Increasing System Reliability for Safety-Critical Applications

Ernesto Sánchez

CAD & Reliability Group

Goal

- To provide some highlights about how to effectively improve the reliability of safety critical applications by using on-line testing.

Outline

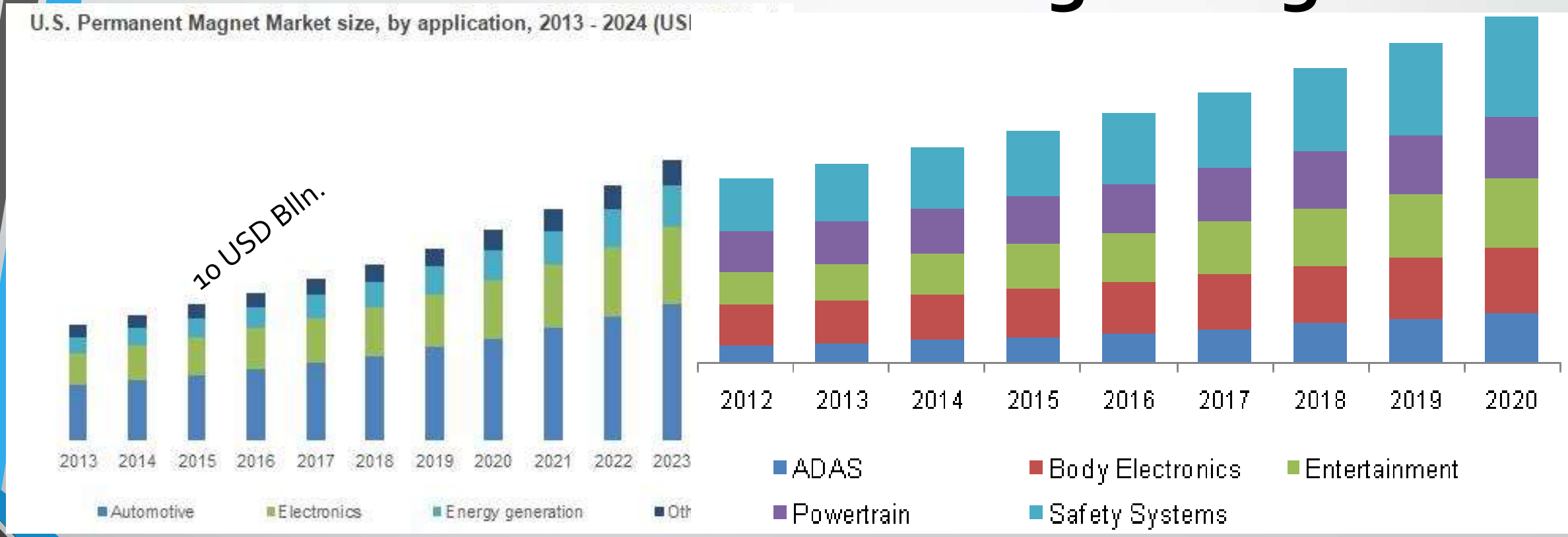
- Introduction & motivation
- On-line testing solutions for safety critical applications
  - Hardware-based solutions
  - Software-based solutions
    - Constraints for on-line generation
    - Development flow
    - Industrial case study
- Conclusions and future work.

Safety-critical system

A safety-critical system can be defined as system whose failure could result in loss of life, significant property damage, or damage to the environment.

- Nuclear plants
- Aircraft flight control
- Spacecraft
- Medical devices
- Automotive devices.

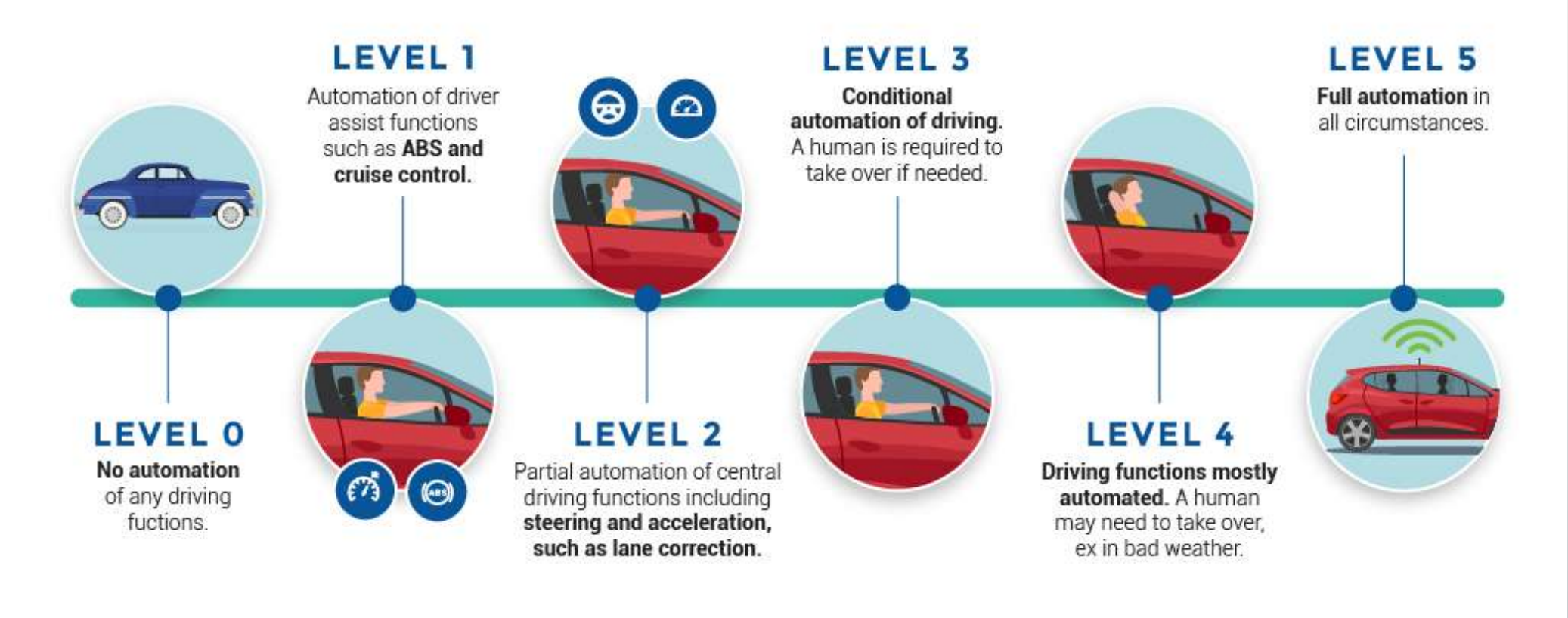
Automotive electronic growing



https://www.openpr.com/

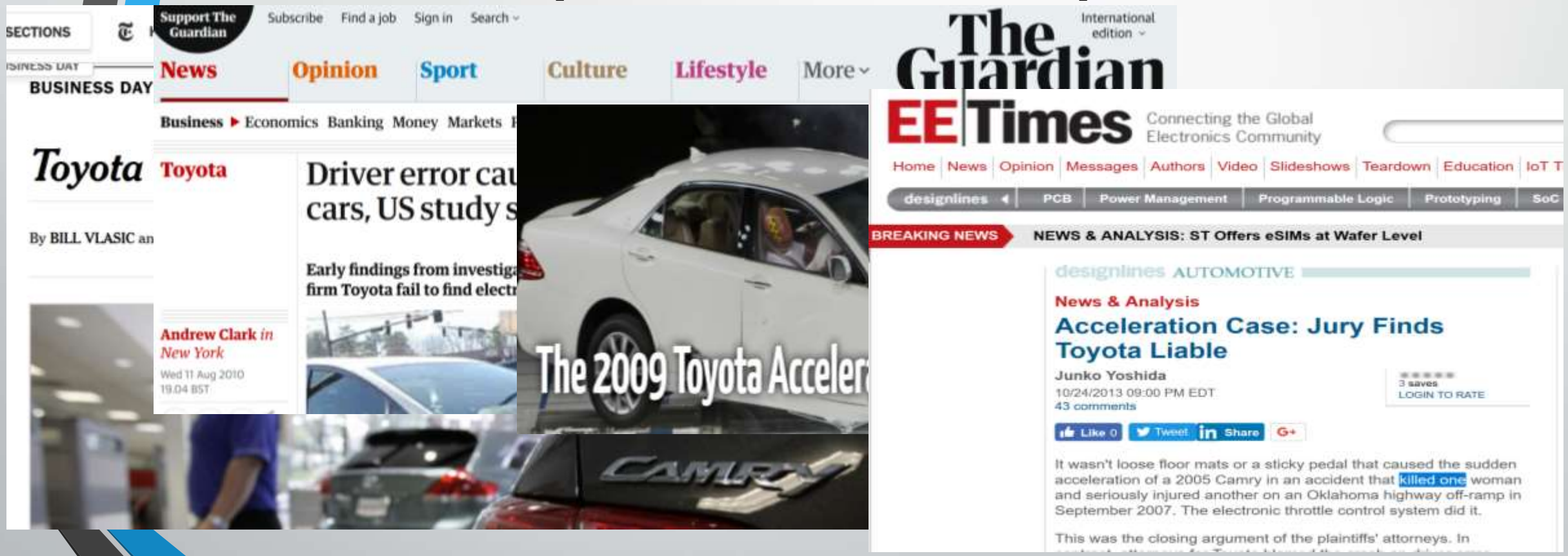
Automotive Electronics Market Analysis GVR

Level of autonomous driving technology



https://www.caa.ca/

Toyota accelerator system



ISO 26262
Automotive Functional Safety Standard

- Functional safety: absence of unreasonable risk due to hazards caused by malfunctioning behavior of electronic system.
  - **A**utomotive **S**afety **I**ntegrity **L**evel (**ASIL**)

Mirrors

Airbag

Steer-by-wire

	ASIL B	**ASIL C**	**ASIL D**
Stuck@ + Transition fault coverage	> 90%	> 97%	> 99%

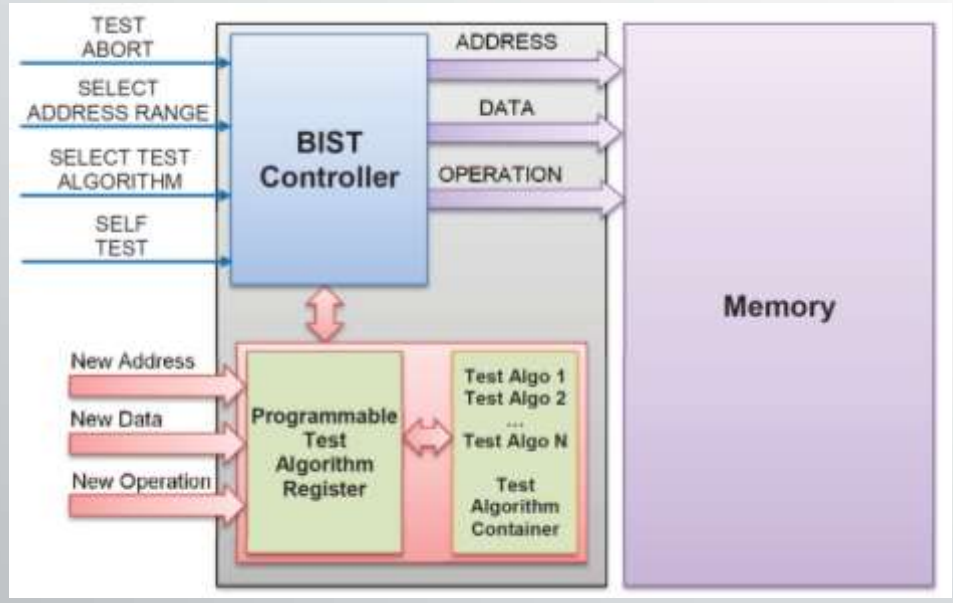
On-line testing techniques

It is required to periodically test the device complying with functional constraints.

- Hardware-based solutions
- Software-based or functional approaches.

Hardware-based solutions

- **Built-in self-test**

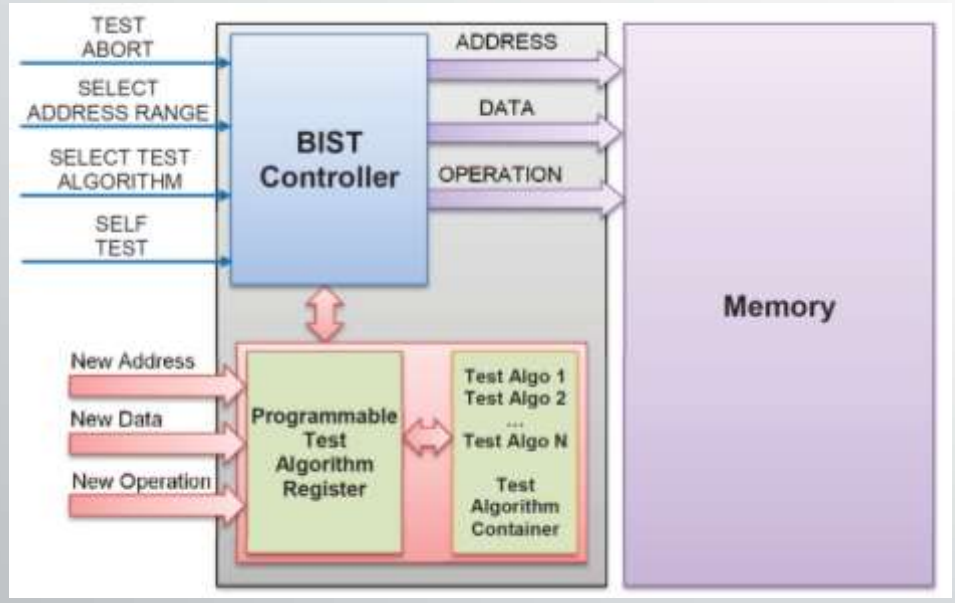


G. Tshagharyan et al. "An Effective Functional Safety Solution for Automotive Systems-on-Chip", IEEE ITC'17

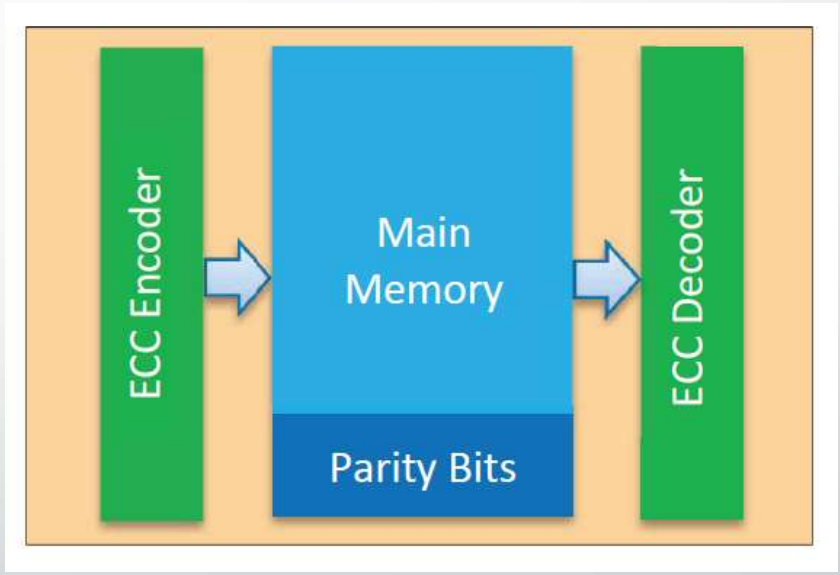
- Suitable for power-on and power-off
- Invasive and costly solution
- Better FC% if able to exploit the scan chains of the device.

Hardware-based solutions

- **Built-in self-test**



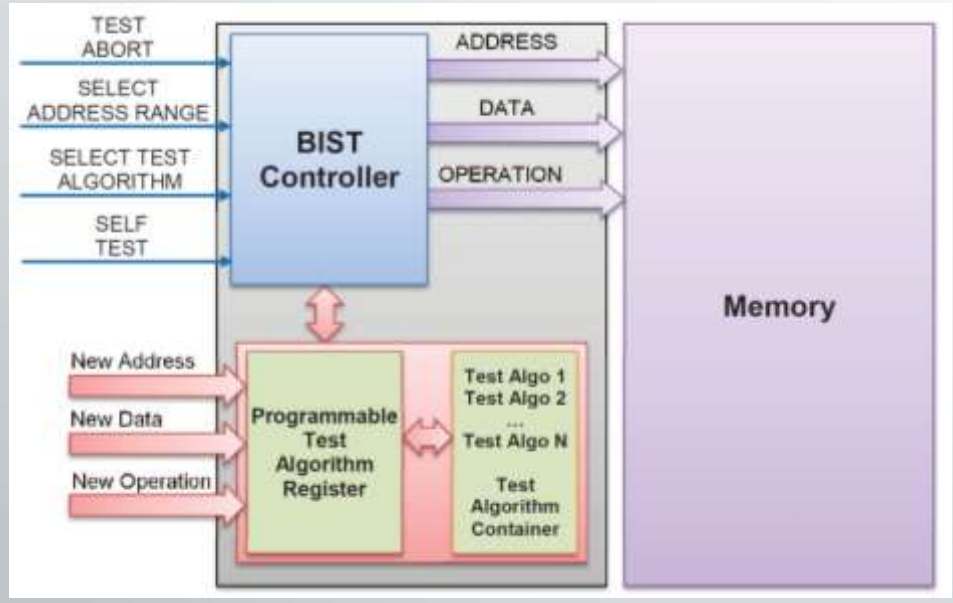
+



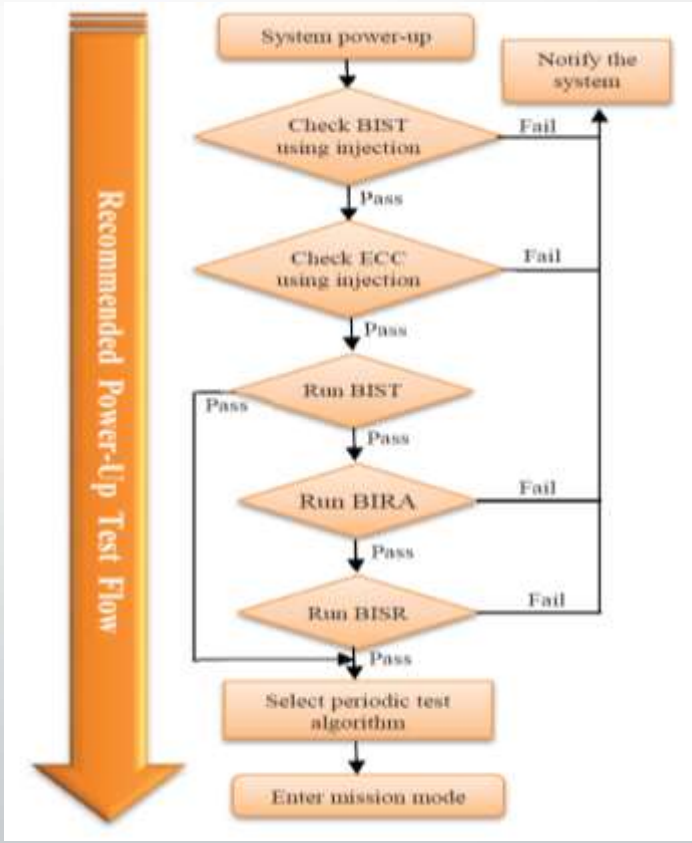
G. Tshagharyan et al. "An Effective Functional Safety Solution for Automotive Systems-on-Chip", IEEE ITC'17

Hardware-based solutions

- **Built-in self-test**

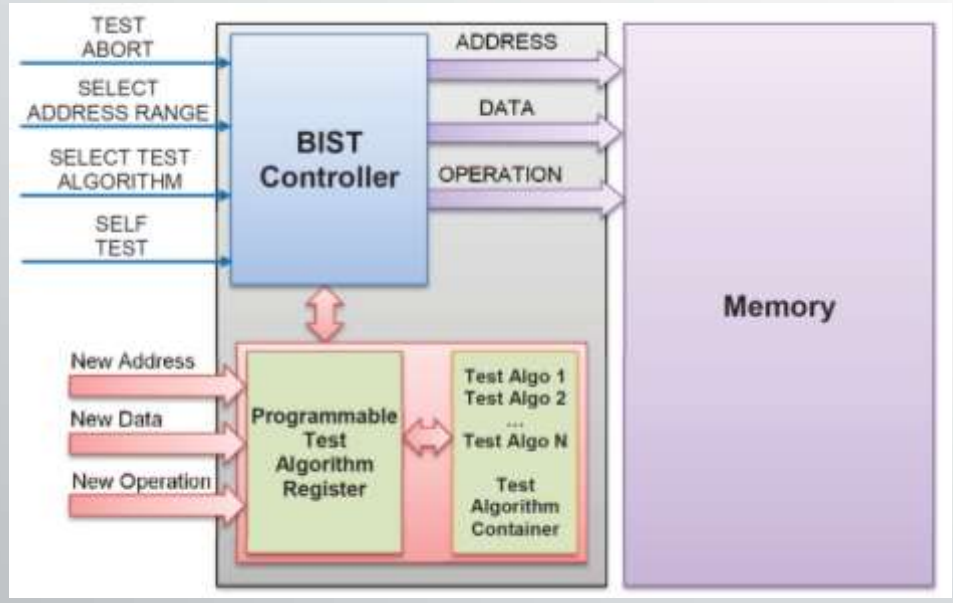


G. Tshagharyan et al. "An Effective Functional Safety Solution for Automotive Systems-on-Chip", IEEE ITC'17

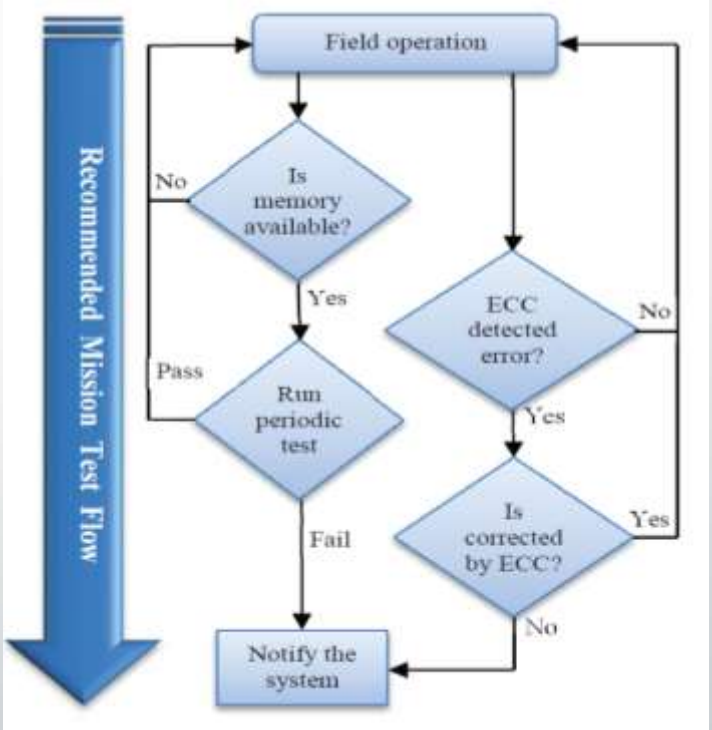


Hardware-based solutions

- **Built-in self-test**

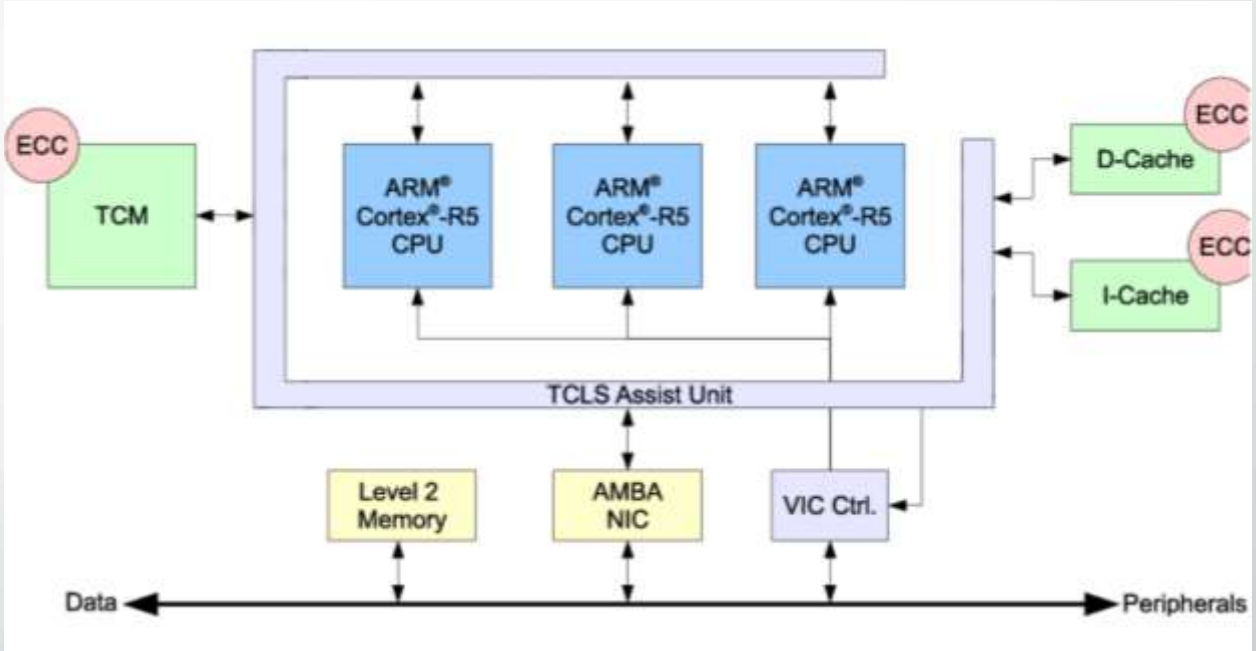


G. Tshagharyan et al. "An Effective Functional Safety Solution for Automotive Systems-on-Chip", IEEE ITC'17



Hardware-based solutions

- **Dual Core Lock-Step**
  - Detection capacity
- **Triple Core Lock-Step**
  - Detection + Correction
- Very efficient but costly solution.



X. Iturbe et al., "Addressing Functional Safety Challenges in Autonomous Vehicles with the Arm TCL S Architecture", IEEE D&T May/June 2018

Software-based or functional solutions

- Software-Based Self-Test (SBST)
  - A suitable ***test program*** is developed
  - The test program is stored in a memory accessible by the processor
  - The processor executes the test program
  - Results are gathered and compared with the expected ones.

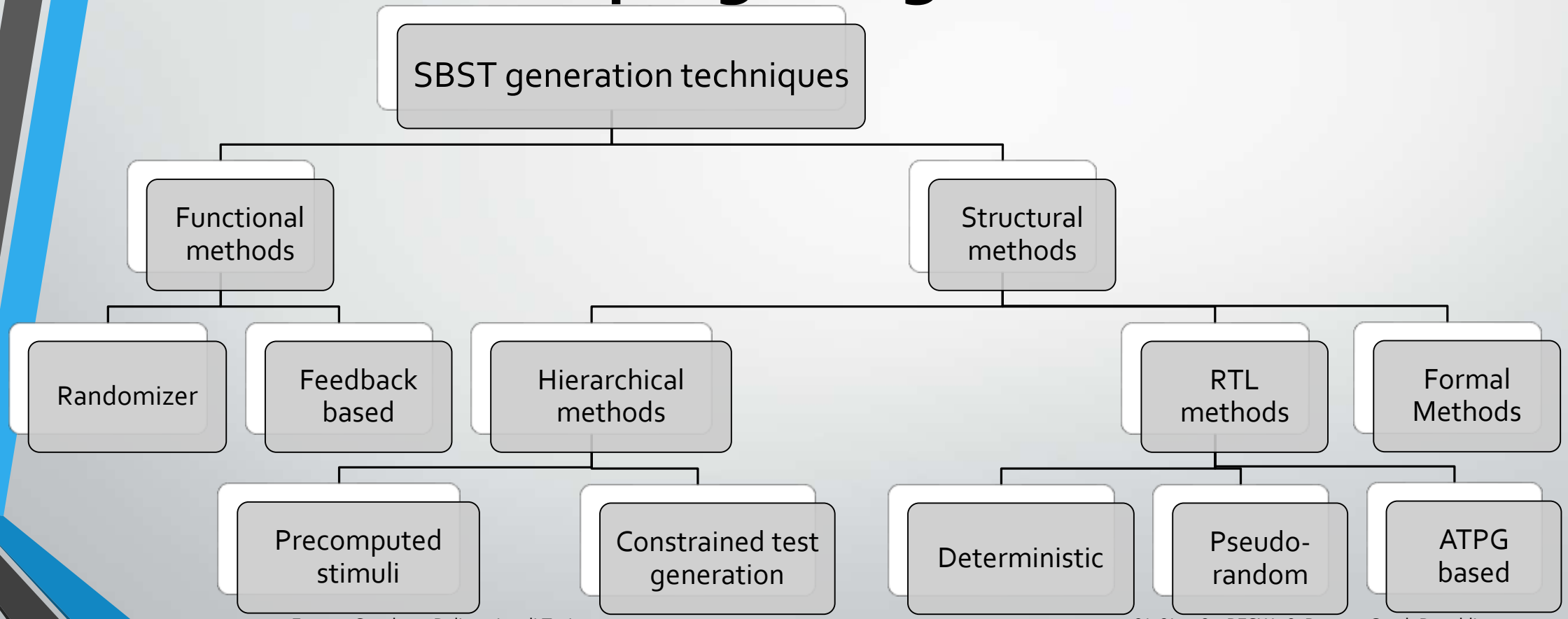
Software-Based Self-Test (sbst)

- Advantages
  - ✓ No additional HW requirements
  - ✓ No HW or performance overhead
  - ✓ At-speed testing
  - ✓ Suitable for on-line testing
  - ✓ Flexible regarding new requirements.

Software-Based Self-Test (sbst)

- Disadvantages
  - ✕ Suitable and compact test programs are required
  - ✕ No EDA tools are available
  - ✕ Manual generation can be very expensive
  - ✕ Grading test programs requires huge efforts.

Test program generation



Test program generation - 2

- ATPG-BASED

  The module under test is extracted and test patterns are generated, then converted to assembly instructions

  - FC%: High
  - Generation time: Low
  - Effort: Medium-low

Test program generation - 3

- Deterministic

  Implementation of documented algorithms targeting some modules

  - FC%: Medium
  - Generation time: Low
  - Effort: Low

Test program generation - 4

- Evolutionary-based or feedback-based

  A set of candidate test programs is improved through a loop-based approach that exploits basic concepts of natural evolution.

  - FC%: High
  - Generation time: High
  - Effort: Medium-high

Test program generation - 5

- Quality comparison
  - Shifter module in an industrial ALU
  - The module counts with 4,196 gates.

Technique	#lines	#clock cycles	Stuck-at FC%
ATPG-based	110	3,549	98.0
Deterministic	42	41,326	90.1
Evolutionary-based	164	1,651	92.9

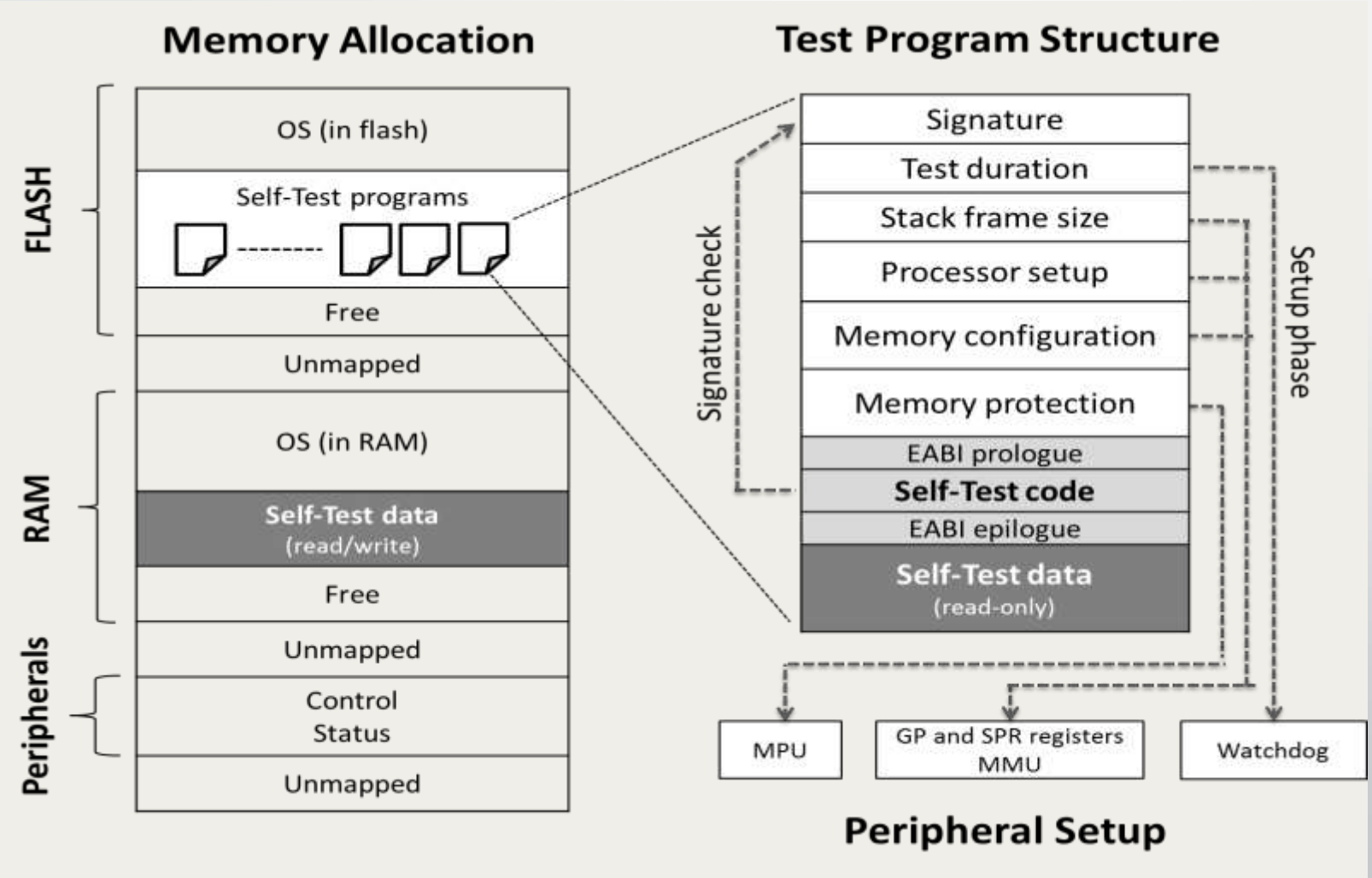
Test program generation - 6

Module	Determ.	ATPG	EVO	FC%	Technique
Combinational functional u.	+	++	+	91,88%	ATPG + evo
Sequential functional u.	++	+	++	93,88%	Det + evo
Muxes	++	-	+	78,03%	Det + evo
Register file	++	-	-	96,72%	Det
Control path	+	-	++	89,50%	Det + evo
Exceptions	++	--	-	72,48%	Det
BTB	++	--	-	72,67%	Det

Constraints for on-line SBST

- Time and memory constraints
- Coexistence with the final application and the O.S.
  - Test encapsulation and metadata
- Context switching
- Robust execution.

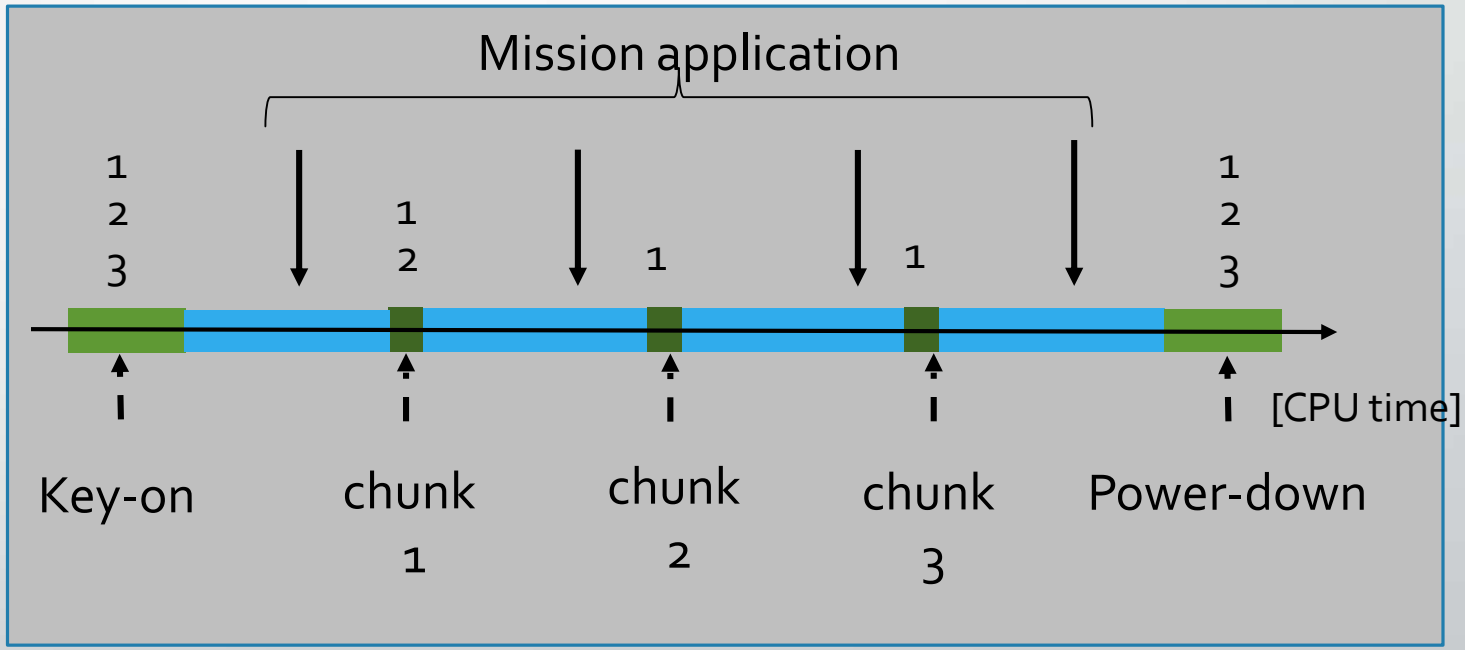
Test encapsulation and metadata



Context switching

Context switching is handled by the EABI interface.
SBST programs can be classified as:

1. run-time tests
   - can be interrupted
2. non-exceptive tests
   - manipulate especial purposes registers
3. critical tests
   - use internal interruptions and peripheral cores.

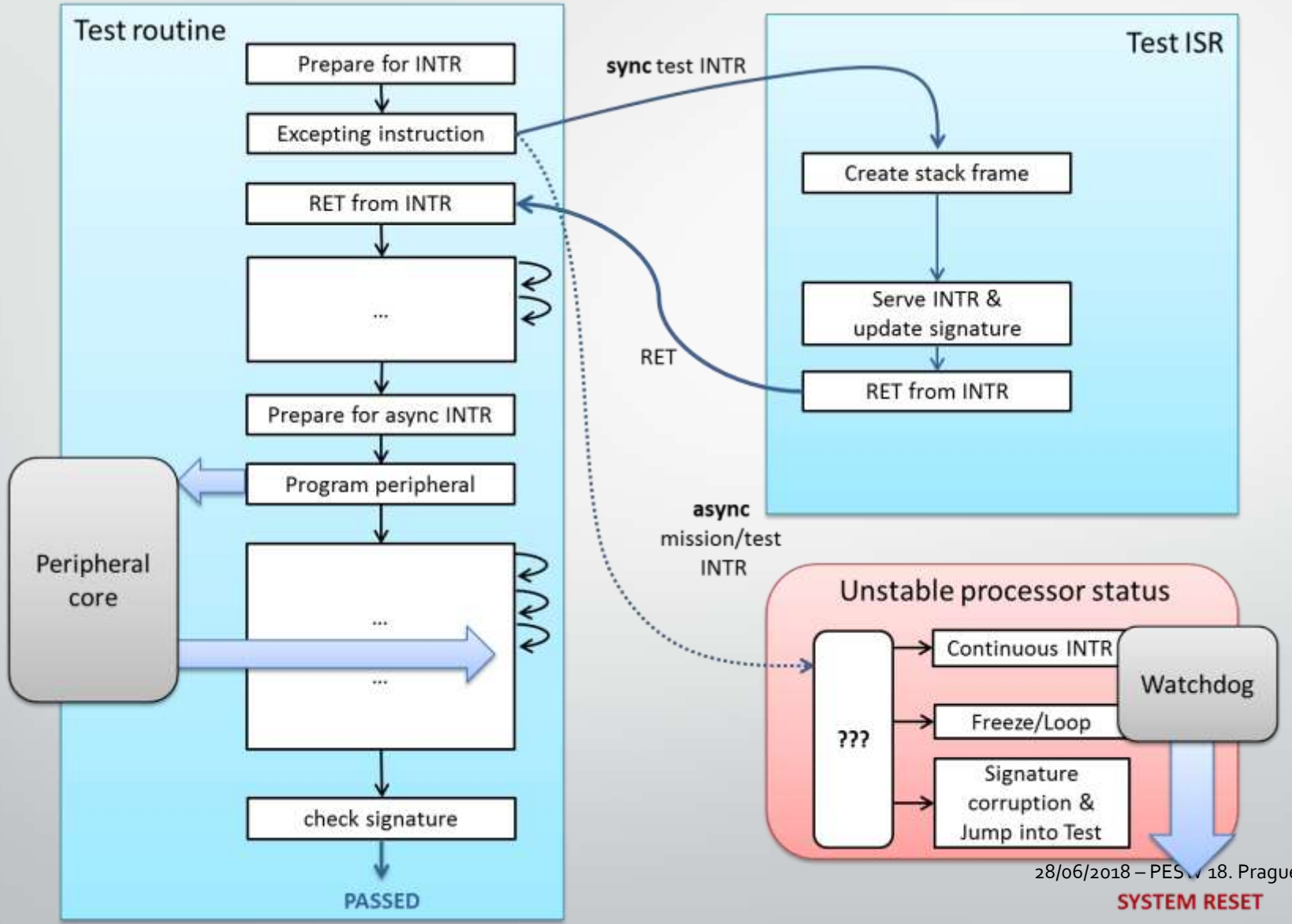


Robust execution

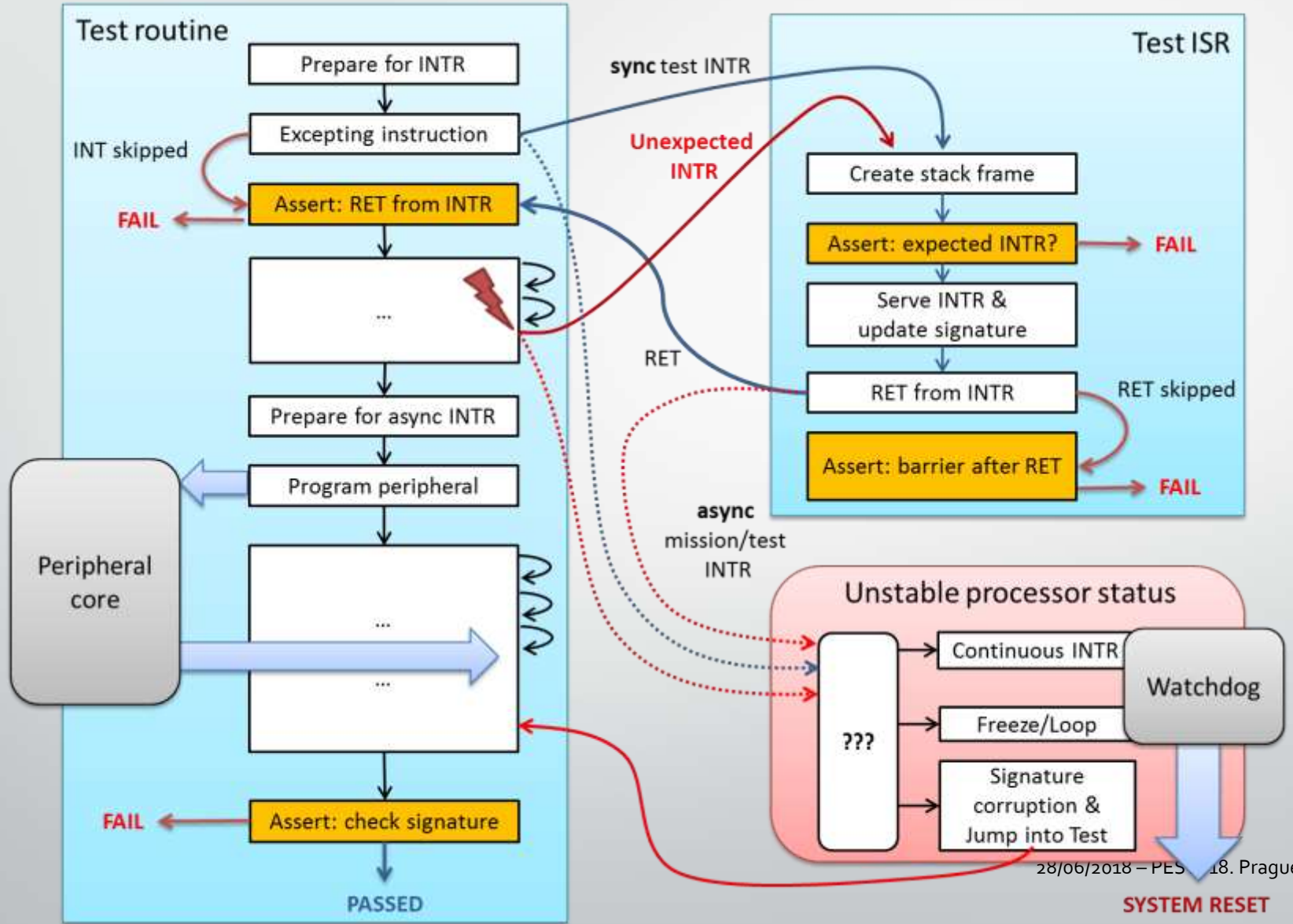
Three interruption types may arise:

- Intentionally provoked
- Unexpected
- Mission mode

Robust execution



Robust execution



SBST Development flow

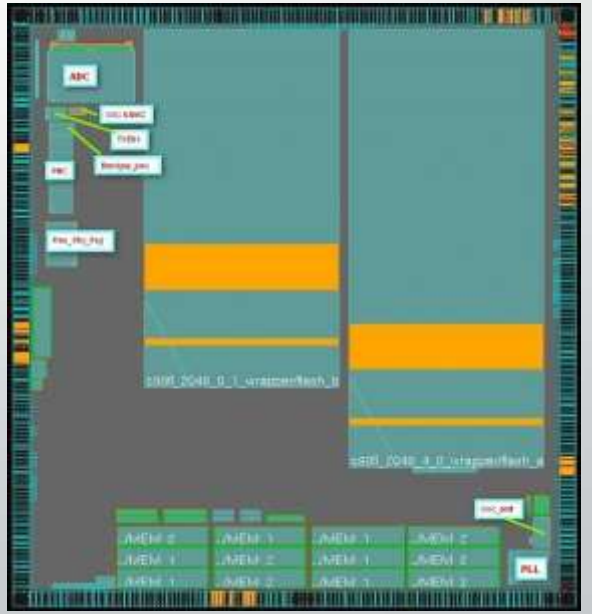
A general SBST development flow may exploit the following principles:

- Modularity
- Parallelization
- Positive side-effects.

An industrial case study

SoC employed in safety-critical automotive embedded systems, such as airbag, ABS, and EPS controllers. SoC is currently manufactured by STMicroelectronics.

- 32-bit pipelined microprocessor
- In-order dual issue
- Multiple functional units
- Branch prediction unit
- Multiport register file
- Power Architecture™.



An industrial case study

Final SBST set of programs:

- Max. execution time for a single run-time program: 512 clock cycles
- Max. FLASH mem: 256kB
- Max. DATA mem: 1kB
- 73 test programs
- Full execution time: 0.8ms (@150MHz).

An industrial case study

Sub-module	#faults	Single 1A FC [%]	Synchro 1A FC [%]	Single 1B FC [%]	Synchro 1B FC [%]	Single 2A FC [%]	Single 2B FC [%]	Synchro 2A+2B FC [%]	Single 3 FC [%]	Synchro 3 FC [%]
Functional Units	140k	**86,89**	89,21	--	--	--	--	89,51	--	91,78
Branch Units	72k	--	--	**75,07**	75,52	--	--	76,43	--	78,28
Register Bank	210k	--	70,19	--	--	**89,54**		93,53	--	95,38
Addressing modules	31k	--	--	--	66,29		**80,34**	81,59	--	82,33
Pipeline modules	278k	--	--	--	--	--	--	64,59	**79,91**	81,10
Glue logic	19k	--	--	--	--	--	--	--	--	63,36
TOTAL	**760k**	**--**	**36.07**	**--**	**9.74**	**--**	**--**	**76.87**	**--**	**87.23**

Conclusions and future works

In this presentation some of the most relevant issues regarding on-line testing of safety-critical applications were described.

Hardware and software-based solutions were introduced highlighting the possibilities of SBST based ones trough an industrial case of study was also provided.

Is there space for hybrid solutions, combining hardware and software advantages?

Thank you