Reduction of Test Time and Improvement of Coverage in DFT ATPG

EE3405 Semester Internship Report

Rayi Giri Varshini
EE22BTECH11215
IC Design and Technology
Indian Institute of Technology, Hyderabad

Internship Duration: 16th January 2025 - 11th July 2025

Abstract

This report summarizes the work performed during the semester-long internship at STMicroelectronics, Greater Noida. As an Associate Front-End DFT Engineer in the Dig FEM TR&D SILICON QUAL HUB (TestChip) team, I worked on tasks across 18nm, 28nm, and 90nm nodes aimed at improving DFT architecture, reducing test time, and enhancing fault coverage. The following document outlines the technical contributions, methodologies, and results obtained during the internship period.

Contents

1 Introduction

The purpose of this internship was to gain practical exposure to industrial-grade DFT and ATPG methodologies and contribute to real-world problems involving scan-based testing, RTL optimization, and automation. During this period, I worked on 12 tasks dealing with RTL modification, simulation, pattern generation, automation using Python, and silicon debug for memory blocks.

2 Internship Details

- **Name:** Rayi Giri Varshini
- **Roll Number:** EE22BTECH11215
- **University:** Indian Institute of Technology Hyderabad
- **Department:** IC Design and Technology
- **Industry Mentor:** Mr. Sachin Bakshi
- **Internship Period:** 16th January 2025 - 11th July 2025
- **Organization:** STMicroelectronics, Greater Noida
- **Team:** SQH Dig FEM TR&D (TestChip)
- **Role:** Associate Front-End DFT Engineer

3 Tasks Undertaken

3.1 Task 1: RTL Modification in Register Bank

Objective:

Modify REGBANK RTL to introduce a common capture register while retaining individual update registers, aiming at reducing area and power without affecting correctness.

Plan and Modifications:

- A common capture register of length 96 is required, where each register will continue to have dedicated update registers, which remain unchanged.
- `USER_REG` has both capture and update blocks where the first capture happens, then update using `USER_FF` for each. So, separate modules of `USER_REG_CAPTURE` and `USER_REG_UPDATE`, where capture is instantiated once and update is instantiated five times for 5 registers. Reduces 10 USER_FFs to 6 by optimized instantiation.
- Instead of instantiating two clock gating cells in `USER_FF`, using only one CGT in `USER_FF` and adding the other CGT in `USER_CAPTURE` and `USER_UPDATE` eliminates multiple instantiations and effectively reduces area.

Tools Used: Xceliumagile (Simulation), Genus (Synthesis)

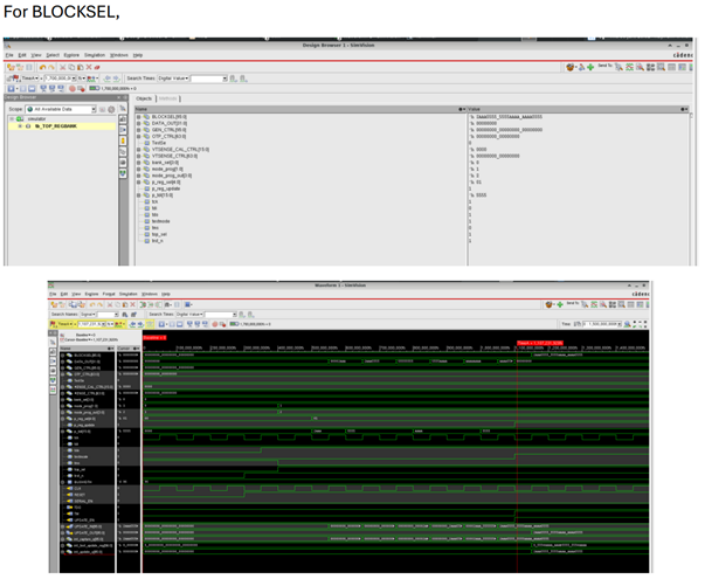


Figure 1: Simulation Results

Results:

Metric	Original	Common Capture	Optimized CGT
Area (μm^2)	29884.98	19833.01	15153.47
Power (mW)	0.0811	0.0385	0.0235
Timing Slack (ns)	94.2	92.78	93.45

3.2 Task 2: Error Correction Codes (ECCs) in REGBANK

Objective

To explore Error Correction Codes (ECCs) for preventing error propagation in a shared-capture register architecture by supporting single-bit error correction, double-bit error correction, and triple-bit error detection.

Motivation

In the REGBANK design, a common capture mechanism has been added for all registers. If an error occurs in one register, it can propagate to others through this shared capture path, increasing the risk of systemic corruption. Although resets are applied after every register, the possibility of propagation remains high. Therefore, ECCs are explored to mitigate this.

What are ECCs?

Error Correction Codes (ECCs) are mathematical methods used to detect and correct errors in digital data. They achieve this by adding redundant bits to the data in such a way that the original information can be recovered even in the presence of certain errors.

Why ECCs?

Error Correction Codes are essential in modern digital systems due to:

- Low-voltage operation and reduced noise margins, which make digital logic more susceptible to faults.
- The need for robust data integrity in cryptographic hardware and memory systems.
- Avoid loss of time in post-silicon validation or automation flows (like cronjobs), where errors might otherwise go undetected during runtime.
- Evaluating the Timing-Area-Power (TAP) trade-offs for designs with and without ECC in the presence of a shared capture mechanism.

Hamming Code (Standard)

A widely used ECC for detecting and correcting single-bit errors. It works by adding parity bits to the data word to form a code word.

- Detects and corrects single-bit errors.
- Uses XOR logic and parity positions.
- The number of parity bits p required for d data bits follows: $2^p \geq d + p + 1$.
- Uses block parity and is efficient for forward error correction (FEC).

Extended Hamming Code (SECDED)

This version adds an overall parity bit to the standard Hamming code, allowing:

- Correction of single-bit errors.
- Detection (but not correction) of double-bit errors.
- Known as SECDED - Single Error Correction, Double Error Detection.

BCH Code (SECDECTED)

Bose–Chaudhuri–Hocquenghem (BCH) codes are more advanced, capable of correcting multiple errors within a data block. They are constructed over Galois fields and support:

- Single/Double error correction
- Triple error detection
- Higher flexibility at the cost of increased complexity and logic area

Can We Extend Hamming for Triple Error Detection?

Although theoretically possible, extending Hamming for triple-bit error detection becomes inefficient and increases the area and logic complexity. BCH or other cyclic codes offer a more scalable and optimized solution for such cases.

Highlights and Future Work

- Studied and compared Hamming, Extended Hamming (SECDED), and BCH Codes.
- Evaluated trade-offs in triple-bit detection using extended logic.
- Future work: Perform TAP (Timing, Area, Power) analysis of ECC integration in REGBANK under shared capture vs. independent capture architectures.

3.3 Task 3: ATPG Pattern Generation and Simulation

Objective

Top-level ATPG Pattern Generation and Simulation for Zero Delay and SDF (FF & SS) conditions for `MEMBLOCK1_SPHD_LOLEAK` and `MEMBLOCK1_SPREG_LOLEAK` in `MEMBLOCK1_ST`, across various modes, fault types, and pattern modes.

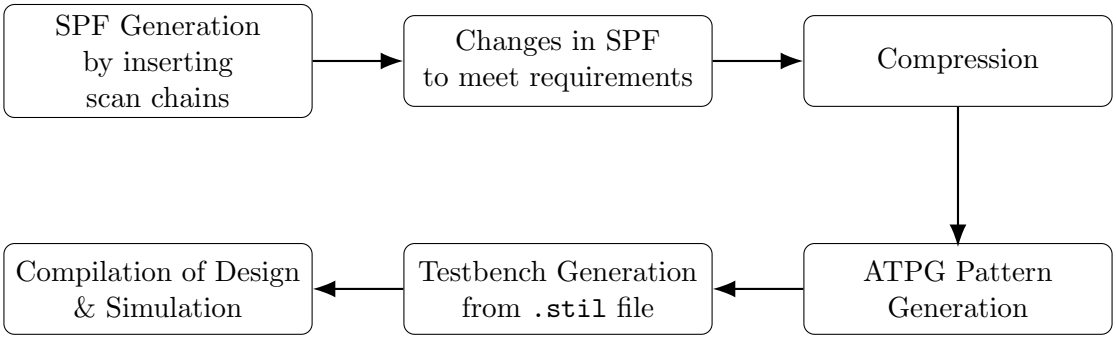
Tools Used

- **Tetramax**: For ATPG pattern generation
- **Xceliumagile**: For simulation of the testbench

Modes and Configurations

- **Modes:** LOGIC, LOGIC_MEM, LOGIC_COMP, LOGIC_MEM_COMP, ONLY_MEM, REGBANK
- **Process Corners:** Zero Delay, SDF (FF and SS)
- **Pattern Modes:** Parallel (par), Serial (ser)
- **Fault Types:** Stuck-at, Low Speed, At Speed
- **ATPG Modes:** Auto, Chain Test, Functional Test

Process Flow



Modifications in SPF for ATPG Simulations

Several adjustments were made to resolve simulation mismatches and enhance testbench reliability:

- Mismatches caused by analog signals and manual test setups
- Need to determine optimal strobing values for simulation accuracy

Solutions Implemented

1. Removed analog signals `MEASRA_MB1` and `MEASRA_MB2`
2. Introduced incremental flow to reduce total test time
3. Inserted two test vectors with clocks turned off (dummy cycles) to meet setup/hold constraints
4. Added reset vector definitions in the test setup using `Macrodefs`
5. Integrated PLL programming and pre-shift cycles for high-trans simulations

3.4 Task 4: Prolib3D28 FDS ATPG SPHD_BB_COMP At-Speed and Low-Speed TF

Objective

To analyze and resolve ATPG test failures observed on silicon for the `SPHD_BB_COMP` module in the `Prolib3D28 FDS` test environment. These failures were specific to low-transition (low-trans) and high-transition (high-trans) test patterns.

Observed Issues

- **High-Trans Failures:** 2 failures observed on silicon
- **Low-Trans Failures:** 9 failures observed
- Failures were seen at specific DOUT pins during certain clock cycles
- No such failures were seen in FF or SS corner simulations, making debug more challenging

Initial Debug Strategy

- Backtraced from `mem_bist_wrapper`, assumed to be involved in failure
- Focused on failures in DOUT[3] and DOUT[10] in high-trans tests
- These correspond to:
  - `C84 → DOUT[3]`
  - `C91 → DOUT[10]`
- Used schematic trackers and hierarchy tracing tools to analyze drivers and loads
- Multiple potential failure candidates were identified but root cause remained uncertain

Root Cause Analysis

- No failures seen in FF or SS corner simulations
- Indicates a mismatch between silicon environment and simulation assumptions
- Potential causes categorized as below:

Possible Root Causes and Next Steps

1. **Test Pattern Mismatch:**
   - Ensure the same ATPG patterns are applied in both silicon and simulations
   - Validate capture/strobe windows and scan chain alignment
2. **Backtracing Critical Paths:**
   - Investigate CUTs C84 and C91 at DOUT[3] and DOUT[10]
   - Examine signal dependencies and logic cones in design
3. **Timing Differences:**
   - Check for hold/setup violations in at-speed paths
   - Investigate timing closure in ATPG synthesized paths
4. **Signal Integrity Issues:**
   - Analyze layout-related issues like crosstalk, ground bounce, and IR drop
   - Perform simulations with extracted parasitics if not already done
5. **Environmental Mismatch:**
   - Cross-check simulation vs silicon environment parameters (temperature, voltage corners)
   - Review tester setup and silicon probing configurations

Conclusion

The failures observed on silicon during high- and low-trans ATPG testing require detailed correlation between testbench simulations and real-silicon conditions. Emphasis is placed on refining backtracing methodology, ensuring test pattern integrity, and investigating physical effects in silicon to isolate and resolve the discrepancies.

3.5 Task 5: ATPG Chain Test for Maximum Shift Frequency

Objective

To determine the maximum working frequency and optimal strobing range for `SPHD` and `SPREG` memory blocks in the scan chain test using `LOGIC_COMP` and `LOGIC_MEM_COMP` modes for Stuck-at Faults.

Approach

- ATPG patterns were generated specifically for the chain test mode.
- Simulations were run for varying frequencies, rise/fall times, and strobing windows.
- Evaluated results across two process corners: FF and SS.
- The primary performance metrics evaluated:
  - Maximum shift frequency without mismatches.
  - Minimum and maximum working strobing times.
  - Effect of process corner variation on timing margins.

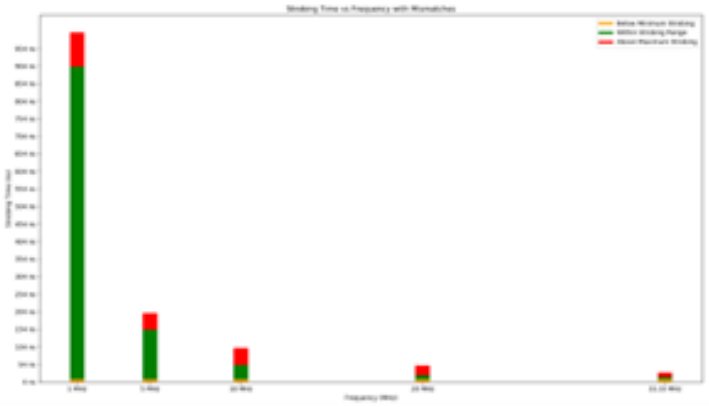


Figure 2: For FF Corner

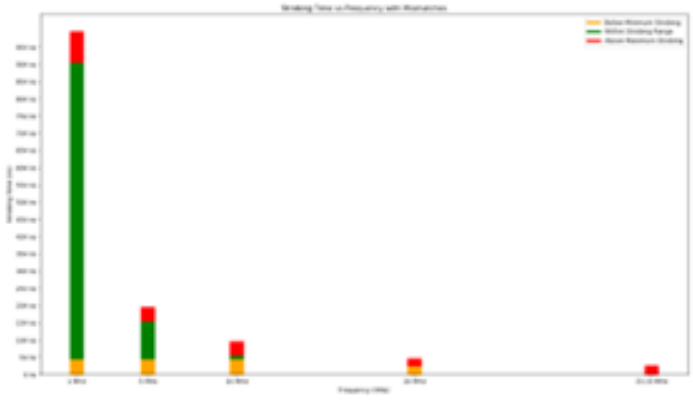


Figure 3: For SS Corner

Results and Observations

1. **Maximum Shift Frequencies:**
   - `SPHD Block`: 9.52 MHz
   - `SPREG Block`: 12.5 MHz

Freq (MHz)	Time period, Tr, Tf (ns)	Strobing range - SPHD LOGIC_MEM_COMP (ns)	Strobing range - SPHD LOGIC_COMP (ns)	Strobing range - SPREG LOGIC_MEM_COMP (ns)
1	1000, 900, 930	12-902	10-902	11-902
5	200, 150, 180	12-152	10-152	11-152
10	100, 50, 80	12-52	10-52	11-52
20	50, 20, 40	12-22	10-22	11-22
33.33	30, 15, 25	12-17	10-17	11-17

Figure 4: For FF Corner

Freq (MHz)	Time period, Tr, Tf (ns)	Strobing range - SPHD LOGIC_MEM_COMP (ns)	Strobing range - SPHD LOGIC_COMP (ns)	Strobing range - SPREG LOGIC_MEM_COMP (ns)
1	1000, 900, 930	72-908	71-908	48-908
5	200, 150, 180	72-158	71-158	48-158
10	100, 50, 80	Fail	Fail	48-58
20	50, 20, 40	Fail	Fail	Fail
33.33	30, 15, 25	Fail	Fail	Fail

Figure 5: For SS Corner

2. **Strobing vs Frequency Trade-off:**
   - As frequency increases, the available strobing window decreases.
   - This limits the effective margin for timing alignment, especially at SS corner.

Conclusion

The ATPG Chain Test provides crucial insights into the frequency-strobing interaction in memory scan chains. With FF corner showing tighter margins and higher performance, and SS corner indicating longer delays, the strobing time must be carefully tuned to ensure data capture integrity. These findings help in defining robust test conditions for production testing and silicon validation.

3.6 Task 6: Python Automation Flow for Maximum Chain Test Frequency and Simulation

Objective

To automate the process of maximum frequency evaluation and simulation for chain tests using Python scripts, minimizing manual intervention while enhancing efficiency and consistency across test setups.

Script 1: Maximum Frequency Sweep Automation

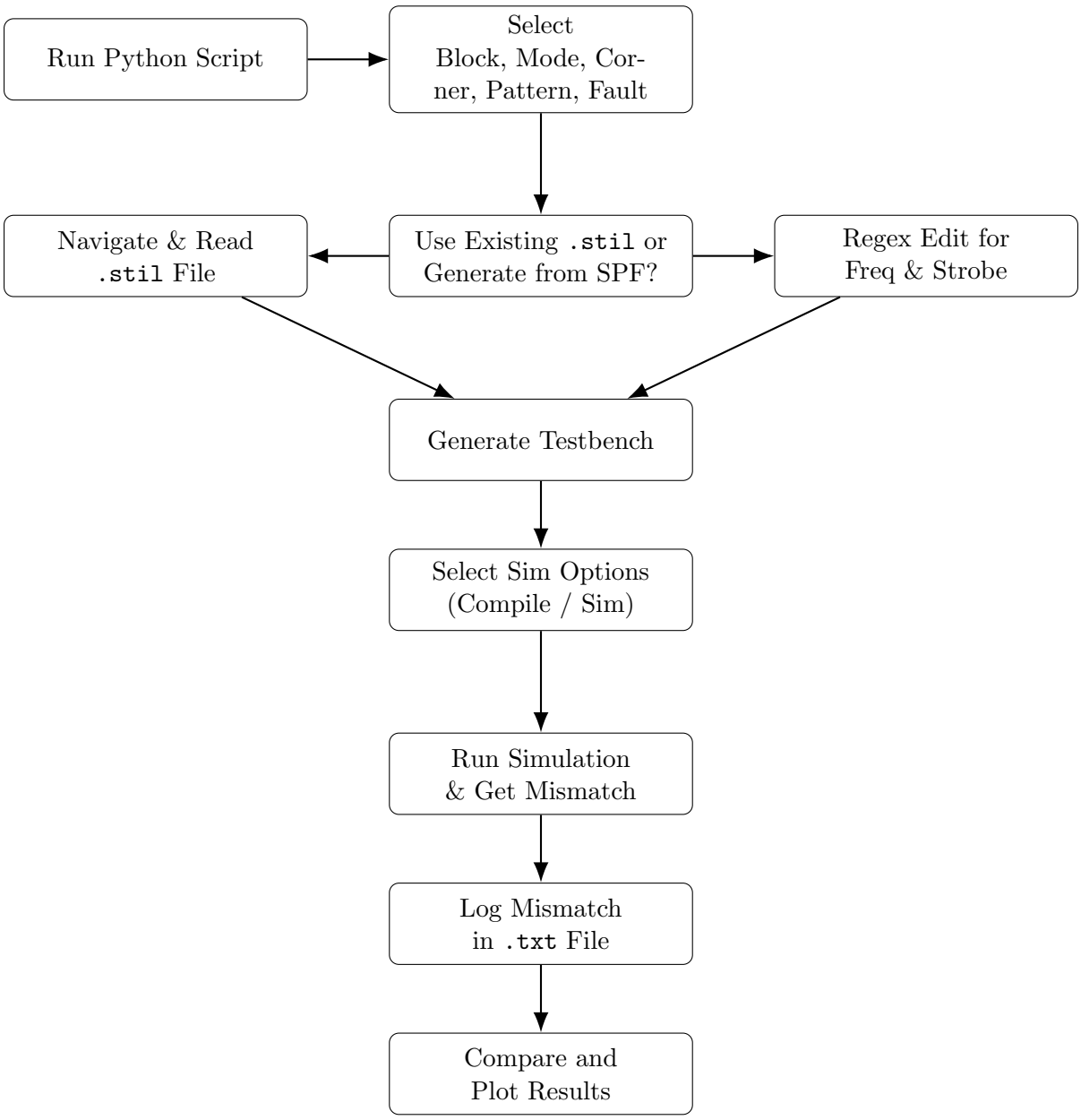
- Run the Python script and follow prompts.
- Select: Block, Mode, Process Corner, Pattern Mode, Fault Type.
- Choose whether to generate a `.stil` file from SPF or use an existing one.
- Based on the selection, navigate to the correct directory and read the `.stil` file.
- Provide frequency and strobing values; the script uses regex to modify the content accordingly. Generate testbench from the updated `.stil` file.
- Proceed to: Block, Mode, Process Corner, Pattern Mode, Fault, Frequency Selection.

- Prompt whether Compilation or Simulation is needed and execute accordingly.
- Finally, extract mismatch count from log files, parse mismatch data and plot mismatch vs. strobe time graph.

Script 2: Focused Compile/Sim Automation

- Run the script and input: Block, Corner, Mode, Pattern Mode, Fault, Frequency.
- Prompt for Compilation: If yes, proceed with compile step then simulate.
- After simulation, parse mismatch data from `.txt` file.
- Automatically compare and visualize mismatch trend.

Automation Flowchart



Conclusion

This Python-based automation flow significantly simplifies ATPG chain test execution. It abstracts repetitive tasks and ensures a consistent simulation pipeline with built-in logging and plotting of mismatch behavior. The two scripts support both exploratory sweeps and focused reruns, providing flexibility to test engineers.

3.7 Task 7: Capturing PI, PO Maximum Frequency

Objective

In ATPG chain tests, maximum frequency was identified from SI to SO during scan shifting. However, to analyze functional behavior when `SE = 0`, primary input (PI) and primary output (PO) faults were introduced to determine the maximum operating frequency of the functional logic.

Approaches Evaluated

- **Approach 1:** Introduce only PI and PO faults using `add_fault` command.
- **Approach 2:** Disable scan shifting during simulation and observe PI-PO behavior under functional test mode.

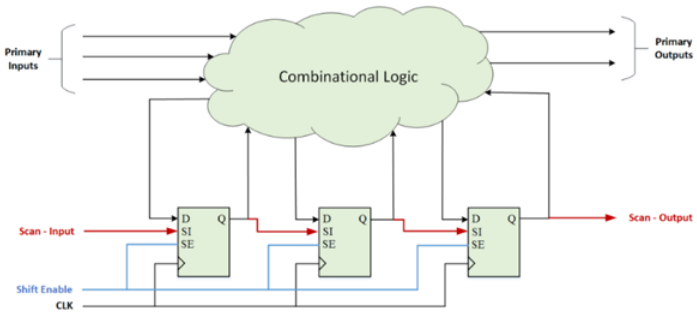


Figure 6: PI, PO Paths

Conclusion

- PI and PO paths can be evaluated effectively using dummy cycles inserted before and after functional patterns.
- These dummy cycles help meet setup/hold requirements during PI-PO switching in at-speed tests.

PI, PO Hierarchies Explored

To trace the exact PI and PO instances in the netlist, multiple naming hierarchies were evaluated:

1. `A_CORE/<signal>`

2. `<signal>` (direct signal name)
3. `A_CORE/MEMBLOCK1_ST_inst/MEMBLOCK1_SPREG_inst/<signal>`

Final Inference

- Top-level PI and PO are found to lie on critical paths.
- Ensuring correct strobing and functional sampling for these paths is essential to prevent mismatches during at-speed simulations.

Implemented Solution

Dummy cycles are introduced before and after the functional pattern application to enable accurate sampling of PI and PO values at high speed. This allows the system to meet setup and hold requirements for functional path testing at maximum frequencies.

3.8 Task 8: Pipelining Flops Insertion at Block Level

Objective

To enhance the shift frequency performance of `SPHD` and `SPREG` blocks by inserting pipelining flops at the block output in all scan modes.

DFT Commands Used

- `set_dft_configuration -pipeline_scan_data enable`
- `set_pipeline_scan_data_configuration -head_pipeline_clock CLK0 -tail_pipeline_clock CLK0 -head_pipeline_stages 0 -tail_pipeline_stages 1 -head_scan_flop true`

Challenges Encountered

The design consists of both compressed and non-compressed modes operating under multiple clocks (`CLK0`, `CLK1`, `CLK2`). While inserting tail pipeline stages using the above configurations, the following issues were initially faced:

- **Clock Visibility:** After inserting pipeline stages, scan chain clocks were not visible. This was resolved by proper clock configuration for all scan chains in multi-clock design.
- **Lockup Latches:** Verification was required to ensure lockup latches were correctly inserted and reported. This was later resolved.
- **Pipeline Flops:** Confirmation was needed to check if pipeline flops were properly added in both compressed and non-compressed modes. This was verified and cleared.

- **Compressor/Decompressor Logic:** A design challenge remains regarding modifying XOR/decoder logic to allow same clock domains to process certain scan chains, potentially eliminating lockup latches and avoiding clock mixing issues. This is under exploration.

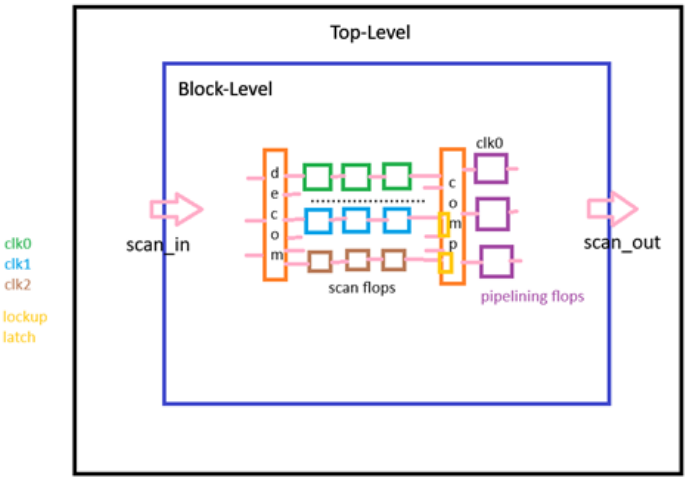


Figure 7: Pipeline Flops Structure

Observations

- The pipeline registers are named in the form:
  - `SNPS_PipeHead_SI_<pin_name>_stage` (for head pipeline)
  - `SNPS_PipeTail_SO_<pin_name>_stage` (for tail pipeline)
- If a clock is not defined, a default clock named `SNPS_PipeClk` is automatically created.

Recommended Flow for Pipelining

1. Identify critical paths with tight timing margins.
2. Insert flip-flops at the head and tail of scan chains.
3. Ensure pipeline registers are clocked and reset properly.
4. Modify data paths to accommodate extra stages.
5. Perform timing analysis to check setup and hold constraints.

`Input` → `Pipeline Register` → `Stage 1` → `Pipeline Register` → `Stage 2` → `Pipeline Register` → `Stage 3` → `Pipeline Register` → `Output`

Future Explorations

- Perform pipelining insertion at the top-level scan path.
- In codec logic, evaluate use of advanced tools such as:
  - **DFTMax**
  - **DFT Compression with Serializer**
  - **DFT Ultra Compression**
  - **DFT SEQ Compression**
- These advanced techniques may reduce or eliminate the need for lockup latches and improve compression efficiency.

3.9 Task 9: Mismatches in Serial Sims due to Dummy Cycles

Objective

To identify and resolve the root cause of mismatches on `DOUT[3]` in serial simulations at the top level during ATPG.

Observations

- All mismatches occur in scan cell 0 (last scan cell – 685th) of the `DOUT[3]` scan chain.
- These mismatches appear at the end of scan patterns for `DOUT[3]`, which is the longest scan chain and lacks shadow cells for `CLK1`.

Procedure

1. During the last shift stage, two dummy cycles were inserted:

   ```
   V{"CLK0"=0; "CLK1"=0; "TCK"=0;}
   ```

   These cycles are added where clocks do not toggle, but serve to meet the setup/hold requirements (minimum 60ns required between scan enable and clock) before entering capture mode.

2. Backtrace the flop connected to `DOUT[3]` in the schematic viewer at the mismatch cycles.
3. Compare ATPG pattern cycles with simulation waveform for both:
   - The scan flop at the end of the chain
   - The output `DOUT[3]`

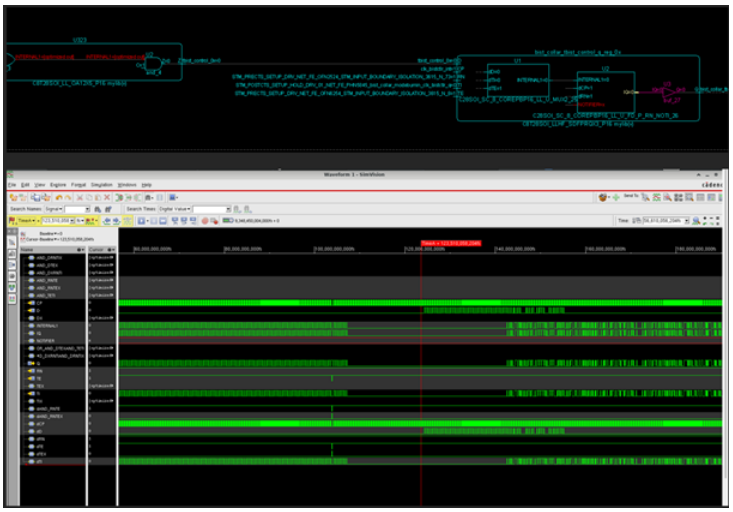


Figure 8: Simulation Result

Results

- During shift mode, the Q-output of the flop and `DOUT[3]` match with the expected value.
- However, during dummy cycles, the Q-output does not update (remains static), which leads to mismatches.
- There is no issue observed between the path from the flop to `DOUT[3]`.

Conclusion & Solution

The mismatch occurs because dummy cycles interfere with expected transitions, especially without toggling clocks. The effective solution is to add a mask for `DOUT[3]` in the SPF, ensuring that these cycles are ignored during simulation checks.

3.10 Task 10: BIST DFT/ATPG Design Issues

Objective

To analyze and resolve issues related to **DFT/ATPG** in BIST-enabled memory designs, specifically focusing on clocking glitches, signal misuse, and their impact on fault coverage and silicon behavior.

Key Issues Identified

- **Issue 1:** Glitch on memory clock in ATPG due to dual port clocks (e.g., in `PROLIBP18 V3.0 CUT 28{31`).
- **Issue 2:** `mem_atpg_mode` is not present in BIST. Although it should ideally be set to 1 during ATPG, constraining it in simulation causes a coverage drop (e.g., in `PROLIBP18 V5.0 RF2 Memory`).

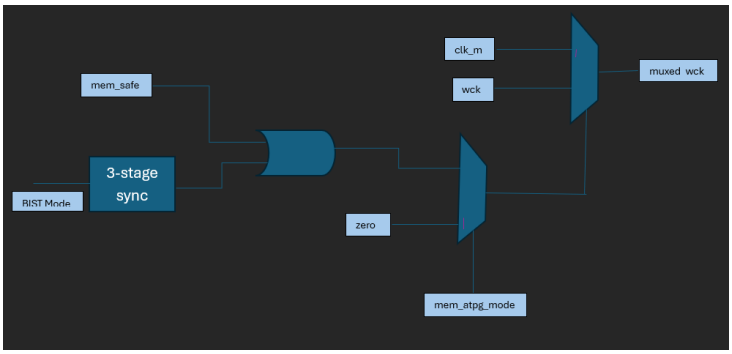


Figure 9: Modified Structure

Clocking Path and Logic Explanation

- In the design, both `mem_safe` and `BIST_Mode` signals are passed through a 3-stage synchronizer and OR'ed together.
- This OR'ed output becomes the select line of a 2:1 MUX controlled by `mem_atpg_mode`.
- The MUX controls the memory clocks — `clk_m` (read clock) and `wck` (write clock), producing a final `muxed_wck`.

Behavior of MUX depending on `mem_atpg_mode`:

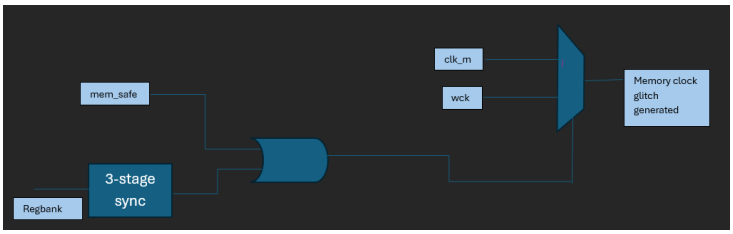
- `mem_atpg_mode = 1`: Output is directly `wck` (synchronizer bypassed).
- `mem_atpg_mode = 0`: Output is selected based on synchronized value of `mem_safe` or `BIST_Mode`.

Problem

- Using a single signal (`mem_atpg_mode`) for both bypassing synchronization and controlling test clock paths introduces conflicts.
- When constrained to 1 in ATPG (as required), it bypasses synchronizer logic and causes unexpected behavior.
- Ideally, separate control signals should be used for synchronization bypass and clock gating.

Glitch Mechanism in V3.0

- `wck` becomes transparent due to mismatched domains when `clk_m` (read clock) is used inside the synchronizer but control shifts to `wck`.
- The output of the OR gate becomes domain-dependent and leads to glitch when transitioning.
- **Workaround:** Use synchronizer scan cell values as either `000` or `111` (i.e., stable values) to hold state during capture for at least 3 cycles.



Design Guidelines and Learnings

- Prefer single-port memories over dual-port unless absolutely required.
- Avoid using the same signal (e.g., `mem_atpg_mode`) across multiple functional blocks with different intent.
- MUXing logic like in previous revisions should be reintroduced to eliminate glitch possibility.
- 3-stage synchronizers are functioning correctly; glitches are introduced only during domain crossings.
- Use `add_cell_constraints` to control values during shift/capture cycles.

DFT/ATPG Problem in Address Path

- `ADD[0]` and `TADD[0]` are inputs to a 2:1 MUX controlled by `TBIST (TEA)`.
- `ADD[0]` comes from top-level for direct memory test (DMT) but limited by tester delays (max 10 MHz).
- `TADD[0]` is used for high-speed BIST path.
- However, setting `mem_atpg_mode = 1` in ATPG leads to `TEA = 1`, which forces only `TADD[0]` to be used.
- This constrains the input path and results in fault coverage drop, which is not desirable.

Conclusion

The issues are identified and communicated to the BIST team for correction. Key recommendations:

- Separate control signals should be introduced for synchronizer bypass and clock muxing.
- Avoid sharing `mem_atpg_mode` for different functional paths.
- Always validate designs in both simulation and silicon, especially for dual-clock/dual-port memory configurations.

3.11 Task 11: Verigy 93K Conversion from ATPG

Objective

To convert ATPG-generated `.stil` patterns into ATE-compatible format using Verigy 93K tester and validate simulations in both compressed and non-compressed modes for `SPHD` and `SPREG` blocks across different modes and corners.

Motivation

The conversion and validation process is required only for the **MEM block** due to its differing architecture. Other blocks such as **LOGIC** and **REGBANK** remain unaffected and are excluded from this flow.

Flow Overview

- **Start:** Use ATPG patterns (.stil) for SPHD and SPREG.
- **Convert:** STIL → ATE → VT using Verigy 93K environment.
- **Simulate:** Use only **serial simulations** generated by the conversion for compressed and non-compressed configurations.
- **Modes tested:**
  - SPHD_Compressed
  - SPHD_Non-Compressed
  - SPREG_Compressed
  - SPREG_Non-Compressed
- **CTRL Pin Settings:**
  - FF corner: CTRL[1] = 0, CTRL[4] = 0 (HS0LS0)
  - SS corner: CTRL[1] = 1, CTRL[4] = 1 (HS1LS1)
- **Execute:**
  1. Launch conversion: `sh clean.sh`, place `.stil` in `vcd_file/`
  2. Execute tester flow: `sh cmd_to_tester`
  3. Simulate the testbenches across 6 modes:
     - `LOGIC_MEM_sa`
     - `LOGIC_MEM_lowspeed`
     - `LOGIC_MEM_atspeed`
     - `LOGIC_sa`
     - `REGBANK_sa`
     - `ONLY_MEM_sa`

Testbench Paths and Case Example

For **FF1** (FF corner, SPHD, Non-compressed):

- All 6 pattern sets were used: LOGIC_MEM_sa, LOGIC_MEM_lowspeed, LOGIC_MEM_atspeed, LOGIC_sa, REGBANK_sa, ONLY_MEM_sa
- Pattern categories:
  - `1_high_trans`
  - `1_high_trans_comp`
  - `2_stuck_at_low_trans`
  - `2_stuck_at_low_trans_comp`
  - `3_Debug_patterns`

Simulation Summary

- **Simulation Modes:**
  - Compressed and Non-Compressed
  - Operating Corners: FF and SS
  - Blocks: SPHD and SPREG
- **Result: 0 mismatches** observed across all simulations.
- **Validation:** Confirms correctness of conversion and testbench generation.

Conclusion

The Verigy 93K conversion process successfully validated all pattern modes for SPHD and SPREG blocks. CTRL[1] and CTRL[4] combinations were handled appropriately for both FF and SS corners, and the correctness of the conversion was confirmed by simulation results with no mismatches.

3.12 Task 12: AI in VLSI (Future Explorations)

Objective

Explore the role of Artificial Intelligence (AI) in advancing VLSI design and automation processes. Investigate how AI techniques can optimize design workflows, improve verification, and accelerate time-to-silicon.

AI Fundamentals in VLSI

Understanding key concepts forming the foundation of AI-driven solutions in chip design:

- **AI:** Artificial Intelligence – Mimics human-like decision-making.
- **ML:** Machine Learning – Learns from data patterns and improves over time.

- **DL:** Deep Learning – Uses neural networks with multiple layers.
- **GenAI & LLM:** Generative AI and Large Language Models for code/document generation, synthesis aid.
- **Neural Networks:** ANN, CNN, GNN, RNN and their relevance in VLSI optimization.

Benefits of AI in VLSI

- **Increased Design Efficiency:** Automates repetitive or manual-intensive steps.
- **Improved Accuracy:** Enables early defect detection and validation.
- **Cost Reduction:** Reduces resource overhead and accelerates development.
- **Scalability:** Handles growing complexity in system-on-chip (SoC) designs.

Challenges in AI-VLSI Integration

- **Data Requirements:** Requires large, clean, and representative datasets.
- **Model Interpretability:** Difficulty in understanding the decision logic of deep models.
- **Integration Complexity:** Compatibility with existing EDA workflows is non-trivial.
- **Computational Cost:** Training and inference require significant hardware resources.
- **Security Concerns:** Trust and vulnerability in AI-generated designs.

Applications of AI in VLSI

- **Automated PnR:** Intelligent placement and routing strategies.
- **Timing and Power Optimization:** Predictive timing closure and power estimation.
- **Yield Prediction and Process Tuning:** Detect potential failures and improve manufacturability.
- **Bug Detection and Fault Prediction:** Learn from historical failures and improve test coverage.
- **Dynamic DVFS:** AI-assisted voltage/frequency scaling based on workload.

Key Areas for AI Usage

- **Physical Design:** Placement, routing, timing closure, and sign-off.
- **Analog Design:** Topology suggestion, sizing, and biasing automation.
- **Manufacturing:** Lithography-aware optimization, defect analysis.
- **Verification:** Regression automation, test generation, corner case identification.
- **FPGA Emulation and Testing:** Test planning and pattern optimization.

Future Exploration Areas

- Analog and Digital Design Optimization using Reinforcement Learning and Graph Neural Networks.
- AI in System-level multi-physics and mixed-signal simulations.
- AI-assisted Technology and Standard Cell Library Development.
- Documentation generation and Design Consultation using LLMs.
- Chat-based design debugging and exploration tools.

Conclusion

AI is increasingly becoming an integral part of the VLSI ecosystem. With the rising complexity of designs and shrinking technology nodes, AI-driven design automation can enable scalable, accurate, and faster design methodologies—bridging the gap between time-to-market and performance optimization.

4 Conclusion

This internship has provided me with rich exposure to the world of front-end DFT engineering. I had the opportunity to work on cutting-edge nodes and collaborate with industry professionals. Each task has helped me understand practical workflows, tools, and design challenges that are critical in VLSI testing and automation.

Acknowledgements

I would like to express my sincere gratitude to:

- My mentor at STMicroelectronics, Mr. Sachin Bakshi, for his guidance and continuous support.
- The entire SQH Dig FEM TR&D (TestChip) team for creating a collaborative and learning-focused environment.
- My faculty mentor Mr. Siva Rama Krishna Vanjari and instructors at IIT Hyderabad for the academic foundation that enabled me to take up this internship.
- My family and friends for their encouragement throughout this journey.