EE3405 Semester Internship Report (STMicroelectronics)

Rayi Giri Varshini – EE22BTECH11215

Task-1: To modify the RTL of Register Bank to have a common capture flop without change in update flops.

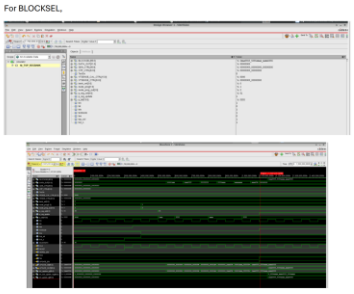
In REGBANK, each register when addressed via the testbench, goes through capture and update flops.

Modes = Serial/Jtag (00), Parallel (01).

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 first capture happens then update using USER_FF for each. So separate modules of capture & update, where capture is instantiated one time and update is instantiated 5 times for 5 registers. This reduces the area (10 USER_FF modules reduce to 6 USER_FF).
- Instead of instantiating two clock gating cells in USER_FF, when only one CGT is used in USER_FF and adding the other CGT in USER_CAPTURE & USER_UPDATE will get rid of multiple instantiations and reduce the area effectively.
- Tools used: xceliumagile for simulation, genus for synthesis.

Simulation Results:



Synthesis Results:

	REGBANK given	REGBANK common capture	REGBANK single CGT cell in USER_FF
Area	29884.983481 µm²	19833.005480 µm²	15153.466126 µm²
Power	8.1102e-02 mW	3.8494e-02 mW	2.3540e-02 mW
Timing (slack)	94.2 ns	92.78 ns	93.45 ns

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

Requirement: In REGBANK as we have added a common capture for all registers if any error has occurred it would propagate leading to the errors in the other registers also. So, we expect the ECC to correct single bit errors, correct double bit errors, detect triple bit errors.

Errors Correction Codes are techniques used to detect and correct errors in digital data. They are essential for ensuring data integrity in communication systems, storage devices and memory systems.

- Standard Hamming Code: It is a popular ECC that can detect and correct single bit errors. It uses parity/redundant bits and XOR logic to detect and correct single bit error/data flip, enabling forward error correction, where errors are automatically corrected when read back.
- Extended Hamming Code: Additional parity bit is added to standard code to detect double bit error resulting in single error correction/double error detection (SECDED).
- Bose-Chaudhari-Hocquenghem (BCH) Code: A class of cyclic advanced ECCs capable of correcting multiple errors within a block of data. Using reduction modulo, encoding, decoding, Galois field for single error correction/double error correction/triple error detection (SECDECTED).

Task – 3: ATPG Pattern Generation and Simulation for MEMBLOCK1_SPHD and SPREG in P18_1

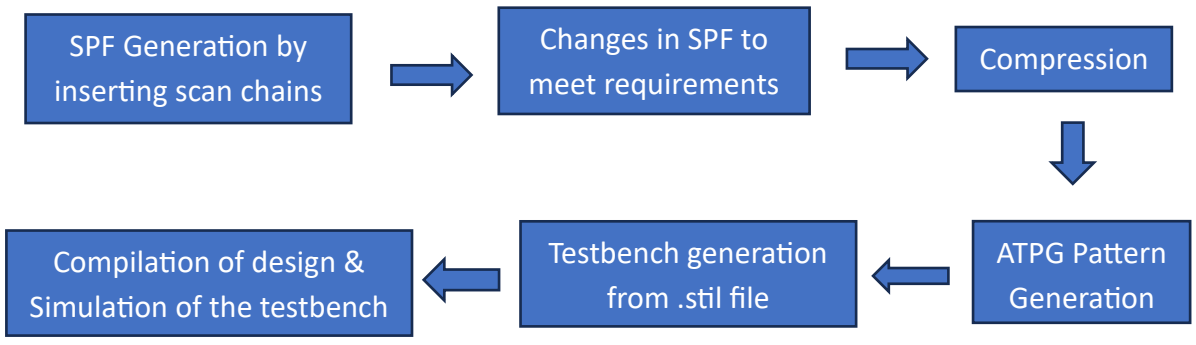
Top Level ATPG Pattern Generation and Simulation for Zero Delay and SDF (FF & SS) for MEMBLOCK1_SPHD_LOLEAK and MEMBLOCK1_SPREG_LOLEAK in MEMBLOCK1_ST for different modes, faults and pattern modes. We already have block level ATPG simulation results.

Tools used: tetramax for ATPG pattern generation, xceliumagile for Simulation of testbench

Modes: LOGIC, LOGIC_MEM, LOGIC_COMP, LOGIC_MEM_COMP, ONLY_MEM, REGBANK

Process Corners: Zero Delay, SDF (FF & SS), Pattern Modes: Parallel (par), Serial (ser)

Fault types: Stuck-at, Low Speed, At Speed, ATPG modes: Auto, Chain Test, Functional Test



Modifications in SPF for ATPG Simulations:

- Mismatches due to Analog Signals, manual modifications in test set-up, finding the optimal strobing values, etc.

Solutions Implemented:

1. Removed the analog signals MEASRA_MB1 & MEASRA_MB2
2. Incremental flow is introduced to reduce the test time
3. Added two test vectors with clock turned off (dummy cycles) to meet setup/hold requirements
4. Vectors defining resets were added in the test-set up in Macrodefs
5. In high-trans, pre-shift cycle test-set up and PLL programming is added

Task – 4: Prolib3D28 FDS ATPG SPHD_BB_COMP atspeed and lowspeed TF

Objectives:

- Issue: ATPG tests on Silicon has issues in low trans and high trans

- 2 failures in high trans and 9 failures in low trans

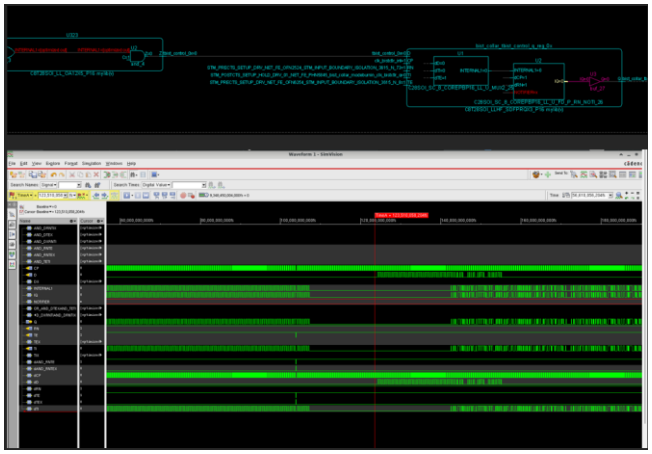
- Specific DOUT pins at clock cycles

- Root Cause Analysis:

- Backtrace & identify the root cause of the failing cases in both FF and SS simulations

- These failures aren't present in simulations, complicating debugging efforts.

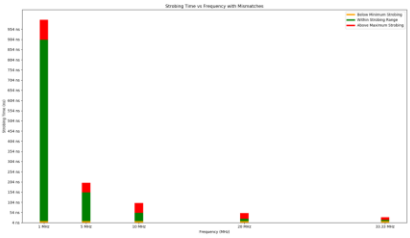
- Starting from mem_bist_wrapper, which is potentially assumed to be causing the issues, back tracing must be done.



Task – 5: ATPG Chain Test for maximum shift frequency

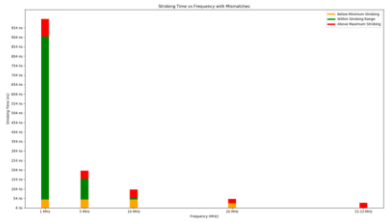
Goal: To find the maximum working frequency and strobing range for SPHD and SPREG blocks in the scan chain test with LOGIC_MEM_COMP, LOGIC_COMP for stuck at fault.

- Generate ATPG patterns for chain test then simulate for a given frequency, rise time, fall time, and strobing time.
- Compare and find the strobing range along with the maximum frequency of operation for the most optimum performance.



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

(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

(SS Corner)

Results:

1. Maximum Shift Frequency SPHD block - 9.52 MHz
2. Maximum Shift Frequency SPREG block – 12.5 MHz

Task – 6: Python Automation flow for Maximum Chain Test Frequency and Chain Test Simulation

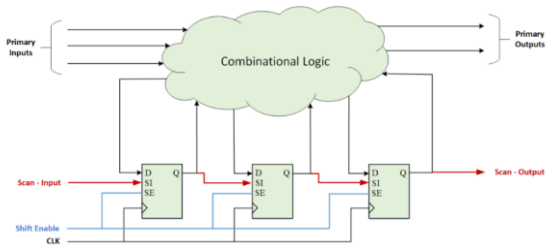
Script – 1: Run the Python Script -> Select block, mode, process corner, pattern mode, fault -> Ask if .stil file is to be generated from SPF or use the existing file -> Based on prompts navigate respective directory and read .stil file -> Select frequency & strobing, and makes changes through regex -> Create testbench from the modified .stil file -> Select block, process corner, mode, pattern mode, fault, frequency for sims -> Ask if compilation or simulation is required & proceed -> After simulation, enter the mismatches count in .txt file -> Compare mismatches count and plot ranges

Script – 2: Run the Python Code -> Select the block, process corner, mode, pattern mode, fault, frequency -> Asks if compilation is required or not, if yes proceeds with compiling -> Asks if

simulation is required or not, if yes proceeds with simulation -> After finishing the simulation, the mismatches count will be in .txt -> Compares the mismatches count and plot using the plot script

Task – 7: Capturing PI, PO maximum frequency

- Objective: In chain test, SI to SO frequency was found. Functional testing is done when SE=0 and PI, PO faults are added.
- Approach-1: Add only PI, PO faults using add_fault.
- Approach-2: Disable the shifting
- Conclusion: PI PO can be handled with dummy cycles.



For SPREG,

Frequency (Mhz)	Time period, Tr, Tf (ns)	Strobing (ns)	FF Mismatches	SS Mismatches (1)	SS Mismatches (2)	SS Mismatches (3)
1	1000, 900, 930	500	0	0	0	0
2	500, 400, 430	400	0	0	0	0
4	250, 180, 210	120	0	0	0	0
4.16	240, 180, 210	120	0	0	0	0
4.34	230, 170, 200	130	0	0	0	0
4.44	225, 175, 205	135	0	3484	5226	0
4.54	220, 170, 200	150	0	3571	5226	3571
5	200, 150, 180	145	0	5995	6097	6968
10	100, 50, 80	45	0	9057	15729	9057
20	50, 20, 40	20	0	9376	13949	9376
33.33	30, 15, 25	15	0	48272	48292	48272

Maximum frequencies: 4.34 MHz, 4.34 MHz, 4.44 MHz

For 3 possible hierarchies,

1. A_CORE/<signal>

2. <signal>

3. A_CORE/MEMBLOCK1_ST_inst/MEMBLOCK1_SPREG_inst/<signal>

Conclusion: Top Level PI, PO are critical paths

Solution: Dummy Cycles in PI, PO

Task – 8: Pipelining Flops Insertion at Block Level

Objective: To insert pipelining flops in all modes at the output to enhance the shift frequency of SPHD and SPREG blocks.

Commands used:

set_dft_configuration -pipeline_scan_data enable

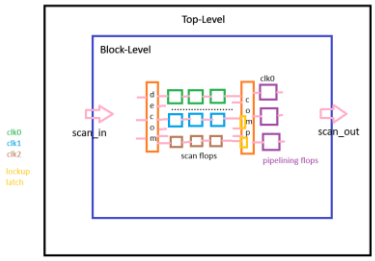
set_pipeline_scan_data_configuration -head_pipeline_clock CLK0 -tail_pipeline_clock CLK0 -head_pipeline_stages 0 -tail_pipeline_stages 1 -head_scan_flop true

Problems encountered: For DFT Insertion using TetraMAX, the design incorporates both compressed and non-compressed modes with multiple clocks (CLK0, CLK1, CLK2)

- Lockup Latches
- Compressor and Decompressor Logic

To add pipelining stages:

- Identify the critical paths where timing margins are tight.
- Add flipflops at the input (head) and output (tail) of design to create pipelining stages also ensure that pipeline registers are clocked and reset correctly.
- Adjust the data flow to account additional pipelining stages so that data correctly passes from one stage to the next.
- Timing Analysis do to verify if it meets required timing constraints, also should check pipeline stages do not add hold time and set up time violations.
- Input -> pipeline register -> stage 1 -> pipeline register -> stage 2 -> pipeline register -> stage 3 -> pipeline register -> output




```
Architecting Pipeline Structures
   Number of Head Pipeline Stages = 0
   Number of Tail Pipeline Stages = 1
Architecting scan compression mode LOGIC_MEM_COMP with base mode LOGIC_MEM
Architecting Load Decompressor (version 5.8)
   Number of inputs/chains/internal modes = 32/165/4
Architecting Pipelined Unload compressor (version 5.8)
   Number of outputs/chains = 32/165
   Information: Compressor will have 100% x-tolerance
Architecting scan compression mode LOGIC_COMP with base mode LOGIC
Architecting Load Decompressor (version 5.8)
   Number of inputs/chains/internal modes = 32/412/4
Architecting Pipelined Unload compressor (version 5.8)
   Number of outputs/chains = 32/412
   Information: Compressor will have 100% x-tolerance
```

Future Explorations:

- Top-Level Pipelining insertion
- In Codec logic, along with DFTMax, explore the other advanced tools like DFT Compression with serializer, DFT Ultra Compression, DFT SEQ Compression, which when used might not require lockup latches.

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

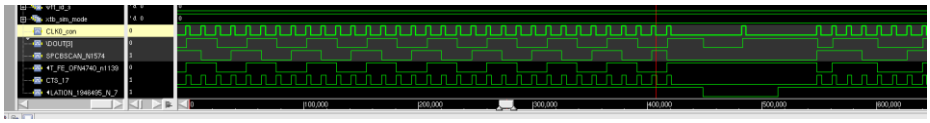
Objective: To find the root cause of mismatch of the pin DOUT[3] in serial simulation at top level in ATPG.

Observations:

- All the mismatches are with scan cell 0 (the last scan - 685th) of the DOUT[3] scan chain.
- The mismatches are at the end of the patterns in scan chain of DOUT[3], which is the longest chain with no shadow cells of CLK1.

At the last shift, adding two dummy cycles V{"CLK0"=0; "CLK1"=0; "TCK"=0;}, where the clock doesn't toggle but to meet the setup/hold violations to enter capture mode (min. 60ns required between scan enable and clk).

Solution to this is adding mask in the SPF.



Task – 10: BIST DFT/ATPG Design Issues

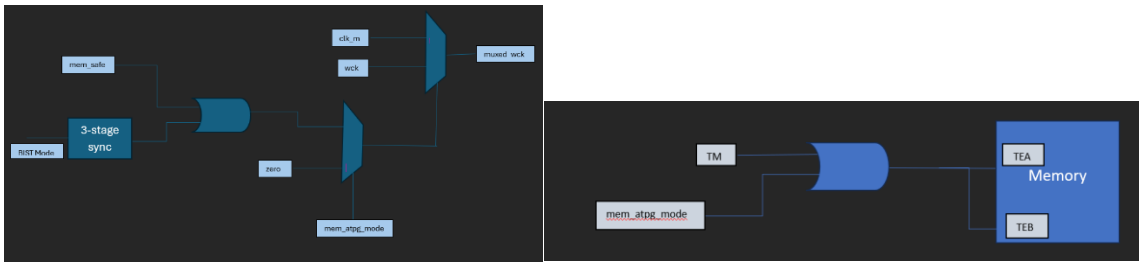
DFT/ATPG Issues and solutions:

Issue-1: Glitch on Memory Clock in ATPG, mem_atpg_mode pin, which is used to Bypass Synchronizer in ATPG, is not preset in BIST [In PROLIBP18 V3.0 CUT 28-31]

Issue-2: Recommended mem_atpg_mode should be 1 in ATPG but when mem_atpg_mode is constrained it leads to coverage drop. [In PROLIBP18 V5.0 RF2 Memory]

In PROLIBP18 V3.0, there is glitch generated due to dual port clocks (CUT 28-31)

Ideal design for no glitches.



The problem in the PROLIB V3.0 version, which generates the glitch is, wck is made transparent.

Task – 11: Verigy 93K Conversion from ATPG

In SPHD and SPREG, we should launch simulations by converting it from ATPG environment to ATE then to VT. There we get all serial simulations, we should run for SPHD compressed, SPHD non-compressed, SPREG compressed, SPREG non-compressed for CTRL[0] and CTRL[4] as 00 (FF corner).

Running the ATPG -> ATE -> VT

- Convert STIL ATPG patterns generated from DFT into ATE format compatible with Verigy 93K.
- Generate the testbench for comprehensive testing of the device for Compression and Non-Compression modes.
- Tabulate the simulation results after simulation using python script.
- Two operating conditions simulated: HS0LS0: CTRL[1] = 0, CTRL[4] = 0 (FF corner), HS1LS1: CTRL[1] = 1, CTRL[4] = 1 (SS corner)

Result: After the conversion to verigy93k ATE and simulating all the patterns in compressed and non-compression for FF and SS for SPHD and SPREG for the different modes we get 0 mismatches indicating the correctness of the conversion.

Task – 12: AI in VLSI (Future Explorations)

Mimics Human Intelligence of learning, reasoning & predicting. AI in CAD tools for faster convergence & Optimization.

- AI Fundamentals in VLSI – AI, ML, DL, GenAI, LLM, NN (ANN, CNN, GNN, RNN)
- Benefits of AI in VLSI – Increased design efficiency & reduced manual effort, Improved accuracy & early defect detection, Cost reduction & enhanced performance, Scalability with growing design complexity.
- Challenges – Large & high-quality data requirement, Model interpretability & integration complexity, Computational resource demands, Security & trustworthiness of AI-driven designs.
- AI Applications in VLSI – Automated PnR, Timing & Power Optimization, Yield Prediction & Process Optimization, Bug Detection & fault prediction, Dynamic Voltage/Frequency Scaling.
- Key AI Usage areas - Physical design (placement, routing, timing closure, sign-off), Analog design (transistor sizing, biasing, topology), Manufacturing process optimization, Design verification, FPGA emulation, testing

Exploration Areas – Analog & digital design optimization, System-level multi-physics simulation, Technology & library development, Documentation & AI-assisted design consultation.