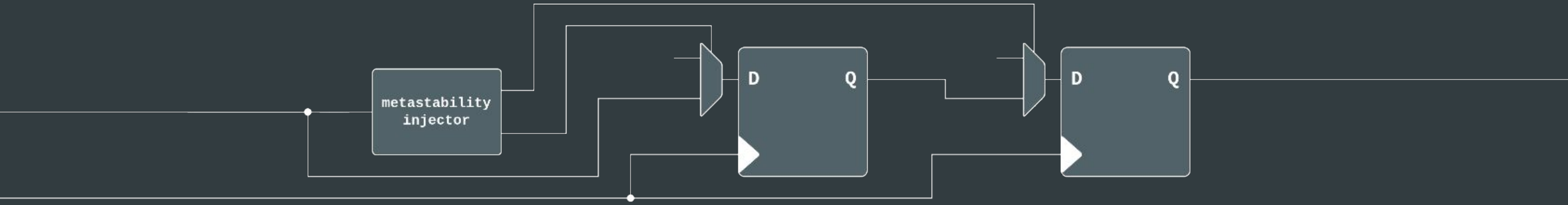
Dynamic CDC Verification

Efficient Approaches for In-House Flows

solutions to simplify.



linkedin.com/in/tchilikov

Christian Tchilikov
July 1, 2025

Overview

- What is Dynamic CDC?
    - How it’s different from Static CDC
    - What we are trying to verify
    - How metastability is simulated
    - Challenges, and how they scale with large designs
- Approaches for Dynamic CDC
    - Proprietary Tools vs Custom Modeling
    - Naive modeling, and semify’s improved and free in-house model
- Modeling with simulation efficiency in mind
- Example use cases

What is Dynamic CDC?

Dynamic CDC	Static CDC
• Simulates the design (requires testbench)	• Analyzes the design structure (no simulation)
• Limited by test coverage	• Exhaustive: checks all paths
• Replaces synchronizers with models that simulate setup/hold violations and internally capture metastable events	• Checks for missing synchronizers and valid synchronization methods
• Checks design functionality under different synchronization latencies caused by metastable events	• Checks for glitch-prone combinatorial paths, reconvergence issues, etc
• Assertions can catch data stability violations	• Designer imposes waivers/constraints based on design intent - which may be wrong

What are we trying to verify?

- That the post-synthesis (silicon) synchronizers behave the same as in simulation
  - Bridge the gap between simulation and silicon
- That a design functions correctly due to varying synchronization latencies caused by metastability
- To fill the gaps left by Static CDC:
  - That Static CDC waivers did not mask potential synchronization bugs
  - Protocol errors can be missed by Static CDC; intent of designer’s synchronization method must be specified to the tool
  - Re-convergence analysis may exceed the limits of simple checks

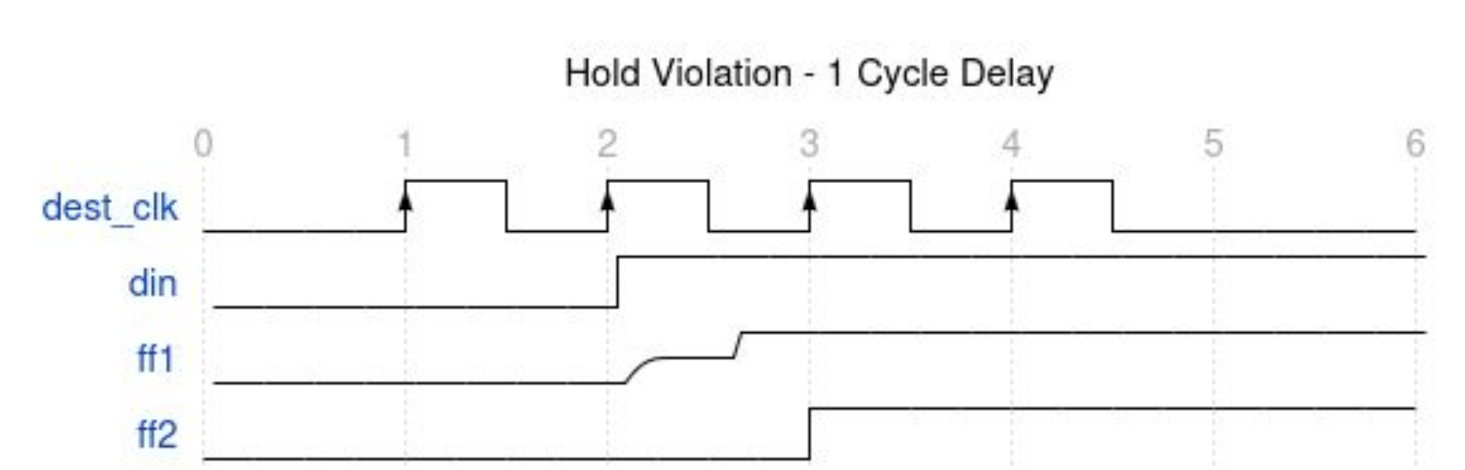
Challenges

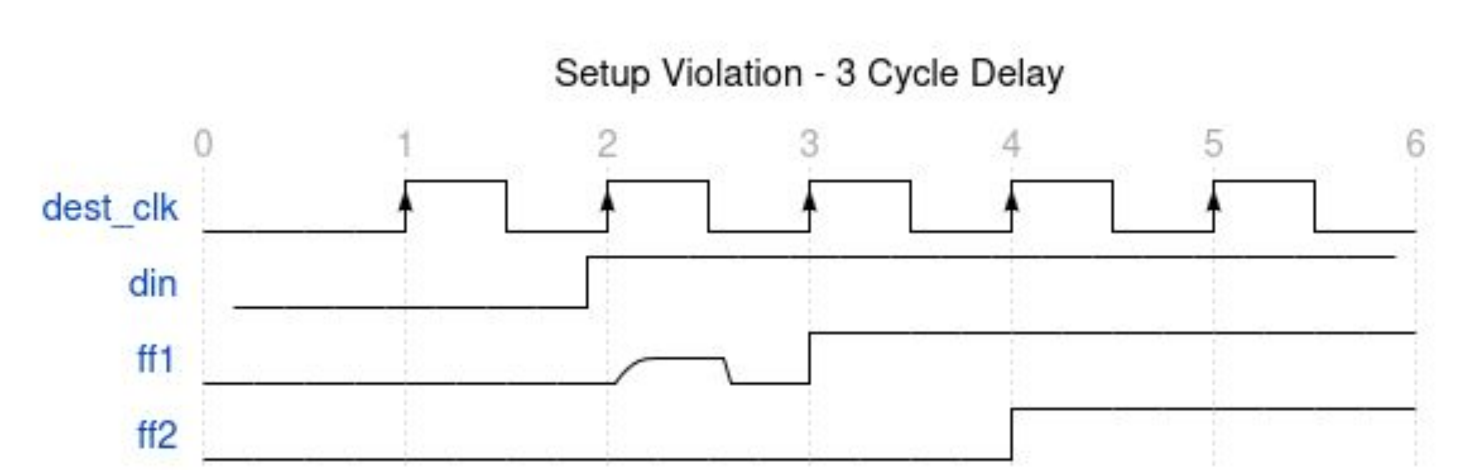
- Model-based Dynamic CDC can not check for missing synchronizers
    - Only verifies the functionality of the design with the already in-place synchronizers
- Dynamic CDC is limited by test coverage – testbench may not exercise all possible scenarios
- Large designs begin to suffer - blackboxing, hierarchical switches, different clock ratios
    - Need a model to account for all of these factors and reduce manual labor

Synchronization Latency

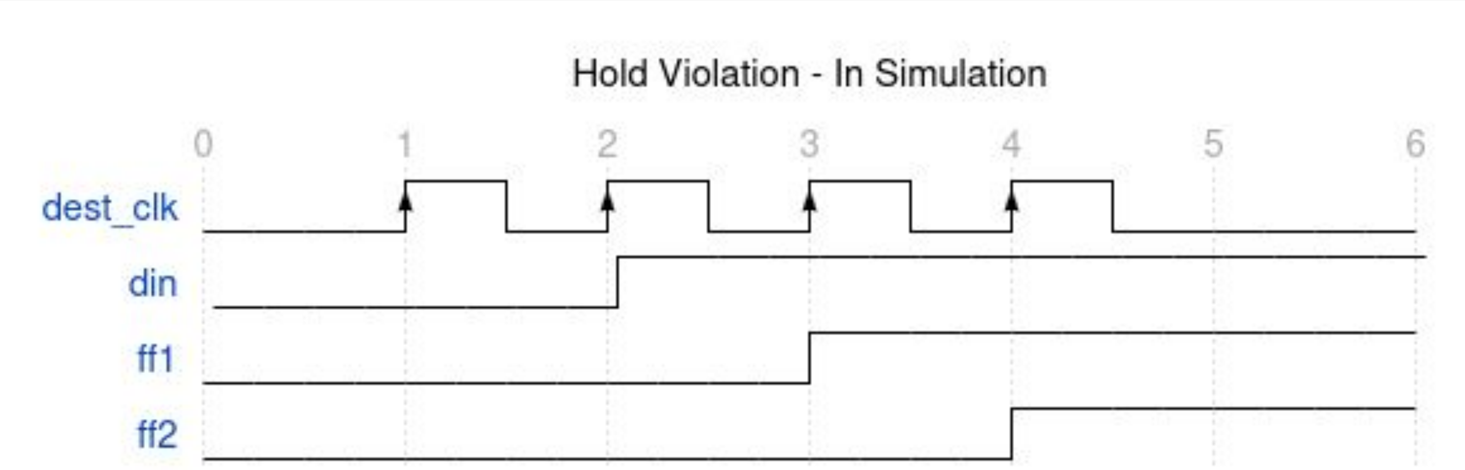
- Time it takes for a signal to be seen at the output of a synchronizer relative to when it was input (number of positive edges of destination clock before signal is seen at output)
  - Consider a 2 DFF synchronizer – what is the synchronization latency?
  - Intuition tells us 2 posedges, one for each flip-flop – but this is not always true…
  - Due to metastability, it is possible for a 2DFF synchronizer to experience non-deterministic synchronization latencies of 1 and 3 posedges as well
  - 2DFF Synchronizers do not exhibit this variation in timing in simulation by default – need to model it either manually or with a tool

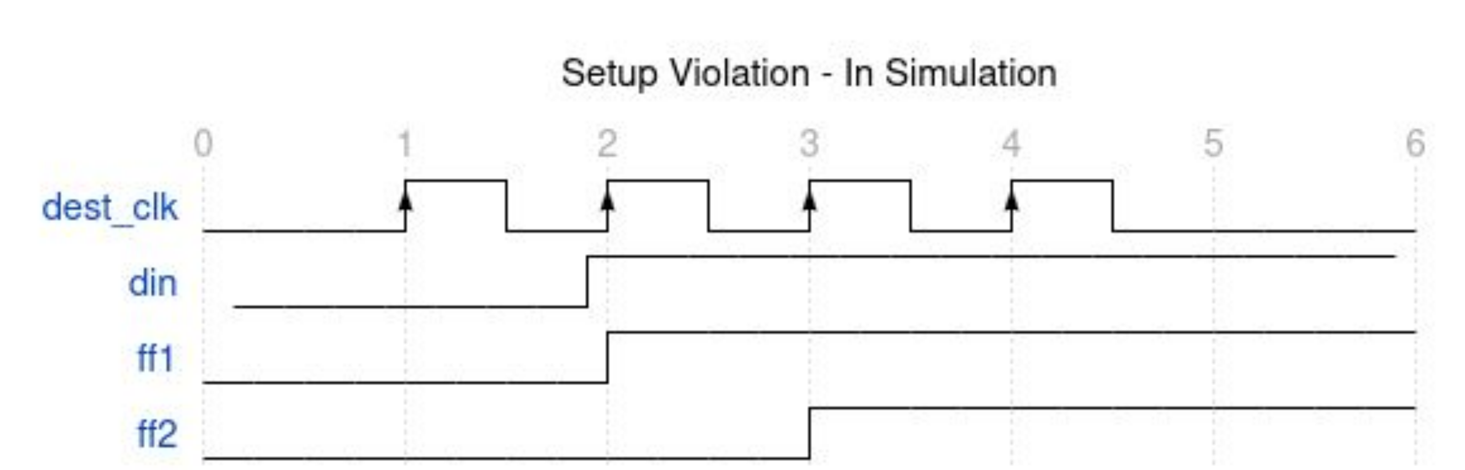
Synchronization Latencies for 2DFF Synchronizer - In Silicon





Synchronizers in Simulation



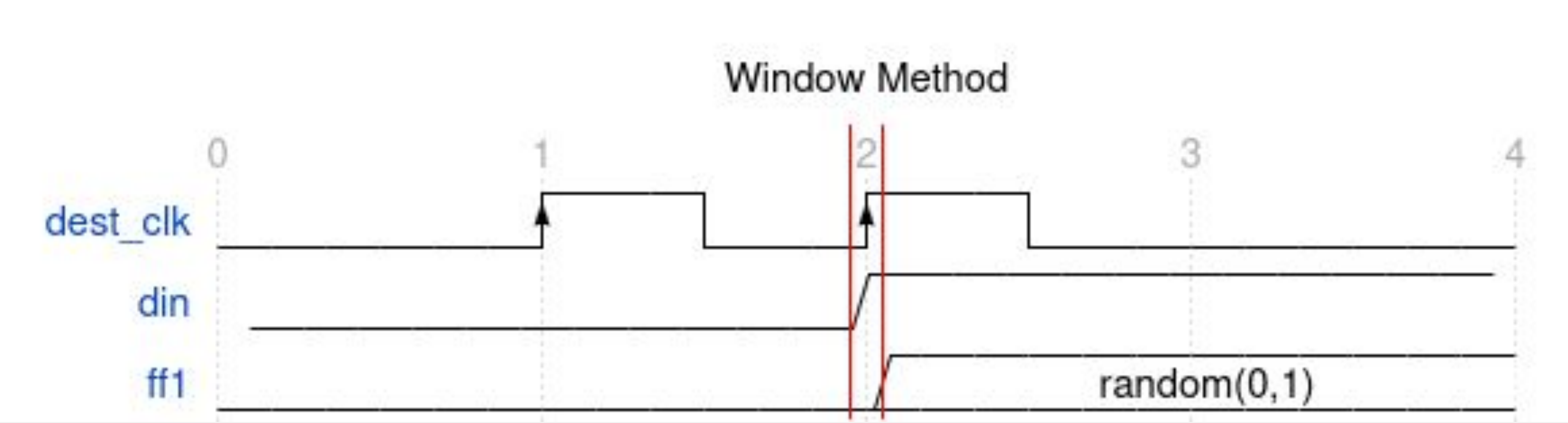


Proprietary Tools vs Custom Modeling

Custom Modeling	Proprietary Tools
● Requires upfront effort to develop a model	● No effort needed to develop custom DFF models - done automatically
● Must manually replace synchronizers with custom model	● Metastability models are automatically inserted into all CDC paths
● Must manually write assertions for protocol checking	● Automatically generates protocol assertions
● Must manually account for clock jitter	● No need to add clock-jitter features to testbench
● Entirely Free (if you create your own model or use an existing one)	● Very expensive: $10k+/year for a license

Naive Modeling

- Define a window size around the posedge of the clock
- Add a randomized clock Jitter
- Data transitions occurring inside the window cause the capturing flip-flop to randomly resolve to either 0 or 1

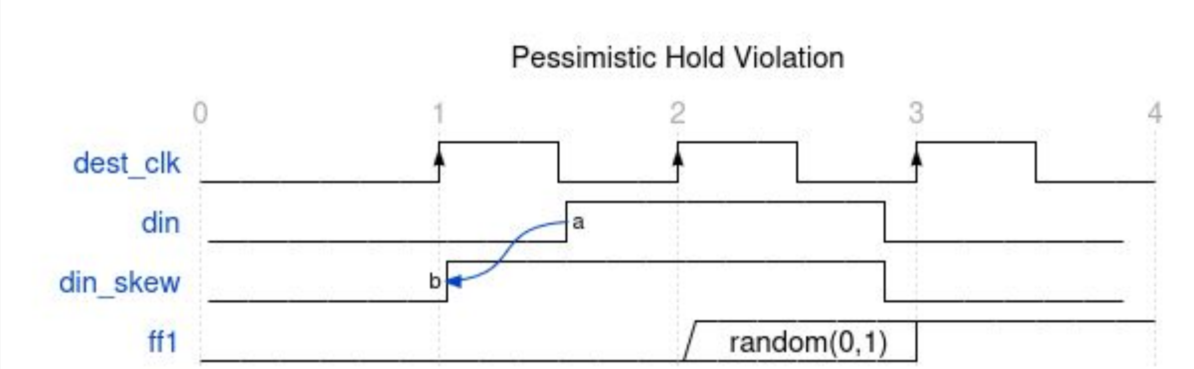


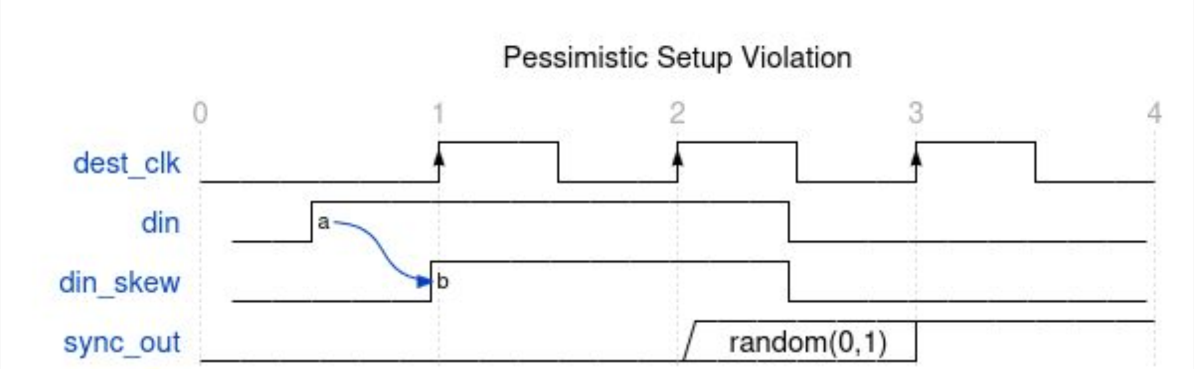
Naive Modeling – Issues

- Difficult to dynamically scale window size with clock period
  - If clock period becomes larger, and window size remains the same, the probability of metastability decreases.
- Would need to manually define different window sizes for different clock relationships to control pessimism – lots of manual work for large designs with multiple clock relationships
- Not pessimistic enough, want to exercise the design fully. A small window may mask bugs.
- Need a more abstract model that implies worst-case scenarios when possible
- Simulation time heavy – checking that data transitions happened within the window requires additional calculations which slow down simulation time, especially noticeable in large designs with many synchronizers

semify's Model

- Available to use entirely for free under the MIT license at github.com/semify-eda/dymanic_cdc
- Pessimistic model - skews timing of data inputs to force metastability and assumes worst case
- Parameterizable - switches for level of pessimism, data validity assertions, synthesis/simulation
- Hierarchical enables/blackboxing without the need for recompilation. Scales with large designs.
- More in-depth technical details of the model are available in the repository
- 2DFF synchronizers used to build more complex synchronization methods (FIFOs, Handshaking, etc) by using them on the respective control signals.





semify

solutions to simplify.

Contact Information

Email:
office@semify-eda.com

Website:
www.semify-eda.com

LinkedIn:
https://www.linkedin.com/company/semify-eda

Address:
semify GmbH
Neubaugasse 24
8020 Graz - Austria