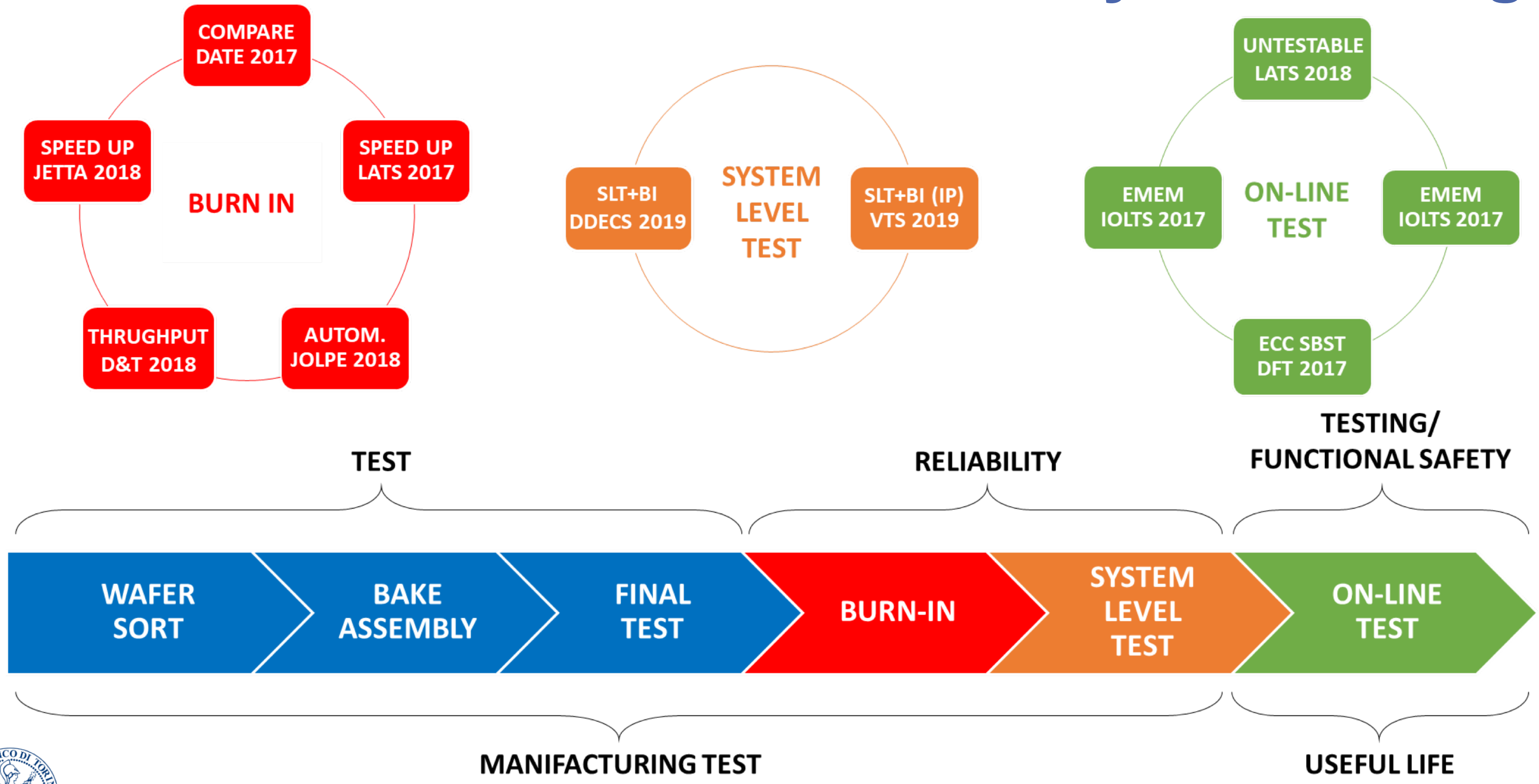
Reliability and Testing of Complex Safety-Critical Automotive SoC

Marco Restifo

Advisor: Paolo Bernardi

Doctoral Examination Committee:
Prof. Alberto Bosio, Université de Lyon, FR,
Prof. Liviu C. Miclea, Universitatea Tehnica Cluj-Napoca, RO
Prof. Zebo Peng, Linkoping University, SW
Prof. Michele Portolan, Université Grenoble Alpes, FR
Prof. Luca Sterpone, Politecnico di Torino

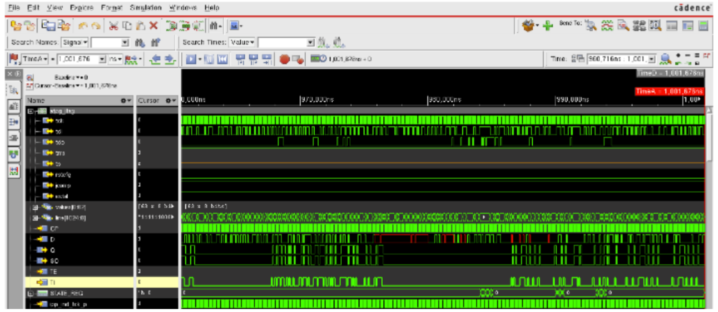
Automotive Reliability and Testing

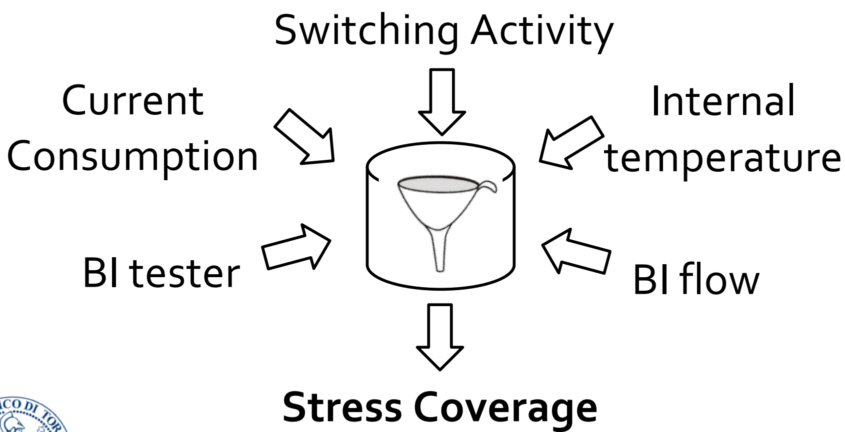


Defense Agenda

- **Burn-In Enrichment**
  - Stress Coverage Metric
  - Parallel Burn-In
  - Program Generation
  - Communication Fail Detection
  - Adaptive scheduler
- **System Level Test Challenges**
- **On-Line Test Hardening**
  - Hybrid Self-Test
  - Error Correction Code SBST
  - Untestable faults

Stress Coverage Metric





$$S_{TOT} = \omega \cdot S_{SW} + \tau \cdot S_{temp} + \theta \cdot S_{current}$$

$$S = \alpha \cdot S^{Strength} + \beta \cdot S^{Distribution}$$

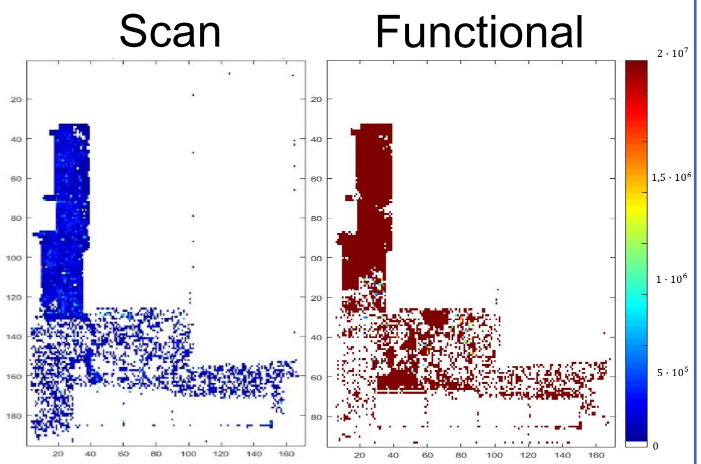
$$S^{t_{eval}} = \alpha \cdot S^{Strength} + \beta \cdot S^{Distribution}$$

$$S^{Strength} = S^{Mean} \cdot S^{Max}$$

$$S^{Distribution} = S^{Mean} / S^{std-dev}$$

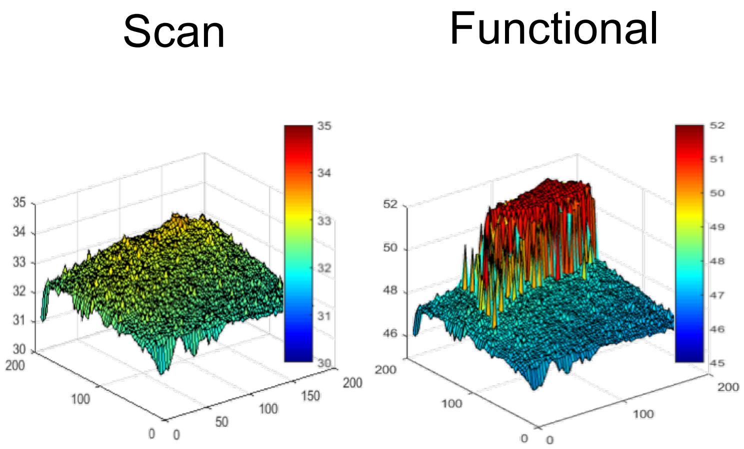
Stress Measurements Scan vs Functional

Switching Activity



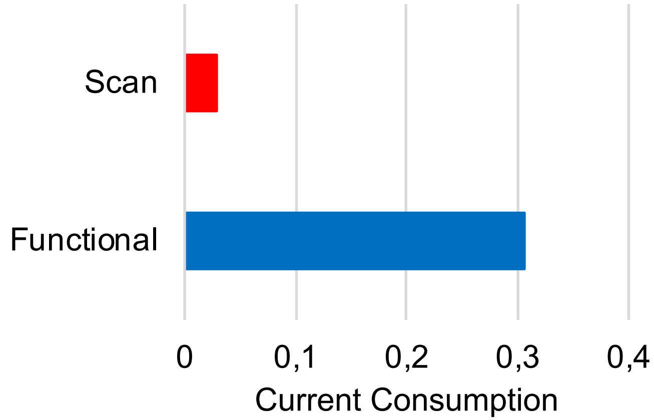
Stress Procedure	Max [SW]	Mean [SW]	Std Dev [SW]
Scan	38013	1366	425
Functional	1874855	81177	3617

Temperature Distribution



Stress Procedure	Max [°C]	Mean [°C]	Std Dev [°C]
Scan	33.80	30.85	1.4
Functional	51.38	47.89	3.17

Current Consumption



Stress Procedure	Current Consumption [mA]
Scan	28.82
Functional	306.08

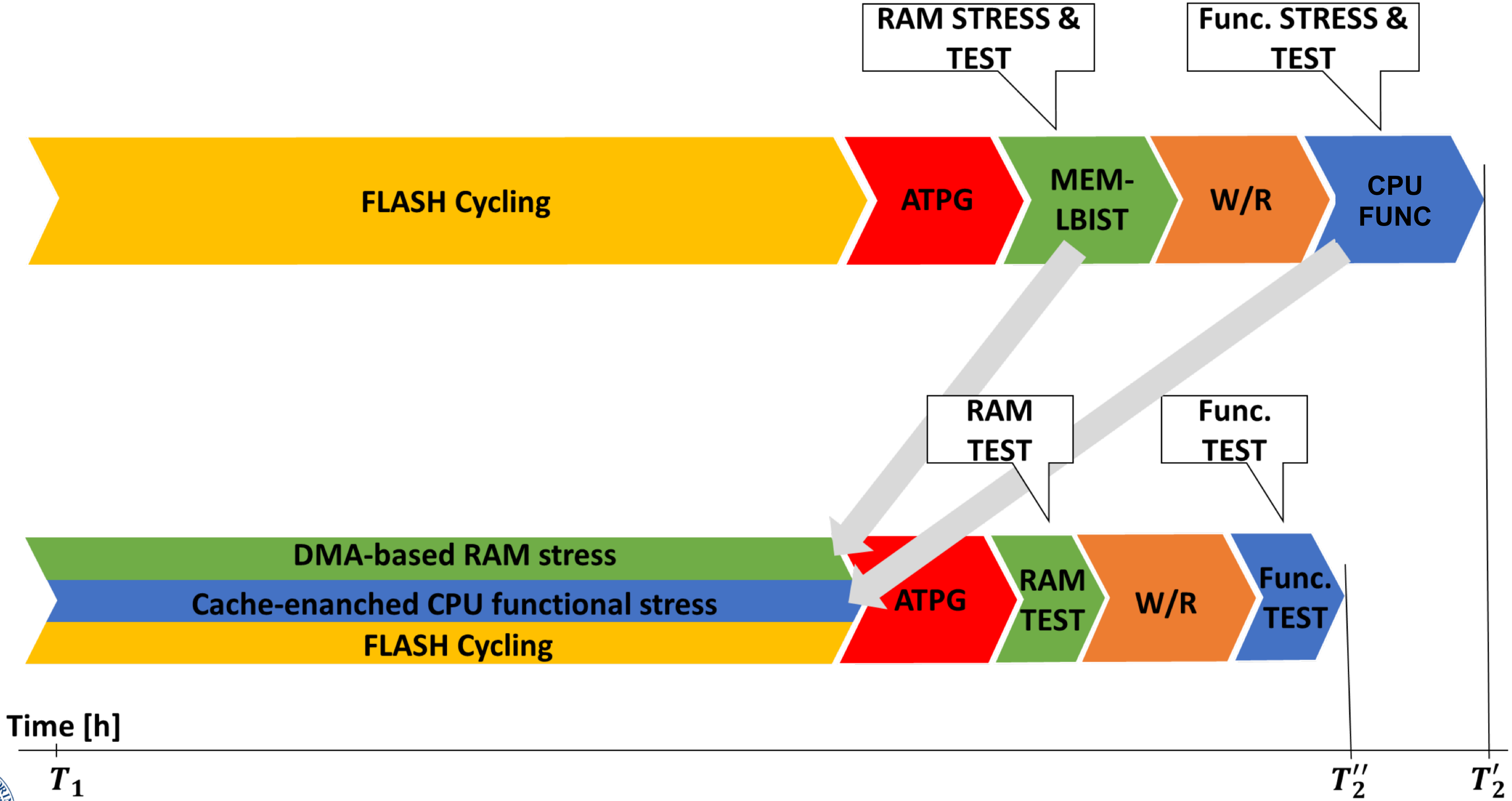
Final Results Scan vs Functional

- The Stress Coverage Metric makes it easy to compare each other stress

Stress Coverage Metric

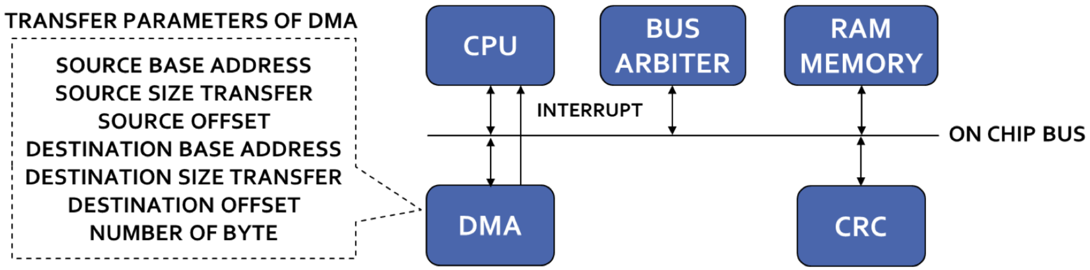
Stress Procedure	Fault class A	Fault class B	Fault class C
SCAN	**0.303**	**0.045**	**0.134**
FUNCTIONAL - DivVect	0.349	0.056	0.489
FUNCTIONAL - FP Div	0.372	0.056	0.480
FUNCTIONAL - Adder	0.384	0.060	0.518
FUNCTIONAL - Logical	0.390	0.061	0.534
FUNCTIONAL - Forwarding	0.392	0.061	0.541
FUNCTIONAL - FP Mac	0.398	0.065	0.578
FUNCTIONAL - MulVect	0.404	0.065	0.583
FUNCTIONAL - Allcore	0.409	0.067	0.592
FUNCTIONAL - composition	**1.000**	**1.000**	**1.000**

Parallel Burn-In



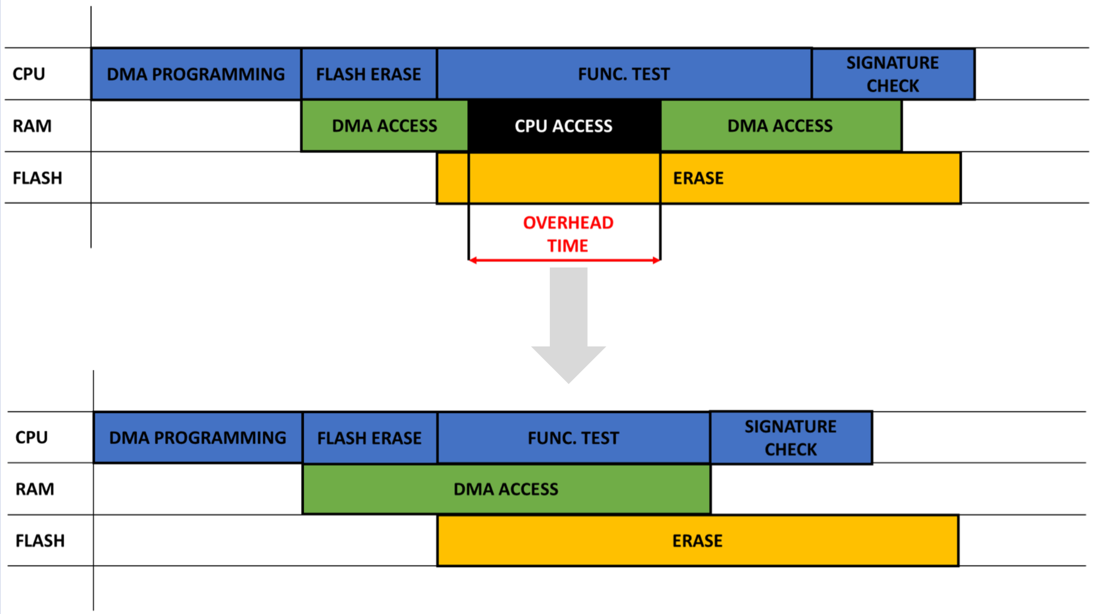
Parallel Implementation

DMA Programming



March element	SOURCE			DESTINATION		
	Base Add.	Size Tx.	Offset	Base Add.	Size Tx.	Offset
$\Uparrow Rx$	Target Add.	Target Size	Target Size	Comp Add.	Comp Size	Zero
$\Downarrow Rx$	Target Add.	Target Size	- Target Size	Comp Add.	Comp Size	Zero
$\Uparrow Wx$	Pattern Add.	Pattern Size	Zero	Target Add.	Target Size	Target Size
$\Downarrow Wx$	Pattern Add.	Pattern Size	Zero	Target Add.	Target Size	- Target Size

Cache Advantage



RAM Stress evaluation

March C-	Clock Cycles	Type of Faults covered	Possibility of parallelization
BIST	2,560	Static & Dynamic	No
SW BIST	3,320	Static & Dynamic	Partial
DMA-based	28,687	Static	Yes

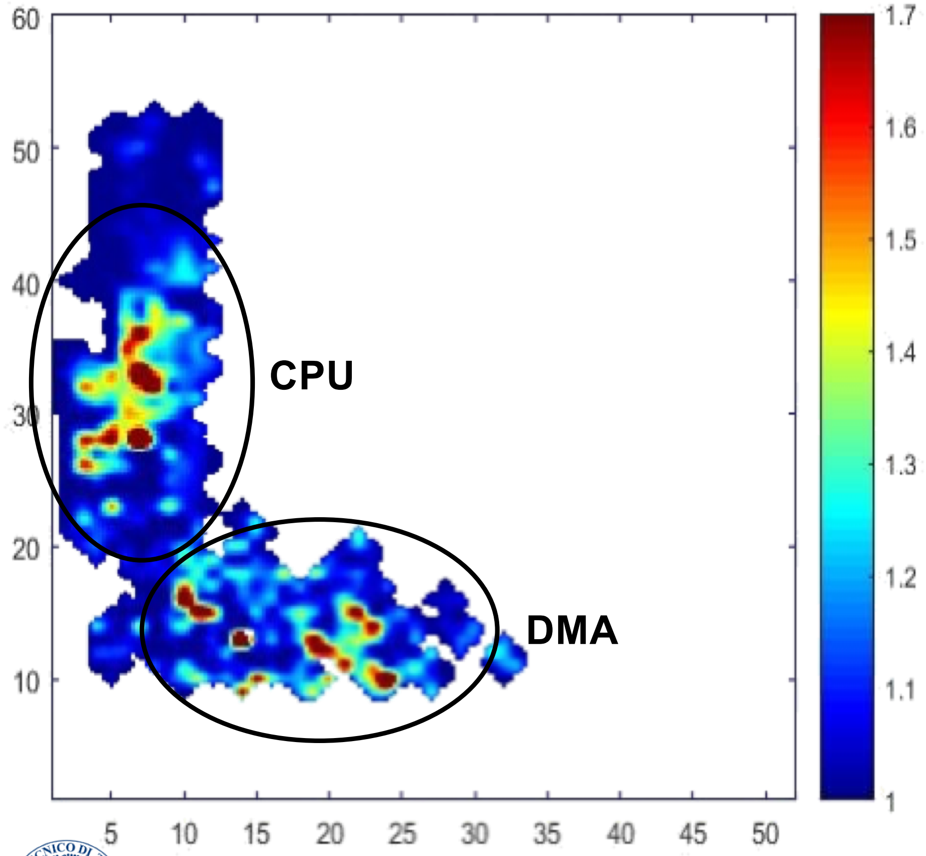
$T_{execution}$ = 4.8 seconds
< Flash cycling (5 hours)

$$T_{execution} = \frac{\text{ClockCycle1Kbyte} \cdot \text{RAMDim} \cdot \#\text{BistBI}}{\text{@Speed frequency}}$$

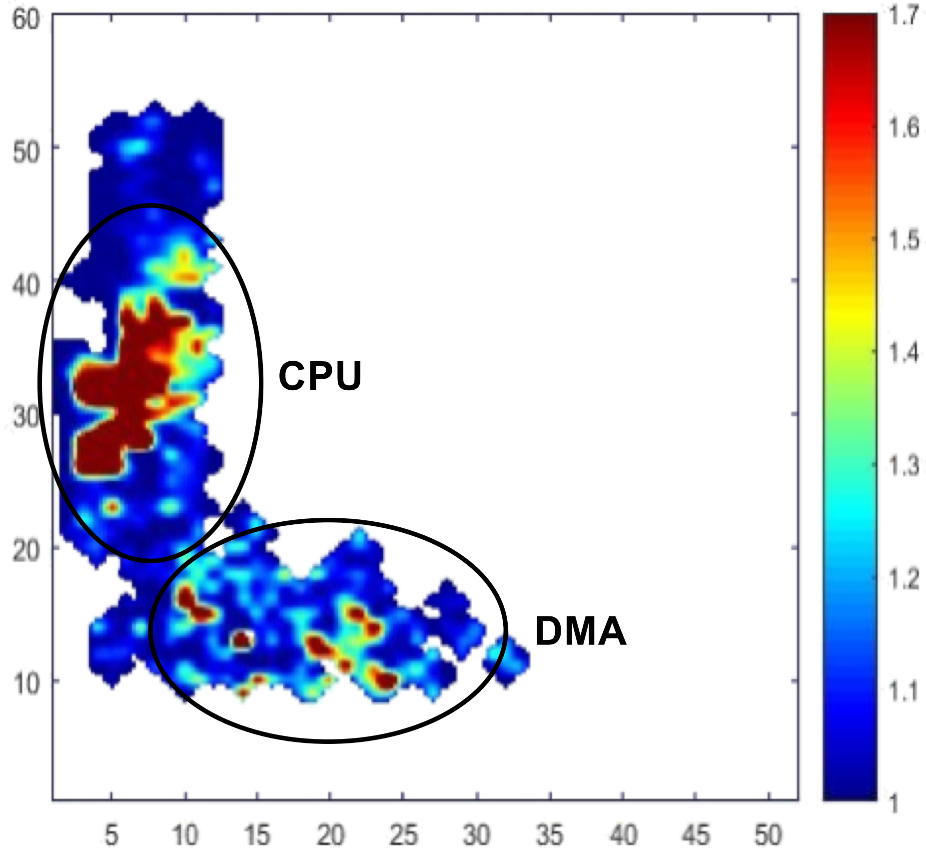
March C-
30K clock cycle/Kbyte
192KBytes
100 BIST executions
120MHz frequency

Parallel stress evaluation - Switching

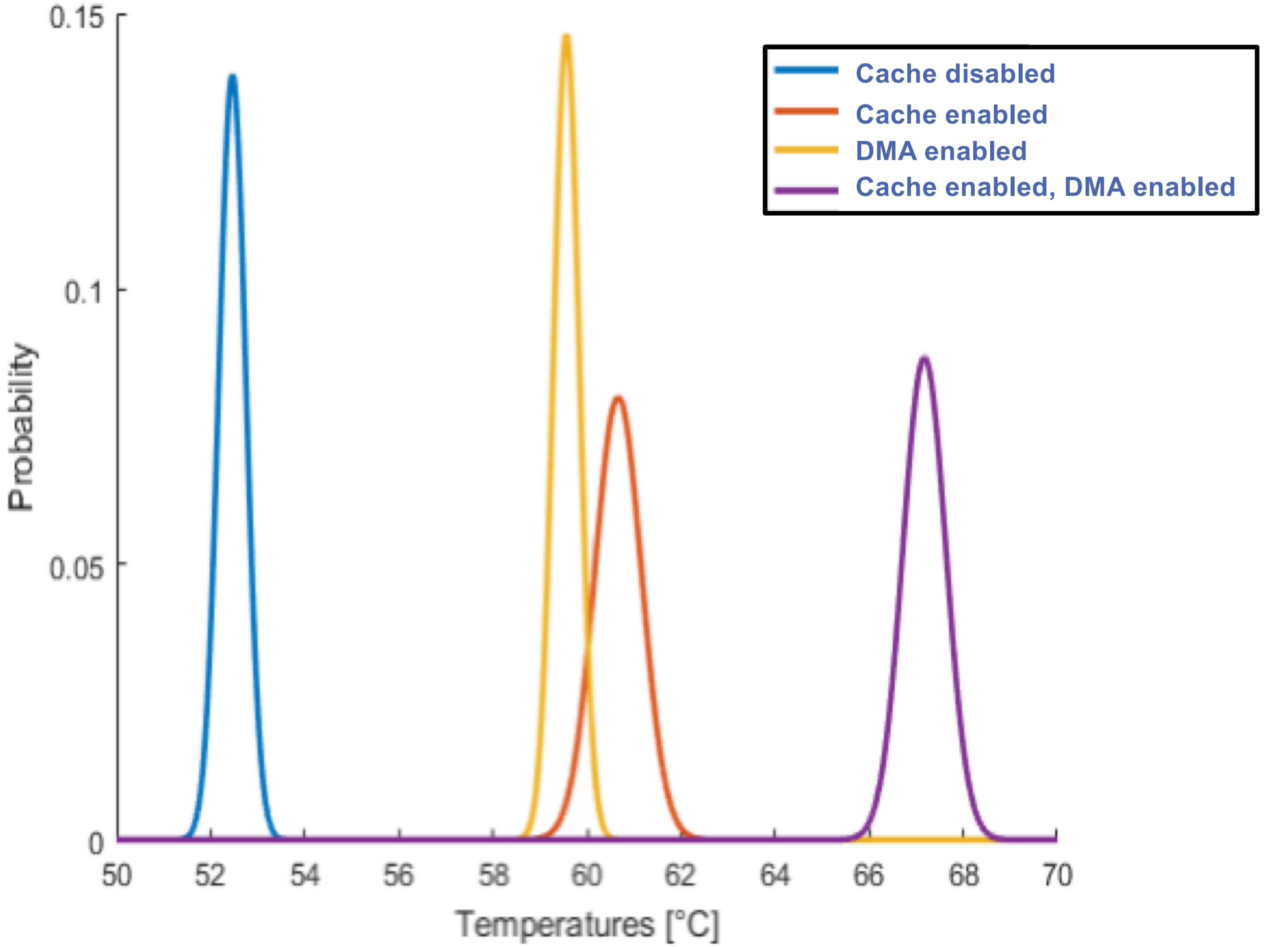
Cache disabled



Cache enabled



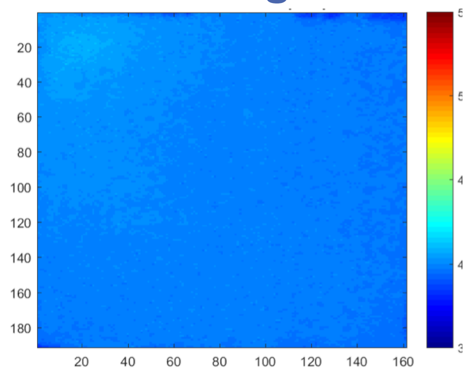
Parallel Stress Evaluation - Temperature



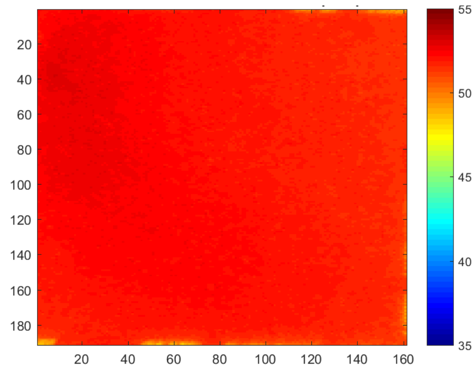
Stress program generation

- The objective is to increase the stress activities of the DUT as much as possible using functional programs.
- This a novel approach optimize the stress procedures at CPU level using an evolutionary algorithm.
- The evolutionary-based framework improves the stress of the CPU in an automatic way

ST Original

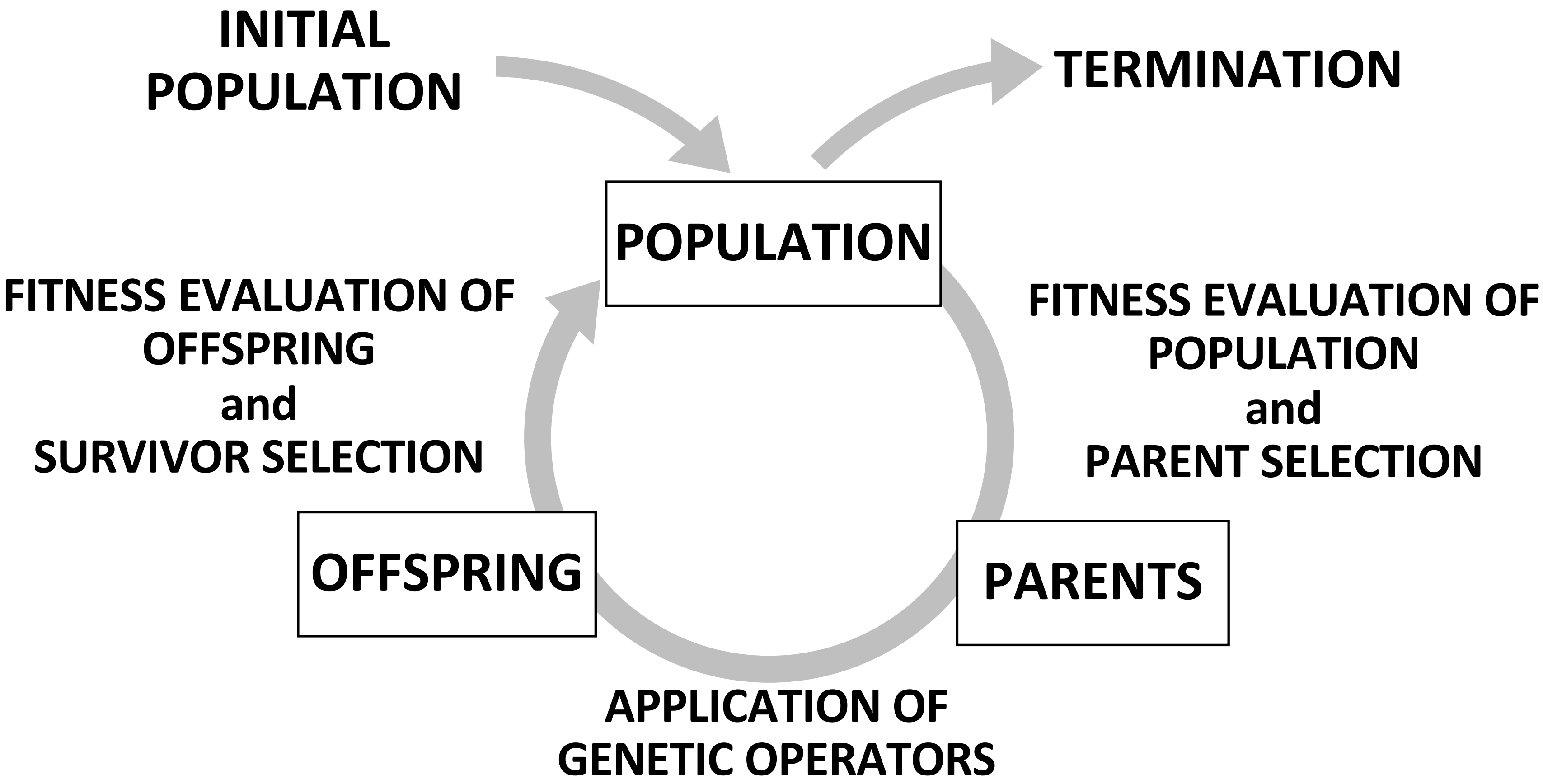


PoliTo

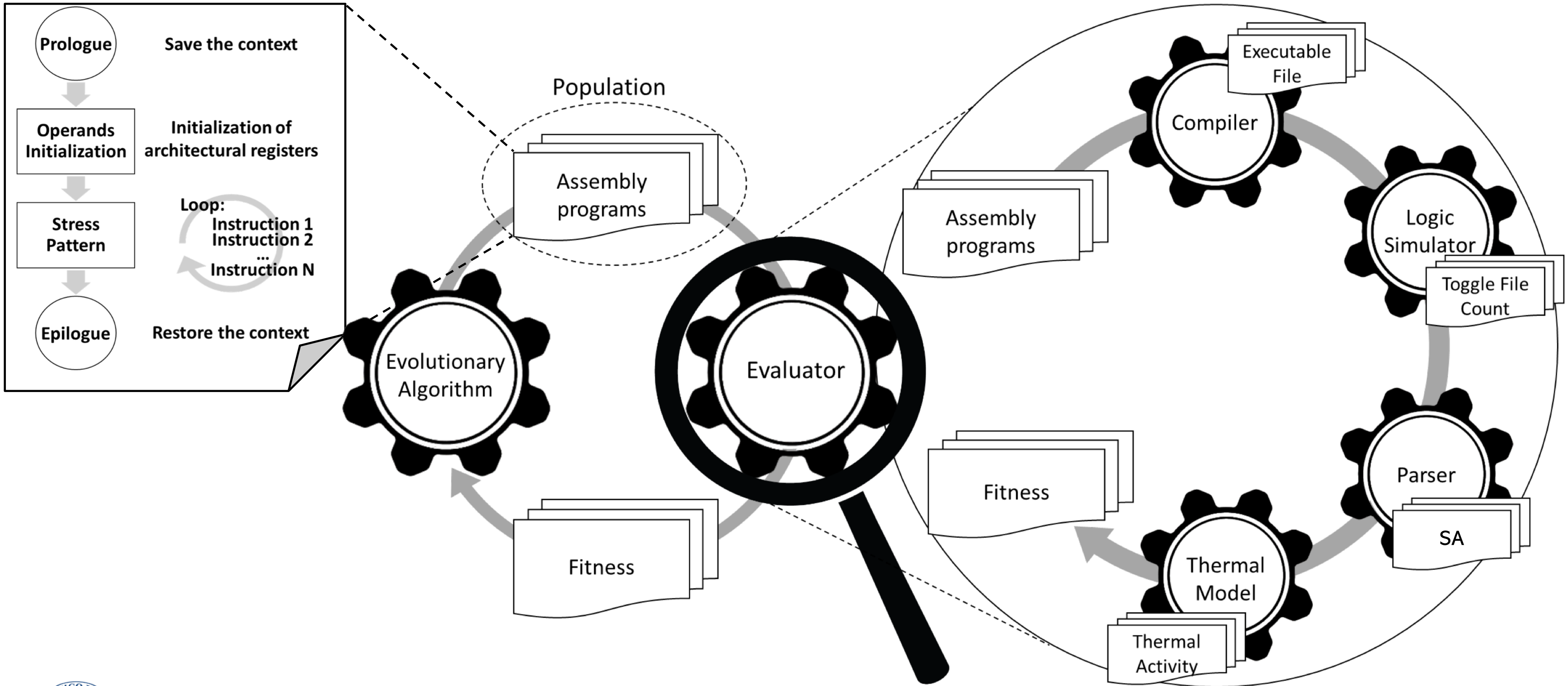


ST Stress Program	PoliTo Stress Program
$T_{MAX} = 41.14°\text{C}$	$T_{MAX} = 53.34°\text{C}$
$T_{AVG} = 40.09°\text{C}$	$T_{AVG} = 50.21°\text{C}$
$I_{MAX} = 0.02930$ A	$I_{MAX} = 0.31574$ A
$I_{AVG} = 0.02882$ A	$I_{AVG} = 0.30608$ A

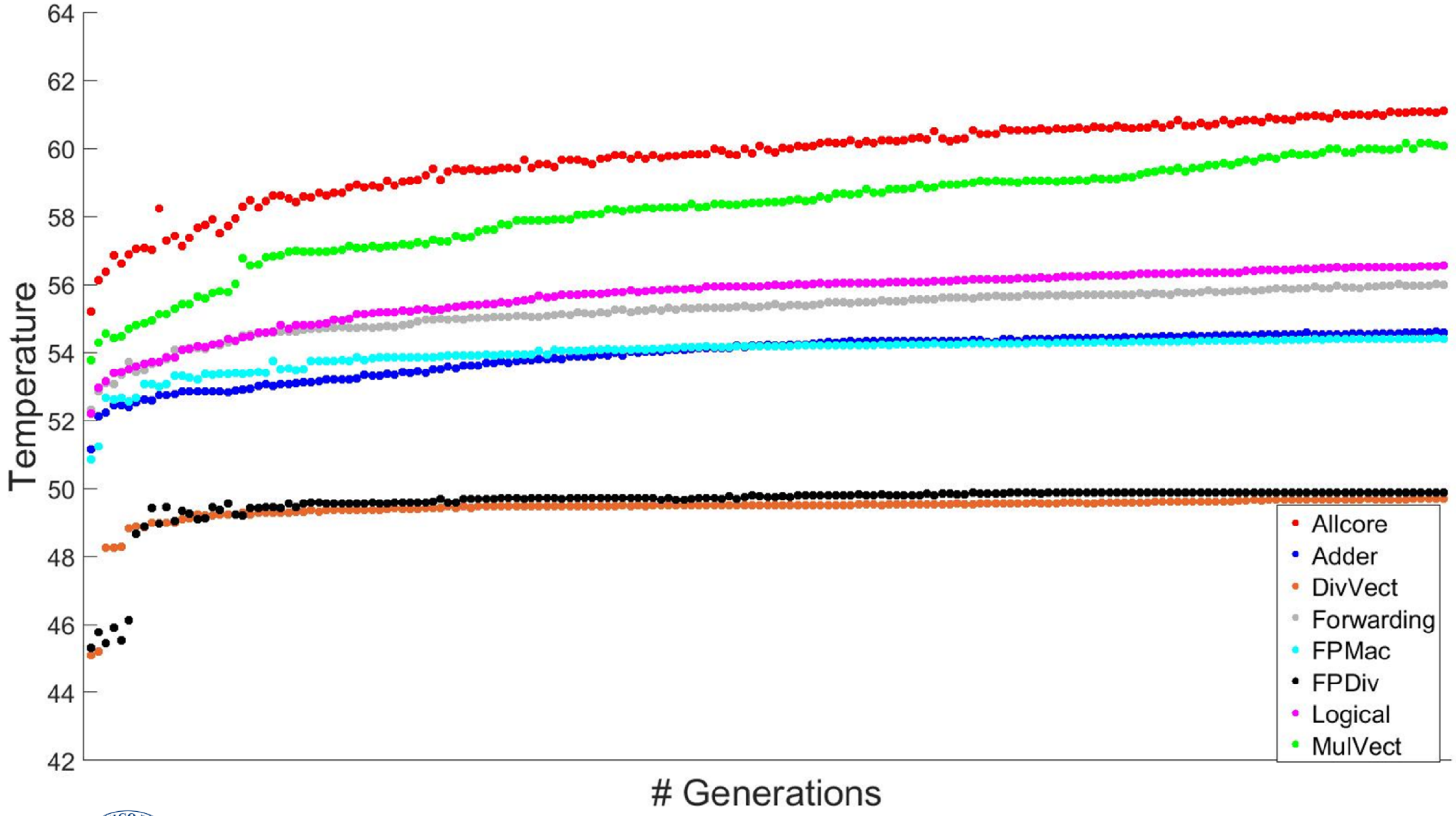
Evolutionary Algorithm



Evolutionary Framework for Stress Program

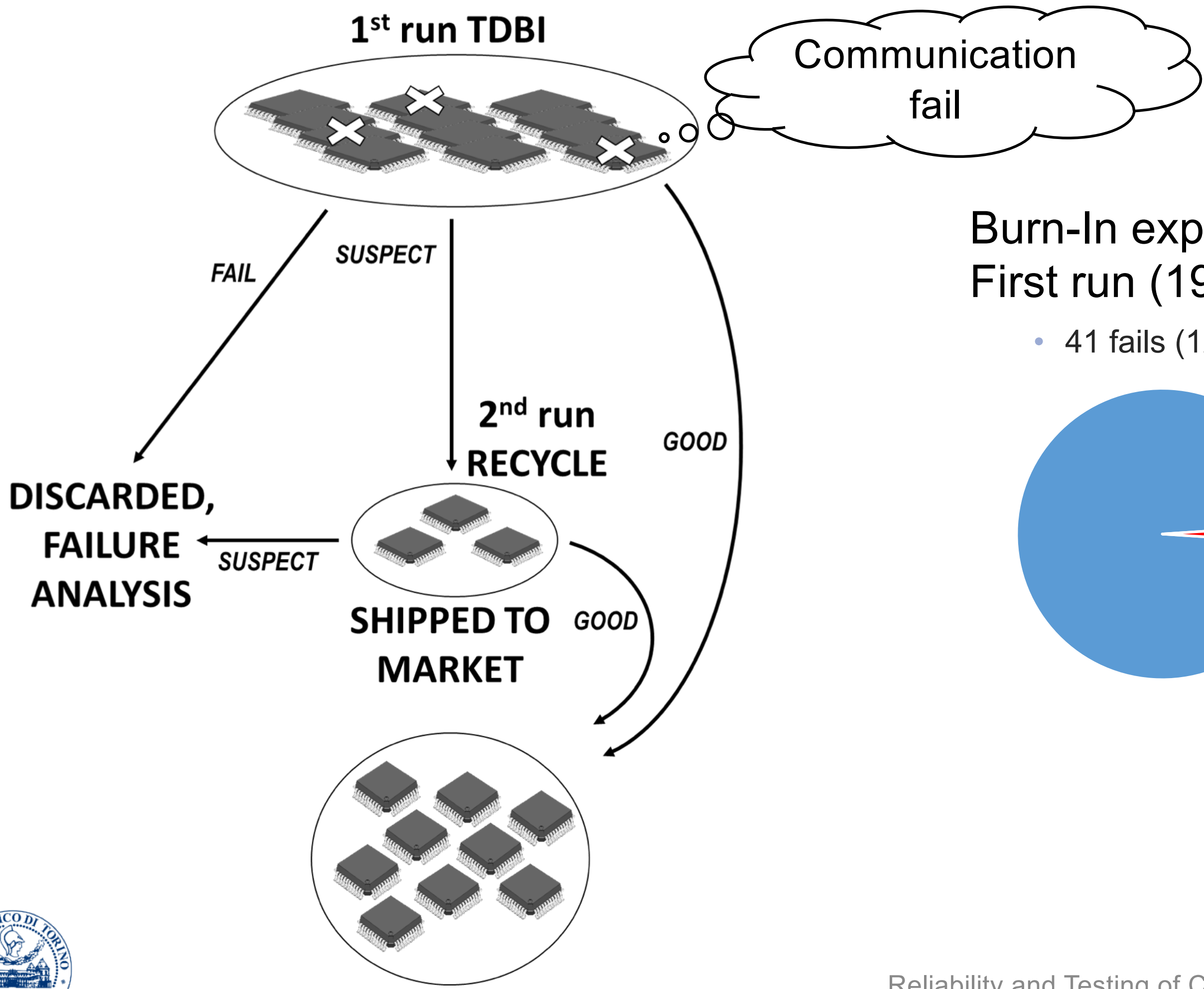


Average Temperature Evolution



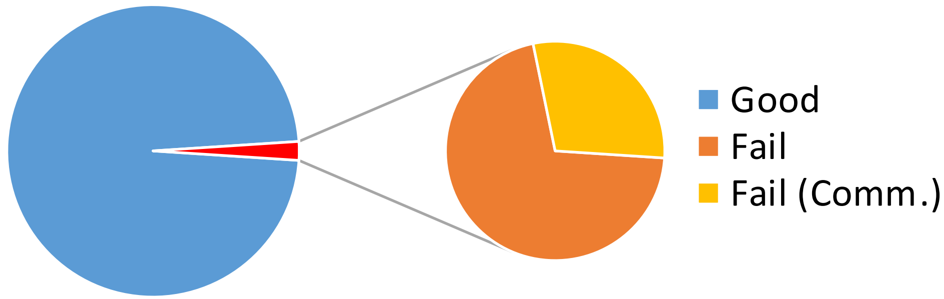
Name Program	Final Average Temperature [°C]
Allcore	61.10
MulVect	60.38
Logical	56.63
Forwarding	56.10
Adder	54.67
FPMac	54.50
FPDiv	49.90
DivVect	49.71

Communication Fails Mitigation

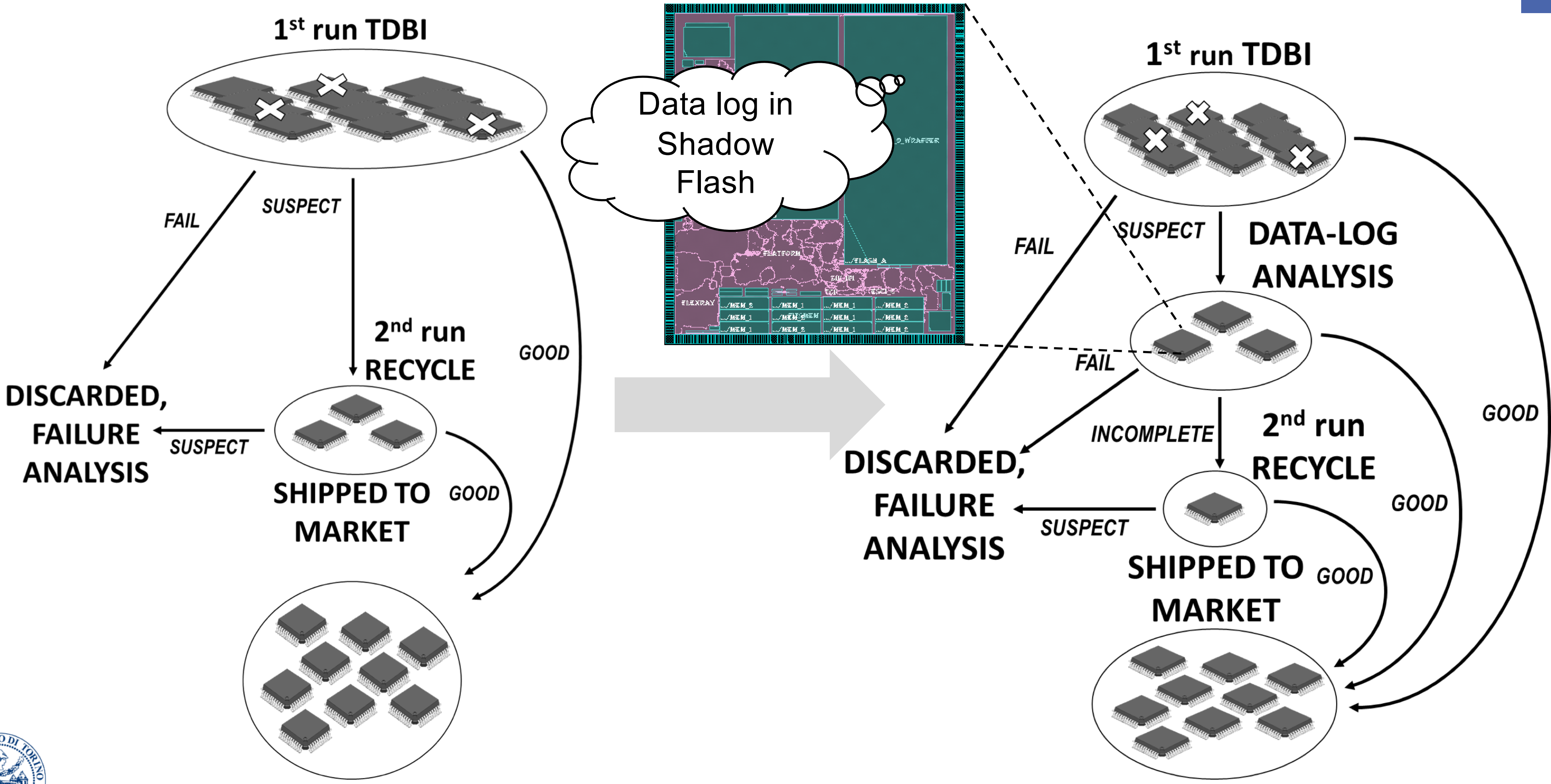


Burn-In experiment 1982 DUT
First run (1982 DUT):

- 41 fails (12 communication fails)



Burn-In Flow Communication Fail Hardening



Burn-In Data-Log Results

SEGMENT	PHASE	SUSPECT FAILS ORIGINAL [%]	INCOMPLETE FAILS PROPOSED [%]
FLASH ERASE	Stand-Alone Flash Cycling	0.14 %	-
CYCLING	Flash Cycling + Functional + RAM stress	-	0.26 % $\rightarrow$ 0 %
DYNAMIC	ATPG stress	0.99 %	0.84 %
BURN-IN	RAM memory BIST	0.11 %	0 %
	TOTAL	1.24 %	1.10% $\rightarrow$ 0.84%

Double Layer Scheduler

SoC Level:

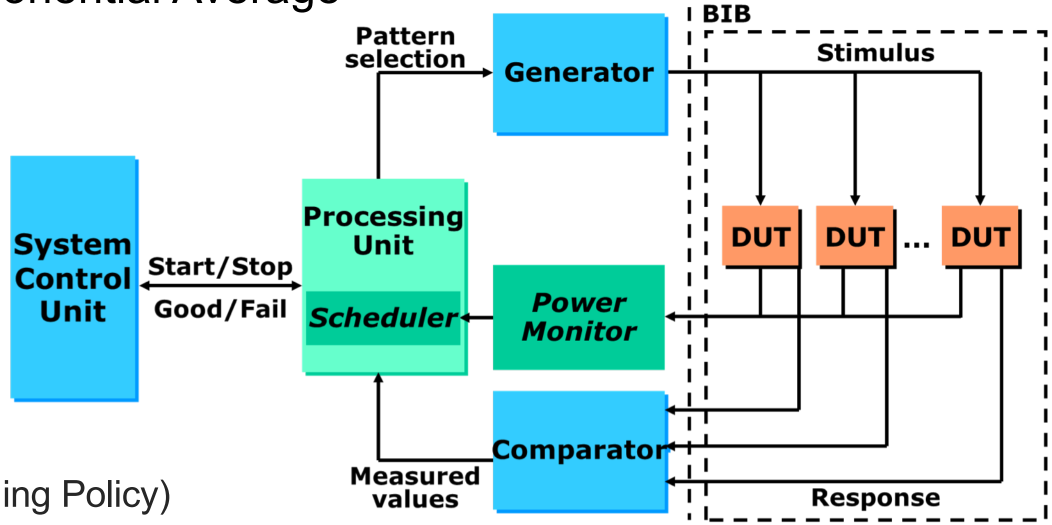
- Flash erase duration depends on temperature and chip
  - Flash Erase time spans between 25 to 45 seconds

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$$

- Flash Erase time prediction using the Exponential Average

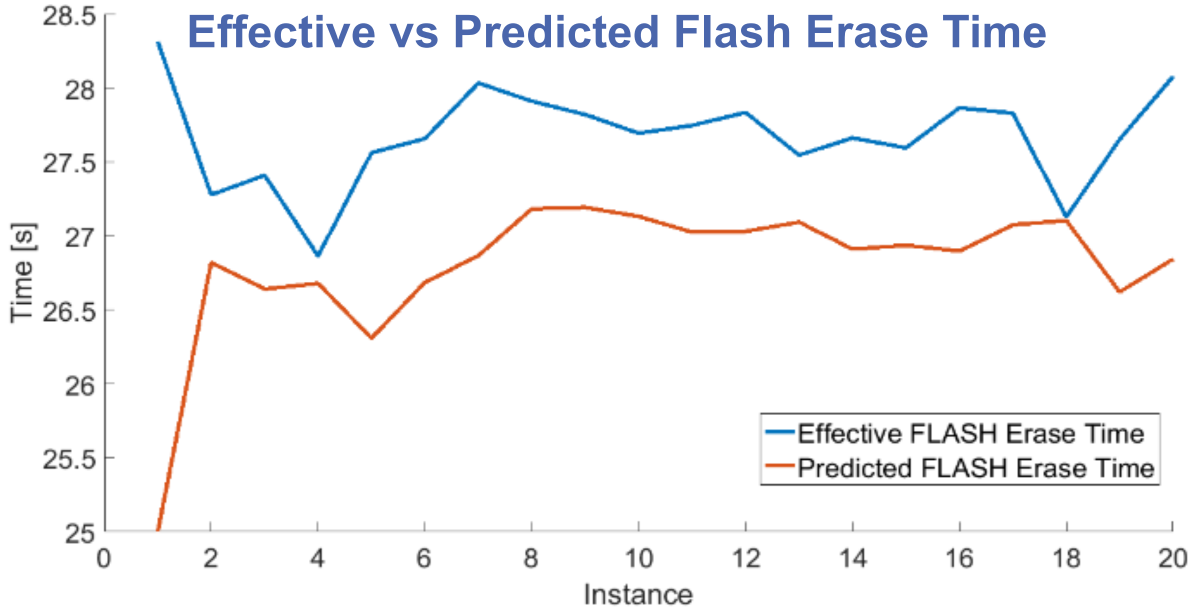
ATE Level:

- Power supply can drive a certain number of DUT
  - TDBI is statically scheduled (sector policy)
- Monitoring the free power budget
  - TDBI is dynamically scheduled (On-line Scheduling Policy)



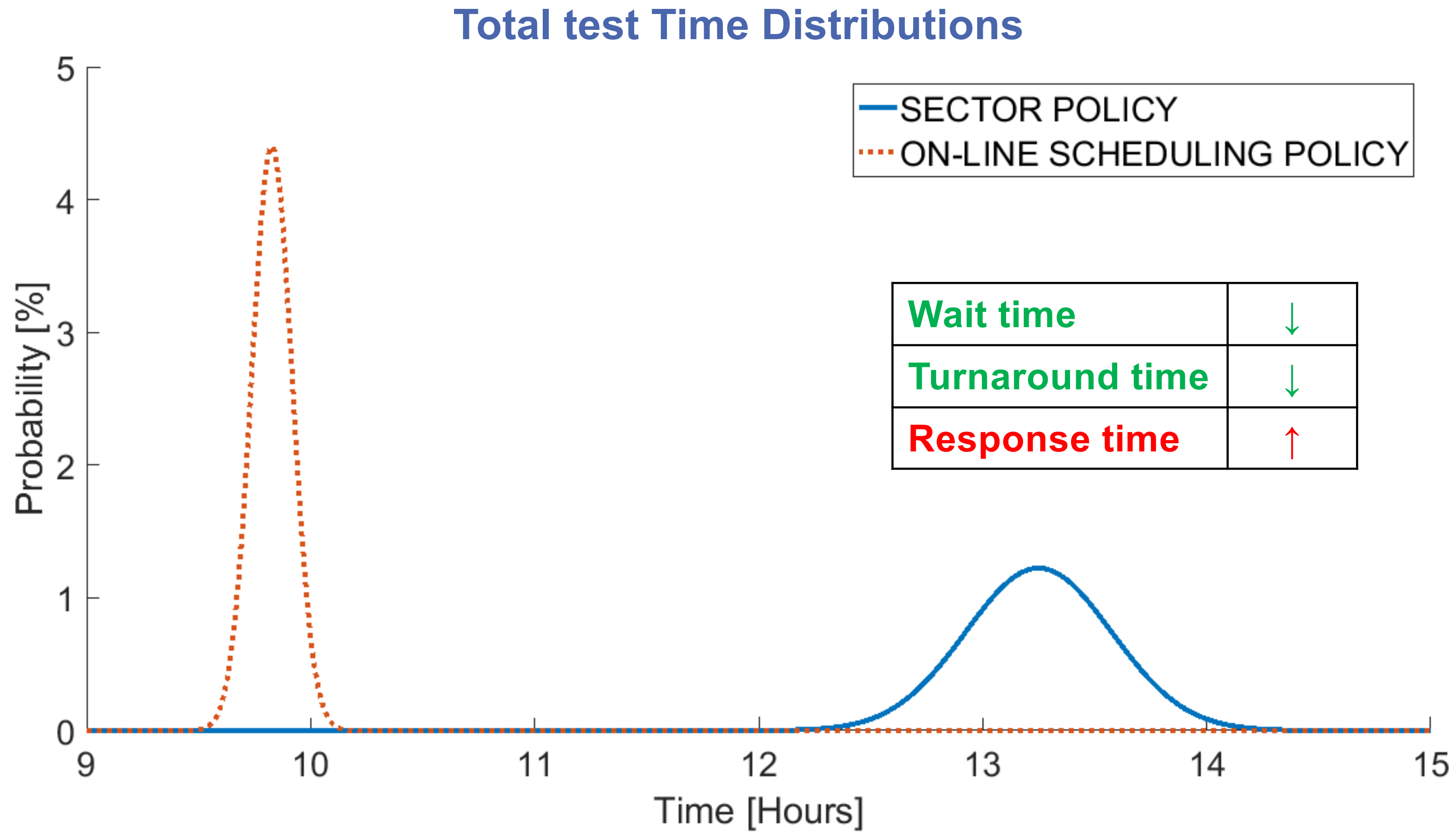
Benefit on Flash Erase Time - SoC Level

- Data collected with the data-log capability
- Exponential Average $\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n + \beta$ with $\alpha = 0.5$ and $\beta = 5$



Accuracy	96.87%
Mean Erase Time	27.67 s
Average Error	±0.87 s

Benefit on Flash Erase Time - ATE Level



Wait time	↓
Turnaround time	↓
Response time	↑

Why System Level Test?

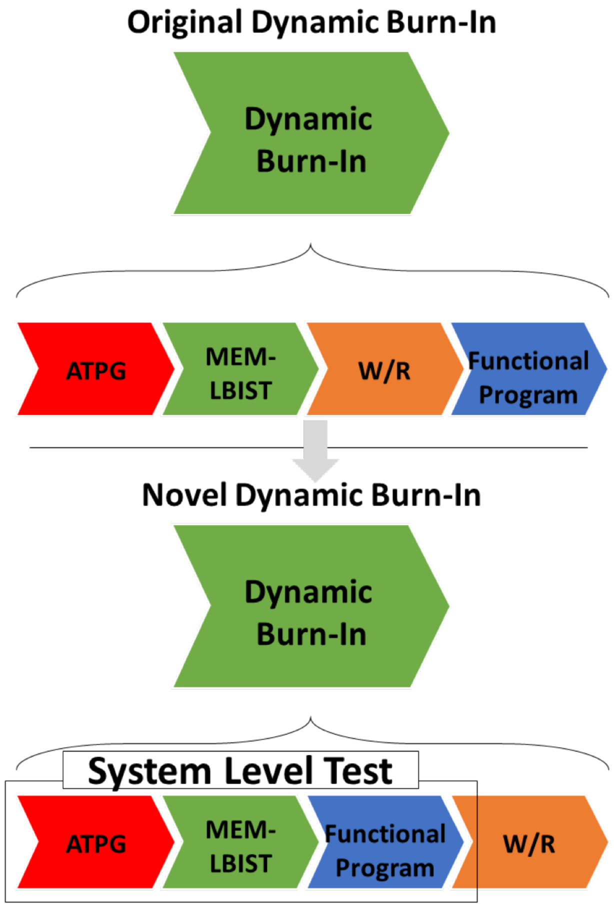
- High accuracy screening has been achieved successfully in the last 30+ years thanks to the Design for Testability and test modes.

- Some defect-free devices (100% tested) can still fail in the application:
  - It does not depends from reliability effects
    - Power/Voltage/Current/temperature derating
    - Metastability
  - Depends from difficulties in achieving exhaustive timing closure in high performance and large devices
    - High speed interfaces
  - Depends from increased complexity and consequent gap left by validation

- Huge PVT variations emphasize the described issues

- SLT is no longer purely a vehicle to validate a DUT in early qualification stages, but a valuable tool to reach maximum test coverage

Merging Burn-In and System Level Test

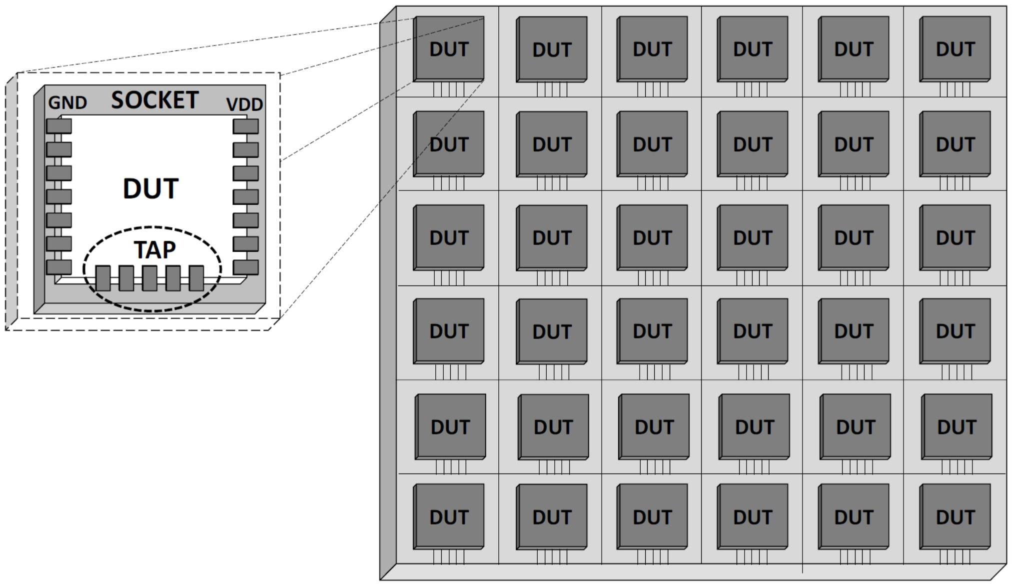


Merging Burn-in and System Level Test allows to:

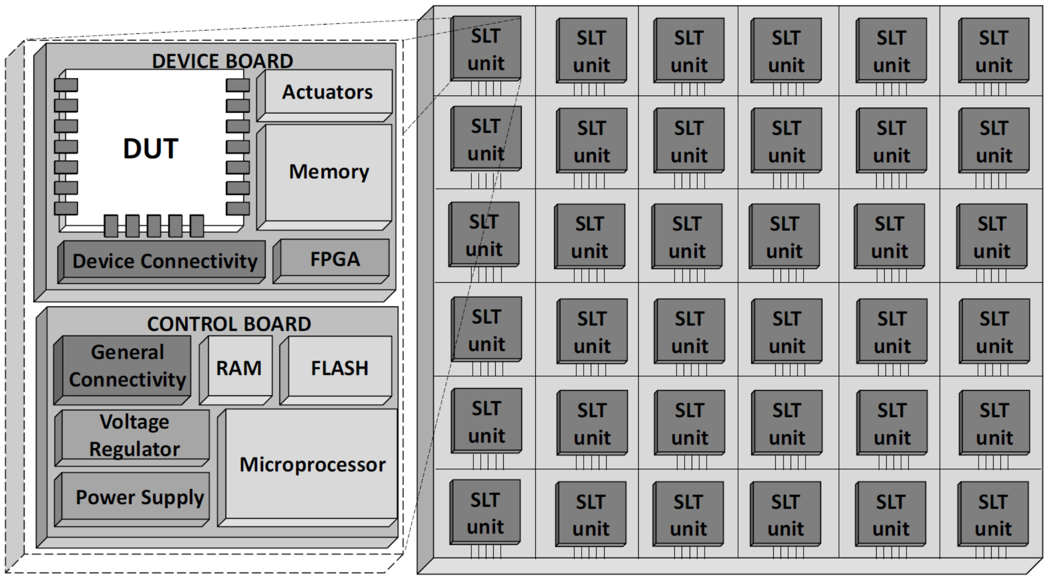
- Avoiding the load and unload phases
  - Semi-manual phase bringing extra time and cost
- Test cost reduction
- Possibility to use high-voltage
  - Reducing drastically the Burn-In time
  - Guaranteeing the Defect Part Per Millions rate

Augmented Burn-In Boards

Structure of a common Burn-In Board

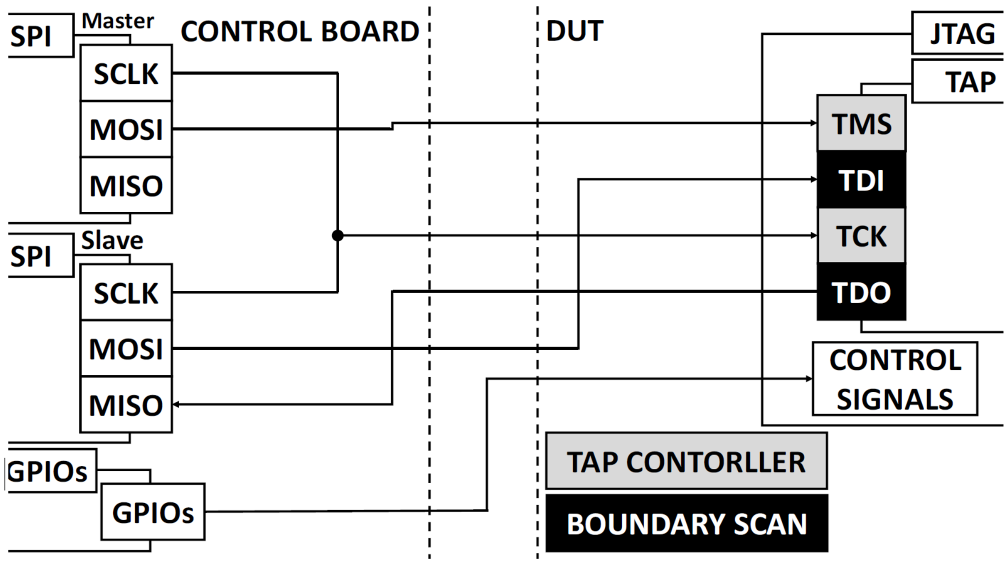


Structure of a new Burn-In Board to enable SLT

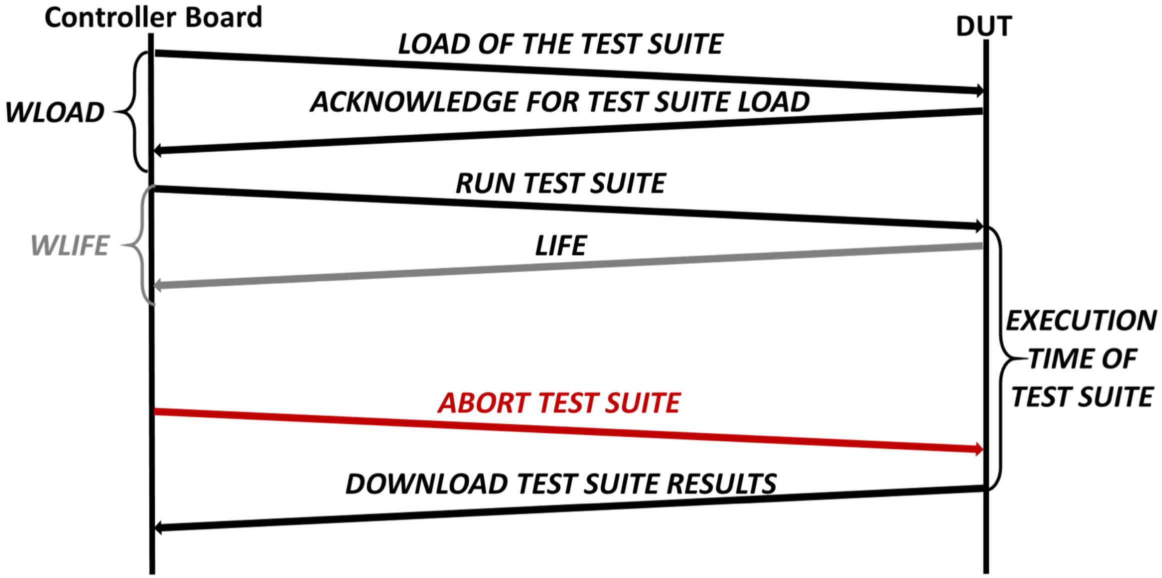


Protocols for merged Burn-In and System Level Test

Protocol for enabling a structural test inside an SLT environment



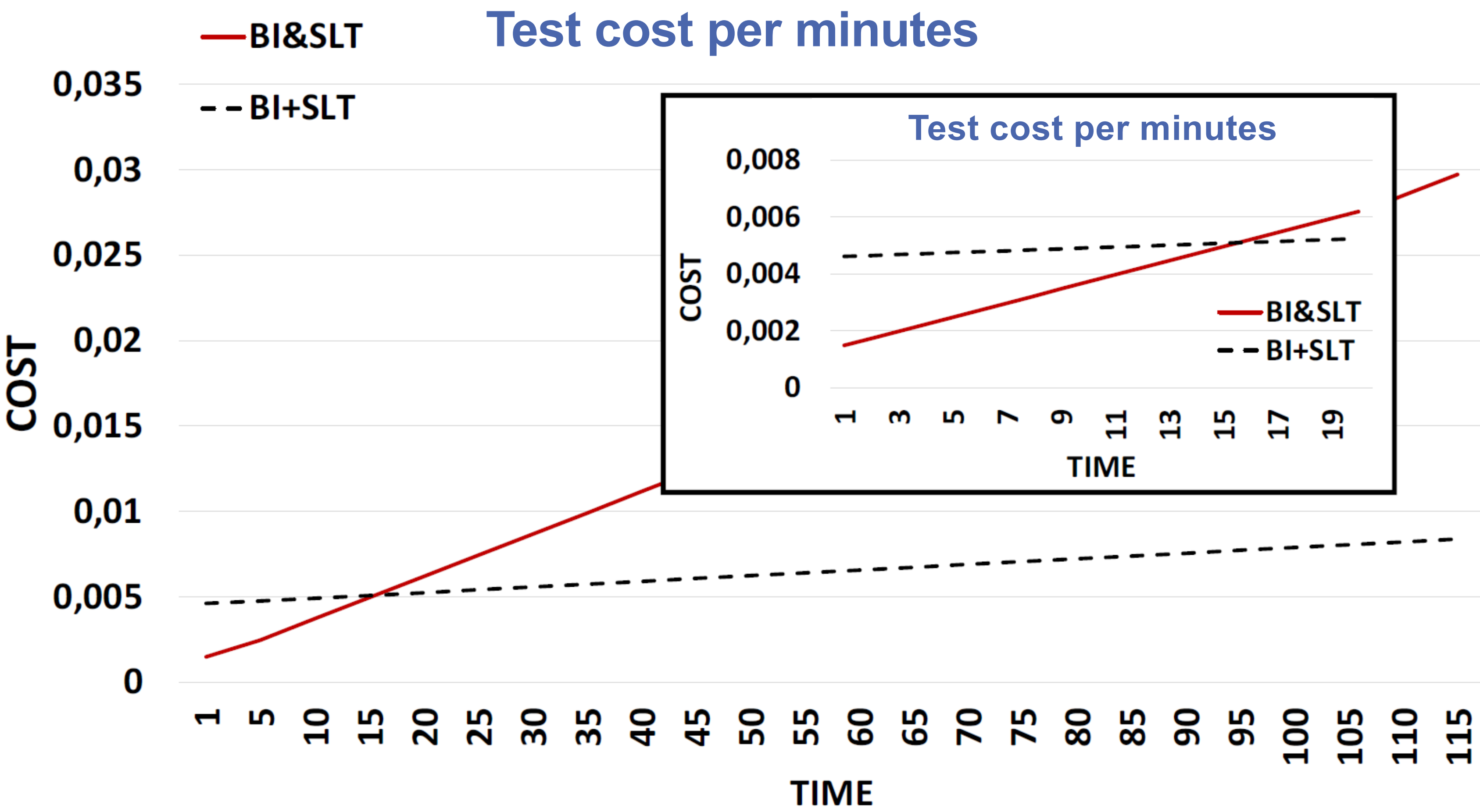
Protocol for enabling a functional test suite in SLT environment



Cost models

	Original BI+SLT		Proposed BI&SLT
N of stages	2		1
Stage	Burn-In	SLT	Burn-In & SLT
Equip. cost (arb. unit)	500,000	500,000	1,000,000
Board parallelism	100	64	64
Additional costs	120 min of Load/Unload devices	None	None
Equip. Depreciation period	6 Years	6 Years	6 Years
Test cost per device per min (arbitrary unit/minutes)	0,0000330	0,0001239	0,0002477

Test cost per minute comparison



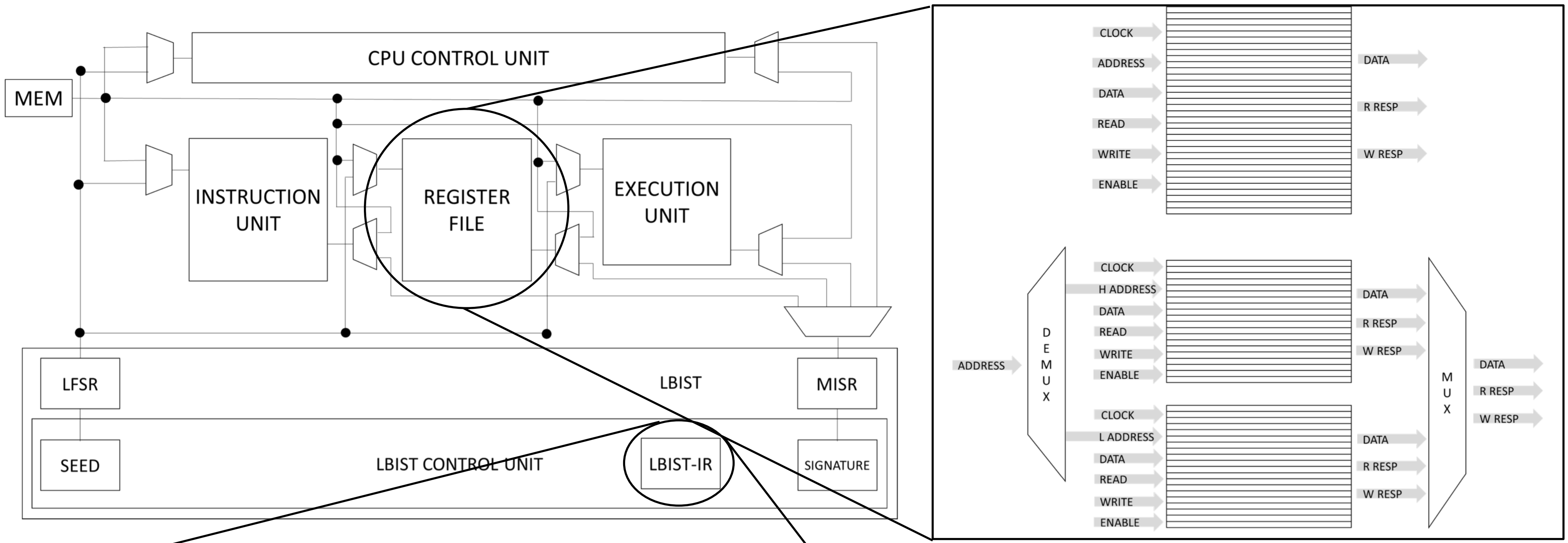
A Hybrid In-Field Self-Test Technique for SoCs

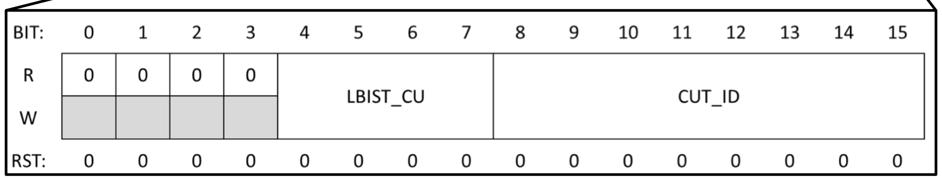
- The coverage reached by SBST and LBIST alone may be insufficient for reaching the required safety level in the safety critical domain

- The hybrid approach combines SBST and LBIST during the on-line self-test and power-on self-test of a processor core resulting in:
  - Higher fault coverage
  - Shorter self-test time
  - Availability (interrupt the self-test)

- Hybrid approach requires to:
  - Find a possible schedule between SBST and LBIST
  - Updates of Software and Hardware (SBST, LBIST, scan chain insertion)
  - Update pipeline components (Register File)

Hybrid Self-Test Architecture

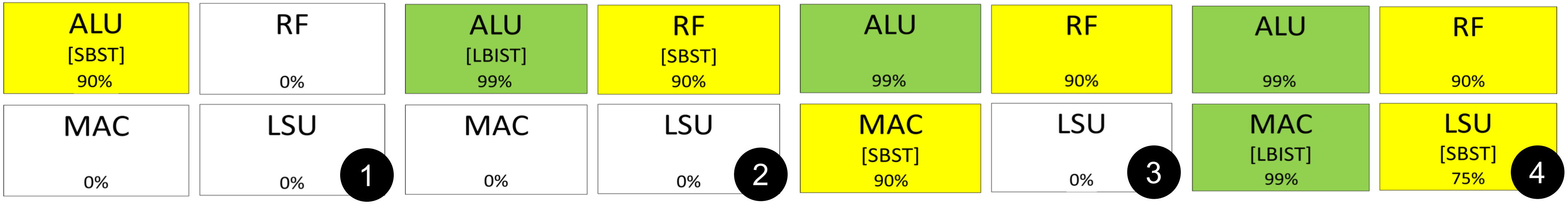




Code	Operation	Description
0001	Run	Triggers the LFSR generation
0010	Stop	Ends the LFSR generation
0100	CUT selection	Selection of the CUT

Hybrid Self-Test Scheduling and Wrapper

- SBST and LBIST on two different modules of the core pipeline



- The wrapper initializes, controls and runs the DfT by means of a SBST

```
SBST ROUTINE{
        LBIST.SEED ← VALUE OF THE SEED
        LBIST .LBIST-IR.LBIST_CU ← CUT SELECTION COMMAND
        LBIST.LBIST-IR.CUT_ID ← CODE OF THE CUT
        LBIST .LBIST-IR.LBIST_CU ← LBIST RUN COMMAND

        SBST CODE

        LBIST.LBIST-IR. LBIST_CU ← LBIST STOP COMMAND
        LBIST.LBIST-IR. LBIST_CU ← LBIST IDLE COMMAND
        LOAD THE VALUE FROM LBIST.SIGNATURE
        COMPARE LBIST.SIGNATURE WITH THE GOLD RESPONSE
}
```

Hybrid Self-Test - Results

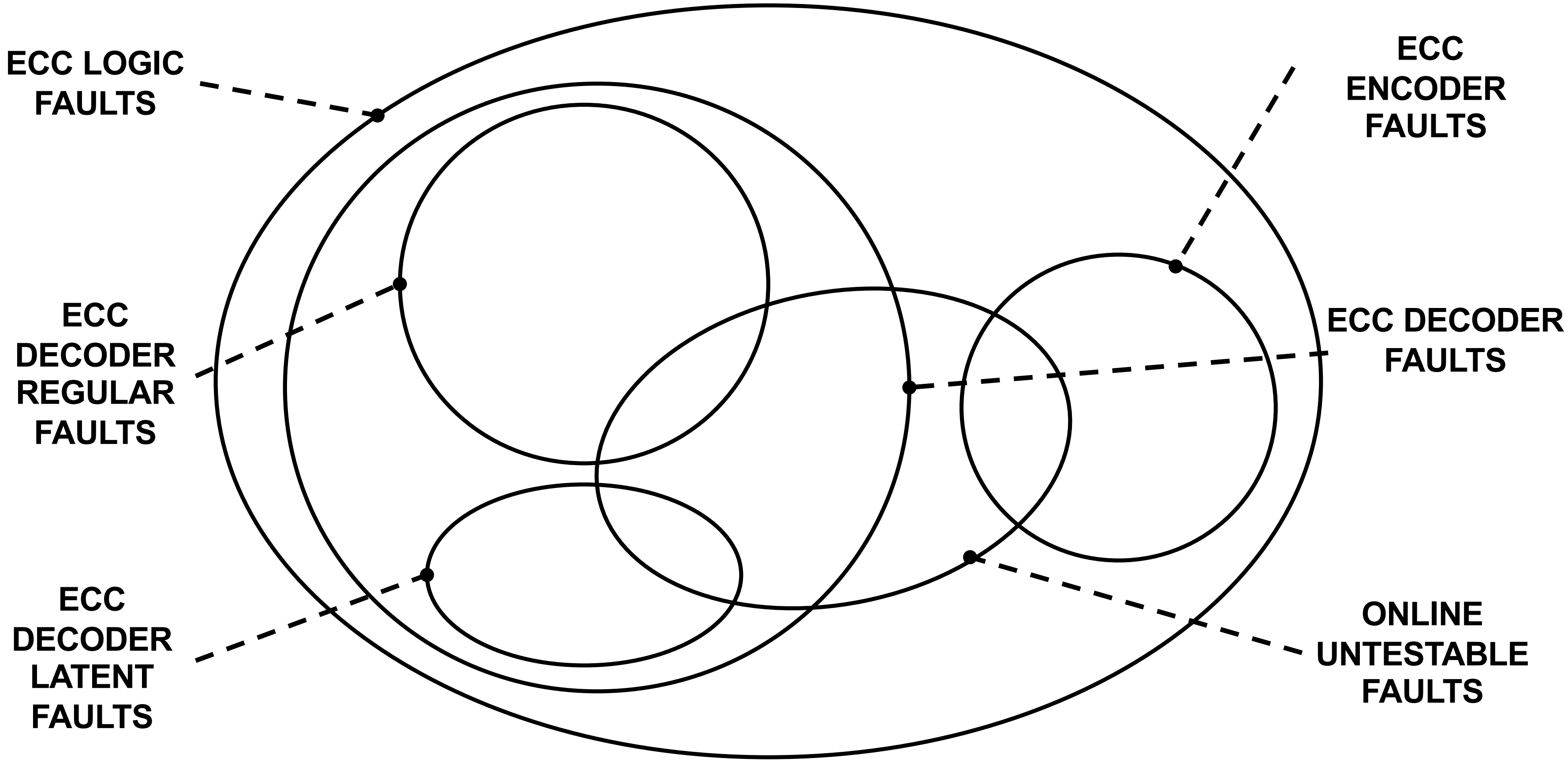
PROGRAM \ MODULE	PC	CTRL	RF	MUX	ALU	MAC	SPRF	LSU	WB	TOTAL
PC	56,21	65,76	65,96	95,19	58,24	23,07	15,33	49,82	64,76	48,39
CTRL	53,03	38,45	39,25	90,89	54,86	52,45	35,18	56,52	71,13	49,07
RF	53,51	19,13	82,51	89,51	5,51 ->0	0 -> 10,89	9,95	51,28	55,76	36,79
MUX	47,1	38,14	37,9	92,5	53,35	43,84	23,7	66,63	71,9	44,79
ALU	54,28	34,91	67,58	90,29	89,05	40,44	10,11	63,58	67,73	57,07
MAC	51,15	44,38	49,07	96,49	50,4	88,78	30,87	51,1	71,95	65,12
LSU	49,64	18,47	61,49	94,84	41	0	9,95	71,72	67,89	34,92
WB	53,03	38,45	39,34	91,46	54,86	52,45	35,18	56,94	71,73	49,14
CPU	56,39	69,41	90,69	96,83	93,83	92,72	39,74	75,56	76,4	88,16

Module	**PC**	**CTRL**	**RF**	**MUX**	**ALU**	**MAC**	**SPRF**	**LSU**	**WB**	**TOT**
SBST	56.39	69.41	90.69	96.83	93.83	92.72	39.74	75.56	76.40	88.16
HYBRID (O)	56.39	69.41	99.56	96.83	99.44	98.87	39.74	75.56	76.40	94.10
HYBRID (P)	95.25	94.24	99.56	97.01	99.44	98.87	76.36	75.56	90.64	98.9

SBST on ECC Logic

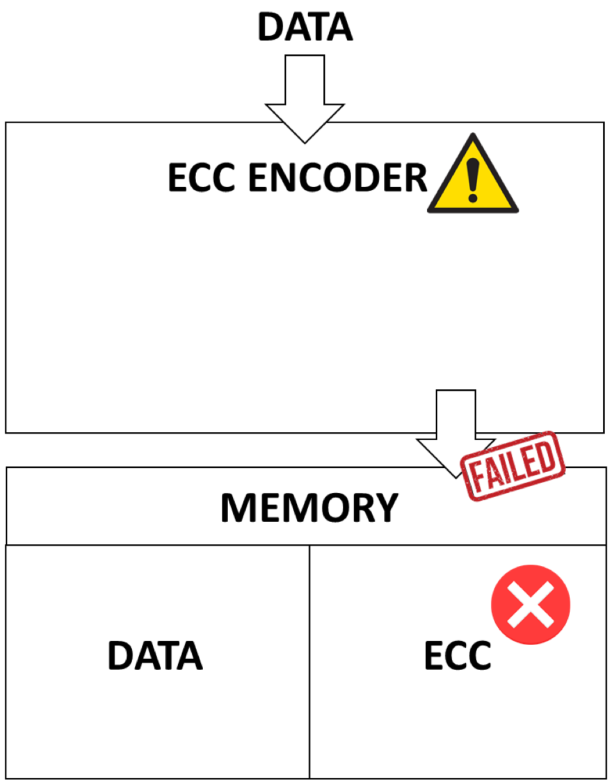
- ECC logic is small but it is essential for reliability
  - It detects and corrects major of data curruptions
  - It takes advantage of additional bits to aguments the information
- ECC logic works and it ages fast because it is deeply involved in the application
  - Fetching Instructions at every clock cycle
  - Data storage (LOAD/STORE)
- A fault in ECC logic can drastically impact on the behavior of the application
  - instructions might not be execute in the right way and the flow might change
  - faulty words might not be corrected properly

ECC Faults Taxonomy

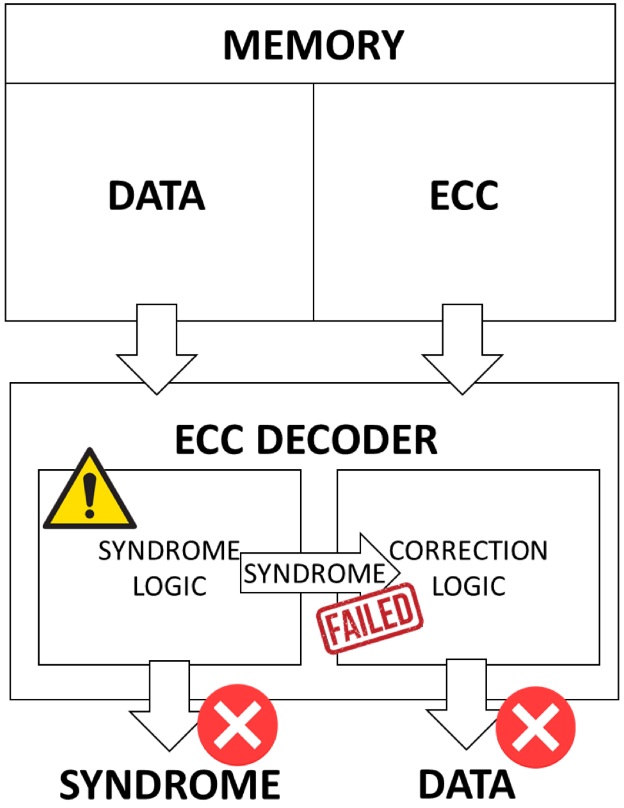


Faulty ECC logic behavior

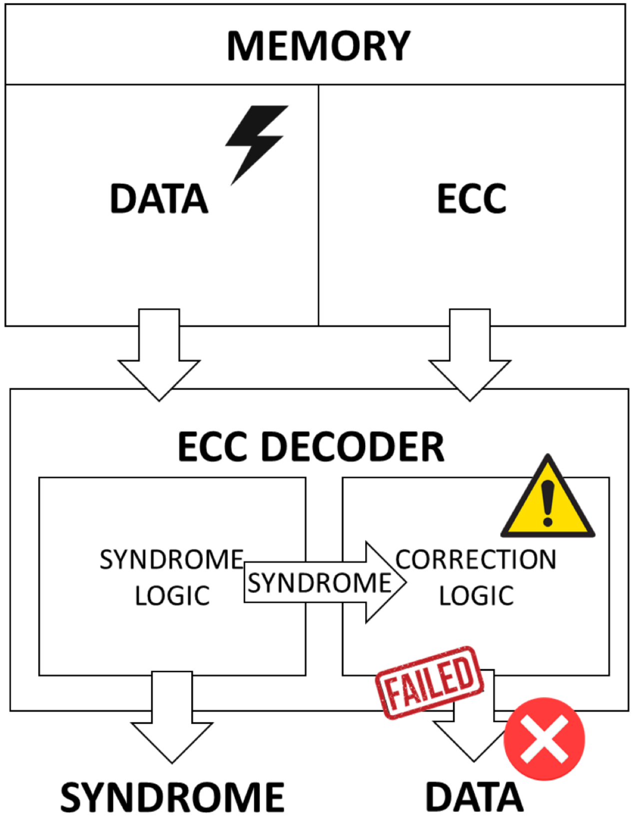
ECC Encoder Fault



ECC Decoder Regular Fault

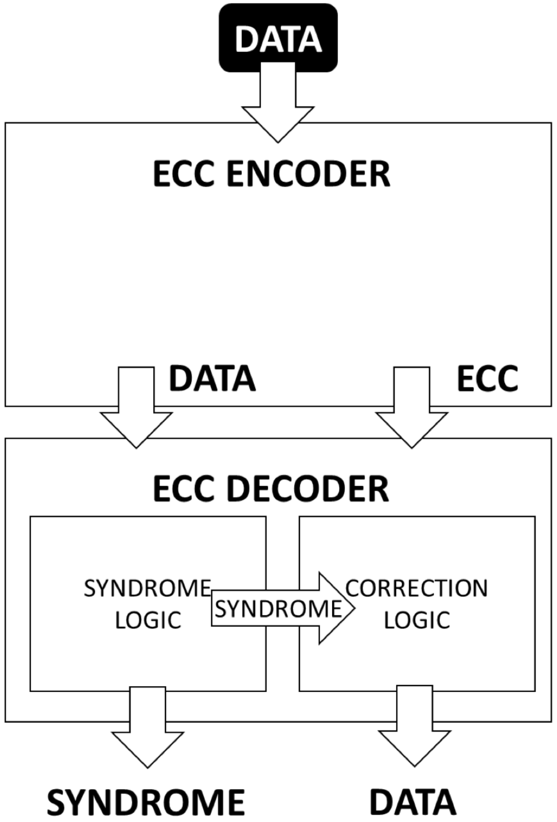


ECC Decoder Latent Fault

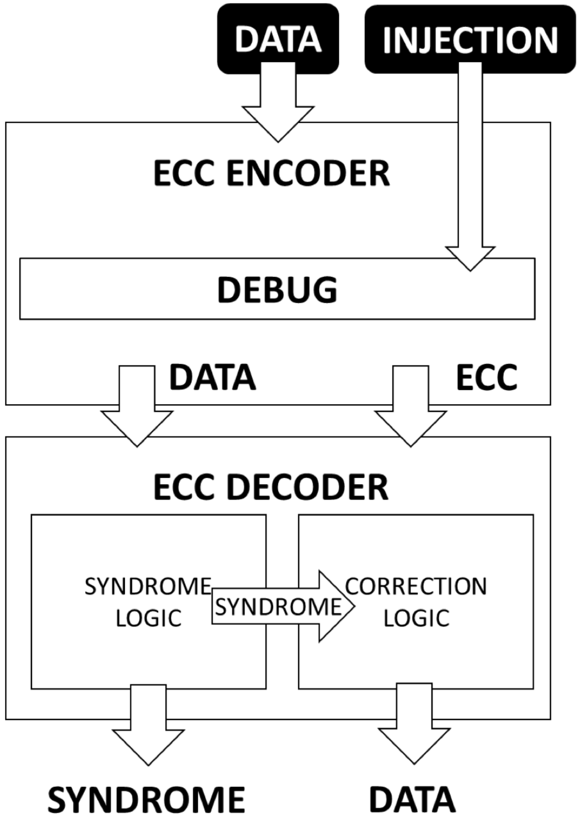


ATPG Framework for testing ECC logic

ATPG for Regular Faults



ATPG for Latent Faults



ECC Logic SBST - Results

LOGIC	#FAULTS	FAULT COVERAGE %		
		SW BIST	SBST Random	SBST ATPG
Ecc logic	**31,608**	**61.60**	**85.97**	**93.00**
Encoder	13,275	85.12	86.77	94.39
Decoder	18,864	44.90	81.25	92.06
no memory corruption	11,379	74.43	78.52	92.22
Latent faults	6,868	0	93.08	96.13
Single bit-flip	6,410	0	93.27	96.44
Double bit-flip	458	0	90.39	91.72
On-line functionally untestable faults	689	-	-	-

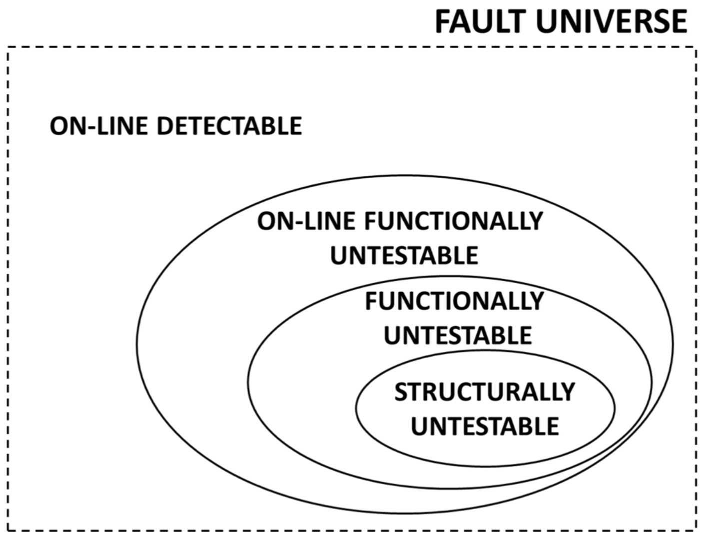
Untestable Faults for Safety-Critical Cores

- The probability that a fault becomes a failure has to be evaluated in safety-critical system.
  - On-line untestable faults can be removed

- A software might not use hardware resources
  - Unused hardware → faults may be untestable → faults may be removed

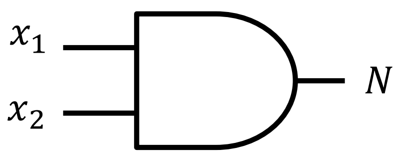
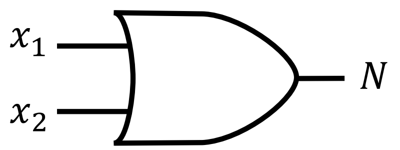
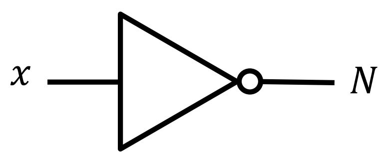
Two contributes:

1. Gates identification theory for on-line untestable faults
2. Semiautomated and scalable method



Controllability

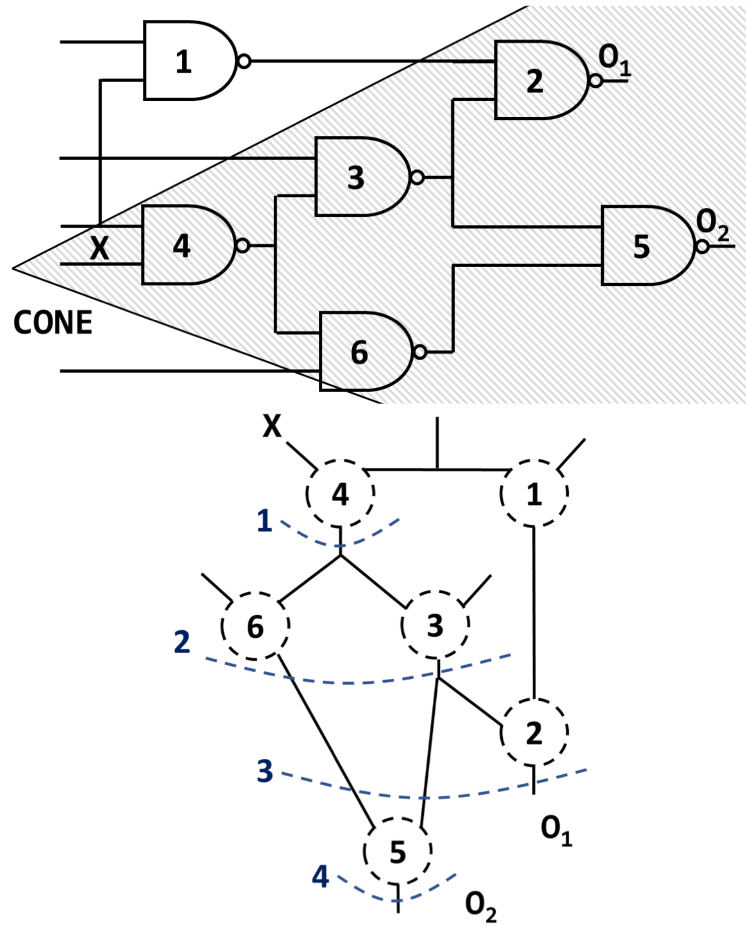
- Probability that a random input vector for a combinational block forces a given line l to the value 1 ($C^1 = 1$) or 0 ($C^0 = 1$)
  - Logic function
  - Controllability inputs probability 0.5

AND	$C^0(N) = 1 - \prod_{i=1}^{N} C^0(x_i)$ $C^1(N) = 1 - C^0(N)$		$C^0 = 0.75$ $C^1 = 0.25$
OR	$C^0(N) = 1 - C^1(N)$ $C^1(N) = 1 - \prod_{i=1}^{N} C^1(x_i)$		$C^0 = 0.25$ $C^1 = 0.75$
NOT	$C^0(N) = 1 - C^1(x)$ $C^1(N) = 1 - C^0(x)$		$C^0 = 0.50$ $C^1 = 0.50$

Cone Partitioning Algorithm

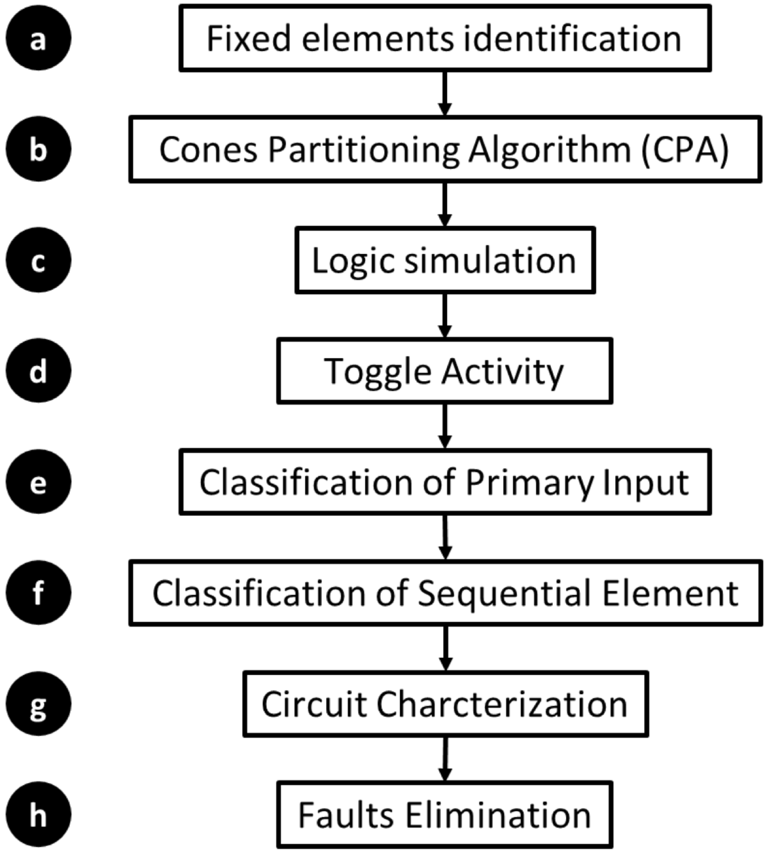
- A cone in a combinational block is the set of all gates that are directly or indirectly fed by a given input signal. The cone starts from input pin X and arrives up to output O^1 and O^2.

- The Cone Partition Algorithm is based on a Breadth First Search algorithm over the graph representation of the combinational block netlist.



Untestable Faults Identification method

- Topology analysis:
  - Identification of fixed element (a)
  - Extracting cone (b)
- Logic simulation (c) (different data sets)
- Toggle activity (d) (different data sets)
- Primary input (sequential element) classification (e, f):
  - FIXED (F)
  - POTENTIALLY NOT-FIXED (PNF)
  - NOT-FIXED (NF)
- Circuit characterization (g)
- On-line untestable faults identification from initial faults list (h)



Untestable on openMSP430

Module	# Faults	UT Faults	Arithmetic	Matrix Mult	Quicksort	CoreMark
clock_module	2,180	86	37.11%	37.11%	37.11%	37.11%
debug	8,340	206	65.56%	65.56%	65.56%	65.56%
execution_unit	18,434	3000	21.79%	18.91%	17.40%	18.61%
frontend	6,268	190	14.16%	14.25%	19.16%	14.15%
mem_backbone	3,512	78	7.03%	13.72%	7.06%	13.72%
multiplier	9,936	130	5.12%	5.12%	5.12%	5.12%
sfr	602	34	14.78%	14.78%	14.78%	14.78%
watchdog	1,568	76	21.11%	21.11%	22.07%	21.30%
glue logic	904	0	14.38%	14.38%	14.38%	14.38%
CPU	51,744	1,100	24.13%	23.57%	30.56%	23.47%

Q&A

[redacted]

Stress Components - Switching Activity

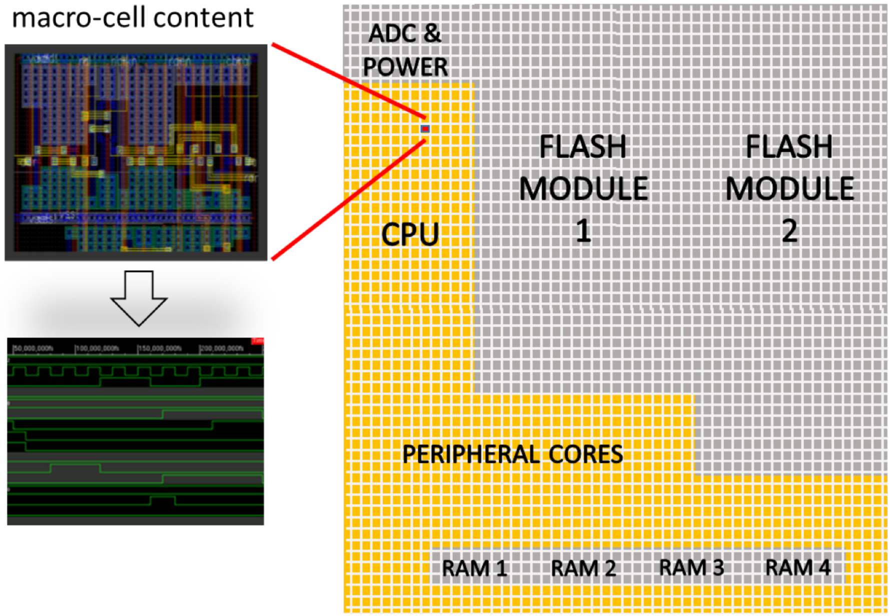
Macro Switching Weighted Fanout (MSWF)

$$MSWF_k = \frac{1}{N}\sum_{i=1}^{N} FO_i^k \cdot SW_i^k$$

$$S^{Max} = Max(MSWF_i)$$

$$S^{Mean} = \frac{1}{N}\sum_{i=1}^{N} MSWF_i$$

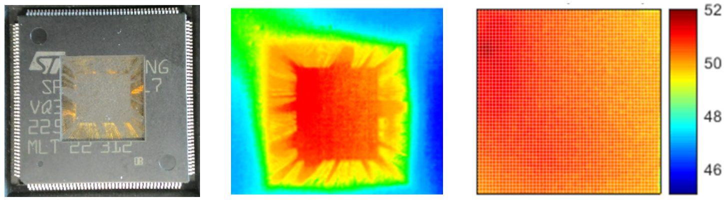
$$S^{std-dev} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (MSWF_i - \mu)^2}$$



t_{eval} ≥ Lowest Common Multiple of the duration of the stress procedures

Stress Components - Temperature Distribution

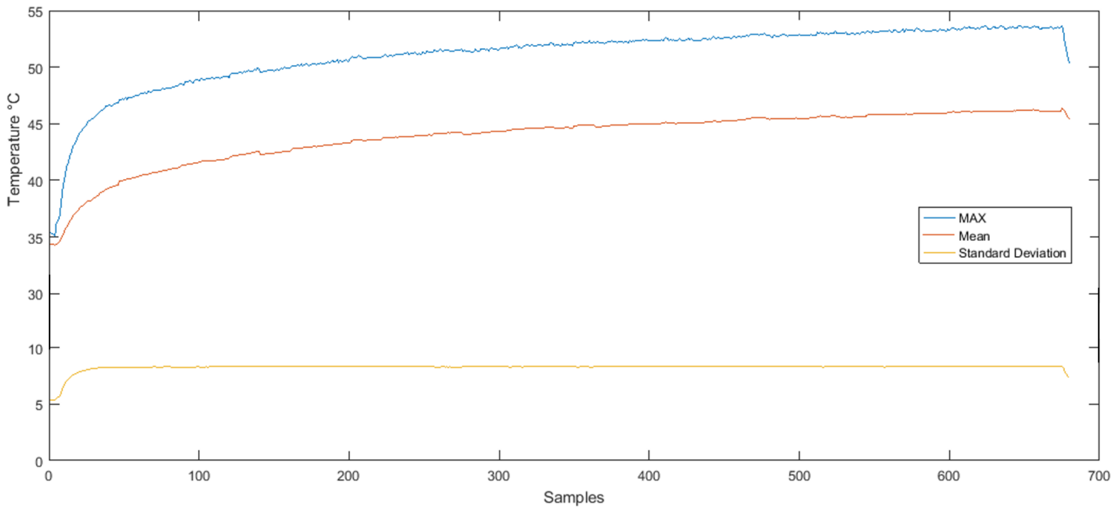
Temperature Matrix $[L \times K]$ with Cell $C_{i,j}$



$$S^{Max} = Max(C_{i,j})$$

$$S^{Mean} = \frac{1}{LK}\sum_{i=1}^{K}\sum_{j=1}^{L} C_{i,j}$$

$$S^{std-dev} = \sqrt{\frac{1}{LK}\sum_{i=1}^{K}\sum_{j=1}^{L}(C_{i,j}-\mu)^2}$$



t_{eval} ≥ temperature stability

Stress Components - Current Consumption

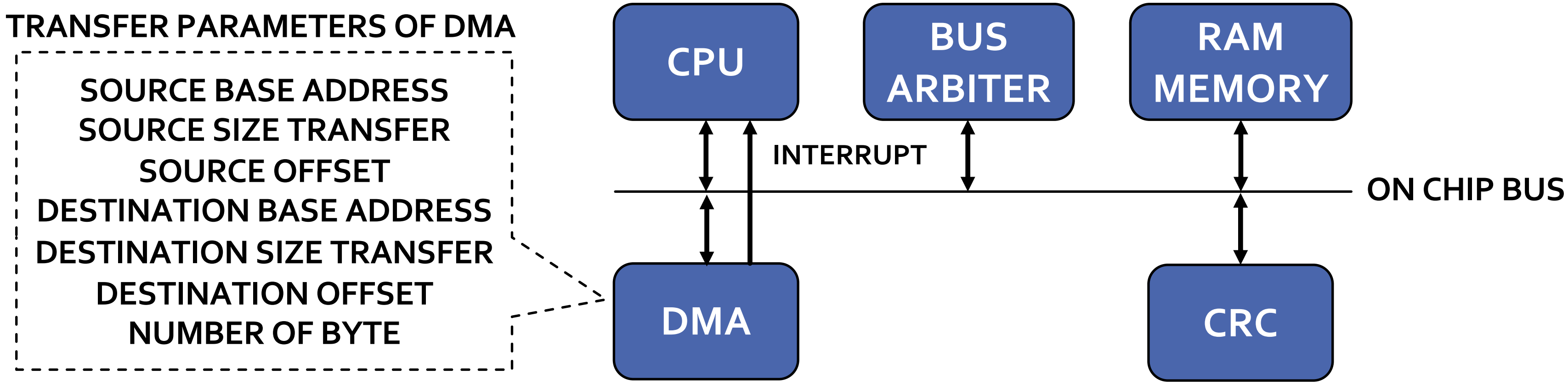
A single point measure of the current during the execution of the stress pattern at package level on the most suitable pin-out grouping with the higher sample rate.

t_{eval} ≥ temperature stability

Fault Classes

Tuning Parameters	Fault class A	Fault class B	Fault class C
α_{sw}	0.7	0.3	0.5
β_{sw}	0.3	0.7	0.5
α_{temp}	0.7	0.3	0.5
β_{temp}	0.3	0.7	0.5
ω	0.2	0.7	0.1
τ	0.6	0.1	0.1
θ	0.2	0.2	0.8

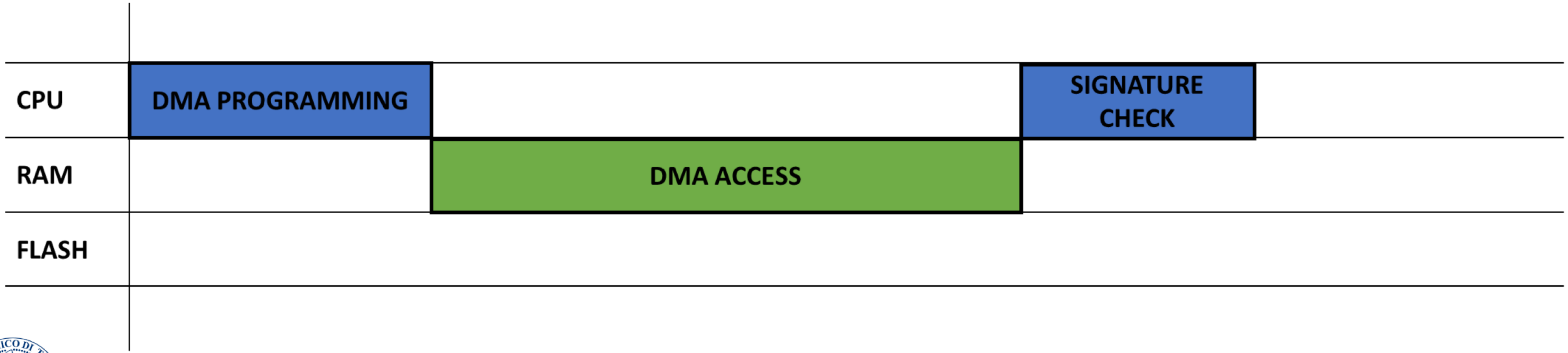
DMA-based BIST



March element	SOURCE			DESTINATION		
	Base Add.	**Size Tx.**	**Offset**	**Base Add.**	**Size Tx.**	**Offset**
$\Uparrow Rx$	Target Add.	Target Size	Target Size	Comp Add.	Comp Size	Zero
$\Downarrow Rx$	Target Add.	Target Size	- Target Size	Comp Add.	Comp Size	Zero
$\Uparrow Wx$	Pattern Add.	Pattern Size	Zero	Target Add.	Target Size	Target Size
$\Downarrow Wx$	Pattern Add.	Pattern Size	Zero	Target Add.	Target Size	- Target Size

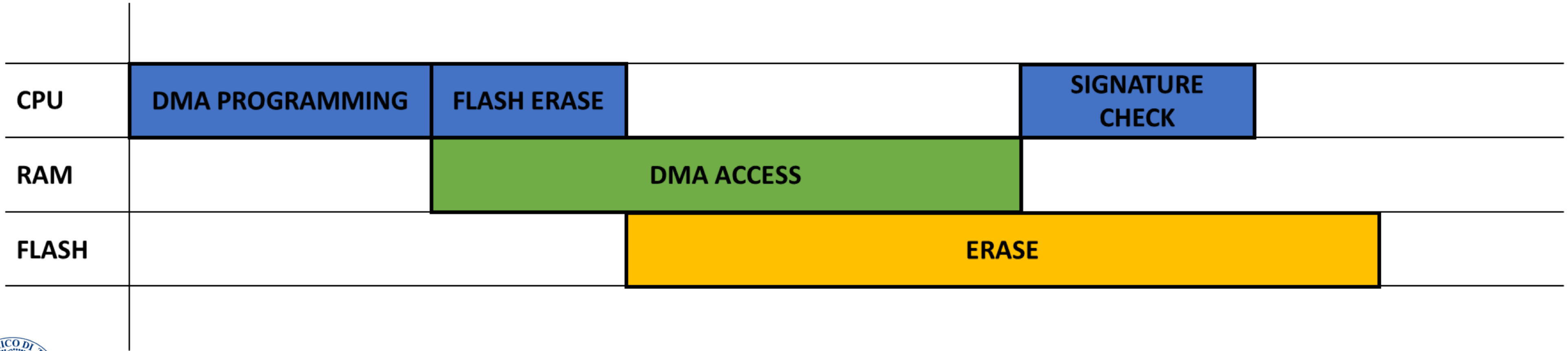
Bus Access

- DMA-based BIST
  1. DMA programming
  2. DMA Test
  3. Signature check
     (i.e., CRC value compared with a precalculated immediate value)



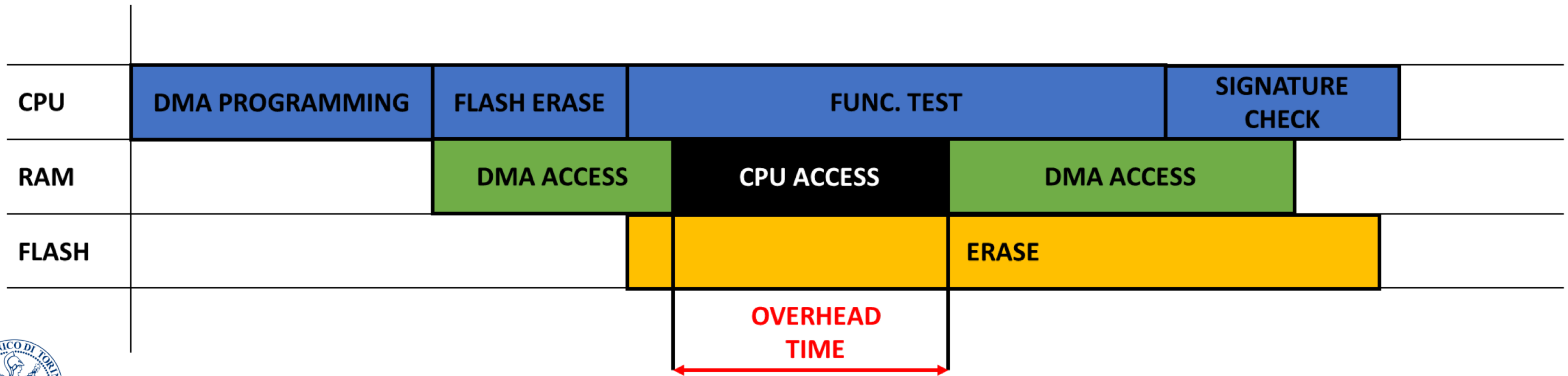
Bus Access

- Parallelization of FLASH ERASE does not impact
  1. FLASH ERASE launch
  2. ERASE is independent (No use of Bus)



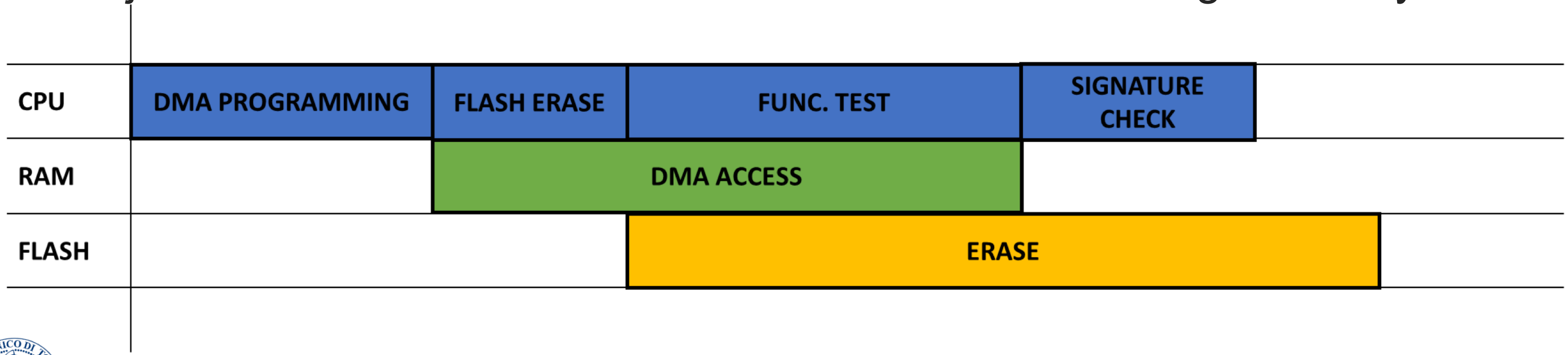
Bus Access

- Parallelization of Functional
  1. Everytime a data need to be fetched from RAM, the DMA access is suspended
  2. If stress/test Functional programs are executed from RAM, the DMA access risks to be continously interrupted



Bus Access

- With CACHE memories
  1. Stress programs are generated in such a way that they can be executed from instruction CACHE once they have been fetched from memory
  2. Data for stress application are generated in such a way that they are just loaded to data CACHE and used without accessing RAM anymore.

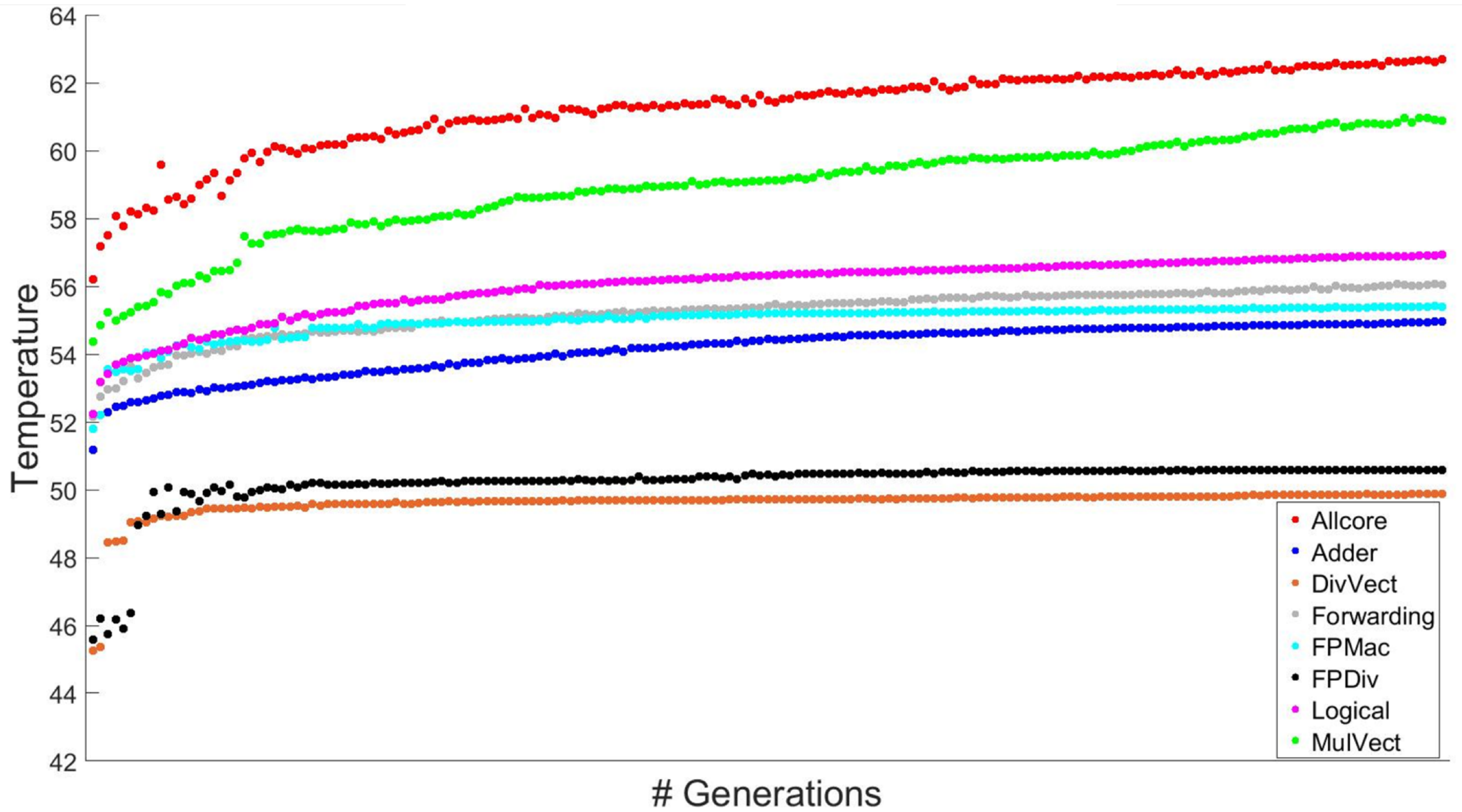


Cache Advantage

- For the sake of completeness, Stress program duration (Clock Cycles) with cache enable and cache disabled scenarios for parallel stress of RAM/FLASH/CPU has been analyzed.

Case of study	**Clock Cycles**
Cache disabled parallel stress	79,988
Cache enabled parallel stress	28,793

Max Temperature Evolution

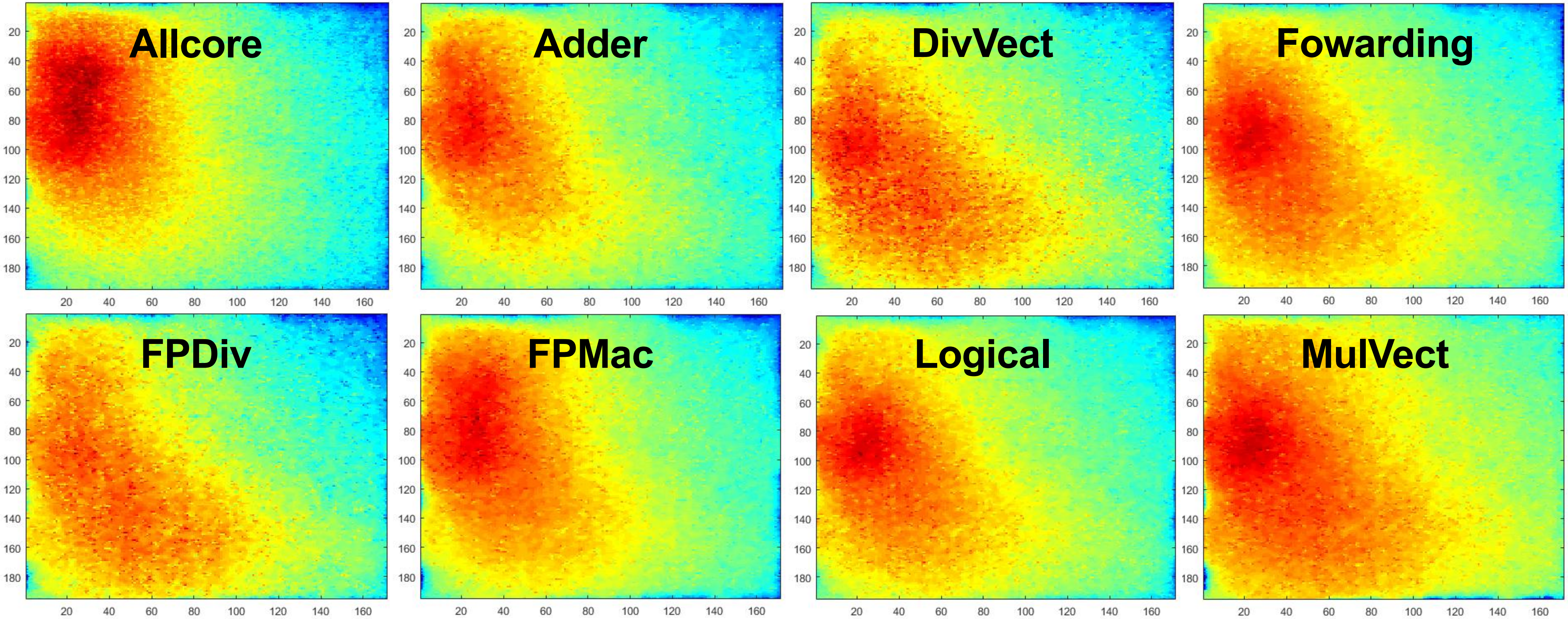


Name Program	Final Max Temperature [°C]
Allcore	62.68
MulVect	61.21
Logical	57.01
Forwarding	56.15
FPMac	55.50
Adder	55.05
FPDiv	50.16
DivVect	49.90

Stress Evolution

Name Program	Final Max Temperature [°C]	Final Average Temperature [°C]	Variance Temperature [°C]	Toggle activity/ Clock	Required Generation Time
Allcore	62.68	61.10	1.54	0.0170	06d 17h 34m 59s
Adder	55.05	54.67	2.82	0.0106	24d 18h 11m 08s
DivVect	49.90	49.71	5.51	0.0075	05d 18h 05m 30s
Forwarding	56.15	56.10	2.08	0.0121	01d 08h 37m 39s
FPDiv	50.16	49.90	4.93	0.0073	04d 18h 37m 47s
FPMac	55.50	54.50	3.50	0.0106	12d 22h 33m 21s
Logical	57.01	56.63	2.13	0.0117	29d 01h 32m 42s
MulVect	61.21	60.38	1.69	0.0135	28d 23h 50m 18s

Temperature Maps



Burn-In Data-Log Capability

Data-log Fields

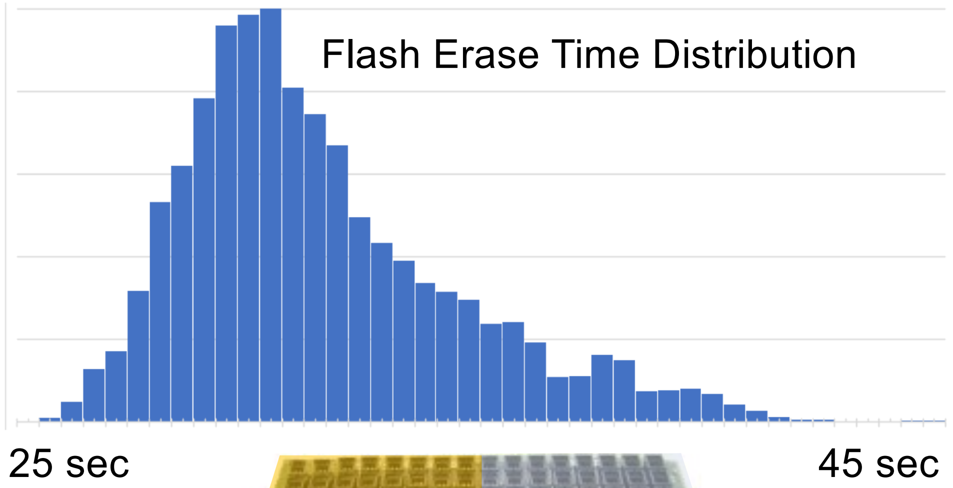
- Seal: BI flow at least once
- Test Fail Flag: failed (a functional tests) at least once
- Failing Test Signature: wrong signature of a failing test
- Individual Test Count: number of successful test executions
- Global Erase Count: counter of performed erases
- Global Test count: count of performed tests
- Communication Fail Flag: DUT-ATE disconnection occurred

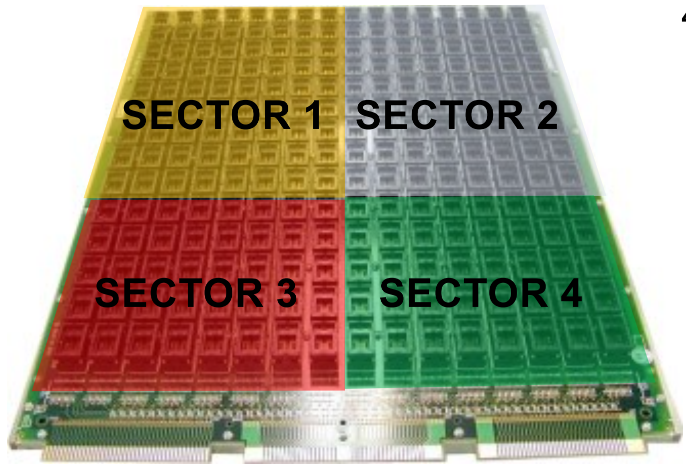
Data-log Analysis

- Asserted Communication Fail Flag:
  - Asserted Test Fail Flag (Discarded)
  - Not-Asserted Test Fail Flag
    - Not-Correct Global Erase Count (Recycle)
    - Correct Global Erase Count (Good)
- Not-Asserted Communication Fail Flag:
  - Asserted Test Fail Flag (Discarded)
  - Not-Asserted Test Fail Flag, incongruent data-log (Recycle)

Flash Erase Time and Power Supply During Burn-In

- Variable flash erase cycling time:
  - Flash erase is Temperature dependent
- A proper management of flash erase time saves time
  - Risk to run a stress procedure more than flash erase time
- BI equipment manages thousand of device in parallel
  - Power supply limitation
- A proper management of power supply saves TDBI time





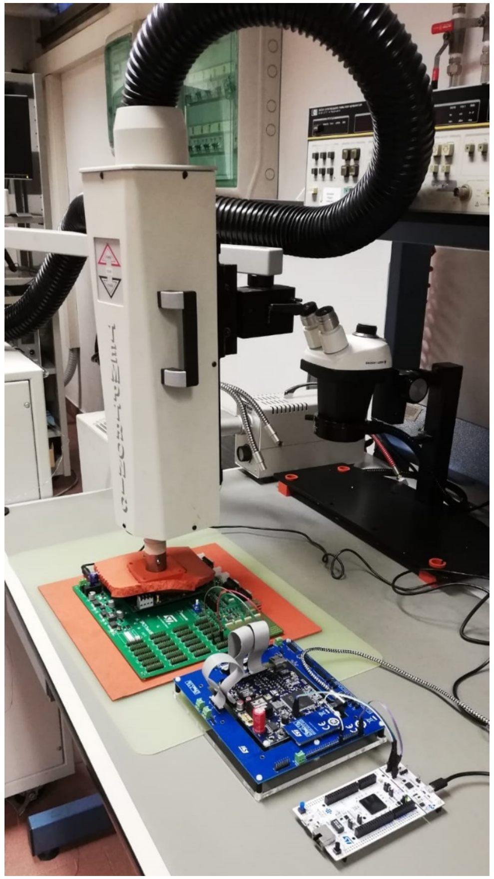
Test Program Characterization

Stress Program	Current consumption [A]	Duration [ms]	Target
Allcore	1.195	489	The whole core
Adder	0.850	308	Integer Adder unit
DivVect	0.785	412	Integer Divider Unit
Fowarding	1.030	304	Forwarding Unit
FPUDiv	0.935	327	Floating Point Divider
FPUMac	1.015	376	Floating Point Multiplier
AllFPU	1.020	329	Floating Point unit
MulVect	1.170	333	Integer Multiplier
Logical	1.175	199	Integer Logic Unit
DMA	1.000	400	Direct memory Access

Sector vs On-Line Scheduling Parameters

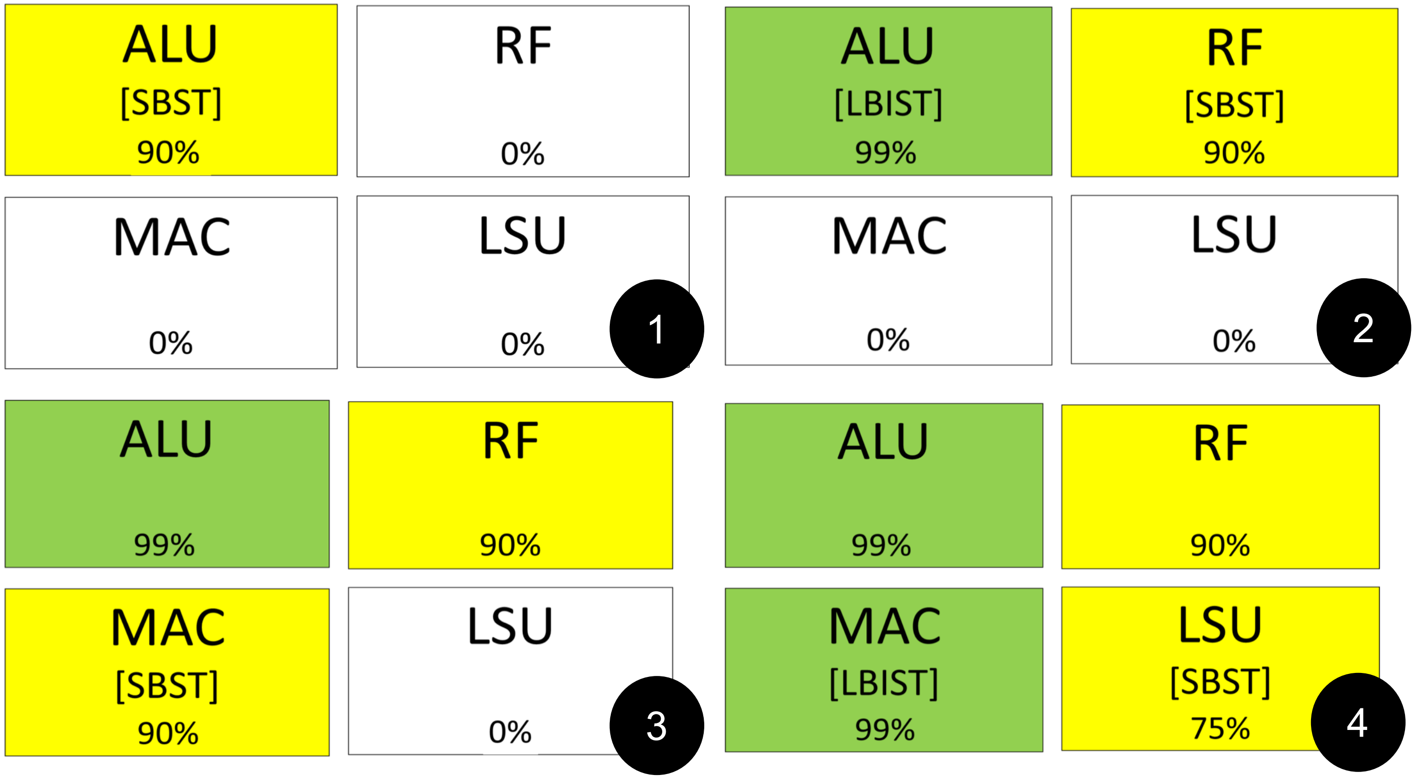
Stress Program	Wait Time [h]		Turnaround Time[h]		Response Time [h]	
	Sector Policy	On-Line Policy	Sector Policy	On-Line Policy	Sector Policy	On-Line Policy
Allcore	6.09	0.56	9.99	1.28	2.45	0.24
Adder	5.99	8.35	9.87	8.78	2.36	7.78
Fowarding	6	9.12	9.87	9.52	2.34	8.77
FPUDiv	6.01	7.53	9.89	7.96	2.39	7.09
FPUMac	6.04	4.74	9.91	6.21	2.4	5.16
AllFPU	6.05	5.68	9.91	5.21	2.39	4.18
MulVect	6.07	2.75	9.95	3.39	2.43	2.32
Logical	6.07	1.66	9.94	2.33	2.41	1.27
DMA	6.08	6.62	9.98	7.09	2.42	6.18

Passive Burn-In



Hybrid Self-Test Scheduling

- SBST and LBIST on two different modules of the core pipeline

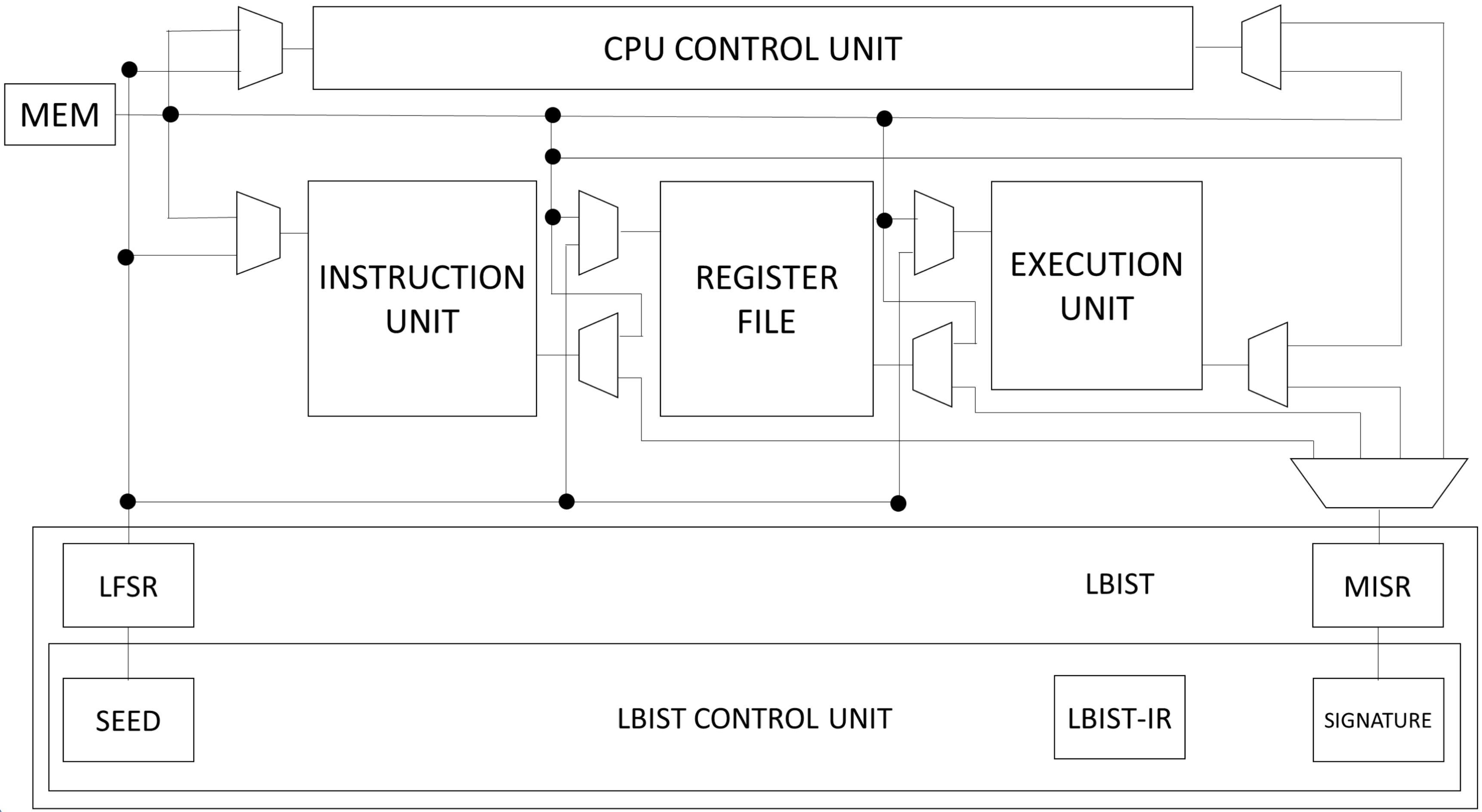


Hybrid Self-Test Wrapper

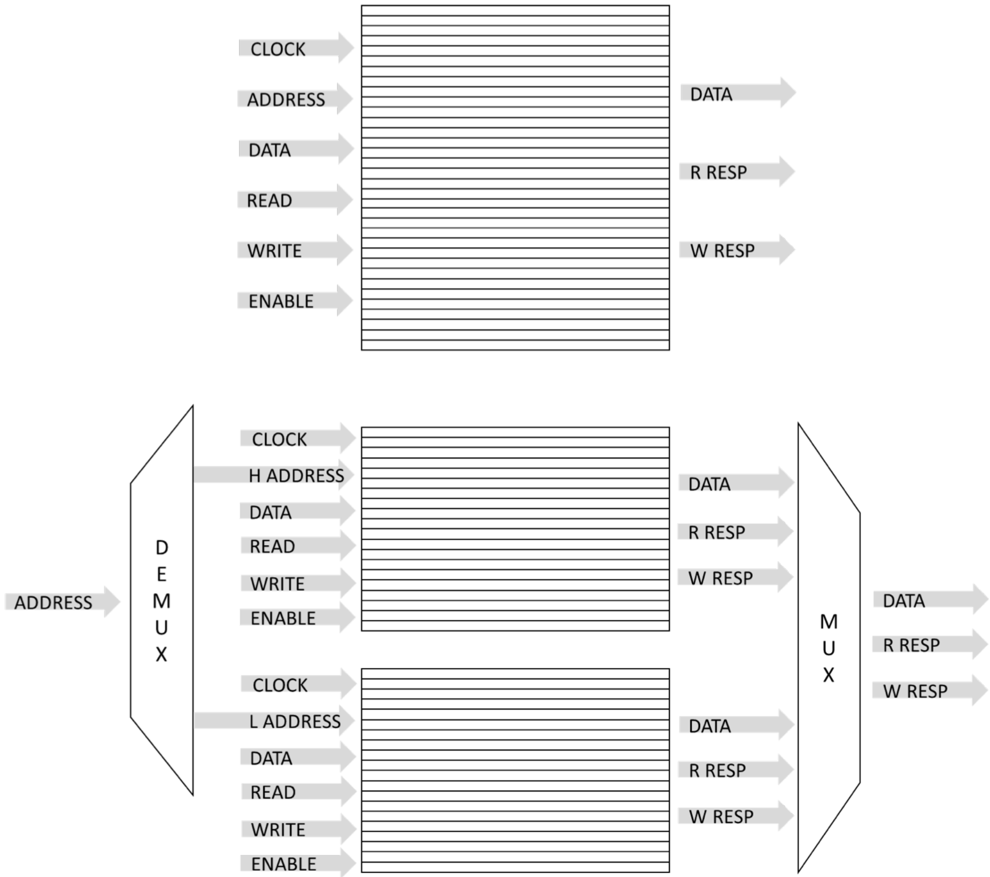
- The wrapper initializes, controls and runs the DfT by means of a SBST

```
SBST ROUTINE{
        LBIST.SEED ← VALUE OF THE SEED
        LBIST .LBIST-IR.LBIST_CU ← CUT SELECTION COMMAND
        LBIST.LBIST-IR.CUT_ID ← CODE OF THE CUT
        LBIST .LBIST-IR.LBIST_CU ← LBIST RUN COMMAND

        SBST CODE

        LBIST.LBIST-IR. LBIST_CU ← LBIST STOP COMMAND
        LBIST.LBIST-IR. LBIST_CU ← LBIST IDLE COMMAND
        LOAD THE VALUE FROM LBIST.SIGNATURE
        COMPARE LBIST.SIGNATURE WITH THE GOLD RESPONSE
}
```

Hybrid Self-Test Architecture

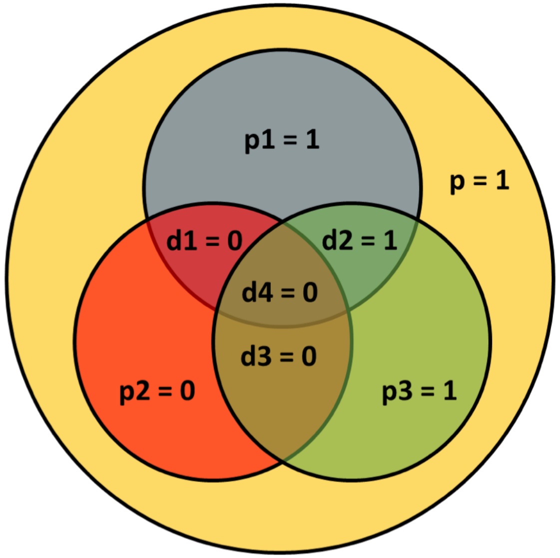


Hybrid Register File Architecture



SEC-DEC ECCC

SEC-DEC behavior



Bit String

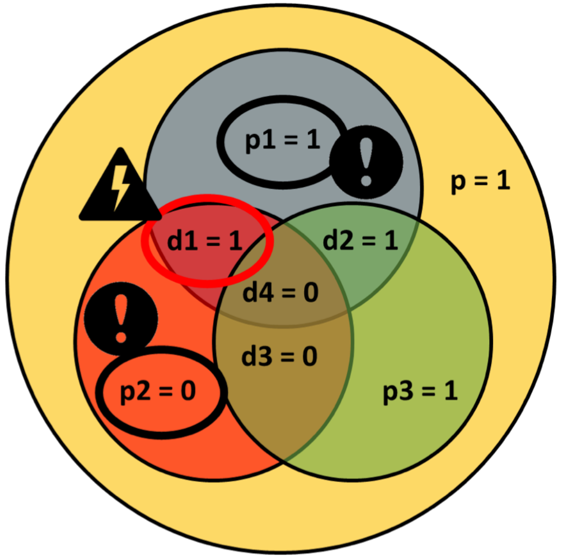
d1	d2	d3	d4	p1	p2	p3	p
0	1	0	0	1	0	1	1

Parity Bit Computation

p1 = d1 XOR d2 XOR d4
p2 = d1 XOR d3 XOR d4
p3 = d2 XOR d3 XOR d4

p = d1 XOR d2 XOR d3 XOR d4 XOR p1 XOR p2 XOR p3

SEC-DEC behavior with memory corruption



Bit String

d1	d2	d3	d4	p1	p2	p3	p
0	1	0	0	1	0	1	1

Parity Bit Computation

p1 = d1 XOR d2 XOR d4
p2 = d1 XOR d3 XOR d4
p3 = d2 XOR d3 XOR d4

p = d1 XOR d2 XOR d3 XOR d4 XOR p1 XOR p2 XOR p3

THANK YOU