design complexity analysis**
- **Full debug capabilities at RTL including**
  - Cross-probing between RTL and violations
  - Graphical FSM viewer
  - Waveform Debug

SDC Signoff Flow



SpyGlass DFT for Early Testability Analysis



Features

- Analysis and coverage estimation for *stuck-at* & *at-speed test*
- Make *RTL scannable*
- *AutoFix test violations* - smart selection of test points
- *RTL SoC connectivity* and *scan chains validity checks* at netlist level
- *Intuitive debug* - cross probe RTL source
- RTL *memory BIST insertion*

Benefits

- Achieve high test quality at RTL
- Improved designer productivity for DFT
- High correlation to ATPG results

SPYGLASS®
FROM ATRENTA

謝謝

Note: The information in this presentation may contain forward-looking statements regarding product development. This development is subject to change at the sole discretion of Atrenta. Atrenta will have no liability for the delay or failure to deliver any features included in this presentation.