Preventing Chip-Killing Glitches on CDC Paths with Automated Formal Analysis

Jackie Hsiung, Mediatek Inc.

Sulabh Kumar Khare, Mentor- A Siemens Business

Ashish Hari, Mentor- A Siemens Business

Introduction

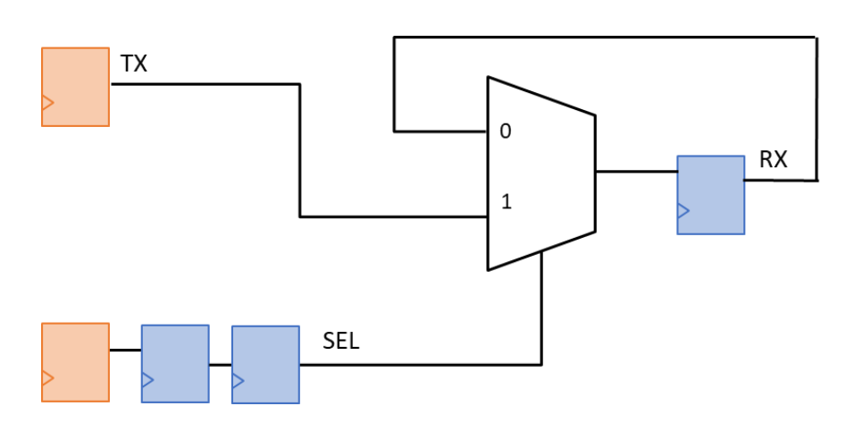
- Glitches are undesired transitions that occur before the signal settles to its intended value
- **Static glitch**: Signal temporarily change its value while it is supposed to remain static at logic 1 or 0

- **Dynamic glitch**: Signal oscillate while changing value from 0 to 1 or 1 to 0 before reaching the final value

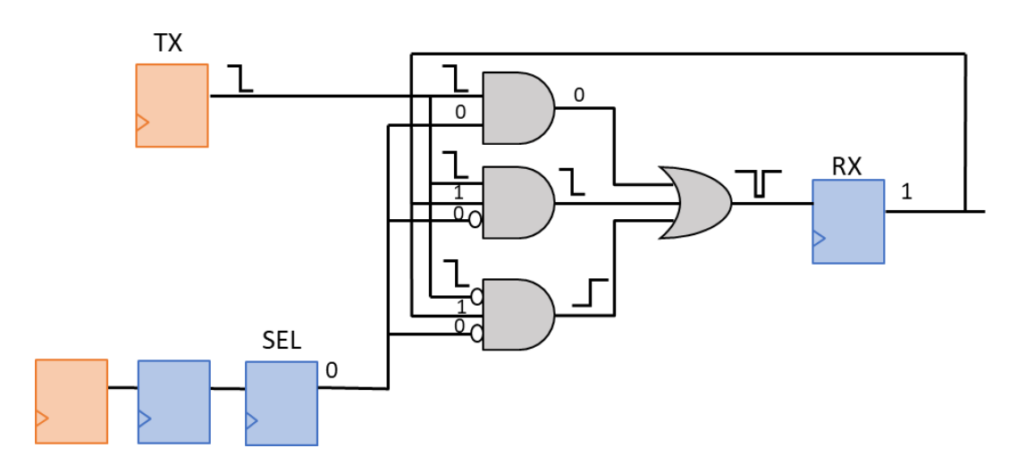
Glitch captured on a Clock Domain crossing (CDC) path can cause chip to fail

CDC Crossing Broken During Synthesis

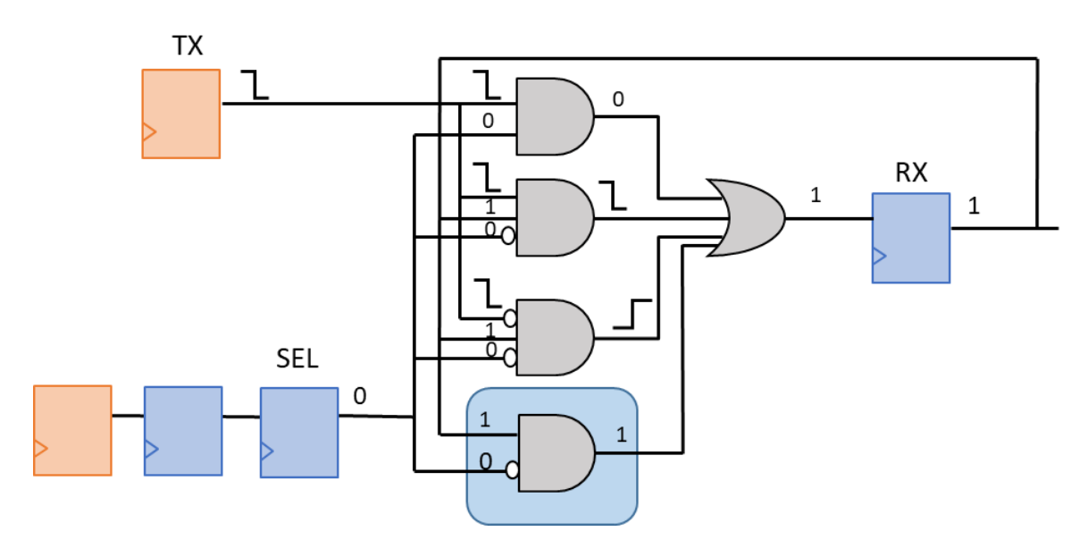
- A CDC path that is completely glitch safe at the RTL can run into chip-killing glitch defects post synthesis if the combinational logic and multiplexer logic in crossing paths is not synthesized in a glitch-free manner



CDC path in RTL is glitch free



Static-1 glitch in CDC path after synthesis



Additional logic added to stop glitch generation

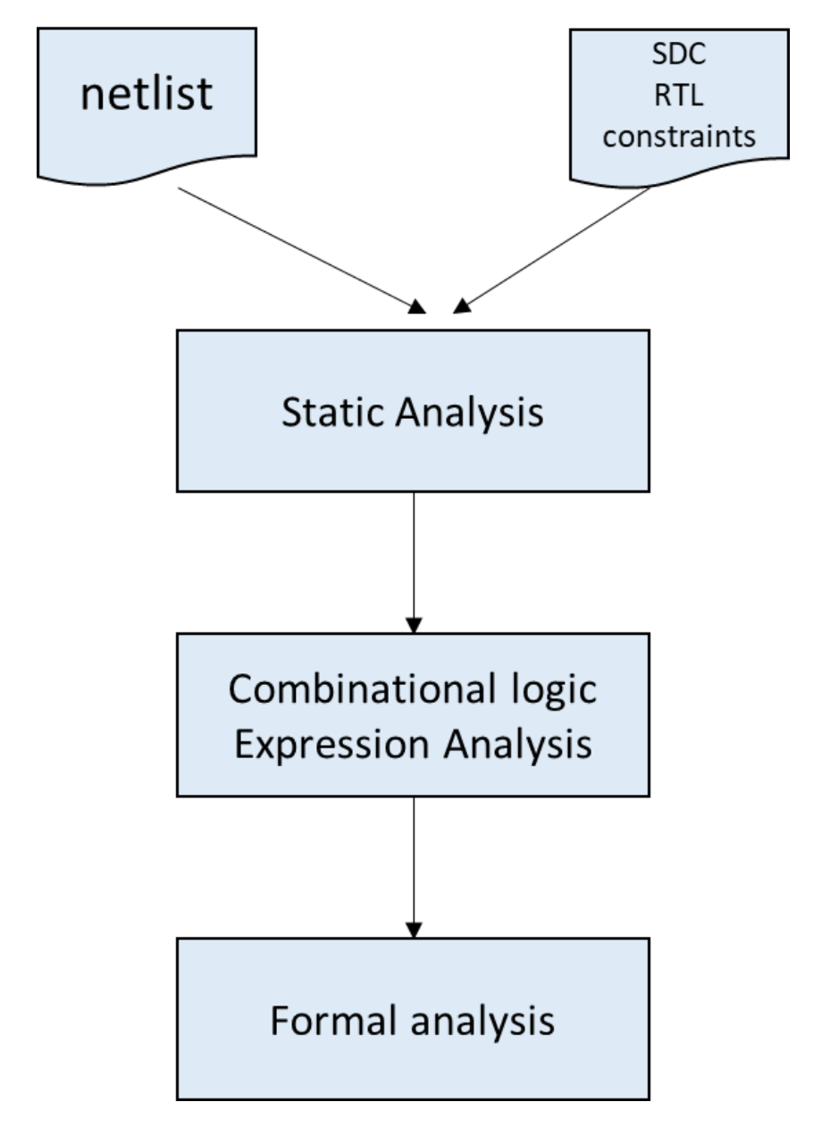
A glitch logic can be fixed by adding additional logic that stops the glitch propagation

Glitch Prone CDC Detection Challenges On Gate Level Netlist

- **Simulation to capture glitch:**
  - Receiver flop may be sampled in gate level simulation to identify glitch presence
  - Simulators have difficulties in sampling signal path when glitch is present
- **Assertions to identify glitch**
  - Assertions to check if glitch can happen in simulation
  - Simulators do not model asynchronous behavior
- **Static checks at the netlist level:**
  - Flagging every CDC path where combinational logic is present
  - Manual inspection to make sure combo logic is glitch free
- **Preventive steps at the RTL:**
  - Marking mux logic is don't touch for synthesis
  - Huge effort and some path may still escape
  - Need to prove at gate level that logic is still glitch free

Automatic Formal Based Glitch Detection Methodology

- The proposed method utilizes a combination of structural CDC analysis, expression analysis, and formal analysis to prune and prove the real glitches in the design at the gate level

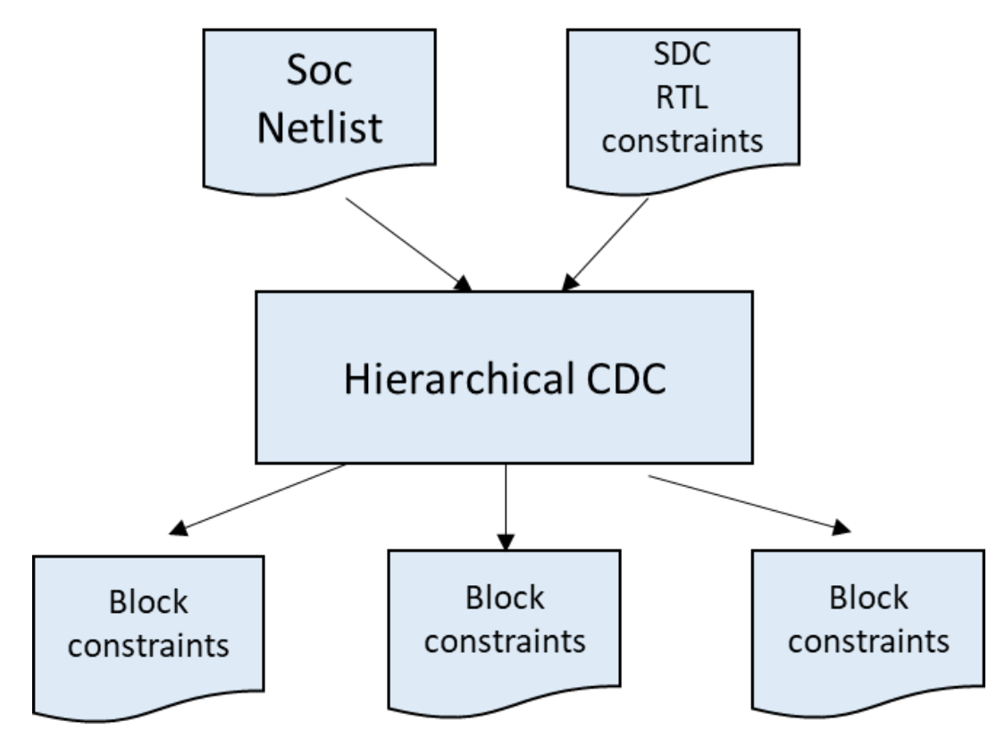


Proposed automatic formal-based glitch analysis

- **Stage1:** Complete static CDC analysis to prune out paths that do not contain any combinational logic at the gate level
- **Stage2**: Comprehensive expression analysis of the combinational logic tree in the data path to identify potential glitch candidates
- **Stage3:** Utilize formal engines to verify the glitch propagation condition conclusively

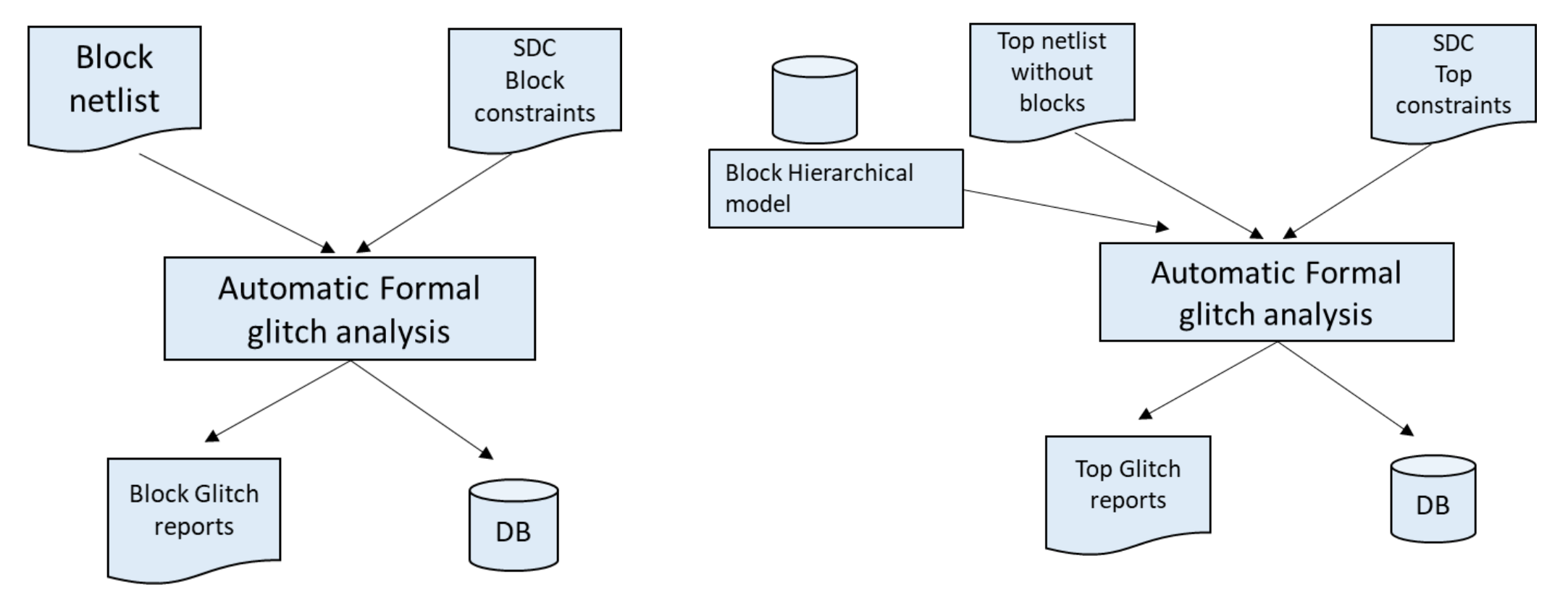
Analysis Of Proposed Glitch Detection Methodology

- **Quality of results and minimal noise**
  - Proposed methodology always resulted in less than a few hundred glitches for millions of paths
- **Ability to run very large SoCs**
  - The hierarchical CDC approach was utilized
  1. Propagate constraints from the top run to the block level:



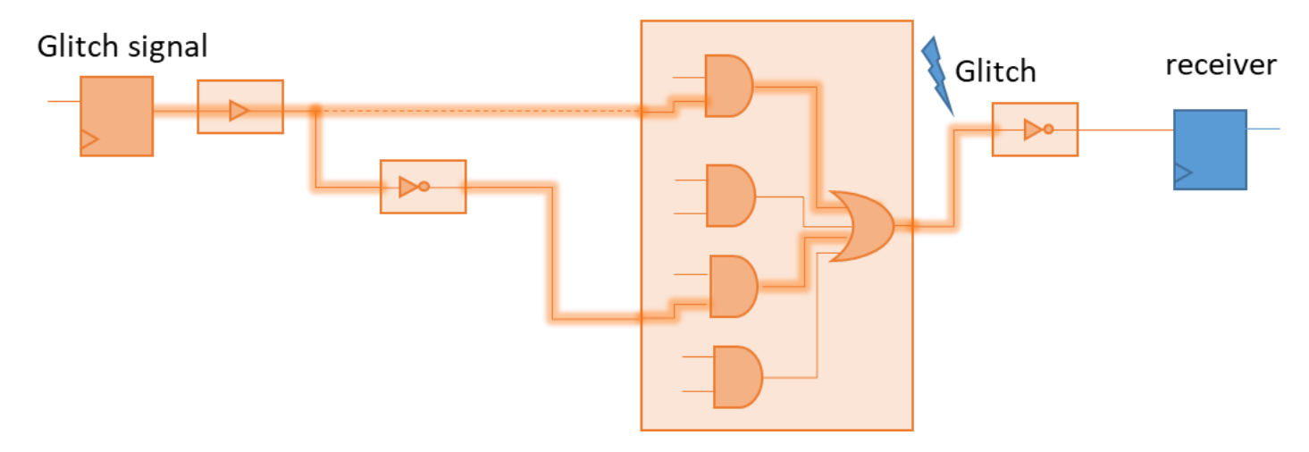
Hierarchical CDC runs to generate constraints for blocks (partitions)

  2. Complete the CDC and glitch analysis on blocks followed by glitch analysis at the top level:



Formal glitch analysis can be done on blocks and top level separately

- **Ease of debug**
  - Due to deep combinational logic on glitch prone paths, it was essential to zero-in on the exact area where the glitch needs to be addressed
  - Paths with glitch signals and the source of glitches are highlighted in the schematic

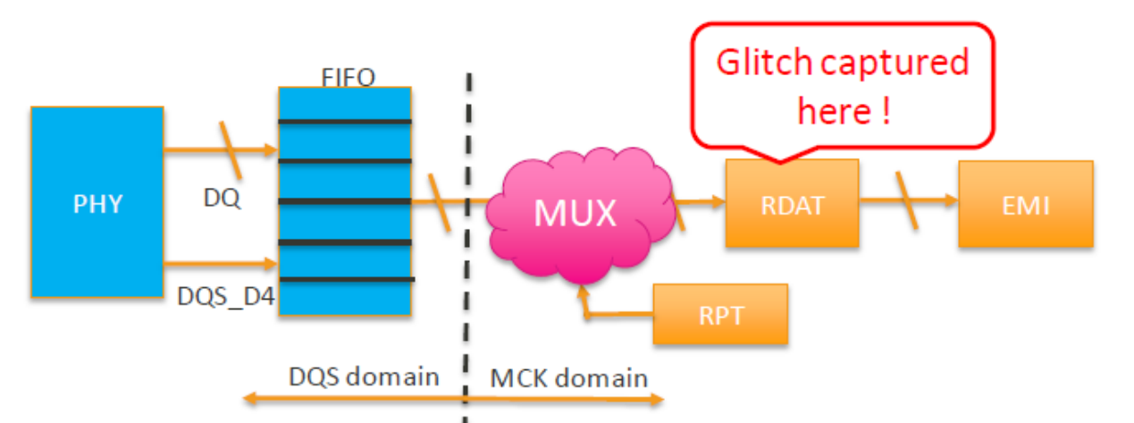


Highlight glitch signal and its converging points

- **Reduction in verification cycles**
  - Significantly reduce the verification cycles as glitch verification can be performed with existing RTL setups and the results pinpoint the exact glitch signal

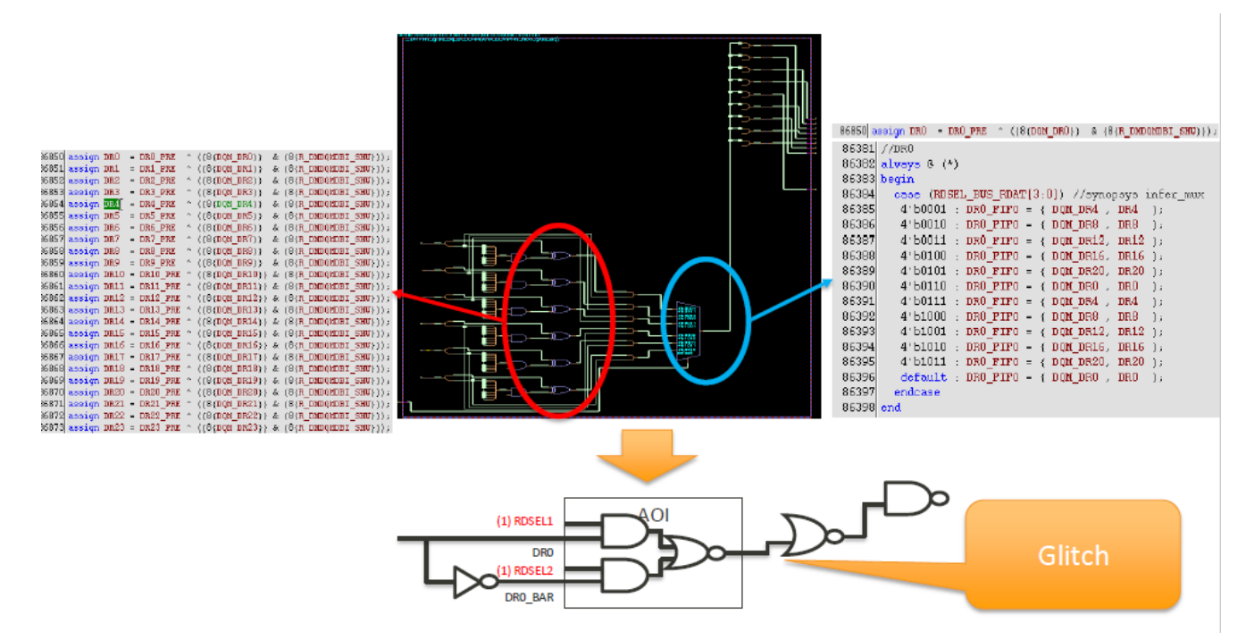
Case Study

- Proposed methodology is illustrated with an example of a real glitch which was observed in gate-level simulation of one of the SoCs
- SoC was signed off for CDC but glitch scenario was identified in a gate-level simulation of the netlist



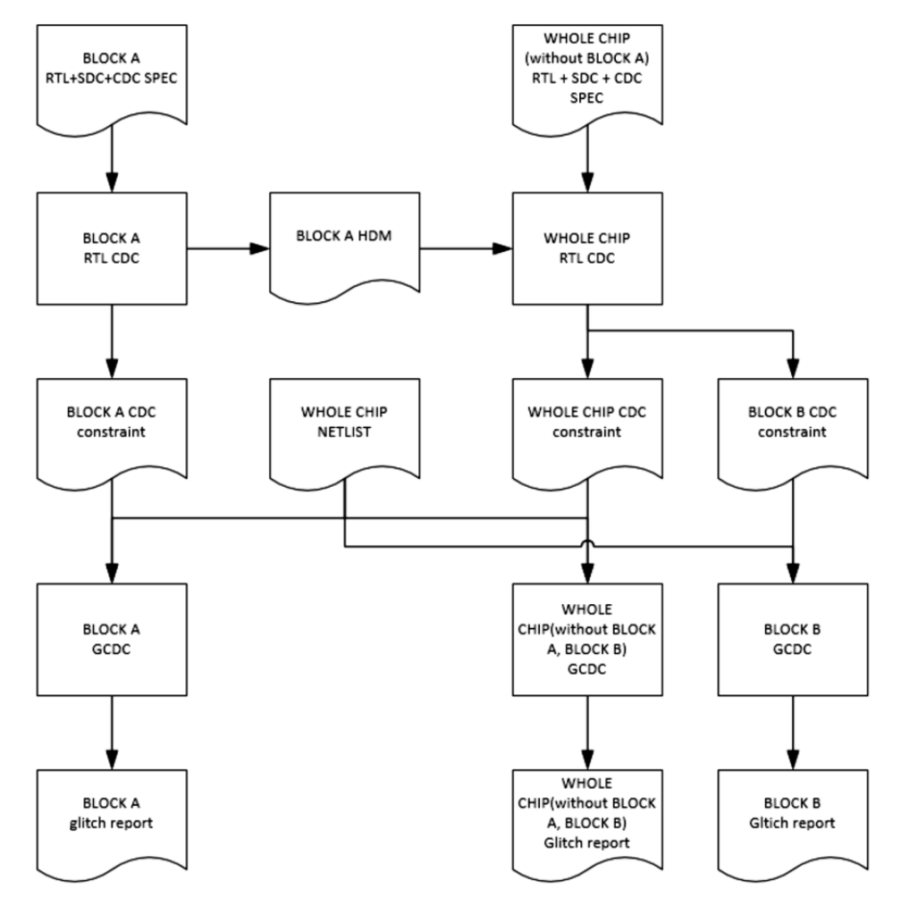
Real glitch scenario in a SoC

- Multiplexer implemented with case was converted to glitch prone logic at gate-level:



RTL to gate-level analysis of the real glitch scenario

- Proposed methodology applied with hierarchical CDC runs to propagate constraints for blocks:



Mediatek glitch verification flow

- Using the proposed methodology, we were able to identify the glitch case with source of the glitch and additional glitch paths with minimal noise:

Table: Glitch results on various SoCs

S.No.	Project Name	Clock Domains	Glitch Sources (All partitions)	Run time (Sum of all partitions /Maximum a partition) hrs	Glitch Result
1	Project A	1155	1848	93/31	data-mux glitch found in one module (288 paths ECO)
2	Project B	1028	1639	87/38	All waived
3	Project C	641	1241	69/23	no-sync glitch found in one module (166 paths ECO)
4	Project D	823	1487	56/6	All waived
5	Project E	828	1534	77/29	All waived
6	Project F	754	2846	103/34	All waived
7	Project G	963	2262	84/32	All waived

Validation on real SoC confirms that proposed flow and techniques are practical, hence must be applied as a signoff methodology for prevention of chip killing glitches before tape-out