2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

accellera SYSTEMS INITIATIVE

Detecting Implementation Glitches in Gate-level Designs Using Advanced Hierarchical Techniques

Shanghai | September 20, 2023

Hongsheng Pan, SiEngine Technology Co., Ltd
Kurt Takara, Siemens EDA
Yuxin You, Siemens EDA

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

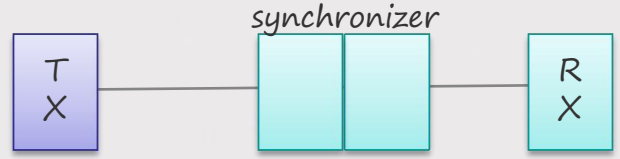
About SiEngine Technology Co., Ltd

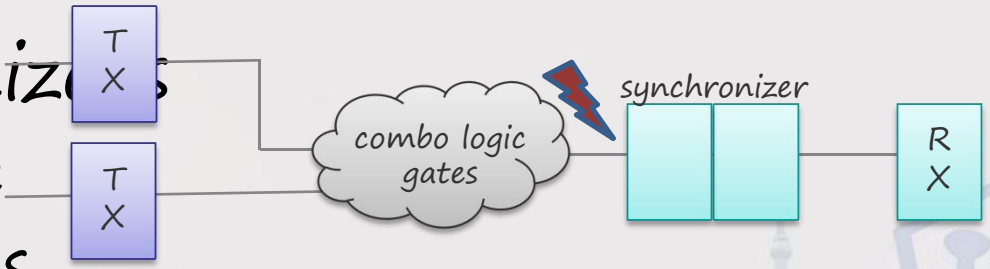
Overall Solutions Supplier for High-end Automotive SoC

- Smart cockpit multimedia SoC
- ADAS L3+/L4
- In-vehicle central computing
- Center Gateway
- Automotive MCU

Different Kinds of CDC Paths

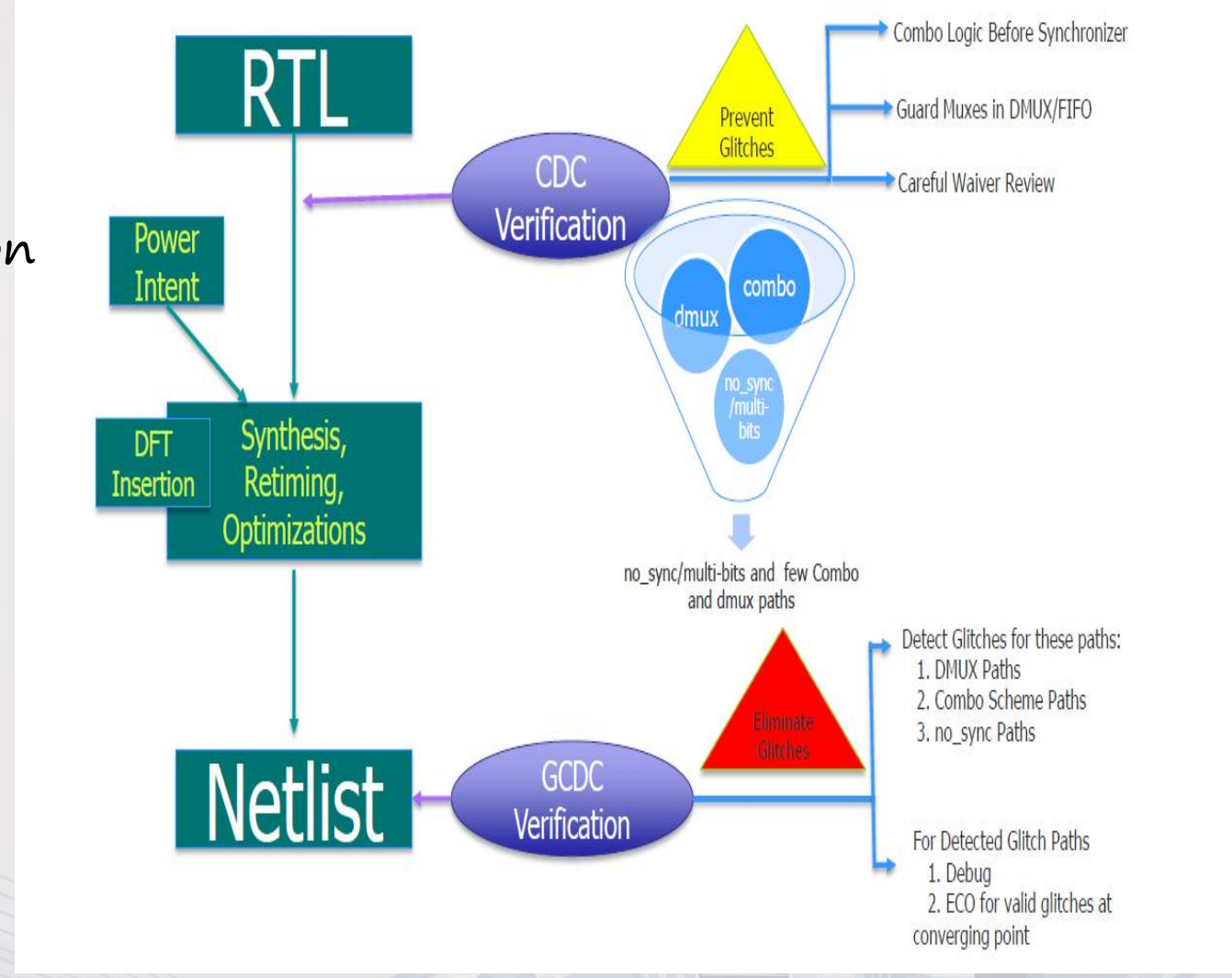
- Glitch Safe CDC Paths
  - TX register drives the synchronizer directly
  - No glitch possible
- Combinational logic before synchronizers
  - Potential glitches in combinational logic
  - Synchronizers may sample wrong values
- Un-synchronized single-bit or multi-bit paths
  - Potential glitches on unsynchronized CDC paths
  - RX may sample wrong values



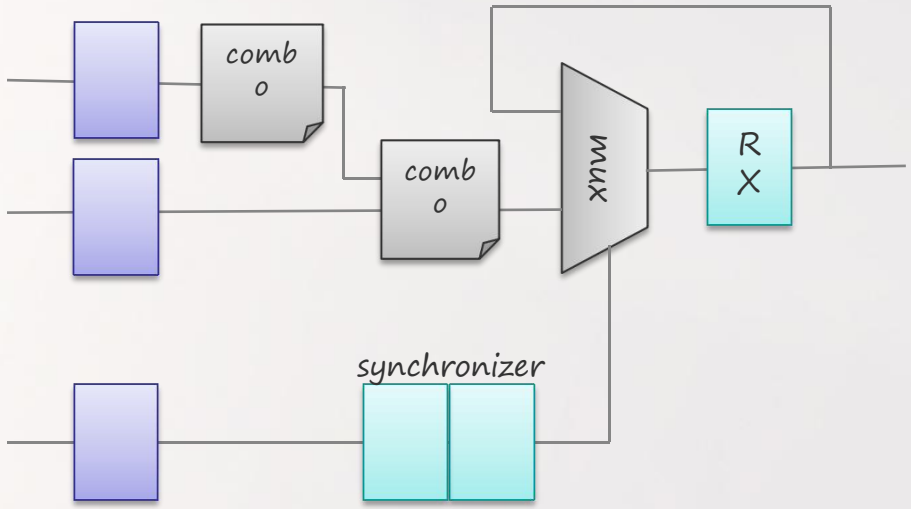


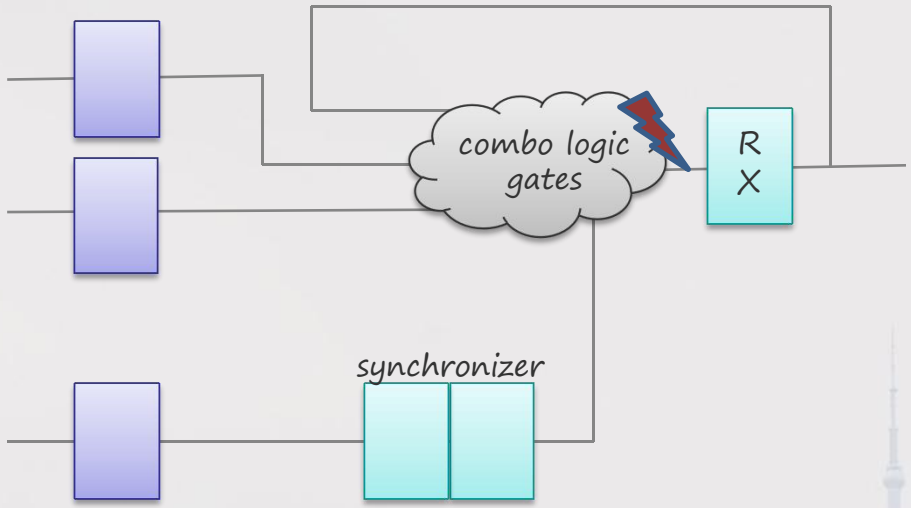
Gate-Level CDC Glitch Detection

- Glitches might be introduced during synthesis
  - Timing optimization of synchronization logic
  - Glitchy implementation of combinational logic
  - Power/DFT circuit insertion may break valid CDC path
- Glitchy logic can be created by optimization of multiple combo-logic blocks.
- Many glitches are detected on unsynchronized paths



Synthesis of DMUX path



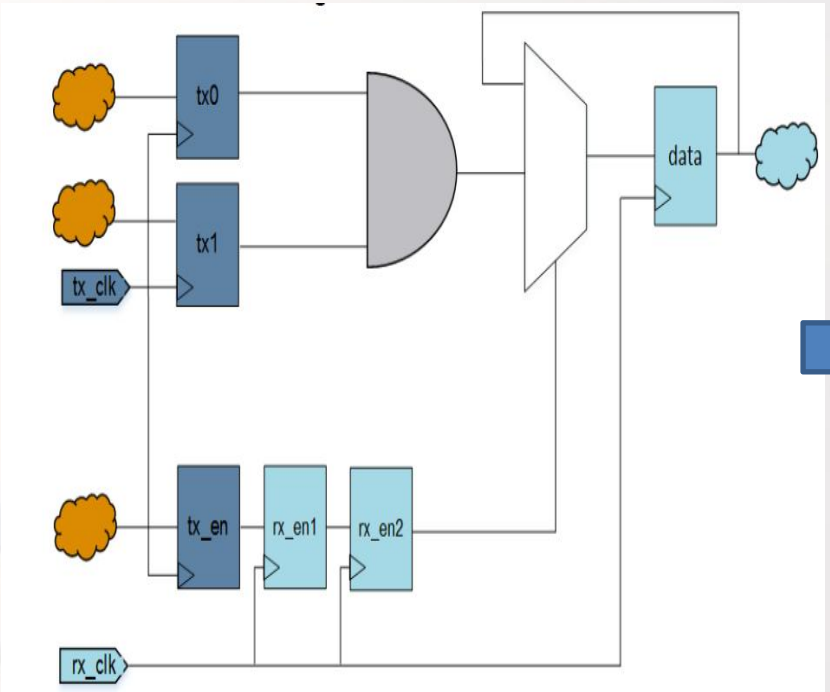


- RTL = Conceptual gate logic
- Mux is used to prevent transmission of asynchronous signals

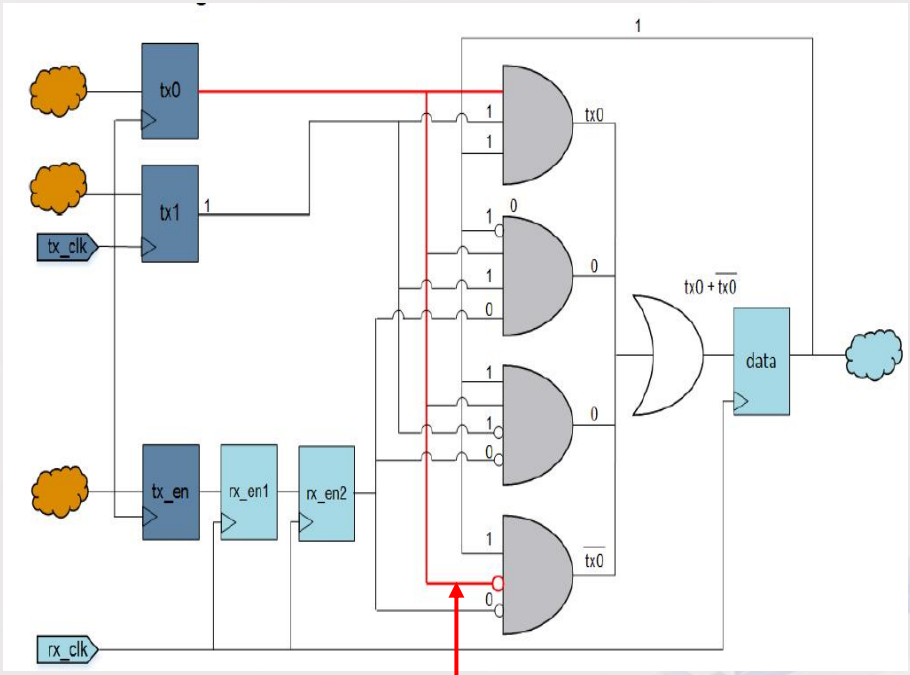
- Mux synthesized with other logic
- Mux transformed to AND/OR gates
- Mux select signal does not block the sampling of the TX

Gate-level CDC Glitch Example

RTL Logic : Mux based synchronizer



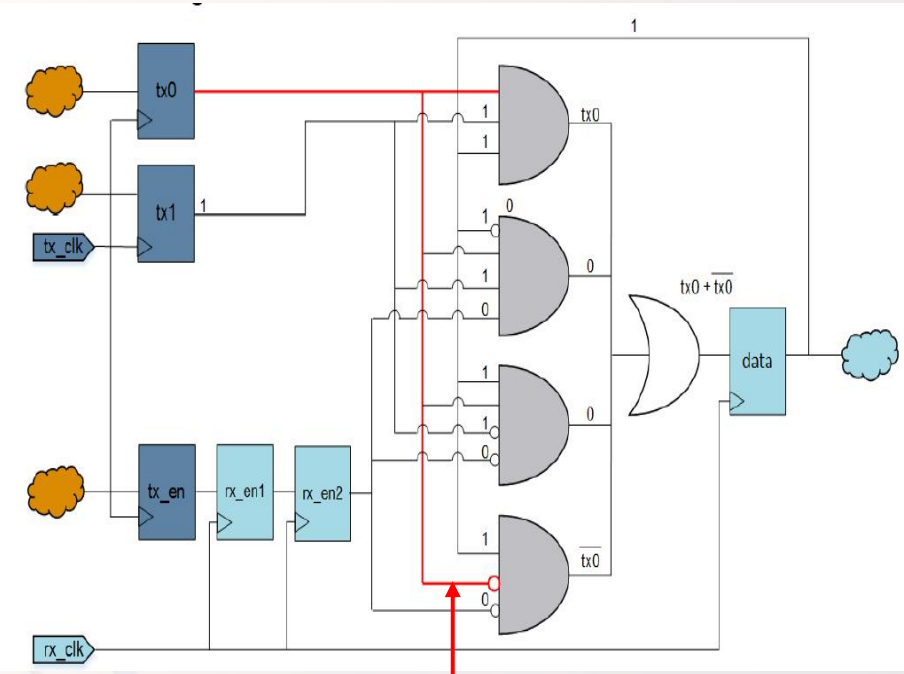
Combo-logic implementation after synthesis



For given constants, logic reduces to (tx0 | ~tx0) which causes an asynchronous glitch

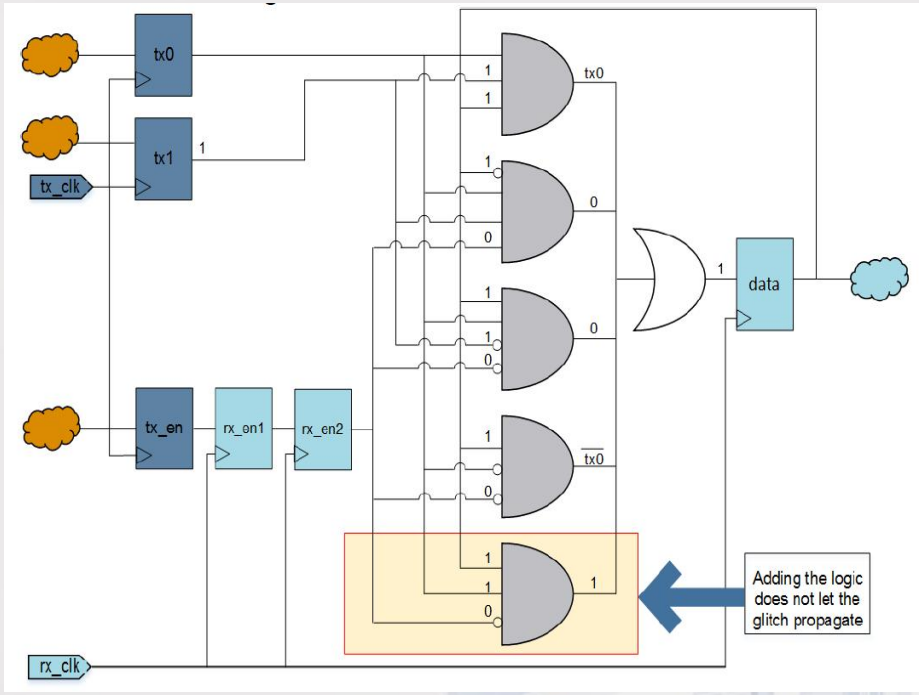
Gate-CDC Glitch ECO Example

Combo-logic implementation after synthesis



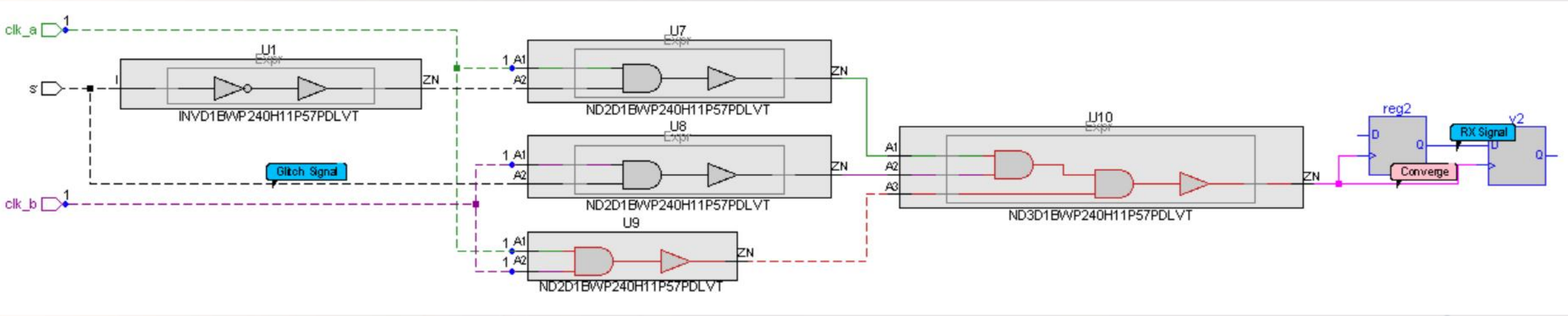
For given constants, logic reduces to (tx0|~tx0) which causes static-1 glitch

Manually corrected implementation



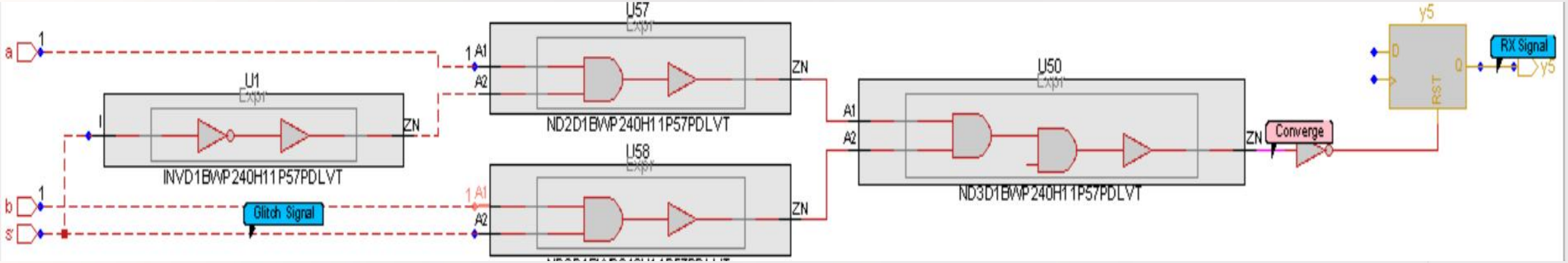
For given constants, (tx0|~tx0) glitch blocked by additional AND gate

Clock Tree Glitch Detection



- Clock tree glitches will generate functional design errors
  - Combinational Logic : [(clk_a && (! s) && clk_b && s && clk_a && clk_b)]
- Detect structural glitches in the clock tree
  - Glitches detected from both clock and non-clock sources
  - Glitch logic can be in same or different clock domain

Reset Tree Glitch Detection

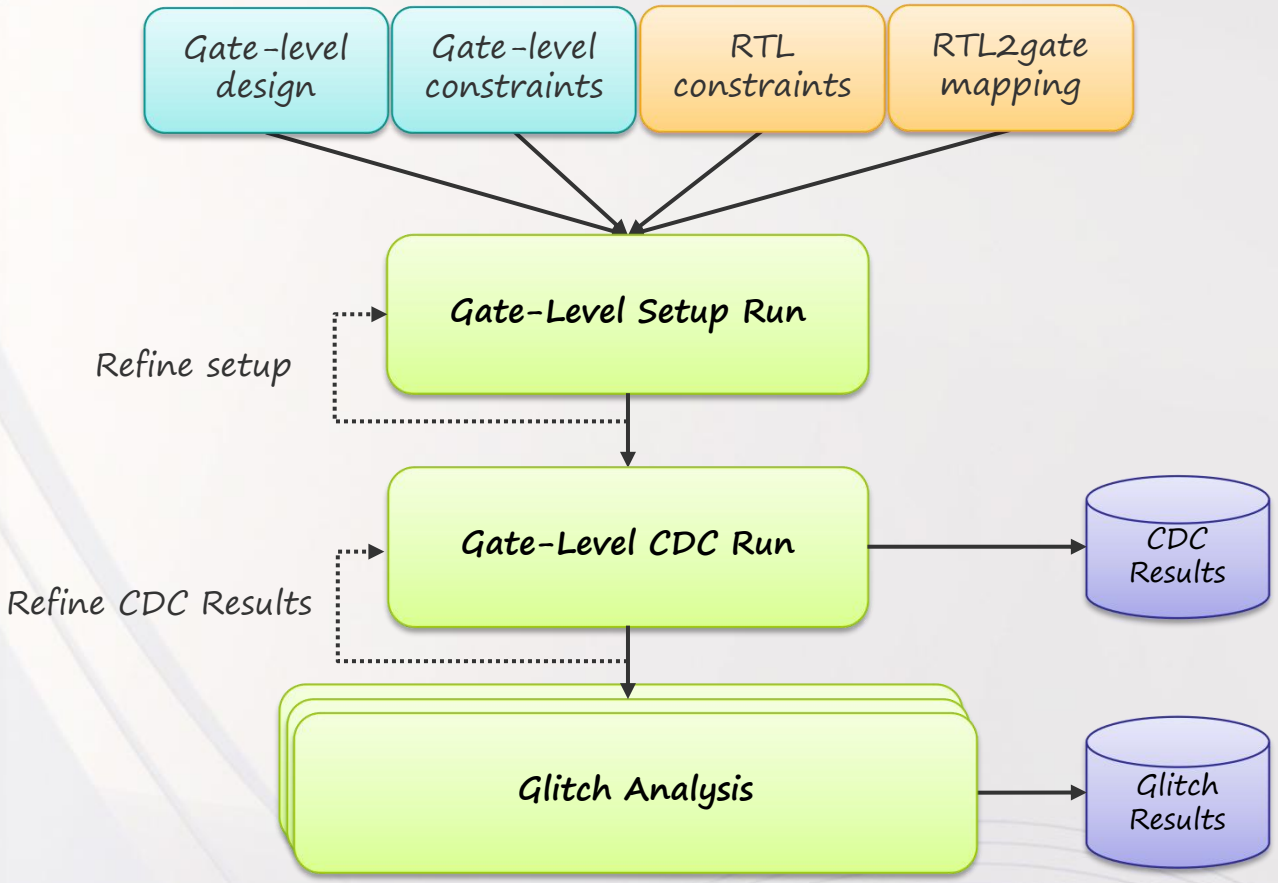


- Reset tree glitches will cause unexpected resets
  - Combinational Logic : [(a && (! s) && b && s && a && b)]
- Detects structural glitches in the reset tree
  - Glitches detected from both reset and non-reset sources
  - Glitch logic can be in same or different reset domain

Gate-level Glitch Analysis Challenges

- High setup effort
  - Incomplete gate-level SDC constraints
  - Difficult to convert RTL constraints
- Inadequate performance and capacity
  - Flat analysis is too slow and cumbersome
- High debug effort
  - Bit-blasted buses increase the number of crossings
  - False glitches reported on non-functional paths
- Hierarchical challenges
  - Difficult to generate lower-level constraints and abstract models

Flat Gate-level CDC Glitch Analysis



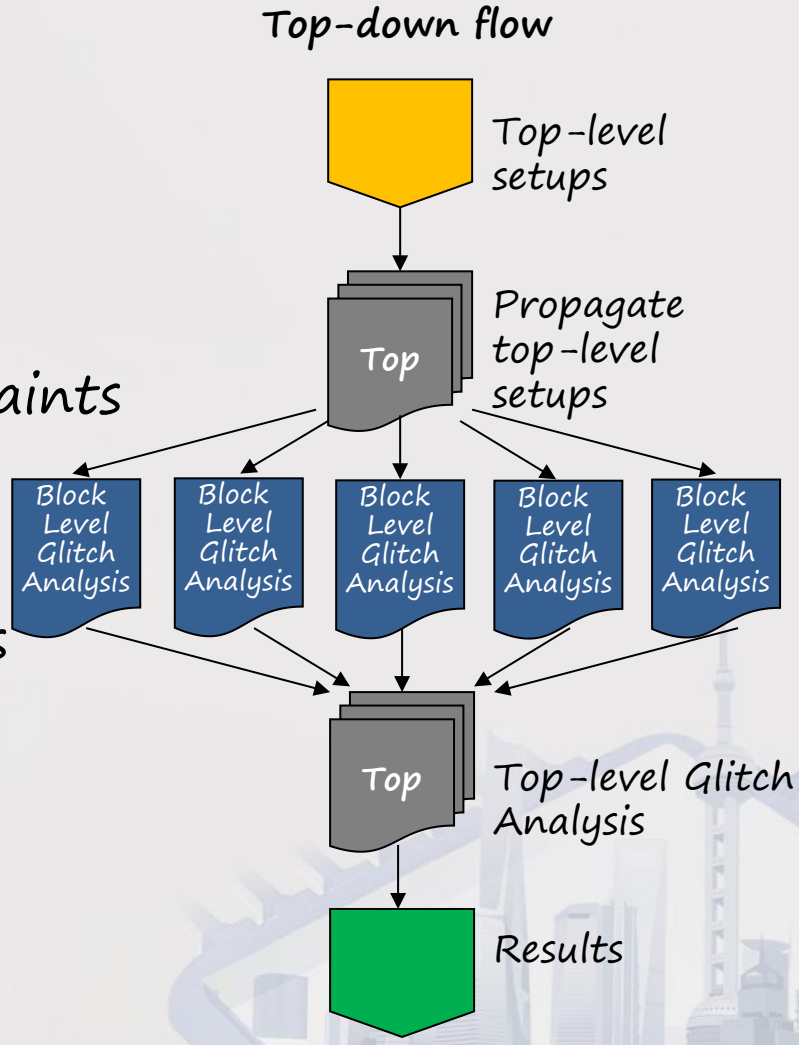
- Setup automation
  - Designer provides RTL constraints
  - Automatically converts RTL constraints
- CDC
  - Identify high risk paths
- Glitch Analysis
  - Identify structural glitches in CDC paths, clock paths, reset paths
  - Formal analysis automatically eliminates false glitch

Proposed Hierarchical Glitch Methodology

- Hierarchical setup automation
  - Designer provides RTL constraints (optionally: RTL-gate name mapping file)
  - Automatically generates partition-level constraints and run scripts
- Partition-level glitch analysis
  - Identifies glitchy structures in design partition
  - Partition-level abstract model automatically generated
- Top-level glitch analysis
  - Uses partition-level abstract models
  - Identifies glitchy structures in top-level integration logic

Hierarchical Setup Automation

- Step 1: Specify modules for abstraction
- Step 2: Run glitch analysis setup
  - Generates constraints & run script
  - Optional: Specify additional block-level constraints
- Step 3: Run hierarchical script
  - Runs block and top-level analysis
  - Automatically generates block abstract models
- Step 4: Review results
  - Step 4a: Review block-level results
  - Step 4b: Review top-level results



Case Study

- Large automotive IVI application subsystem targeted at ASIL B
- 11 clock groups and 15 reset groups

ISO26262-11 – Guidance on

ISO26262-5:2018 Requirement	Technique/Measure
7.4.4 Verification of hardware design	HDL Simulation Formal Verification Requirement Driven Verification
7.4.4 Verification of hardware design	Functional and structural coverage-driven verification (with coverage of verification goals in percentage)
7.4.4 Verification of hardware design	Automatic verification of coding rules ("Coding style") by code checker tool.
7.4.1.6 Modular design properties (testability)	Design for testability (depending on the test coverage in percent)
7.4.1.6 Modular design properties (testability)	Proof of the test coverage by ATPG (Automatic Test Pattern Generation) based on achieved test coverage in percent
7.4.4 Verification of hardware design	Perform **cross clock domain check on gate level netlist**, before and after test insertion

Results

	Estimated gate complexity	CDC signals	CPU run time
Flat glitch analysis	41M gates	27K	9h 1m
Hierarchical glitch analysis			4h 55m
Setup			3h 58m
Partition analysis	4M gates	10K	27m
Top-level analysis	1M gates	16K	30m
Incremental TAT			27m-57m

- Hierarchical incremental TAT 4X-20X faster than flat analysis

Hierarchical Glitch Analysis Advantages

- Runtime improvements over flat analysis
  - Faster turn-around-time over flat analysis
  - Concurrent partition analysis
- Automated setup reduces designer effort
  - Automatic import for RTL constraints
  - Automatic generation of partition constraints and run scripts
- Easier debug
  - Partitioning enables more focused glitch results review
- Reduced debug and triage effort
  - Advanced formal techniques eliminate noise and false violations
- Improved results management & progress tracking
  - Manage review and debug with status constraints
  - Enables project and design reviews

Summary

- By transforming the RTL directives and waivers to the gate-level
  - Reused the hard work that have been performed at the RTL
- Advanced gate-level glitch analysis required for tape-out
  - Identifies structural glitches introduced by implementation (after RTL signoff)
  - Advanced formal methods eliminate noise and false glitches
  - Identify the exact paths contributing to the glitch scenario
- With hierarchical multi-stage and multi-processing
  - Faster turnaround time with parallelized analysis
  - Easy to use with setup automation
  - Easy to debug with partition-focused results