



ASAP 2026



A Synthetic Benchmark Generator for Evaluating FPGA Incremental Tools

Presented by: Elmira Nezamfar

Supervisor: Prof. Steve Wilton

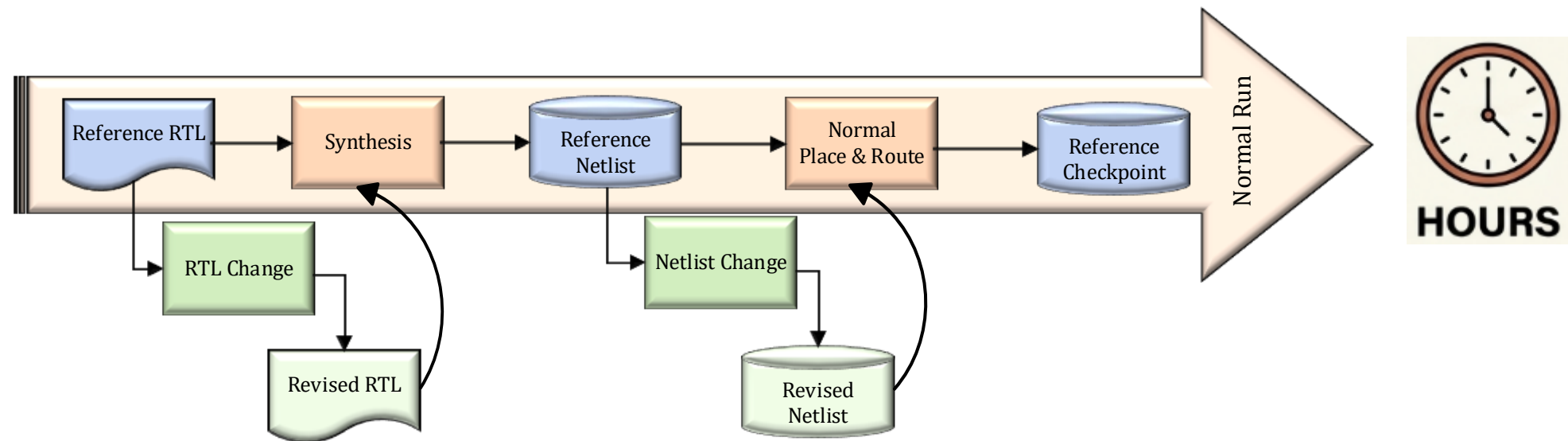
University of British Columbia



Electrical and
Computer
Engineering

September 2026

- Field-Programmable Gate Arrays (FPGAs) are essential for flexible, high-performance computing 
- Long compile time are a major bottleneck =>** Iterative development workflows =>
Hinder debugging, validation, and optimization 



- Modern FPGA toolchains offer **incremental compilation flows**, which reuse placement and routing data from a previous checkpoint to speed up compilation

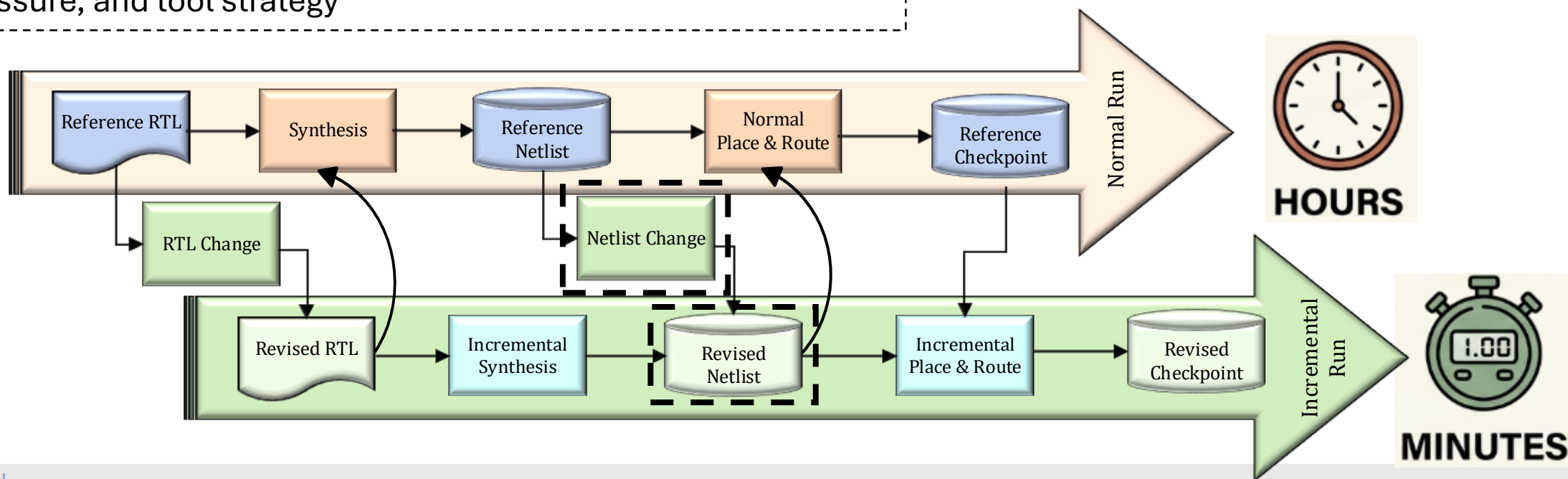


Industry feedbacks and Researches indicate that:

- ✗ Current flows can be **unpredictable** across design changes
- ✗ The effectiveness of incremental flow **depends** on reference selection, design-change type, routing disruption, timing pressure, and tool strategy

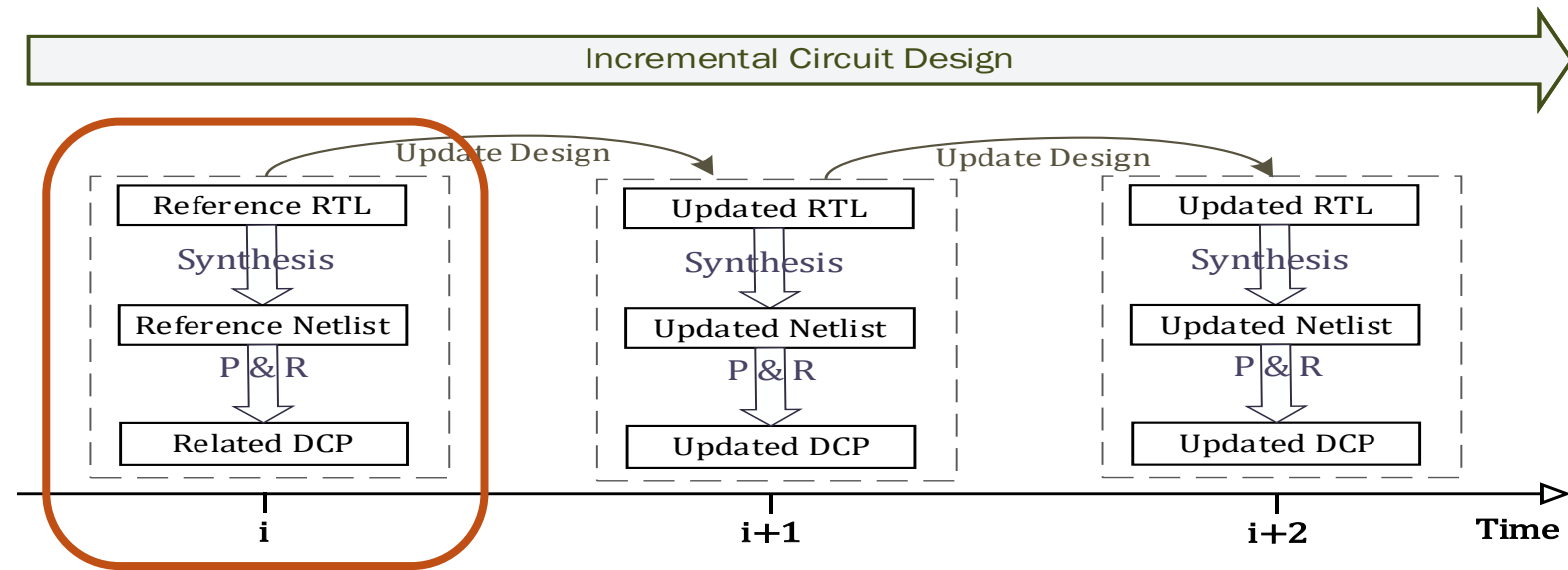


Opportunities for **significant Improvement and further Research**



Evaluating incremental FPGA CAD tools requires benchmarks with **multiple related design versions**

- ❑ Standard CAD benchmark suites (VTR, Titan, Koios, etc) provide **single static design snapshots**



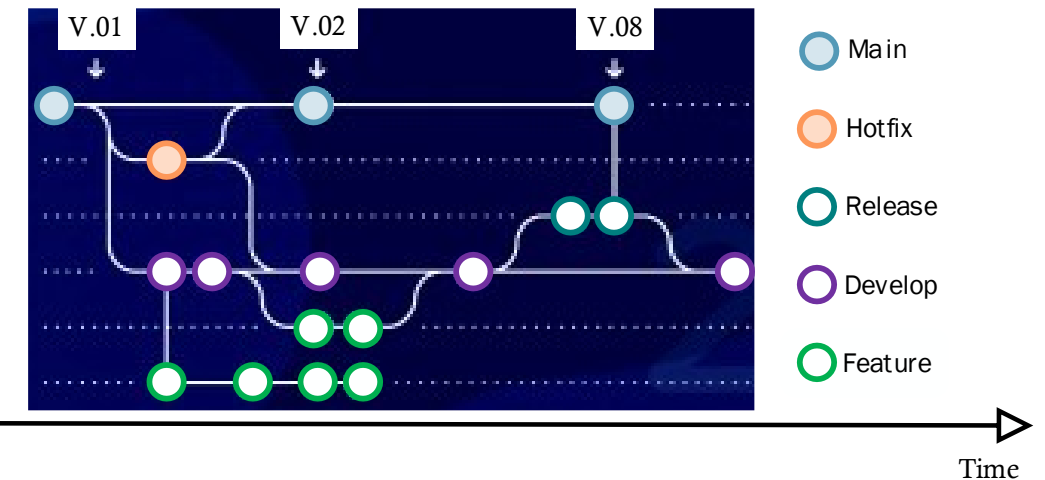
Evaluating incremental FPGA CAD tools requires benchmarks with **multiple related design versions**

Possible Approach

A. Use real design histories from version-controlled repositories [Chronbench¹]

- ✓ Authentic design evolution and real developer behavior
- ✗ Suitable FPGA designs with long, well-documented histories are scarce
- ✗ The number of distinct benchmarks is small
- ✗ Limited control over the size and nature of changes between versions

Incremental FPGA Design Development – Git Version Control



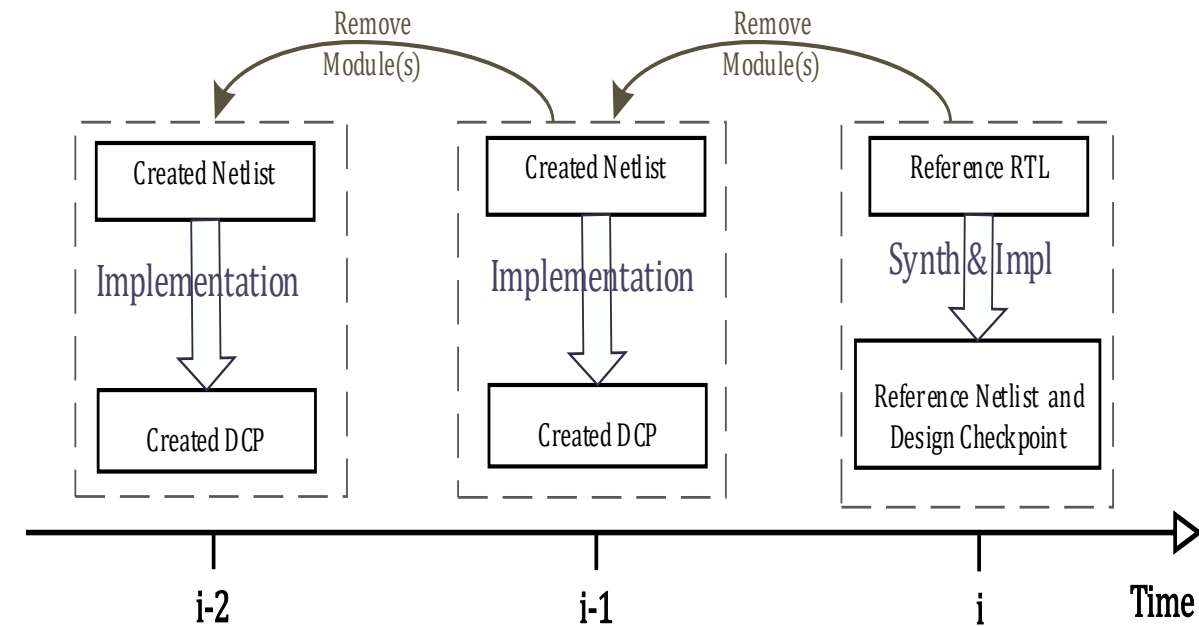
¹Nafziger Z, Wilton SJ. Chronbench: An Incremental HDL Benchmark Suite, IEEE FCCM, 2025.

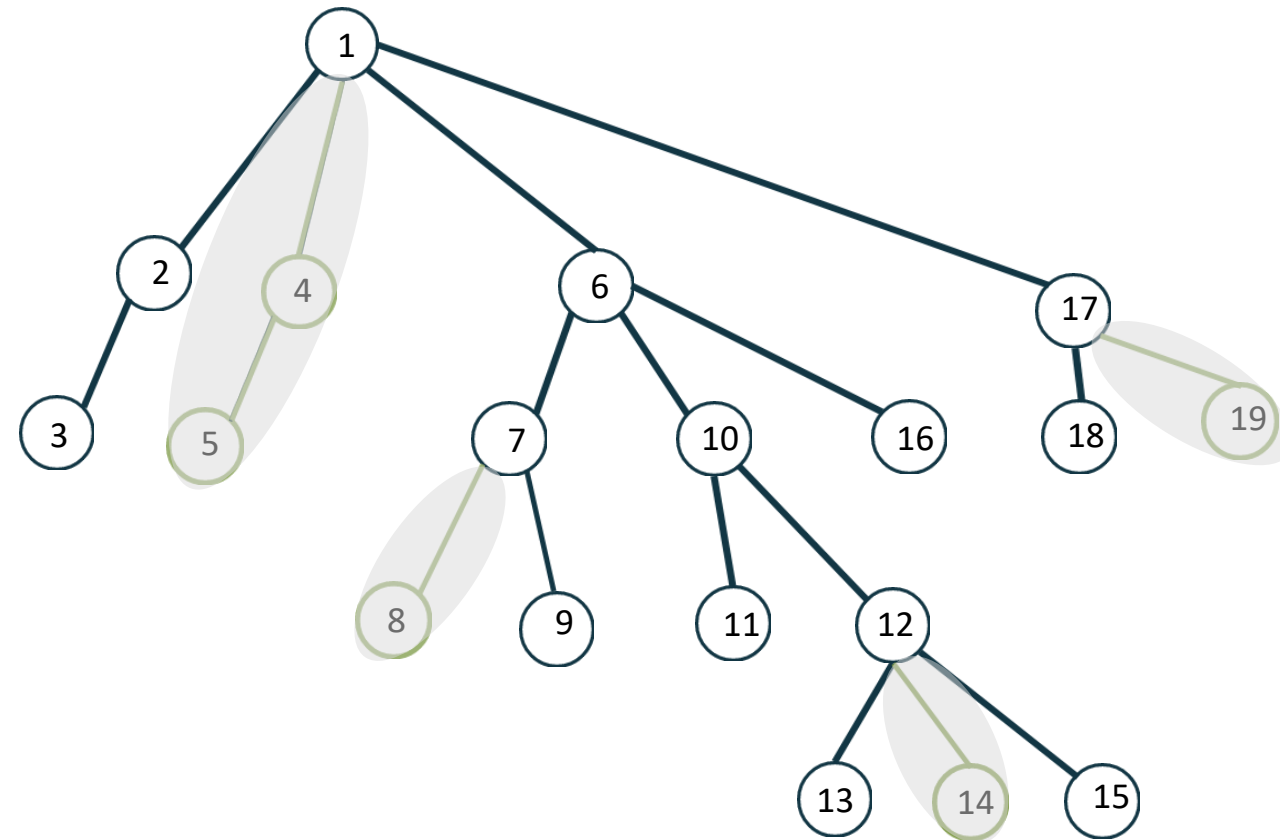
Evaluating incremental FPGA CAD tools requires benchmarks with **multiple related design versions**

Possible Approach

B. Generate synthetic benchmarks

- ✓ **Controlled size reduction** across history steps
- ✓ **Tunable similarity** between consecutive versions
- ✓ **Many histories generated automatically** from one source design
- ✓ **Reproducible** and consistent benchmark creation
- ✓ **Expands single-version** benchmarks into **multi-version** histories



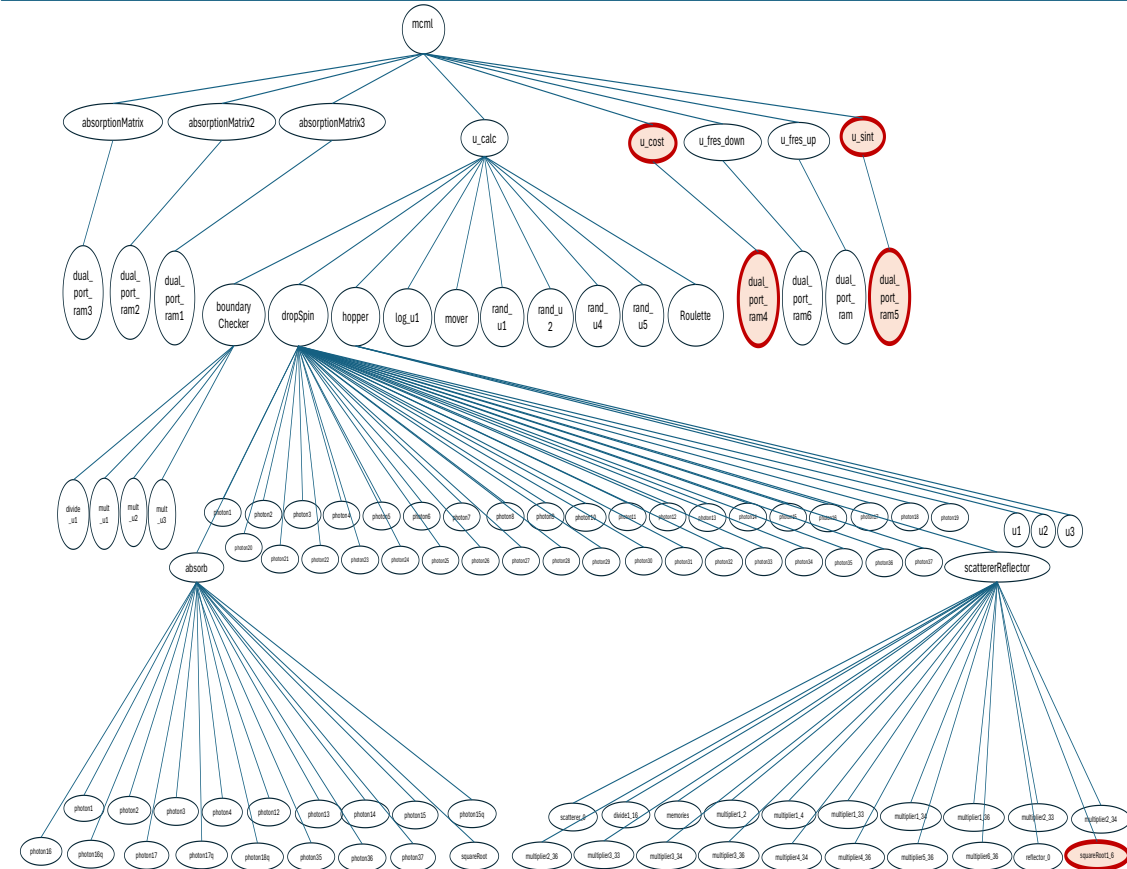


Hierarchy Module Scale Transformation

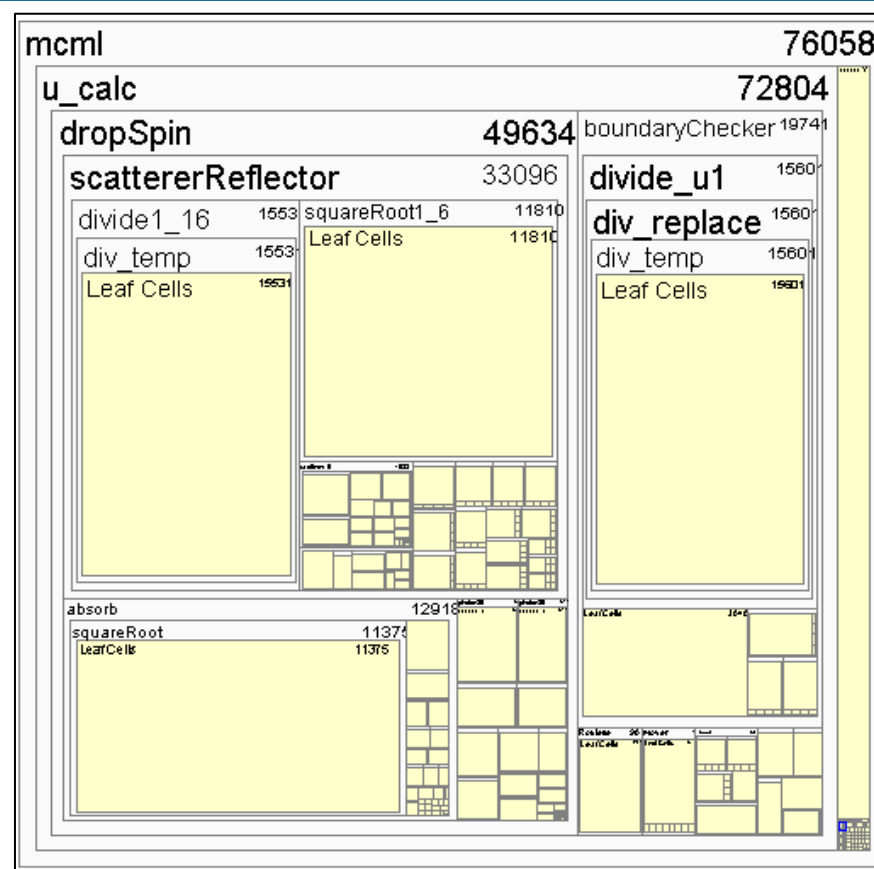
RTL Development Observations

- **Module-Scale Transformations in project progression**
- **Hierarchy-Oriented Changes**
- **Interface Stability During Local Internal Modification**
- **Gate-Level or Random Netlist-Level Modification is Unrealistic**

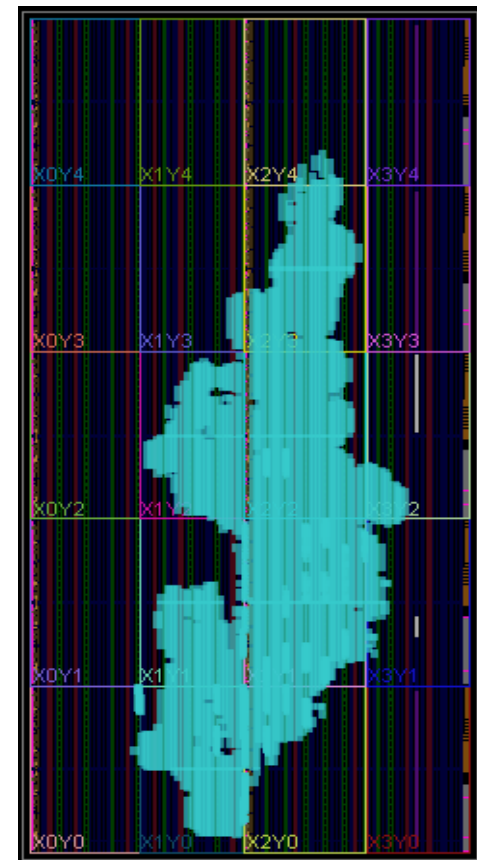
Source Design (D_i)



Hierarchal Tree Structure



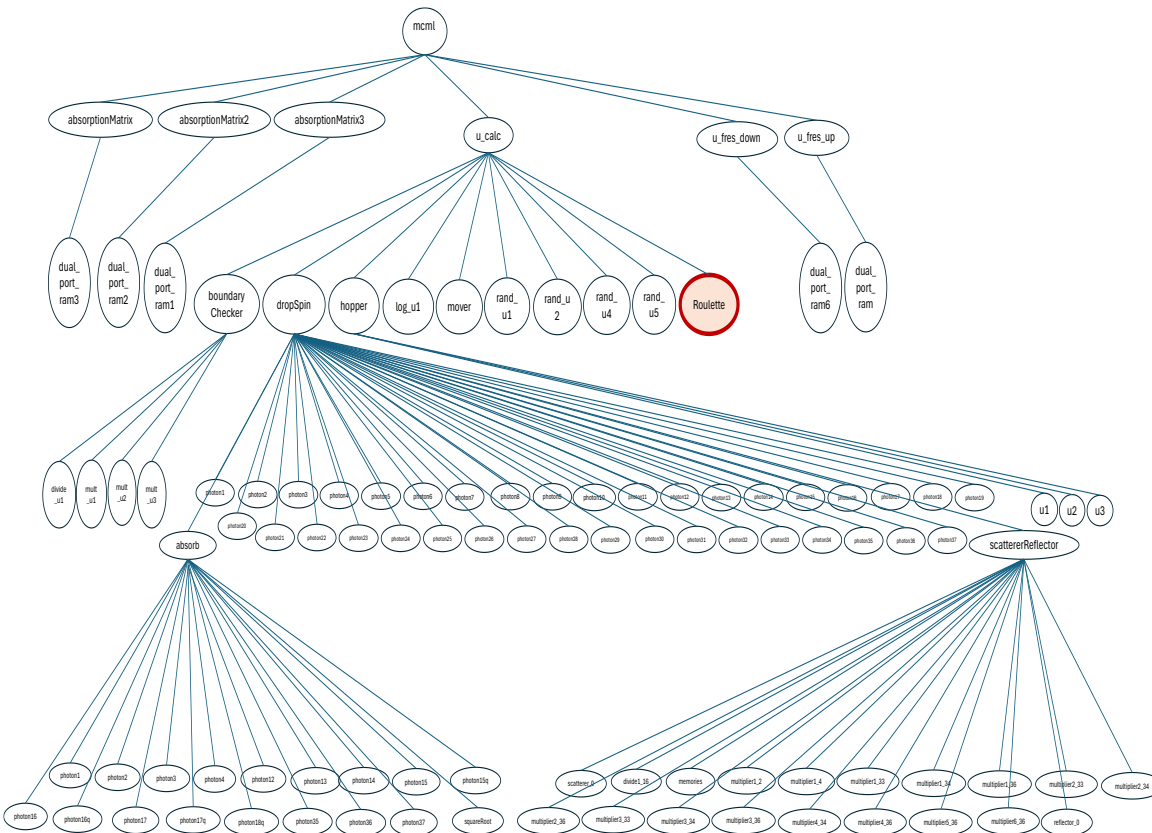
Hierarchal View with Cell Number



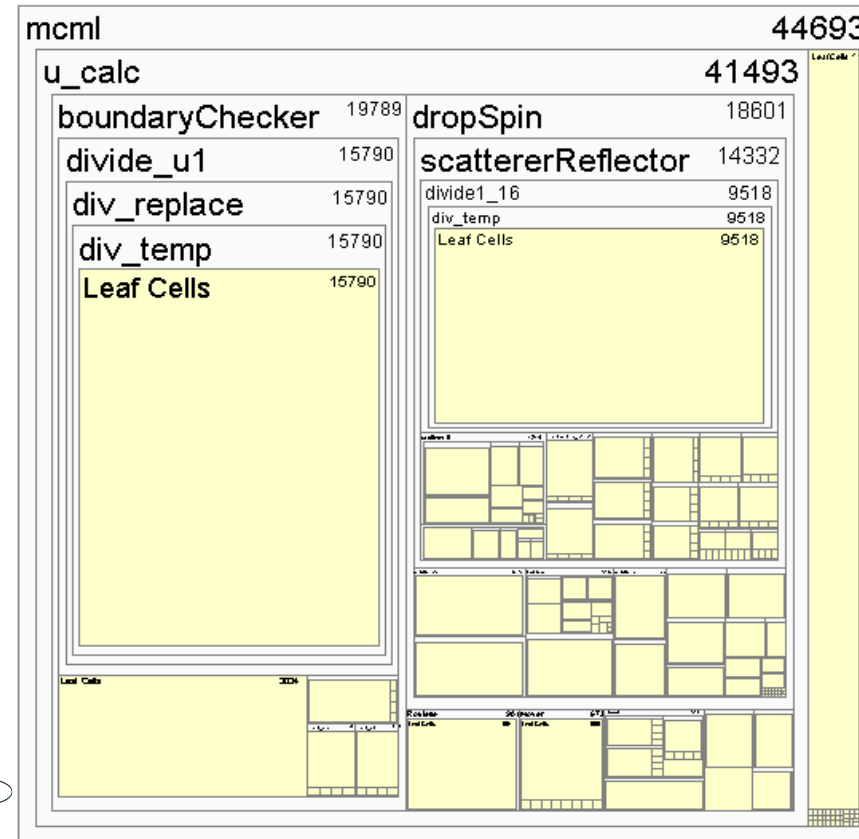
Floorplanning Design View

Instead of making arbitrary assumptions to assemble designs from the base design => starts with a source design (D_i) => systematically removes modules => reconstruct plausible predecessor snapshots ($D_{i-1}, D_{i-2}, \dots$)

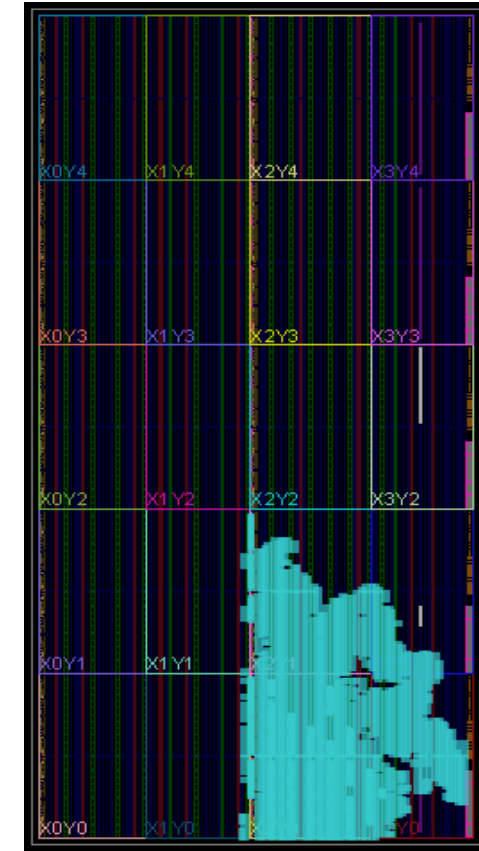
Predecessor Design (D_{i-1})



Hierarchal Tree Structure



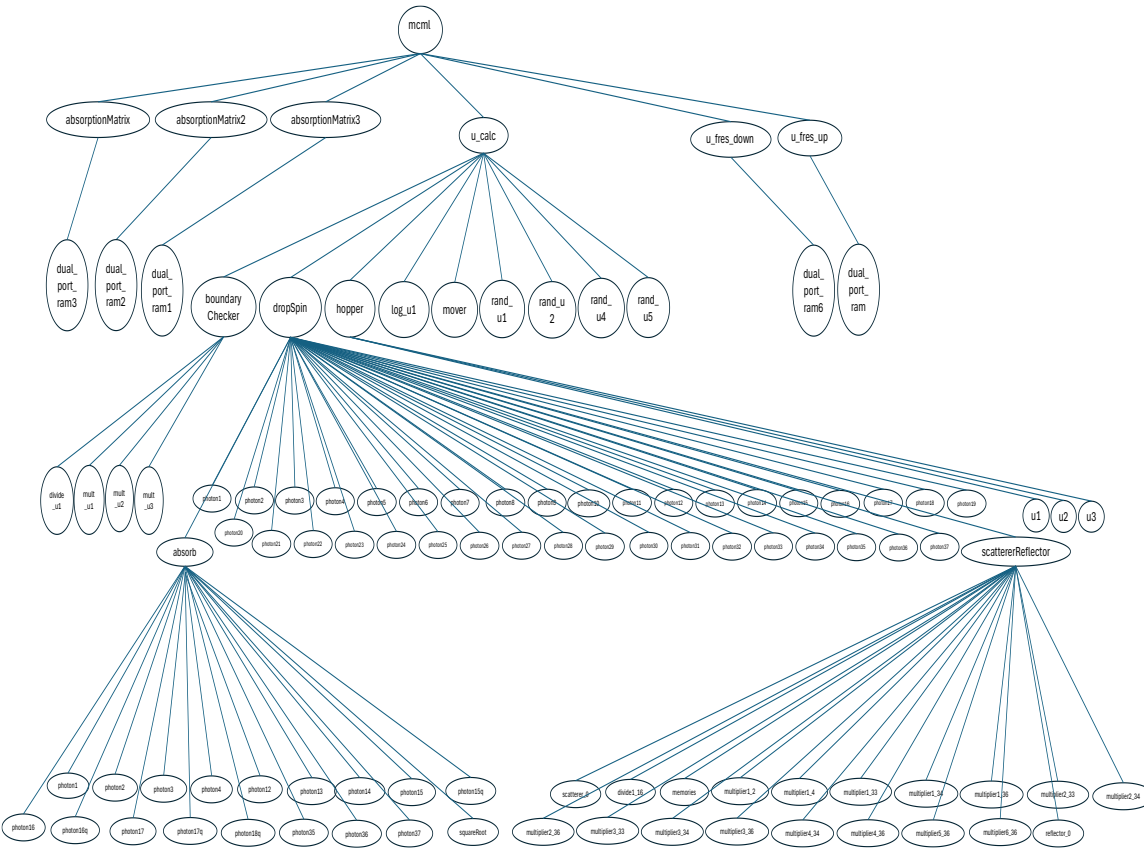
Hierarchal View with Cell Number



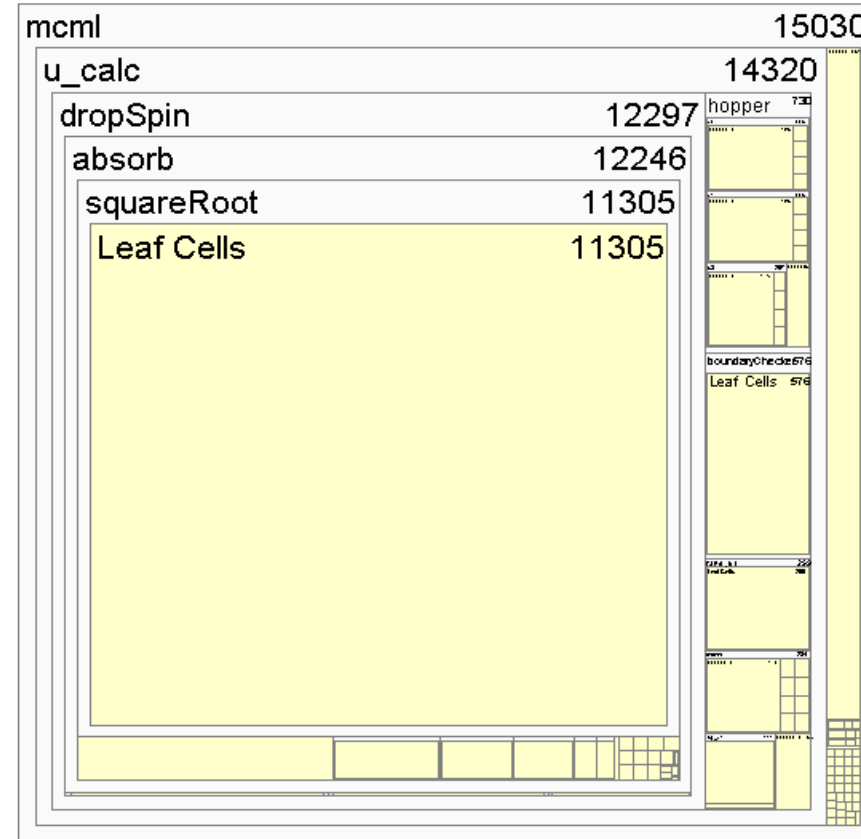
Floorplanning Design View

Instead of making arbitrary assumptions to assemble designs from the base design => starts with a source design (D_i) => systematically removes modules => reconstruct plausible predecessor snapshots (D_{i-1} , D_{i-2} , ...)

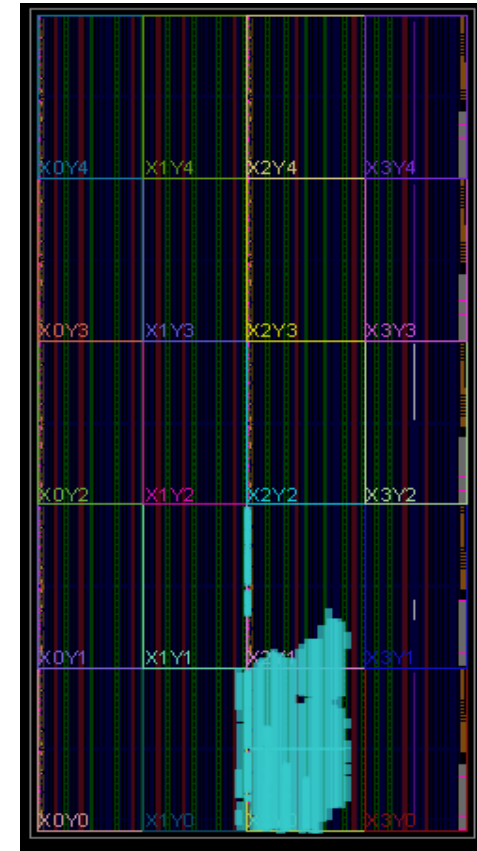
Predecessor Design (D_{i-2})



Hierarchal Tree Structure



Hierarchal View with Cell Number



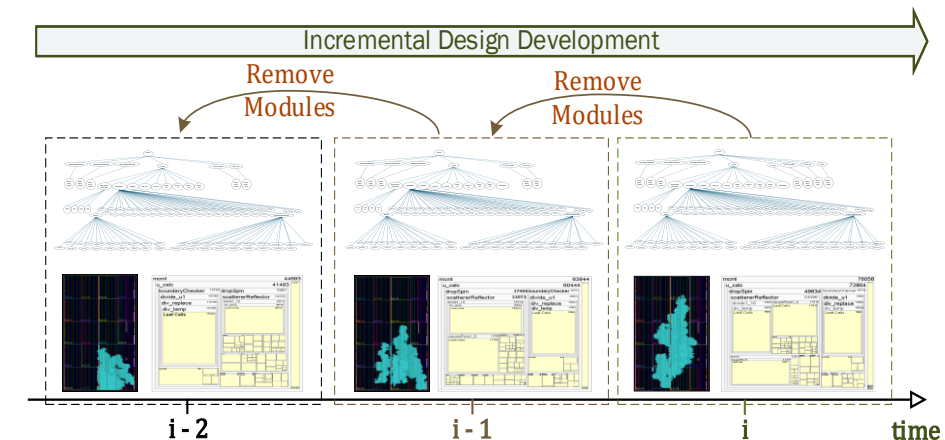
Floorplanning Design View

Instead of making arbitrary assumptions to assemble designs from the base design => starts with a source design (D_i) => systematically removes modules => reconstruct plausible predecessor snapshots ($D_{i-1}, D_{i-2}, \dots$)

- Given a FPGA design at iteration i , D_i
- Generate a predecessor design at iteration $i-1$, D_{i-1}
- With **TWO** primary **Constraints**:

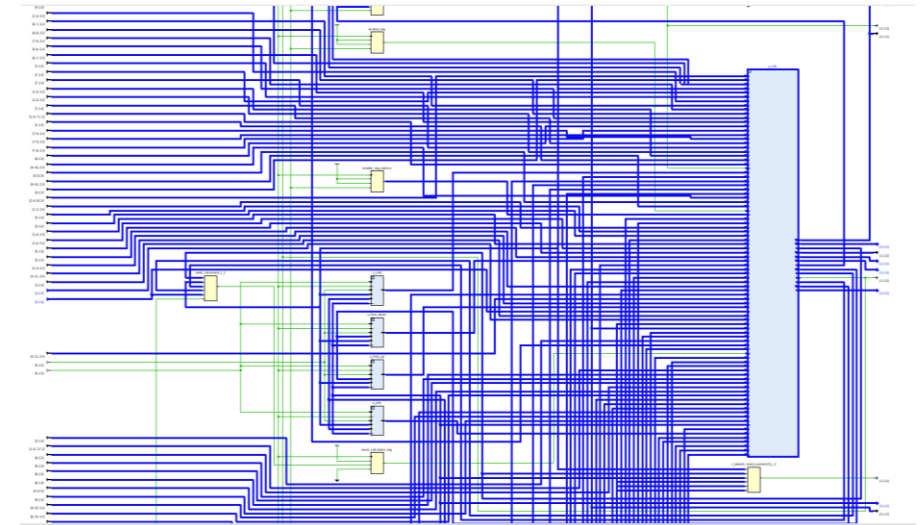
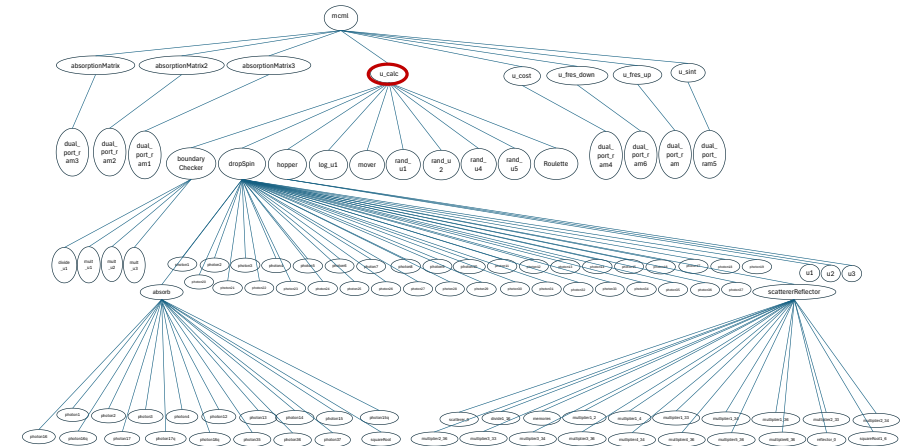
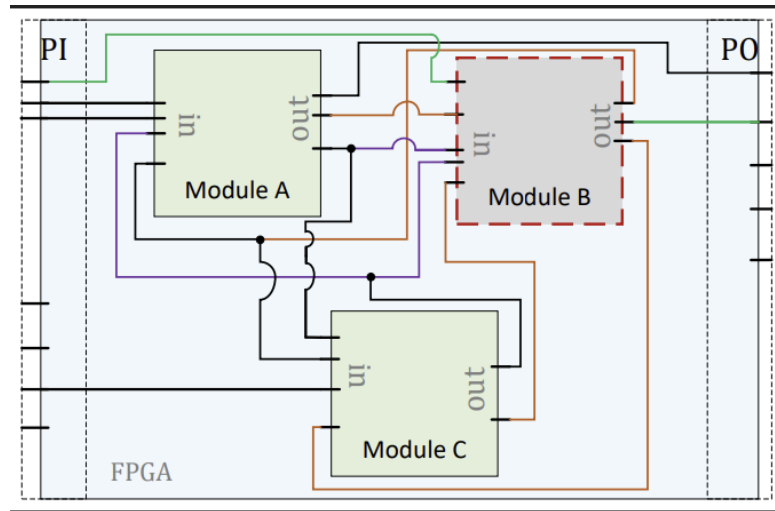
Constraint 1

“Controlling the magnitude of structural change”



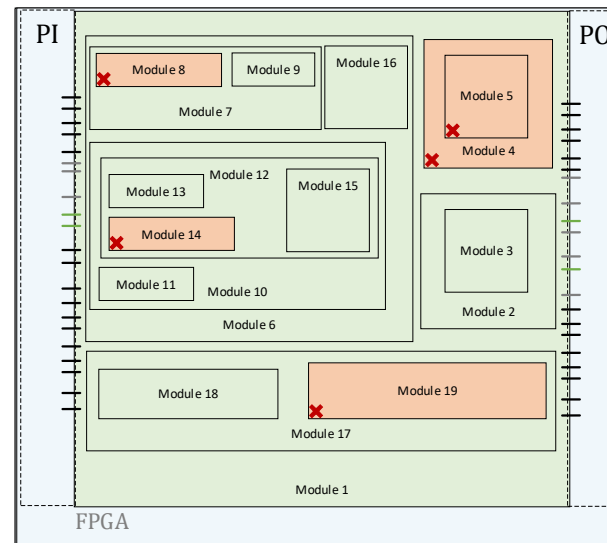
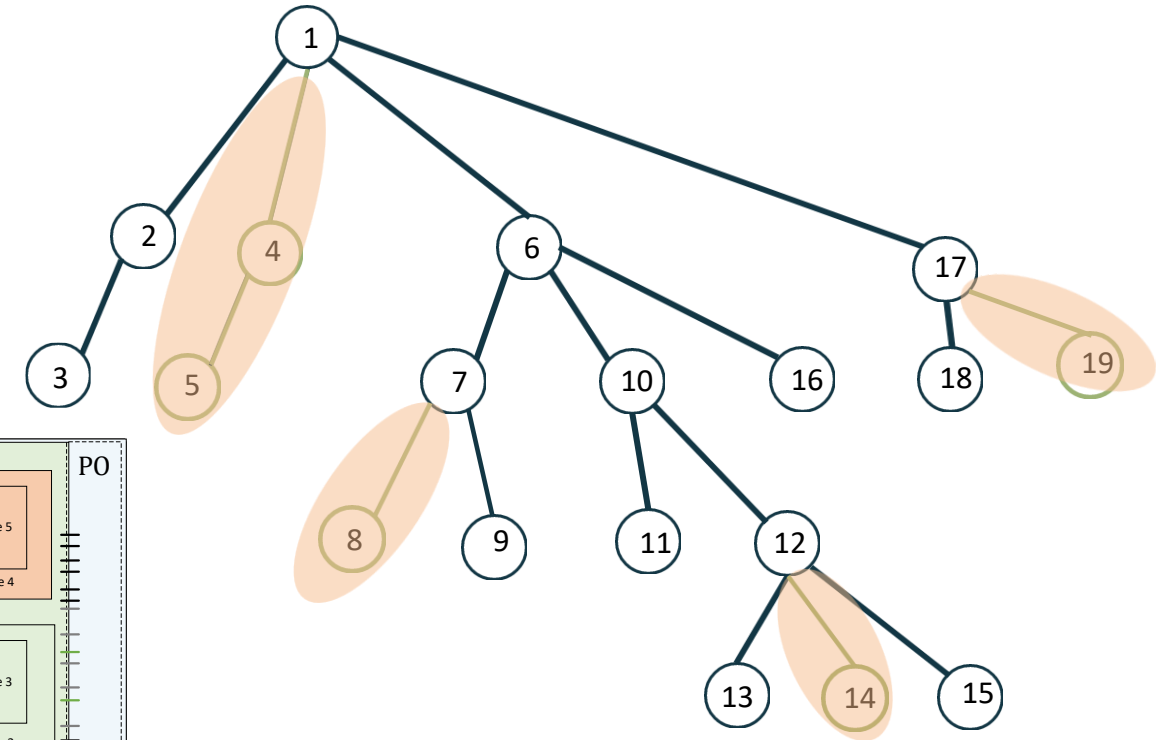
Constraint 2

“Preserving interface stability”



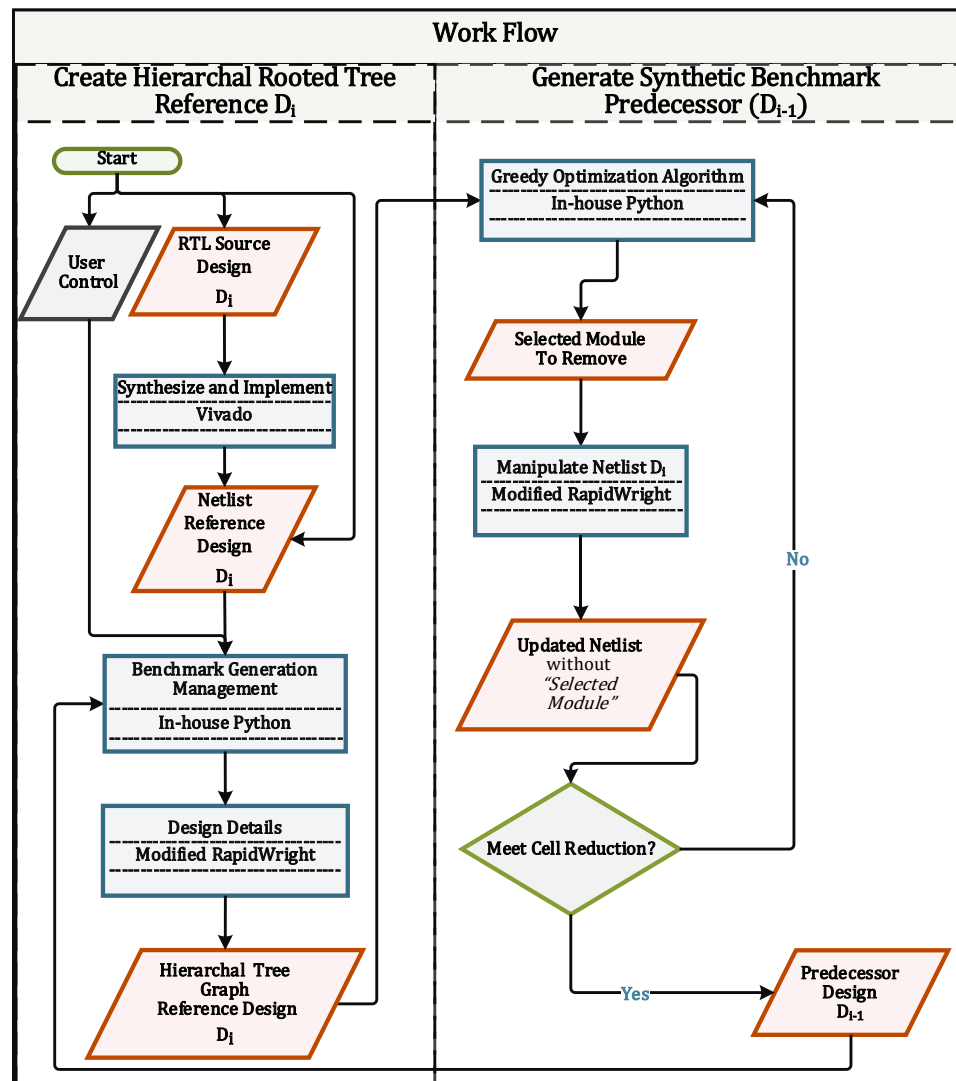
- Formulate module removal as a multi-objective **optimization problem**

$$C_m = a \times \Delta_{cell}_m^{norm} + b \times \Delta_{IO}_m^{norm}$$



Generate a new design at iteration $i-1$, D_{i-1}

- ✓ **Automated** environment for generating synthetic benchmarks
- ✓ Can generate histories for design at **RTL Level** or **Netlist Level**
- ✓ Provides **tunable control** over how aggressively each generated history reduces the design size



- ✓ Built on top of the **RapidWright**, AMD/Xilinx Research Labs' open-source framework



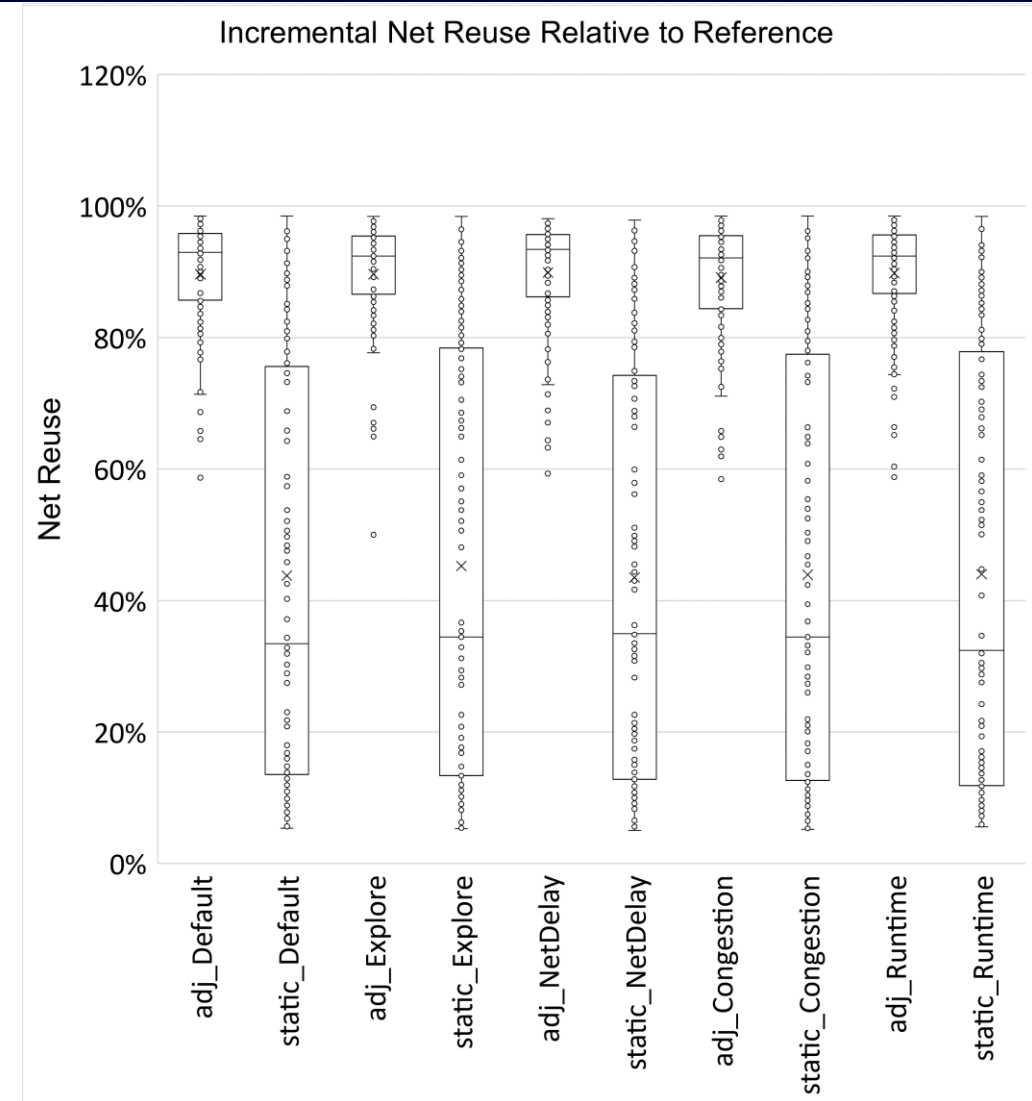
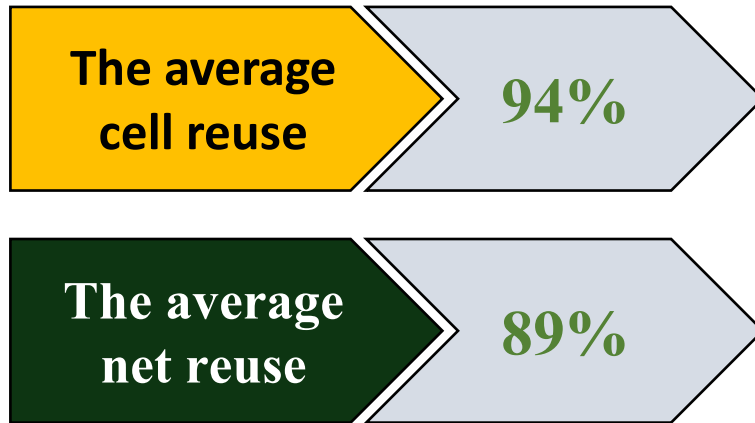
- ✓ **Open-source and publicly available**



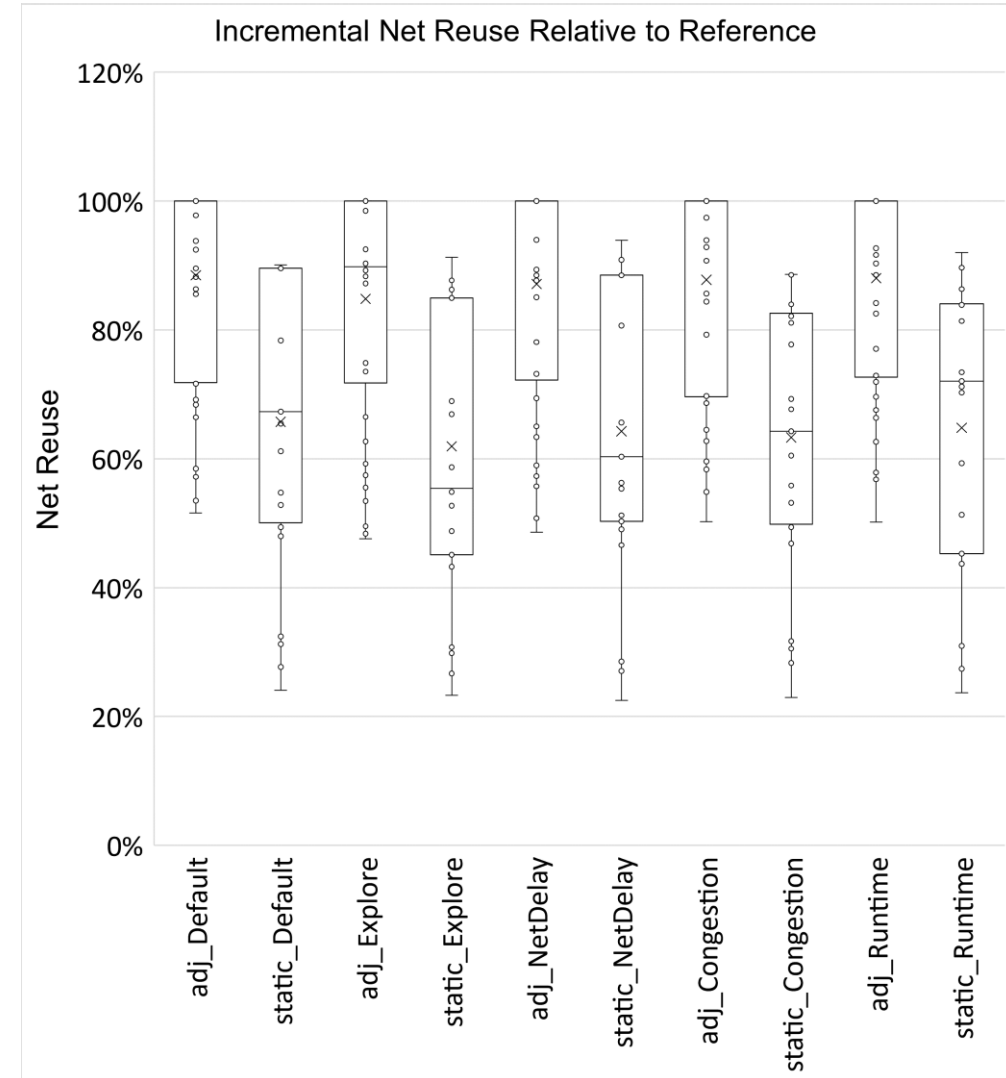
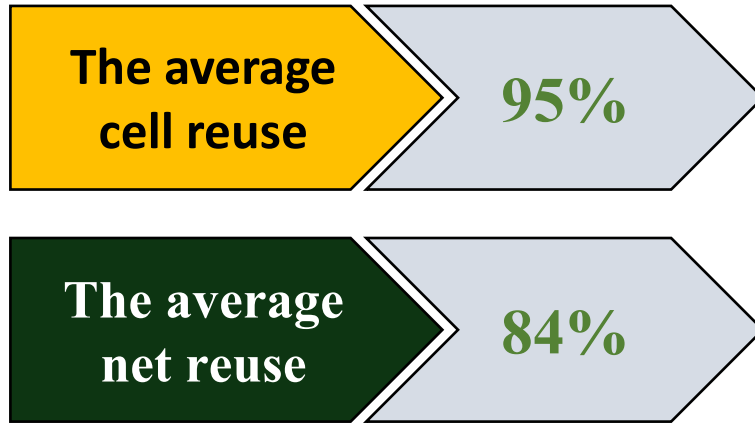
https://github.com/ElmiraNezamfar/Synthetic_Benchmark_Suite_Generator

Benchmark	Source Tree Structure			Synthetic Generation Metrics				
	Hierarchical Nodes (Mod/Inst)	Leaf Nodes (Mod/Inst)	Max Depth	Max Generated Histories	Max Nodes Decrease (%)	Max Leaf Decrease (%)	Depth Stable (%)	Max Depth Decrease (#Level)
attention_layer	248	223	5	9	12.10	9.87	100	0
spmv	553	229	5	21	27.30	26.2	100	0
bgm	150	100	3	24	89.30	89	87	1
raygentop	16	12	3	4	56.25	58.33	50	1 ↓
mcml	137	118	5	8	13.14	15.25	100	0
stereovision0	252	206	3	58	93.25	94.17	100	0 —
stereovision1	233	184	4	12	30.90	19.57	67	1
stereovision2	140	127	2	5	12.14	13.39	100	0
cva5	57	43	3	9	85.96	86.05	56	1

Generated synthetic benchmarks



Real GitHub Histories



Limitation

- ❑ **backward module removal**
=> not capture all real design changes
- ❑ The method depends on the **quality of the input hierarchy**
- ❑ **Functional** equivalence is not preserved by construction

Future Work

- **Extend** the generator to support finer-grained changes, such as sub-module or cell-level modifications
- **Compare** generated histories against a larger set of real open-source repository histories
- **Apply feedback** or machine learning to guide module selection toward specific CAD stress cases, such as congestion, power constraint, or timing degradation

- ✓ Automated FPGA benchmark-history generator using Python + RapidWright
- ✓ Generates structurally valid EDIF histories through module removal and netlist repair
- ✓ Greedy selection balances design-size reduction and interface stability
- ✓ Generated histories preserve key structural properties while introducing useful implementation variation
- ✓ Results show suitability for evaluating incremental FPGA flows: reuse, timing, runtime, and strategy sensitivity