2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 – 5, 2026

Optimizing Functional Fault Grading Flow for Memory Designs

Euisang Yoon, Arun Gogineni, Saurabh Srivastava, Eunjong Oh, Seongwook Lee, Sungyun Yoo, Geonbeom Kwon, Yoseop Lee, Hyojin Choi

Samsung Electronics Co., Ltd,
SAMSUNG

Siemens EDA
SIEMENS

Agenda

Introduction

Background

Proposed Approach

Case Study and Results

Conclusion

Introduction

- Continuous process scaling leads to a higher frequency of **random defects**.
- **Testing peripheral circuits of memory devices** relies heavily on **functional test patterns**, requiring quantitative evaluation of their defect detection capability.
  - Scan-DFT is impractical for memory designs due to area, routing, power, and cost penalties.
- This work proposes **a suite of optimization techniques** to accelerate functional fault injection simulation.
  - The techniques are implemented in a commercial simulator and validated on a production-grade NAND Flash design.
  - Experimental results show up to 3.7x reduction in total simulation time.

Motivation & Our Approach

- Exhaustive fault simulation is infeasible for modern memory peripheral circuits, as the fault count exceeds 10M, leading to a computational explosion.
  - Conventional fault simulation scales poorly.
  - Parallel execution alone is insufficient for large memory designs.
- A scalable solution must reduce simulation workload before and during execution.
  - Our approach combines static filtering, clustering, design pruning, stimulus grading, and dynamic optimization to make function fault grading practical.

Functional Fault Grading Workflow

Functional Fault Grading Workflow – Detailed-1

Fault-list Generation

Static Fault Optimization

Fault Collapsing

- Device Level Collapsing
- Functional Level Collapsing

Untestable Fault Filtering

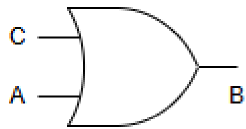
- Single-Stage
- Multi-Stage

Dynamic Fault Optimization

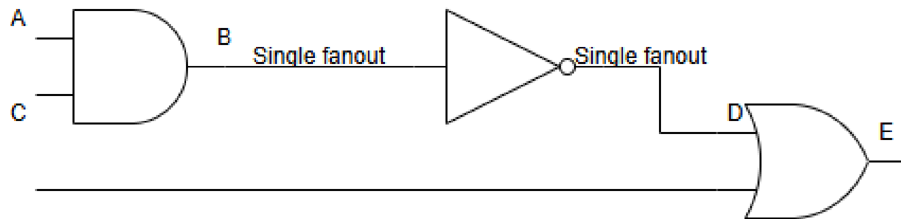
- Injection Based Optimization
- Propagation Based Optimization

Static Fault Optimization

- Fault Collapsing



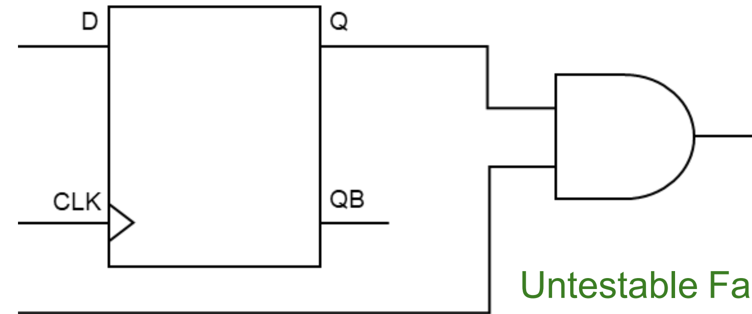
Equivalent Faults : A-SA1=C-SA1=B-SA1 Primary Fault : B-SA1



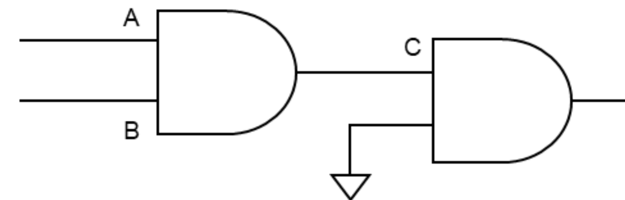
Equivalent Faults : A-SA0=B-SA0=D-SA1=E-SA1 Primary Fault : E-SA1

Group equivalent faults and define primary faults

- Untestable Fault Filtering



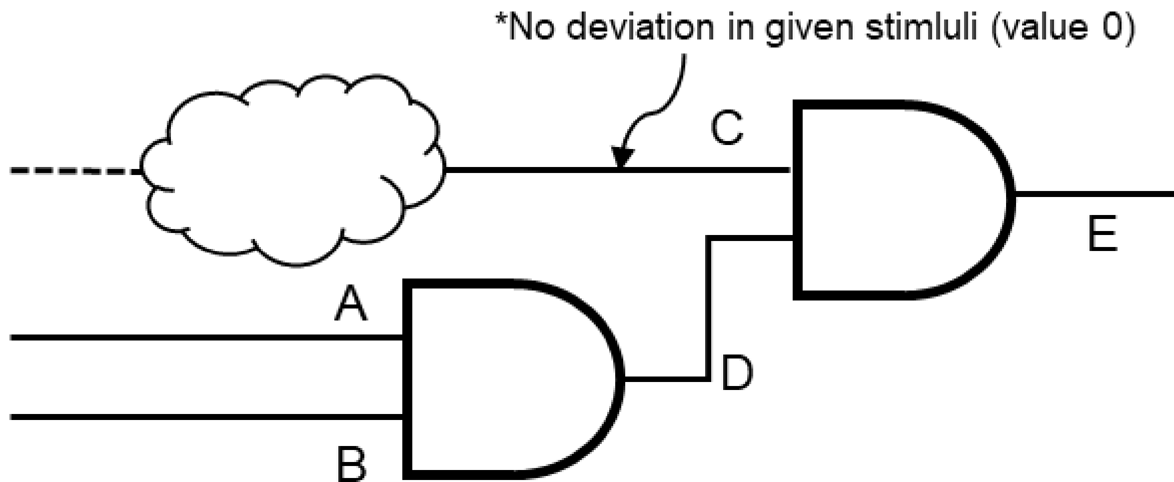
Untestable Faults : QB-SA0, QB-SA1



Untestable Faults : A-SA0, A-SA1, B-SA0, B-SA1, C-SA0, C-SA1

Define and remove untestable faults

Dynamic Fault Optimization



Not Injected Fault : C-SA0

Not Propagated Faults : A-SA0, A-SA1, B-SA0, B-SA1, D-SA0, D-SA1

Define Unobserved Faults on stimulus behavior – Not Injected, Not Propagated

Functional Fault Grading Workflow – Detailed-2

Fault Injection Simulation

Fault Clustering

- Injection Based Optimization

Design Pruning

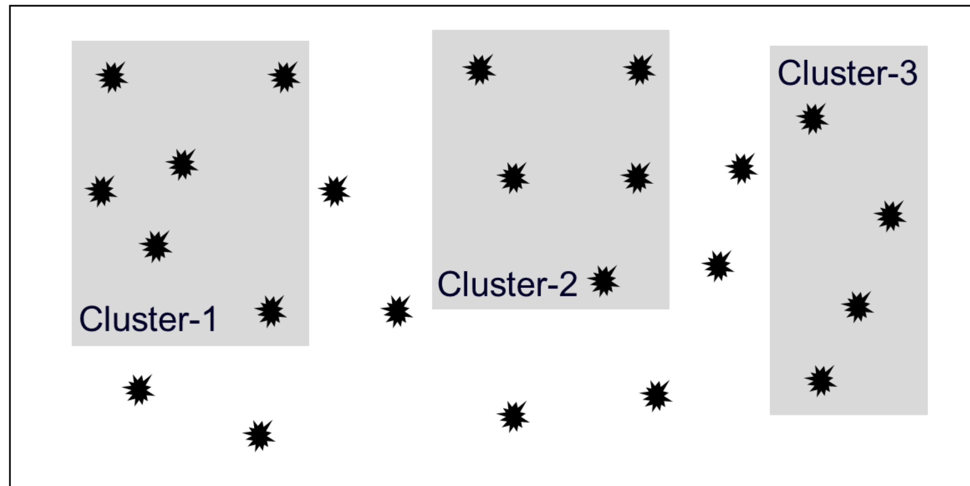
- Propagation Based Optimization

Dynamic Stimulus Grading

- Injection Based Ordering

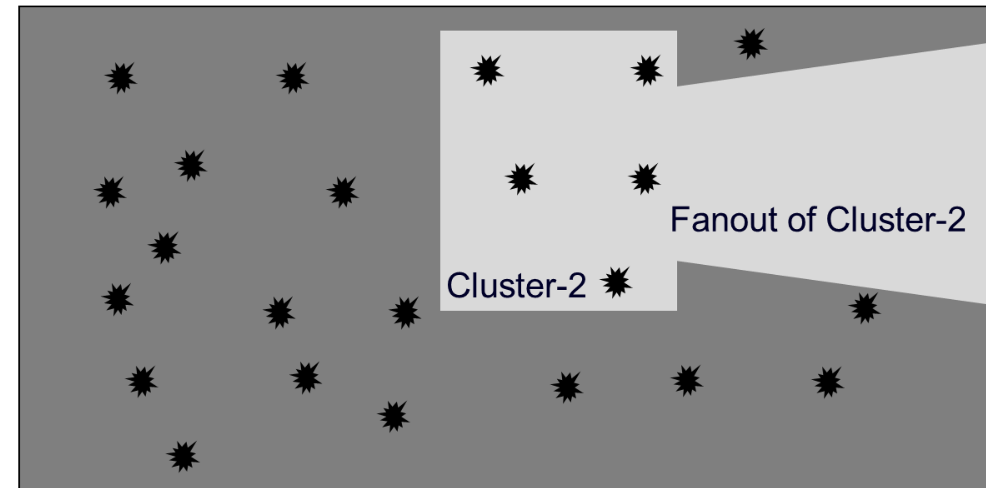
Fault Clustering and Design Pruning

- Fault Clustering



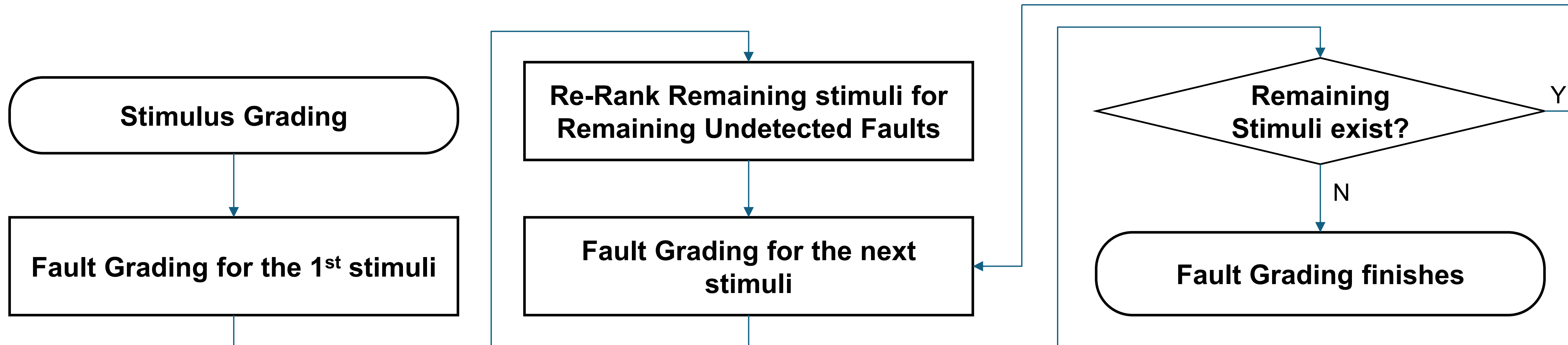
Group faults by injection point-based structure

- Design Pruning



Trim the design by logic cone analysis of the fault propagation path

Dynamic Stimulus Grading



- *Rank and order multiple test patterns based on the fault injectability*
- *Dynamically re-rank test patterns as accordance of updated remaining faults*

Dynamic Stimulus Grading

- example

Total faults : 15,000

Rank	Stimulus	Injectable Count
1	TB_C	12,000
2	TB_A	8,000
3	TB_E	7,000
4	TB_B	4,000
5	TB_D	3,000

Run for TB_C completed.

Detected faults : 4,000
Remaining faults : 11,000

Rank	Stimulus	Injectable Count
1	TB_E	5,000
2	TB_A	4,000
3	TB_B	3,000
4	TB_D	2,000

Run for TB_E Completed.

Detected faults : 8,000
Remaining faults : 7,000

Rank	Stimulus	Injectable Count
1	TB_D	1,500
2	TB_B	1,000
3	TB_A	500

Next run For TB_D

Functional Fault Grading Workflow – Detailed-3

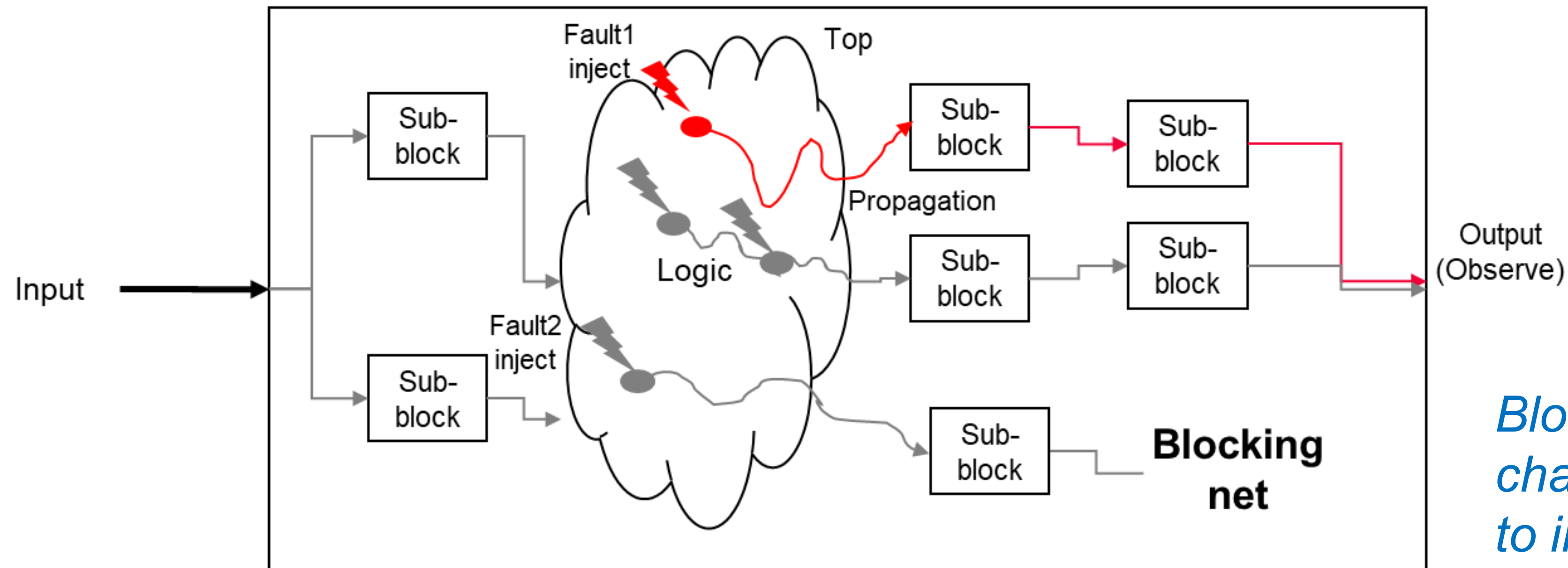
Analysis / Debug

Visualized Fault Debugger

- Multiple Faults Analysis
- Blocking Net Analysis
- Deviating Path Analysis

Visualized Fault Debug

- Aspect of fault propagation and blocking



Blocking net analysis can help changing the design to increase coverage

Visualized Fault Debug

File Edit View Tools Window Help   Search

Hex · ALL · AnyEdge · 68001 · 1ps

Design - Specification_Archit

.GenericToTransport.urep · DU

Instance	Design Unit
urps0	Specificati..
urpc0	Specificati..
urep	Specificati..
urps	Specificati..
urhc_0	Specificati..
urpc	Specificati..
uwec3...	Specificati..
ls	Specificati..
lp3	Specificati..
lp1	Specificati..
ls_0	Specificati..
lp2	Specificati..
la	Specificati..
ud7	Specificati..

Specification_Architecture_Structure_commons.v - Specification_Architecture_Structure.initiatorSocket0

```
7265          input   Sys_Clk_EnS     ;
7266          input   Sys_Clk_RetRstN ;
7267          input   Sys_Clk_RstN    ;
7268          input   Sys_Clk_Tm      ;
7269          output  Sys_Pwr_Idle    ;
7270          output  Sys_Pwr_WakeUp  ;
7271          input   WordErr         ;
7272          input   WordVld         ;
7273          reg   Clr          ;
7274          wire  HdrParErr ;
7275          wire  ParSet    ;
7276          wire  PldParErr ;
7277          assign PldParErr = PldVld & PldErr;
7278          assign HdrParErr = HdrVld & HdrErr;
7279          assign ParSet = PldParErr | HdrParErr;
7280          always @( posedge Sys_Clk or negedge Sys_Clk_RstN )
7281                  if ( ! Sys_Clk_RstN )
7282                          Clr <= #0.001 ( 1'b0 );
7283                  else    Clr <= #0.001 ( ResErr Clr );
```

Logic Cone - 0

Value: · Hex

Extend Logic Level: 1 2 3 · Point1: Clr · Point2:

urep

ursrsrg

RTL

0 O

0->1/0 Reset · 0->1/0 Reset · C · RegIn 0

0 Set

Specification_Architecture_Structure_z_T_C_S_C_L_R_Rsr_Srg_1

Specification_Architecture_Structure_z_H_R_N_T_U_R_Rep_a0a46896

MINIMAP

Fault Debug

Fault List · Alarm List · Observation Point · Hotspot Blocking Nets

Fault Node	Fault ID	Fault Type	Resolution	Cla
...tion_Architecture_Structure.initiatorSocket0_I_main.MainIP.GenericToTransport.la.u_4490	1	SA0	Undetected_Unobserved	Sa
..._Architecture_Structure.initiatorSocket0_I_main.MainIP.GenericToTransport.la.ud.u_c39d	2	SA0	Detected_Observed	Mu
...n_Architecture_Structure.initiatorSocket0_I_main.DuplicateIP.GenericToTransport.urep.Clr	3	SA0	Undetected_Unobserved	Sa
Specification_Architecture_Structure.targetSocket0_R_Valid	4	SA0	Detected_Unobserved	Mu
Specification_Architecture_Structure.initiatorSocket0_main.uremovexz_Lock.Tmp	5	SA0	Not_Injected	Un
Specification_Architecture_Structure.initiatorSocket0_W_Data[42]	6	SA0	Undetected_Unobserved	Sa
Specification_Architecture_Structure.initiatorSocket1_Ar_Id[2]	7	SA0	Undetected_Unobserved	Sa
Specification_Architecture_Structure.initiatorSocket1_W_Data[38]	8	SA0	Undetected_Unobserved	Sa
Specification_Architecture_Structure.initiatorSocket2_Ar_Size[1]	9	SA0	Undetected_Unobserved	Sa

Blocking Nets · Deviating Ports

Blocking Net
Specification_Architecture_Structure.initiatorSocket0_I_main.DuplicateIP.GenericToTranspo

Transcript · Fault Debug

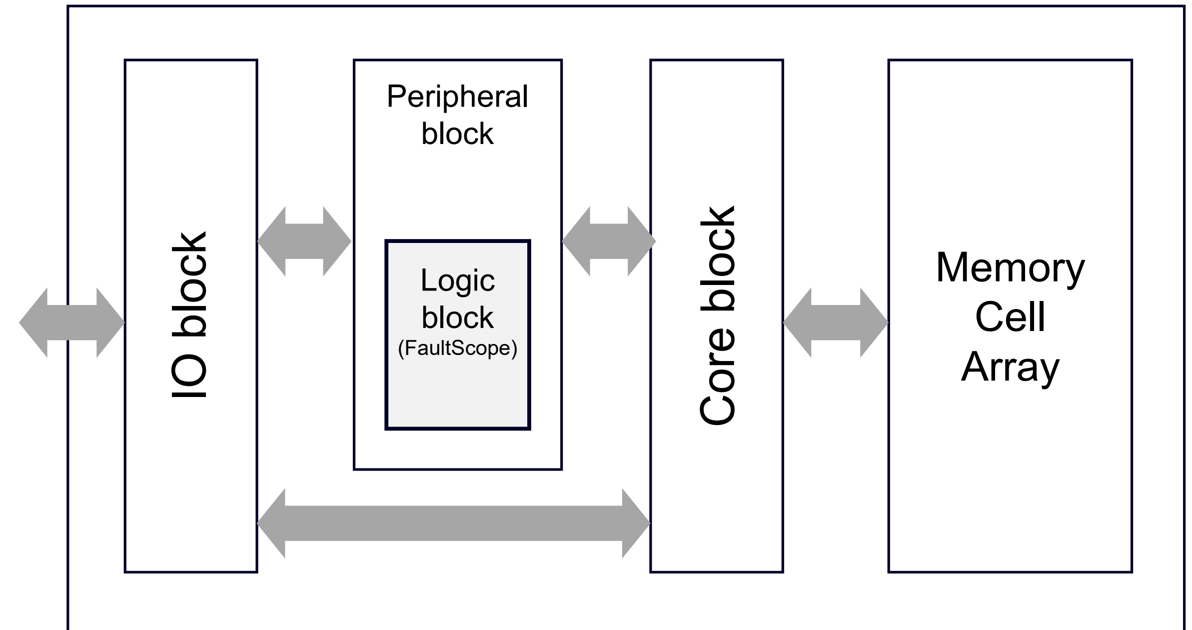
Fault resolutions

Logic cone analysis

Blocking nets / deviating ports

Case Study

- NAND Flash Memory
  - Peripheral logic instance
  - 7 functional testbenches written for test



Results

- Fault-list Optimization – Static and Dynamic

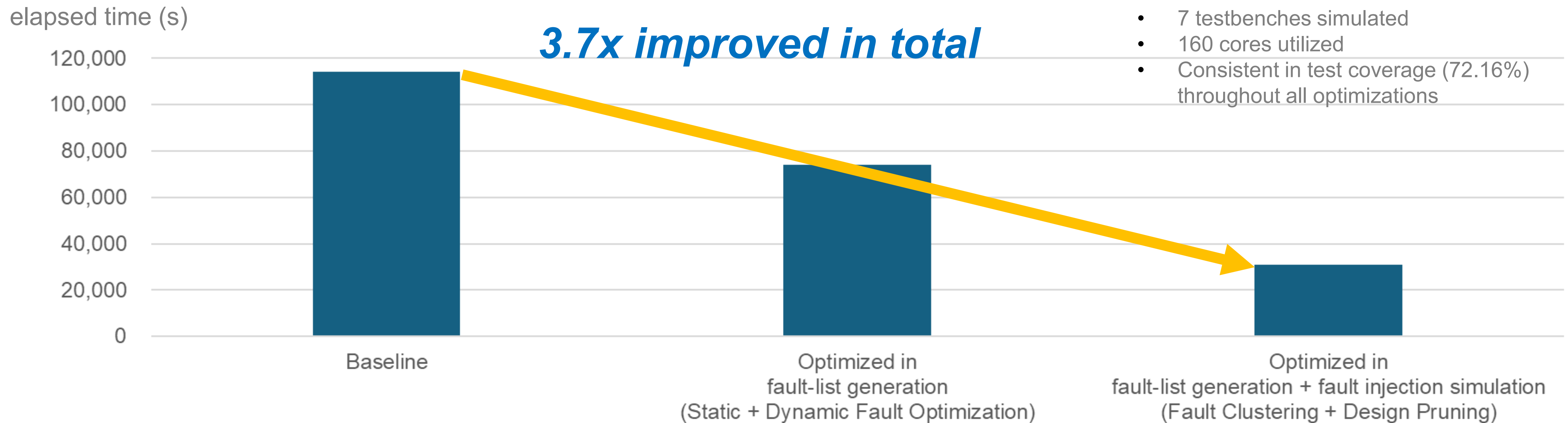
Fault population reduced to 40.4% of its original size

		Number of Faults						
Total Faults		5,302						
Statically Optimized Faults	Collapsed Faults	2,278						
	Untestable Faults	36						
Primary Faults (statically active)		**2988 (56.4%)**						
Dynamically Optimized Faults		TB1	TB2	TB3	TB4	TB5	TB6	TB7
	Not Injectable Faults	494	728	667	649	509	503	507
	Blocked Faults	246	278	272	287	260	244	262
Primary faults (statically and dynamically active)		**2,248 (42.4%)**	**1,982 (37.4%)**	**2,049 (38.6%)**	**2,052 (38.7%)**	**2,219 (41.9%)**	**2,241 (42.3%)**	**2,219 (41.9%)**

2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES SANTA CLARA, CA, USA MARCH 2 - 5, 2026

Results

- Run Time improvement with flow optimizations



Conclusion

- The optimization techniques are integrated into a fully automated flow
- The proposed methodology is applicable to production-grade NAND Flash memory designs
- The approach is scalable to the other memory designs, logic designs, and full SoC designs
- The methodology is also applicable to Functional Safety and Silent Data Corruption (SiDC) evaluation use cases

Questions

Thank you!