2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

VIRTUAL | MARCH 1-4, 2021

Verification of Functional Safety for an Automotive AI Processor

Mihajlo Katona PhD, Veriest Solutions

Veriest

www.VeriestS.com

accellera SYSTEMS INITIATIVE™

Veriest Solutions introduction

- ASIC Engineering services company, founded in 2007
- Headquartered in Israel, 4 additional sites in Europe
- 120+ engineers
- Customers in US, Europe and Israel:
  - Tier1 international Semi companies
  - Start-up companies
  - IP& EDA companies
  - System companies

Some of our professional services

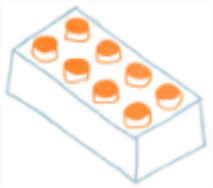
Front-end design

Functional Verification

Formal Verification

Firmware/ Embedded SW

IP/VIP Design



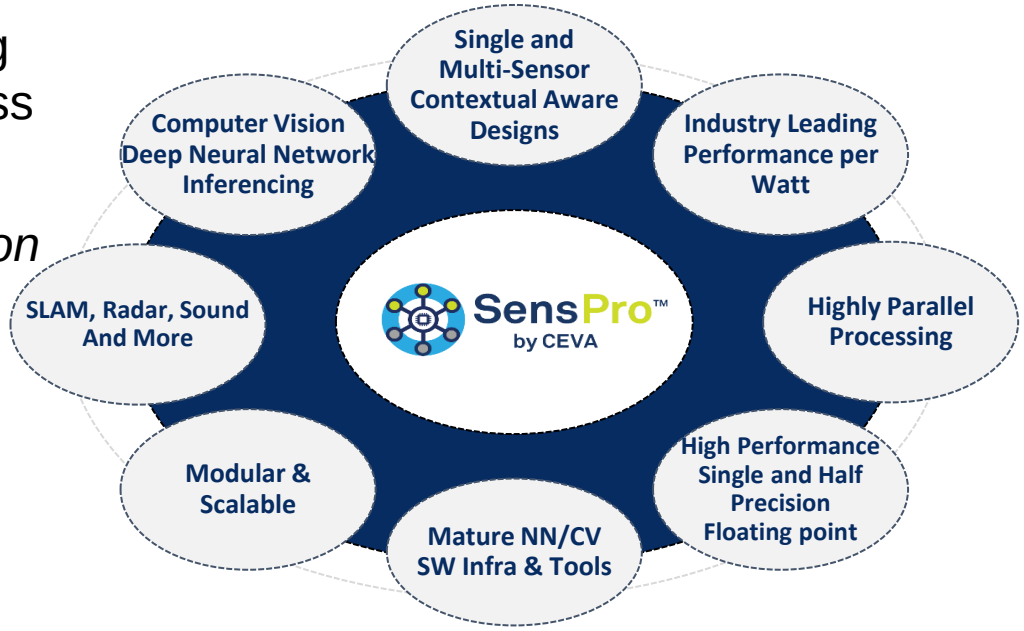
FPGA Design

CEVA DSP IP architecture

The methodology described in this workshop was developed during projects to verify different IPs at **CEVA**, a leading licensor of wireless connectivity and smart sensing technologies.

- *NeuPro - AI Processor Architecture for Imaging & Computer Vision*
- *SensorPro - High Performance Sensor Hub DSP Architecture*

- I'd like to thank ***Mr. Noam Meser***,
  Director of VLSI Verification & Infrastructure at CEVA,
  for his contributions to this presentation



SensPro – Industry's First High Performance Sensor Hub DSP

CEVA IP Technologies

Cellular

5G DSP-based platforms for smartphones, RAN and cellular IoT

Camera

Vision DSPs
DNN accelerators
CDNN compiler for NN inferencing

IoT Connectivity

Comprehensive platforms for Bluetooth and Wi-Fi

IMU

Software & algorithms
Sensor fusion & management
Activity classifiers

Microphone

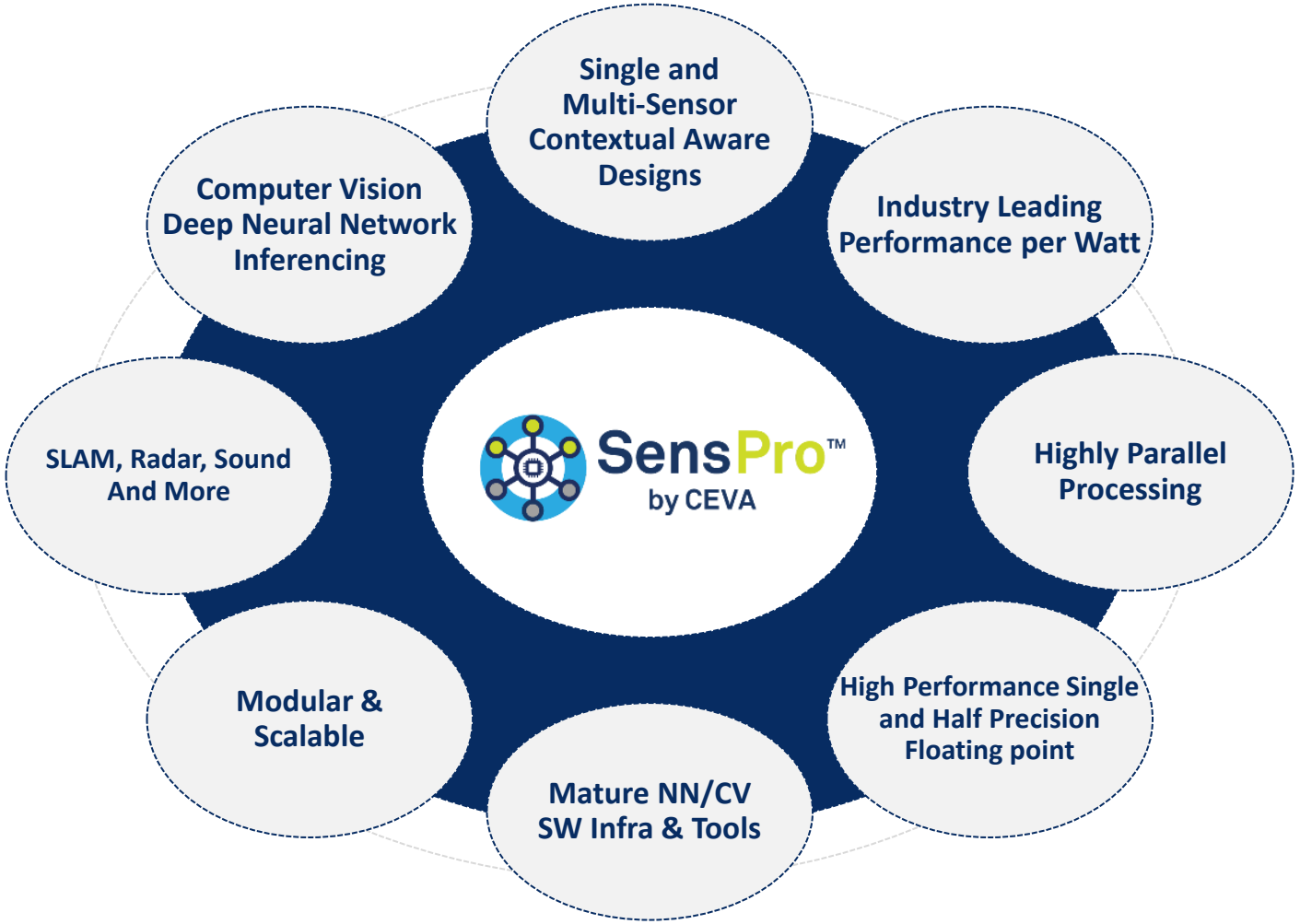
Audio DSPs, SW technologies, speech recognition, noise reduction, audio AI

Wireless Connectivity

Smart Sensing

SensPro – Industry's First High Performance Sensor Hub DSP

SensPro™ by CEVA



20% Frequency Increase*

Up to x8 Performance Improvement*

30% Energy Savings*

30% Code Size Savings*

Agenda

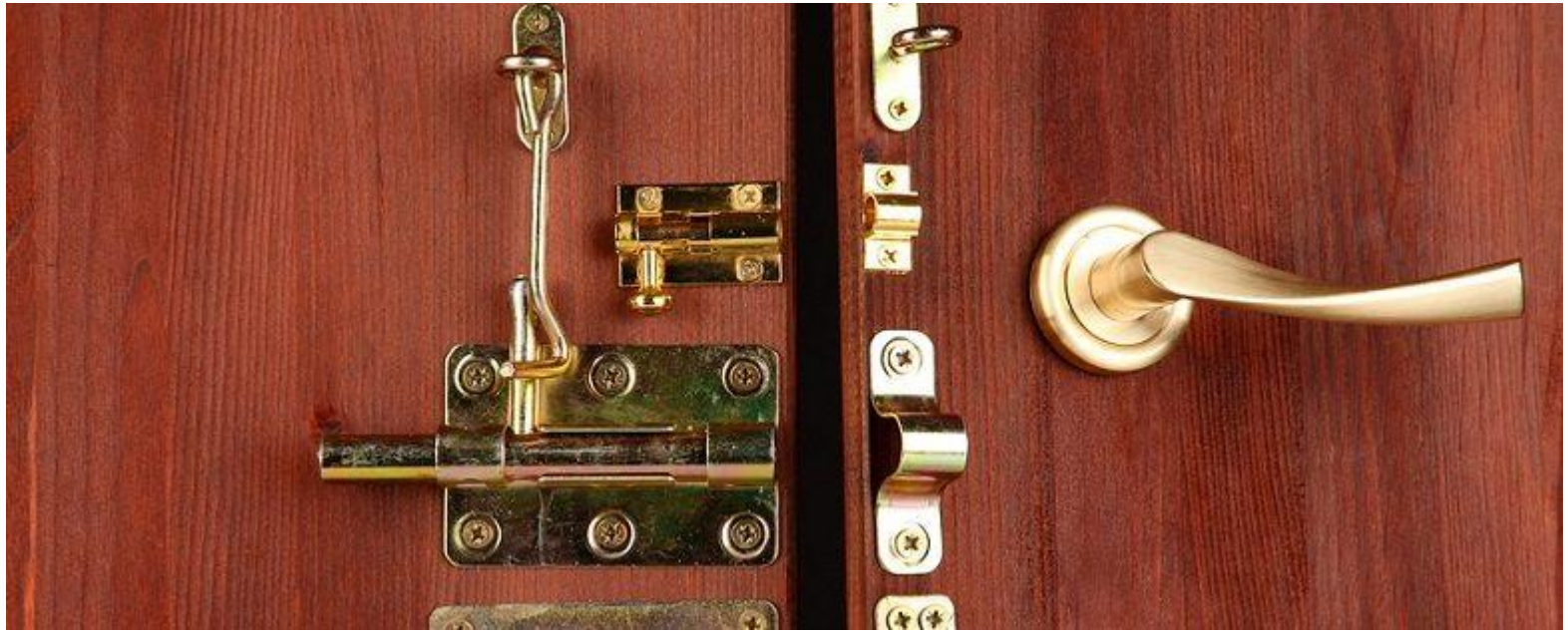
- Functional Safety Introduction
  - Standardization
  - Error Correction Schemes
- Functional Verification and Functional Safety Verification Challenge
- Verification Methodology for Functional Safety Verification
  - Tools
  - Procedures

Agenda

- Functional Safety Introduction
  - Standardization
  - Error Correction Schemes
- Functional Verification and Functional Safety Verification Challenge
- Verification Methodology for Functional Safety Verification
  - Tools
  - Procedures

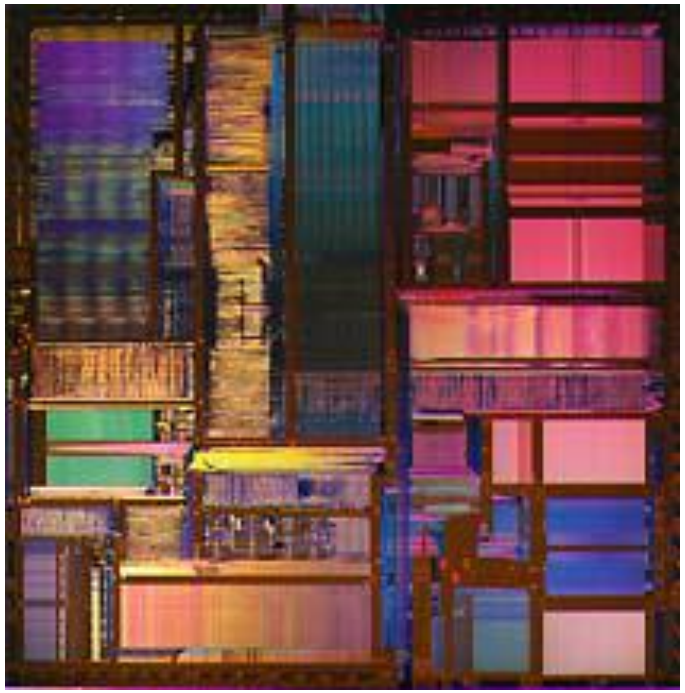
What is Functional Safety ?

It is about safe machinery without causing any risk to human life



Passive safety system

What is Functional Safety ?



- Functional safety covers an ***active system*** that has safety mechanisms in place.
- These mechanisms are activities or technical solutions to ***detect***, ***avoid*** and ***control*** failures or mitigate their harmful effects.
- The safety mechanism is either able to ***switch or maintain the item in a safe state*** or able to ***alert the user*** to take control of the effect of the failure

If at any time these machines fail to perform the intended function, there could be damages.

ISO 26262 Failure Clasification

- **Systematic Failures (Pre-Production)**

ISO 26262 target is to prevent/avoid systematic failures

induced in a deterministic way during development, manufacturing, or maintenance (process issues)
for example, incorrect specification or manufacturing defects

- **Random Failures (Production)**

ISO 26262 target is to control random failures

random defects, process or usage conditions such as radiation or silicone wear out

- *permanent faults* (e.g., stuck-at faults)
- *transient faults* (e.g., single-event-upsets or soft errors)

The Verification Problem

- State of the art Functional Verification Methodology is not directly supporting verification of random failures within ISO 26262 requirements for functional safety

- Verification Methodology is required to distinguish functional safety verification from classical functional verification flow
  - Tools new tools, updated verification approached, etc.
  - Procedures stricter and well documented

Semiconductor and Functional Safety

- It is all about ***data storage*** and ***data movement*** through the system
- Electrical or magnetic interference inside hardware system can cause single bit to spontaneously flip to opposite state

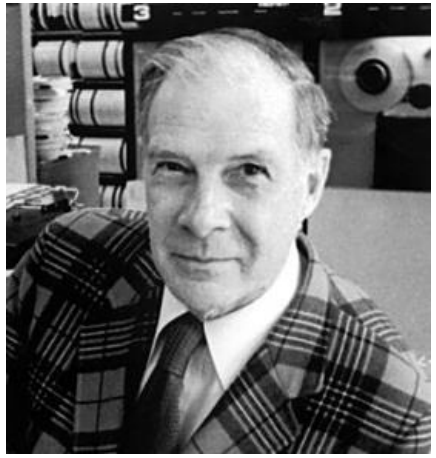
Some statistic: error rates range is *from* 10^{-10} *to* 10^{-17} error/bit in one hour

Roughly for 1GB of memory in range
from **one bit error per hour** to **one bit error per century**

Two main error-detecting codes

Hamming Code

- Invented in 1950 by Richard Hamming
- Used in computer memory systems



Cyclic Redundancy Check

- Invented in 1962 by Wesley Peterson
- Used in Ethernet, USB, wireless, mobile and many other standards

Hamming (7,4) coding and decoding

4 bit data value is encoded to 7 bit by adding 3 parity bits: $X_0X_1X_2X_3 \rightarrow P_0P_1X_0P_2X_1X_2X_3$

Received 7 bit value is : $P_0P_1X_0P_2X_1X_2X_3$

We do index XOR to get bit position of errored bit

$$P_0 = X_0 \oplus X_1 \oplus X_3$$
$$P_1 = X_0 \oplus X_2 \oplus X_3$$
$$P_2 = X_1 \oplus X_2 \oplus X_3$$

Syndrome:

$$S_0 = P_0 \oplus X_0 \oplus X_1 \oplus X_3$$
$$S_1 = P_1 \oplus X_0 \oplus X_2 \oplus X_3$$
$$S_2 = P_2 \oplus X_1 \oplus X_2 \oplus X_3$$

DED condition is:

- Parity bit is zero
- Sx > 0 (syndrome > 0)

Position of error	Error	S_0	S_1	S_2	$(S_2S_1S_0)_{10}$	
---	none	0	0	0	0	*OK*
1	P_0	1	0	0	1	*Error on parity bit, data is not affected*
2	P_1	0	1	0	2	*Error on parity bit, data is not affected*
3	X_0	1	1	0	3	*Single Error Correction*
4	P_2	0	0	1	4	*Error on parity bit, data is not affected*
5	X_1	1	0	1	5	*Single Error Correction*
6	X_2	0	1	1	6	*Single Error Correction*
7	X_3	1	1	1	7	*Single Error Correction*

Cyclic Redundancy Check

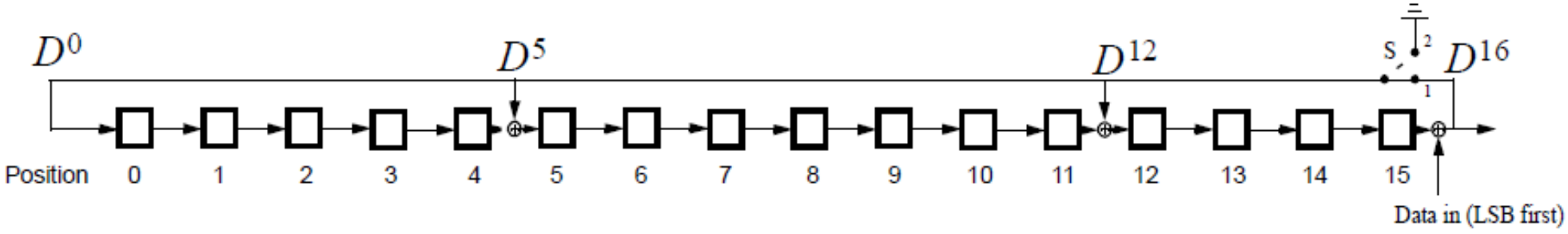
Encodes messages by adding a ***fixed-length check value***, for the purpose of error detection

Payload Data	Check Value (CRC)

Specification of a CRC code requires definition of a so-called ***generator polynomial***

Generator polynomial for Bluetooth Baseband Packet is **$g(D) = D^{16} + D^{12} + D^{5} + 1$**

16-bit LFSR Circuitry: (*Linear Feedback Shift Register*)



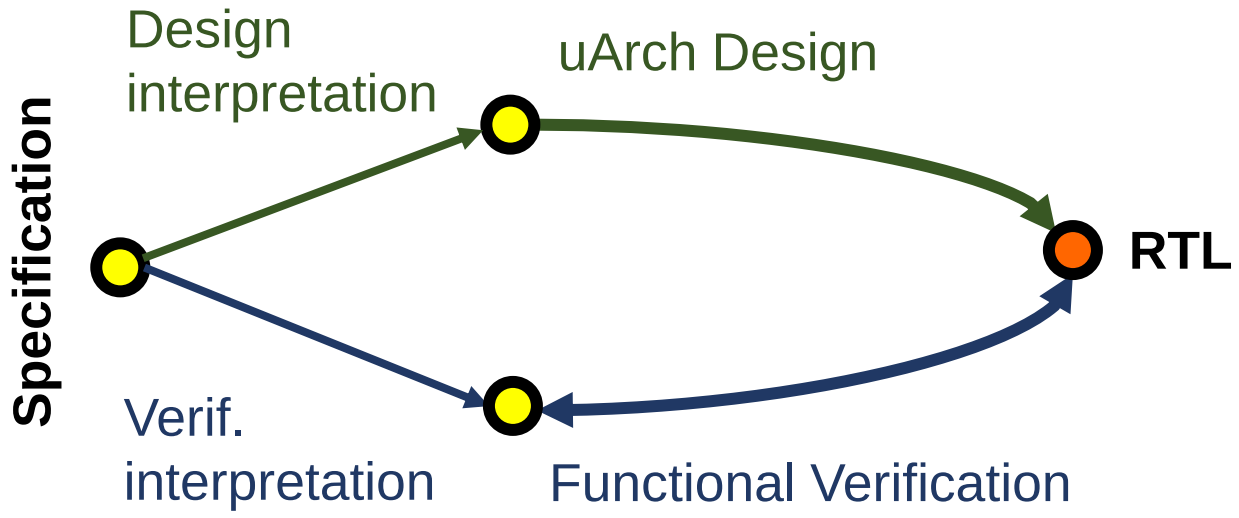
Error Correction Schemes Summary

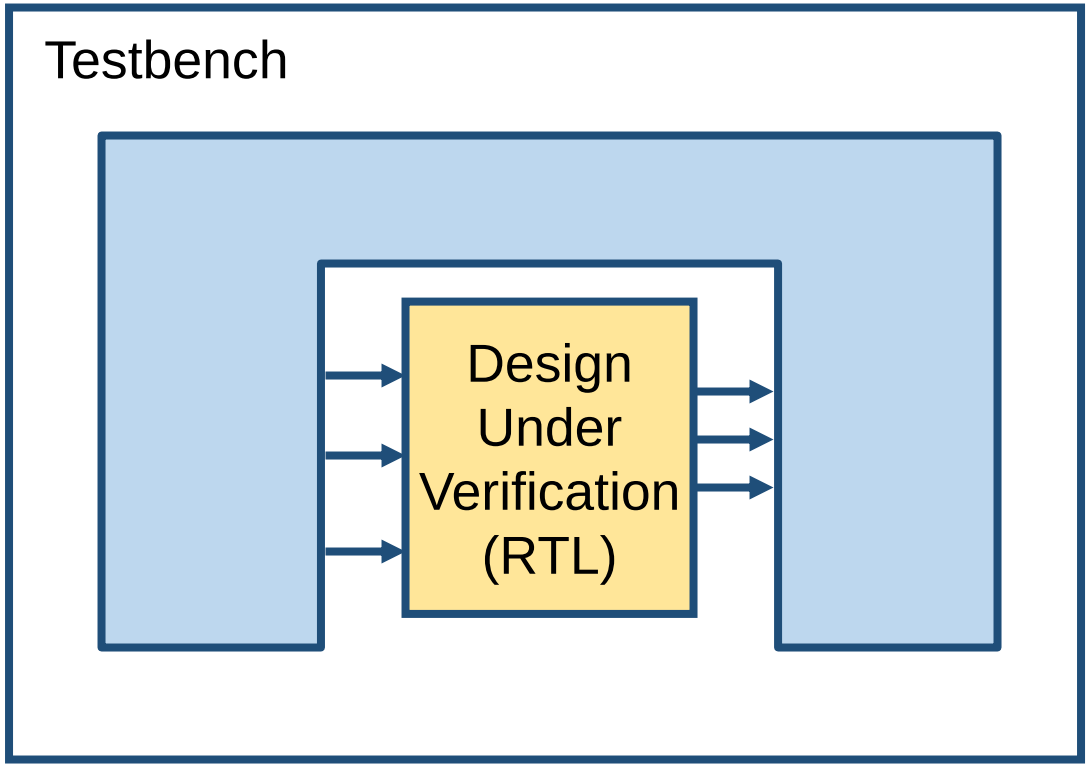
- Background math is quite complex and sophisticated, but implementation is straight forward
- Everything is based on XOR combinatorial networks and shift registers
- Influence on signal propagation through critical design paths (pushing timing constraints)
- Area constraints must be considered when safety targets are defined

Agenda

- Functional Safety Introduction
  - Standardization
  - Error Correction Schemes
- Functional Verification and Functional Safety Verification Challenge
- Verification Methodology for Functional Safety Verification
  - Tools
  - Procedures

Functional Verification

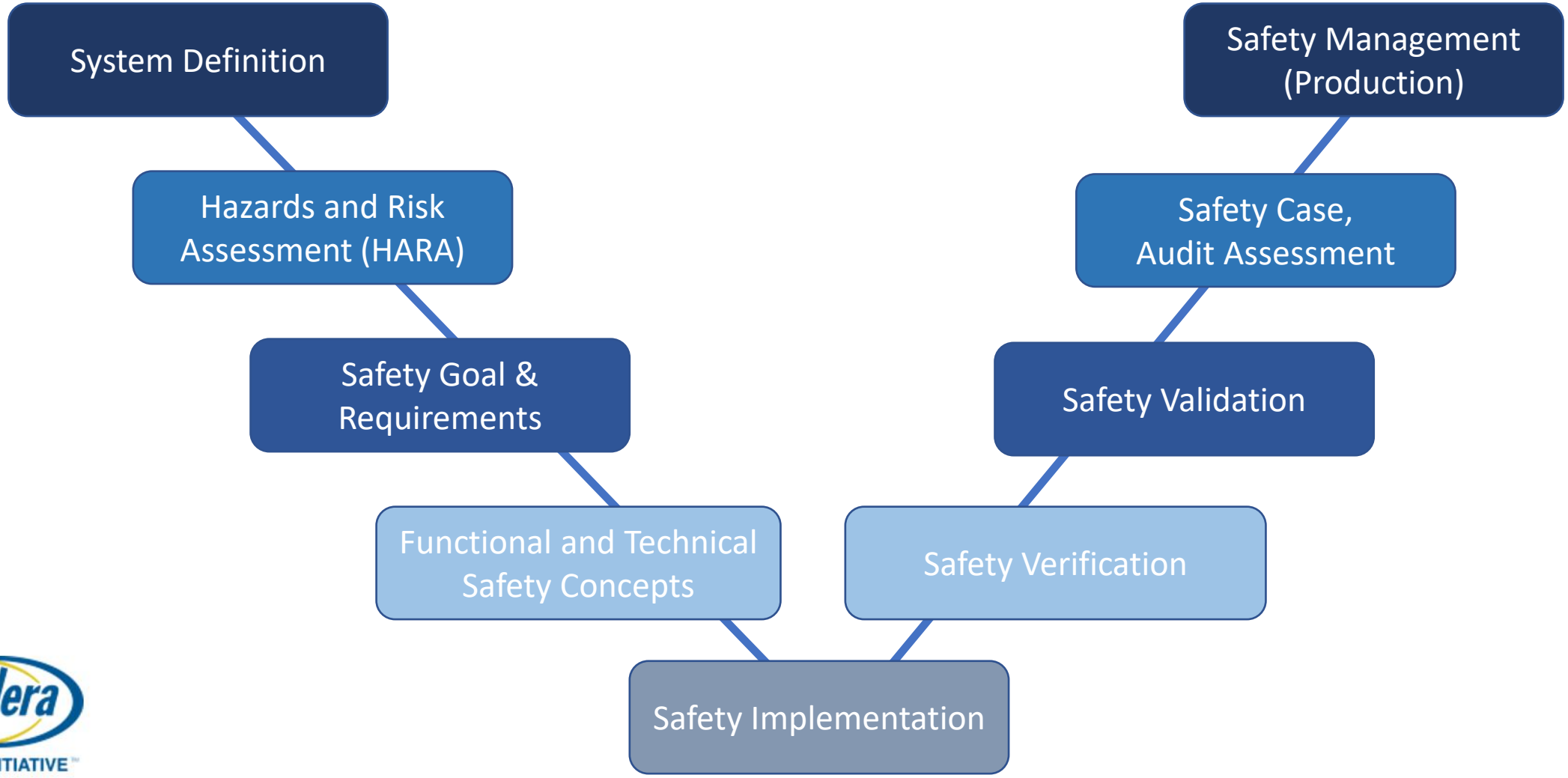




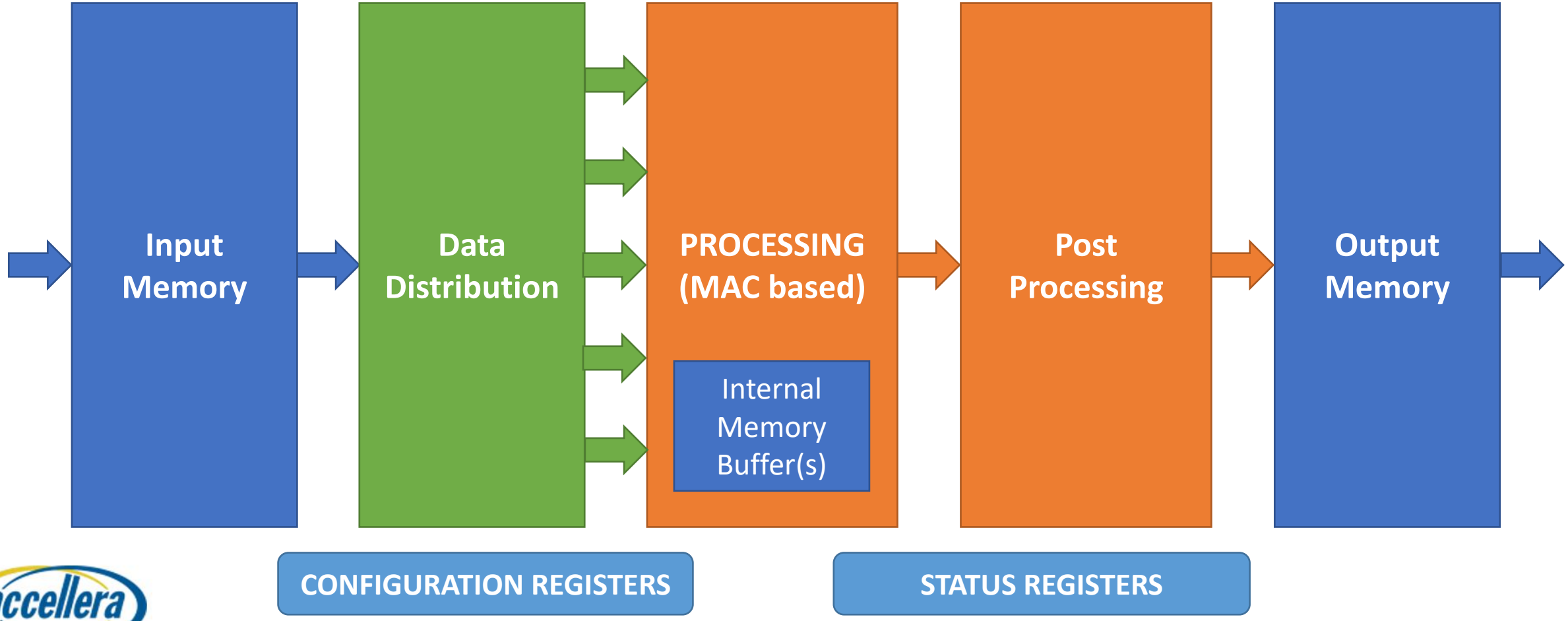
Black Box-Verification Approach

using only available interfaces
without any knowledge of the actual implementation of design

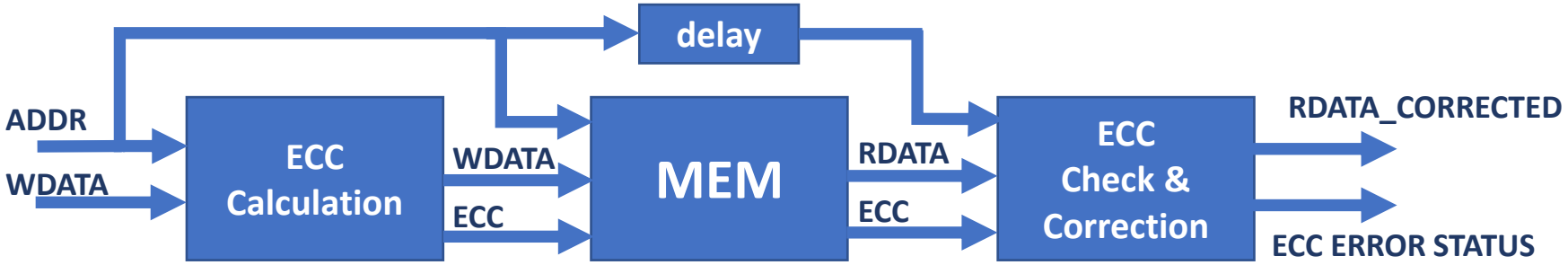
Safety V-Model as per ISO 26262



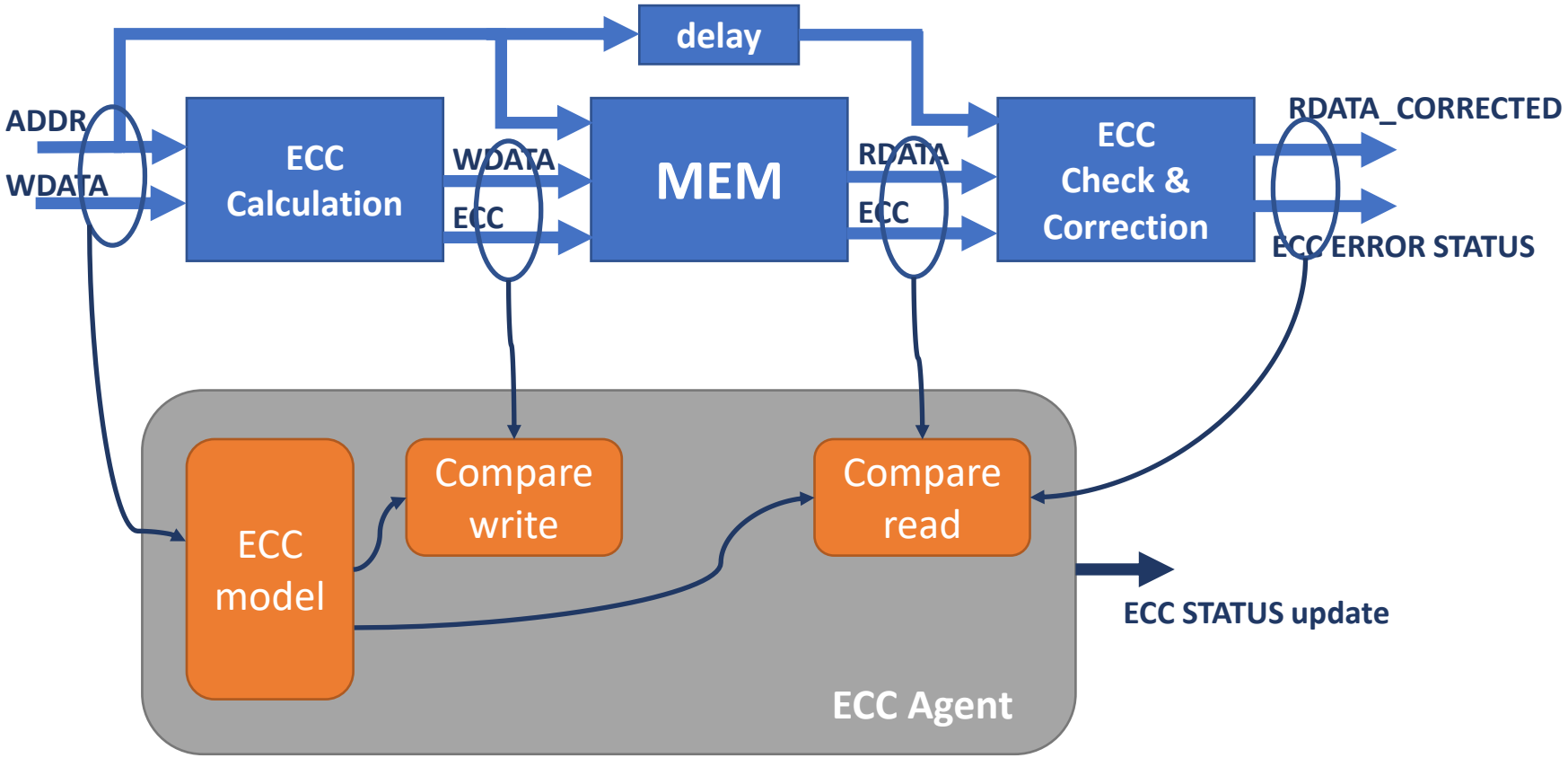
Generic AI Processor Structure



Memory ECC Implementation Example

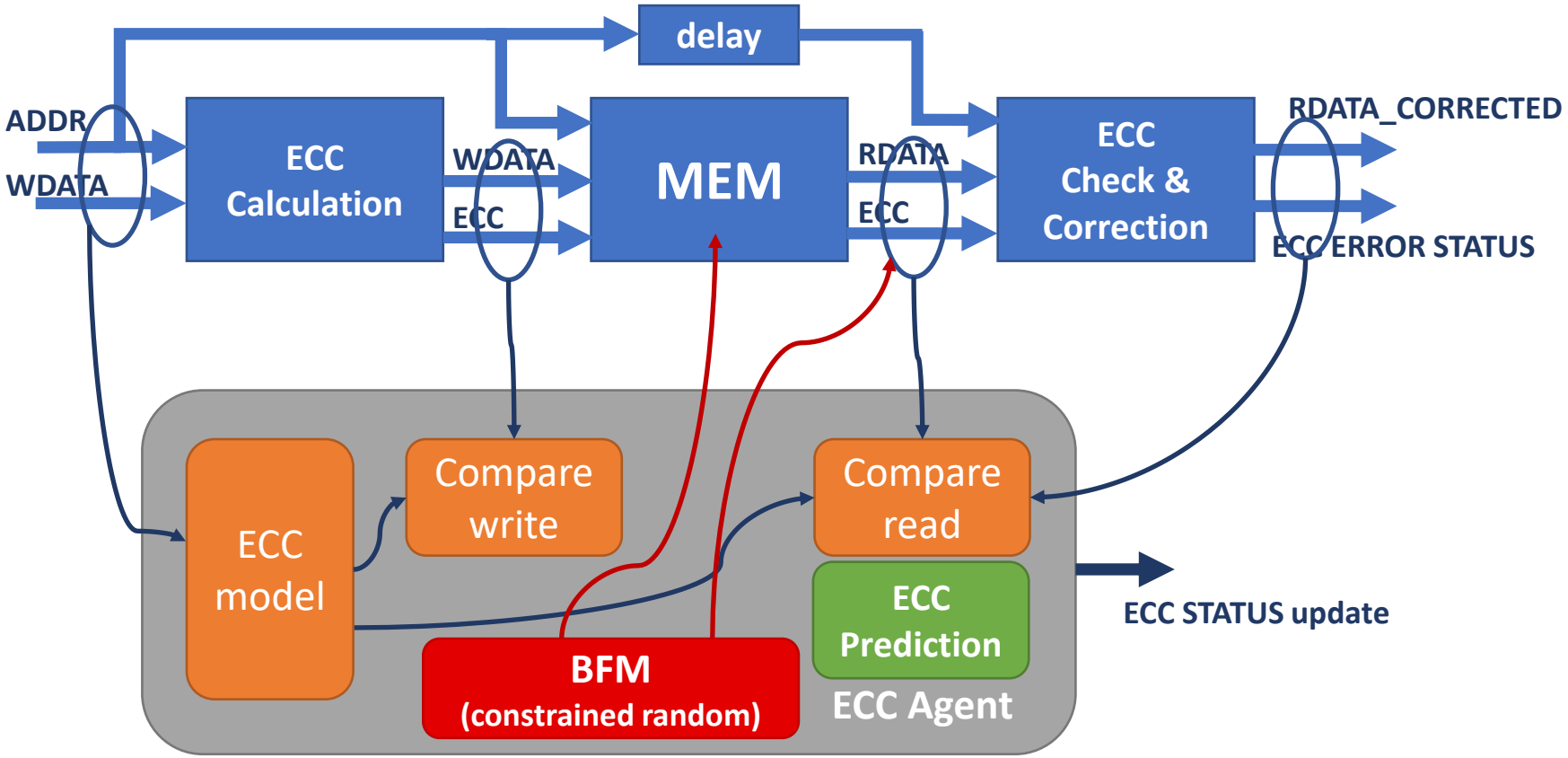


Memory ECC Verification Strategy



Passive Components / Checkers

Memory ECC Verification Strategy, ACTIVE Error Injection



Permanent faults

Transient faults

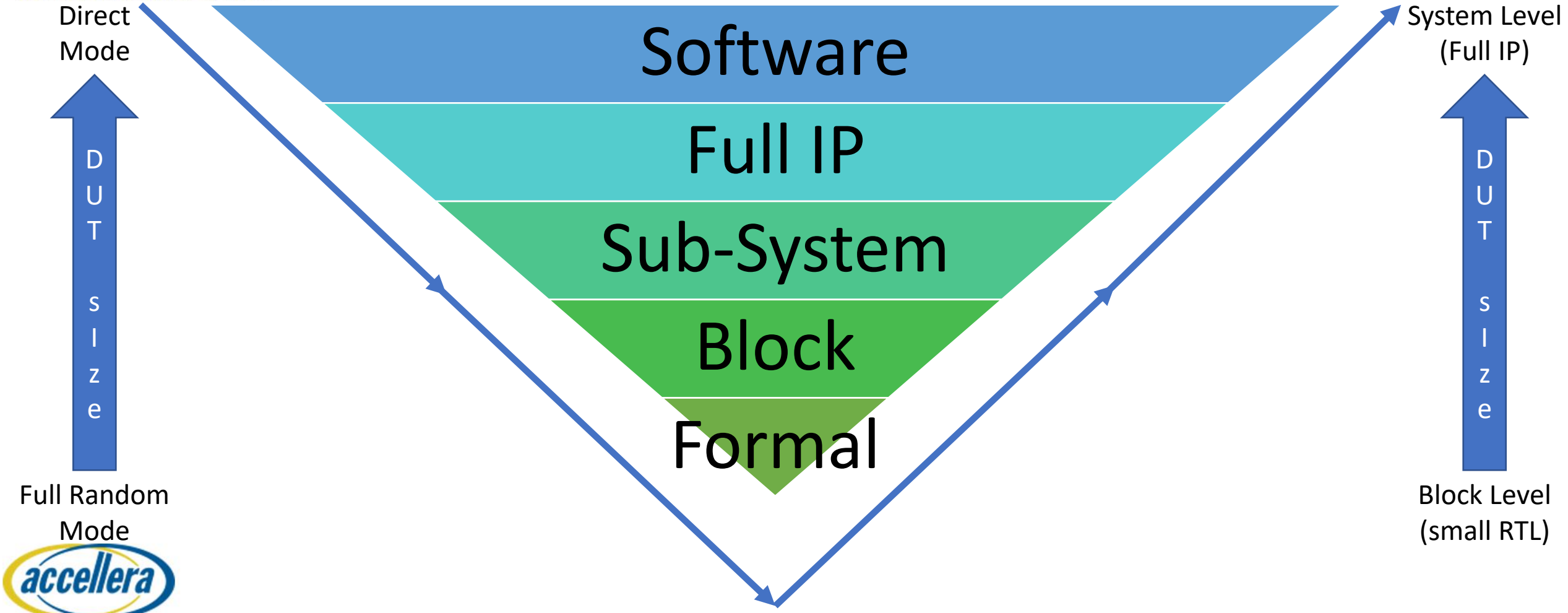
Functional Verification Challenge with Functional Safety

- Error injection inside DUT is required
- ***White-box*** approach is required instead of classical *black-box* functional verification methodology
- Error reports are good
- Classic verification tree is growing new branch to meet ISO 26262 requirements with need for ***white-box error generation***

Agenda

- Functional Safety Introduction
  - Standardization
  - Error Correction Schemes
- Functional Verification and Functional Safety Verification Challenge
- Verification Methodology for Functional Safety Verification
  - Tools
  - Procedures

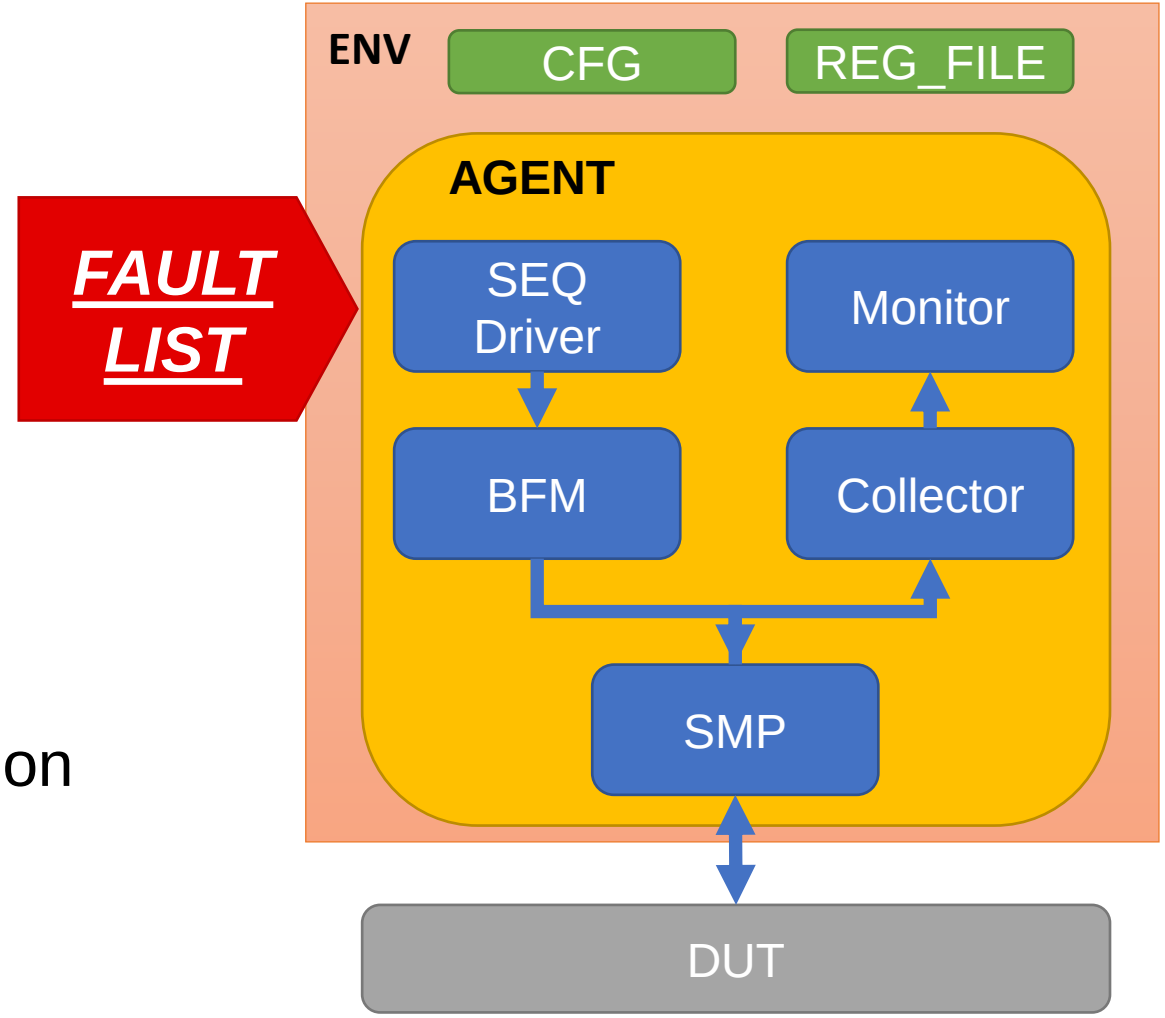
Tools For Safety Analysis V-Model



Safety UVC Architecture

ECC agent needs to have

1. ECC modeling
2. ECC checkers
3. Error injection
4. Error monitor and predictor
5. Error reporting
6. Recovery flow implementation



Error Injection:
Signal Deposit or *Signal Force* ?

- Error Injection must be implemented by bit flipping on design side (*inside DUT*)

- Signals which will be intentionally corrupted from the verification side must be *agreed between design and verification teams*

  - Recommendation: if possible, *target registers* not wires/nets
    - *Nets have a resolution function in Verilog designs*, which resolves a final value when there are multiple drivers on the net -> wired or

Error Injection: *Signal Deposit* or *Signal Force* ?

Signal deposit

```
p_smp.psl_fault_litst[0] = a_faulty_value;
```

Gives a value to a net or register that will propagate forward. The signal retains the deposited value until its next scheduled change

Use for ***Permanent Faults*** or for corrupting memory bits

Signal force

```
force p_smp.psl_fault_litst[0] = a_faulty_value;
```

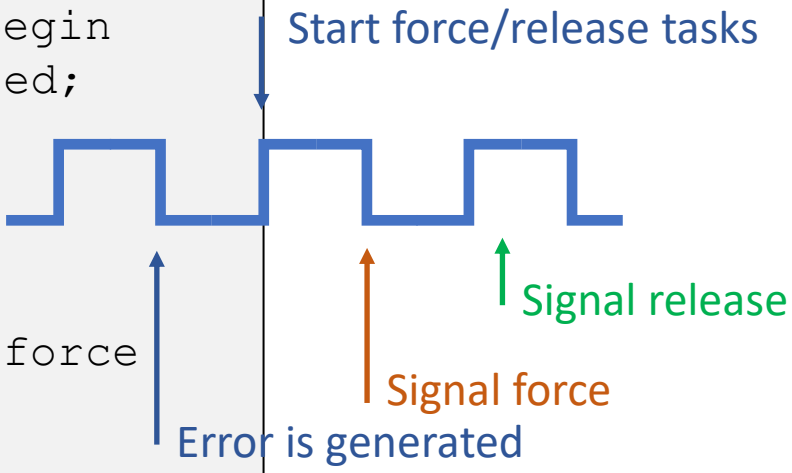
Forces a value to a continuous assignments that will propagate forward. Overrides all other drivers and stays in effect until replaced with another force or canceled with release.

Use for ***Transient Faults*** together with signal release

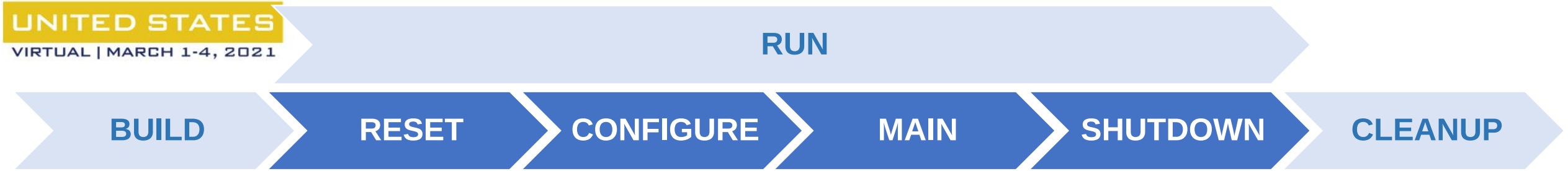
Example for Transient Fault Injection

```
task force_error(...);
  @(negedge if.clk);
  if (a_error_type in [DATA_ERROR, DATA_AND_PARITY_ERROR]) begin
    force p_smp.psl_signal_data[a_sig_idx] = a_data_corrupted;
  end
  if (a_error_type in [PARITY_ERRORS, DATA_AND_PARITY_ERROR]) begin
    force p_smp.psl_signal_parity[a_sig_idx] = a_parity_corrupted;
  end
endtask : force_error

task release_error(...);
  @(posedge if.clk); // skip next rising edge, wait for signal force
  #(CLK_PERIOD/3);   // avoid race conditions with monitor,
                     // release signals after clock rising edge
  if (a_error_type in [DATA_ERROR, DATA_AND_PARITY_ERROR]) begin
     release p_smp.psl_signal_data[a_sig_idx];
  end
  if (a_error_type in [PARITY_ERRORS, DATA_AND_PARITY_ERROR]) begin
    release p_smp.psl_signal_parity[a_sig_idx];
  end
endtask : release_error
```



Reset on Safety Critical Event



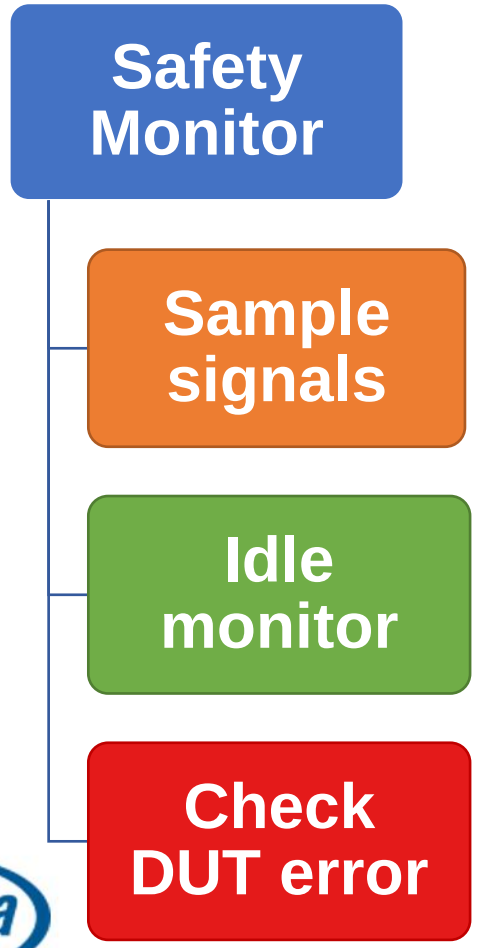
When critical event is detected specification might request that processing is stopped, and reset is executed

With UVM phases this is straight forward to execute with rerun of the reset phase from simple sequence

```
class reset_on_ded_test_c extends basic_test_c;
  ...
  virtual task main_phase(uvm_phase phase);
    if (ded_reset) begin
      phase.raise_objection(this);
      std::randomize(reset_delay_ns) with {
        reset_delay_ns inside {[10:50]};
      };
      #(reset_delay_ns * 1ns);
      phase.drop_objection(this);
      phase.jump(uvm_pre_reset_phase::get());
      ded_reset = 0;
    end
  endtask : main_phase

endclass : reset_on_ded_test_c
```

Safety Monitor

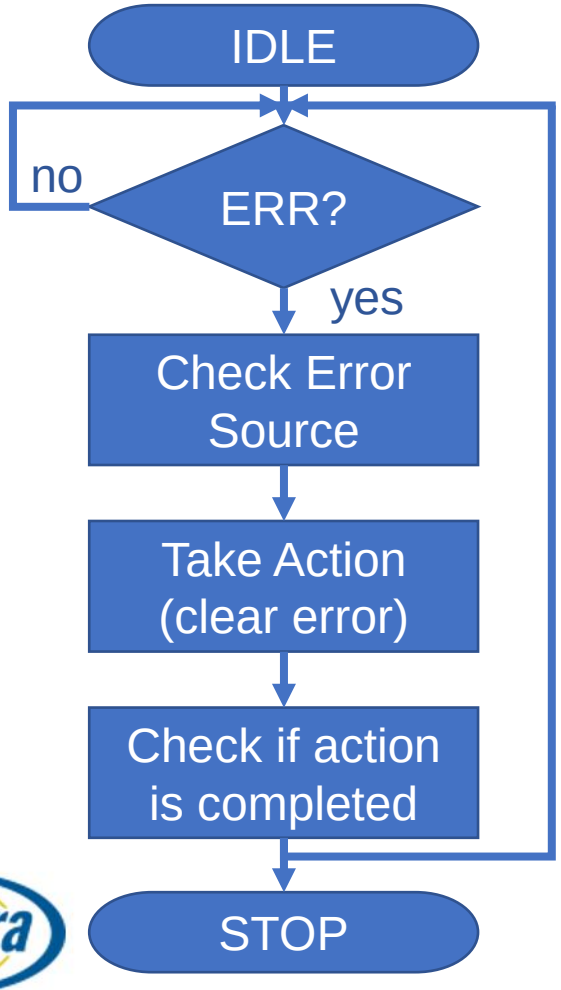



```
class safety_monitor extends uvm_monitor #(my_transaction);
  `uvm_component_utils(safety_monitor)
  ...
  task pre_main_phase (uvm_phase phase);
    forever begin
      @(posedge dut.reset);
      fork
        sample_signals();
        idle_monitor();
        check_dut_error();
      join_none

      @(negedge dut.reset);
      disable fork;
    end
  endtask: pre_main_phase
endclass: safety_monitor
```

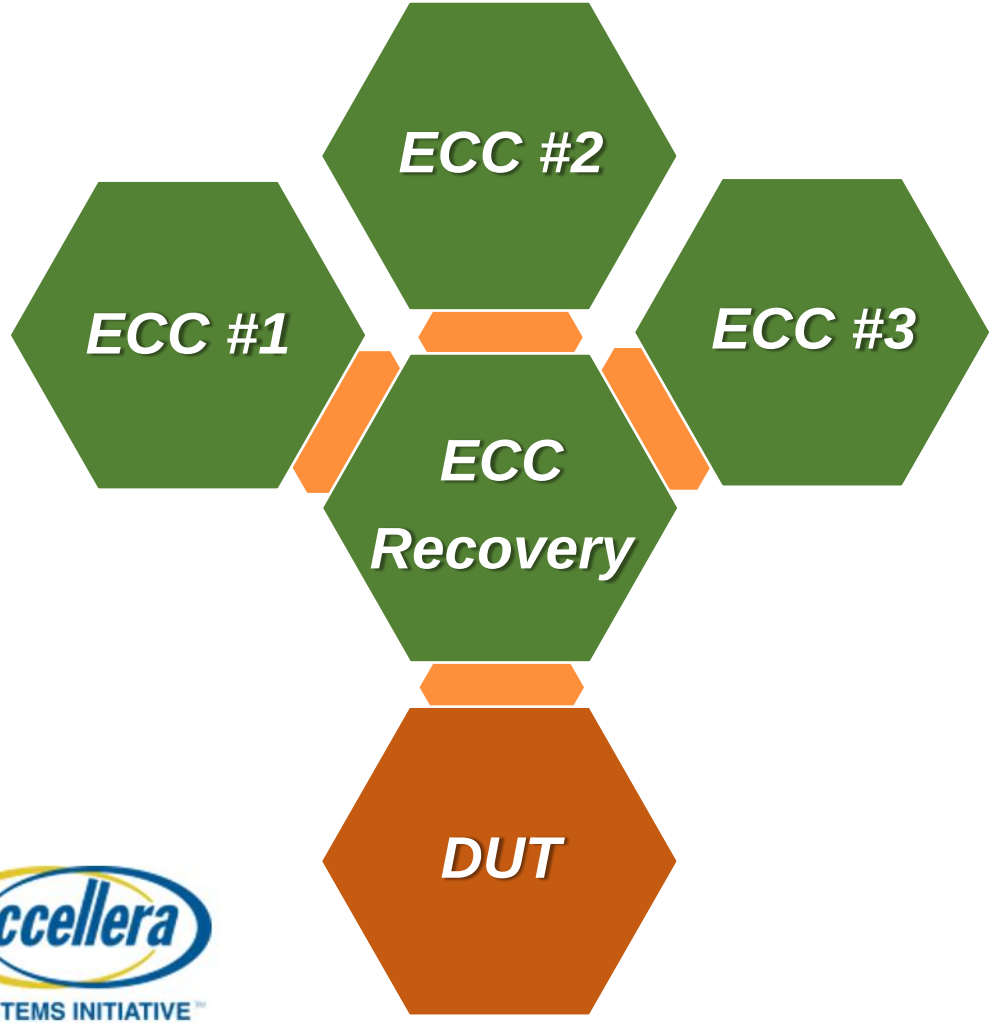
Continuous events which needs action in every clock cycle are monitored in threads started in pre_main phase

Error Recovery Flow Example



- For realistic modeling recovery flow should use front door access to the status information (e.g. APB or AXI transactions)
- This is taking some time, particularly if clock division is enabled on the interface
- New error can be generated while error recovery flow is in progress

Unifying Recovery Flow for all Agents in Environment



Multiple ECC agents connected to DUT and each can start recovery flow from its BFM

Only one physical interface for recovery flow

Unified ECC Recovery Flow

1. Reducing stress on interface transactions
2. Improving sequence predictability
3. Enables atomic approach for ECC recovery

Built-In-Self-Test Verification

Automotive requirements for IC operation time is 15+ years !!!

BIST logic is used for two main cases

1. Lab testing after chip arrives from fab and before is integrated into final product
2. Test sequence while IC is integrated in the system and in use

Part of Design-for-Testing architecture,
crucial for ***in-system testing for lifetime reliability***

Two general categories of BIST techniques for testing random logic[1]

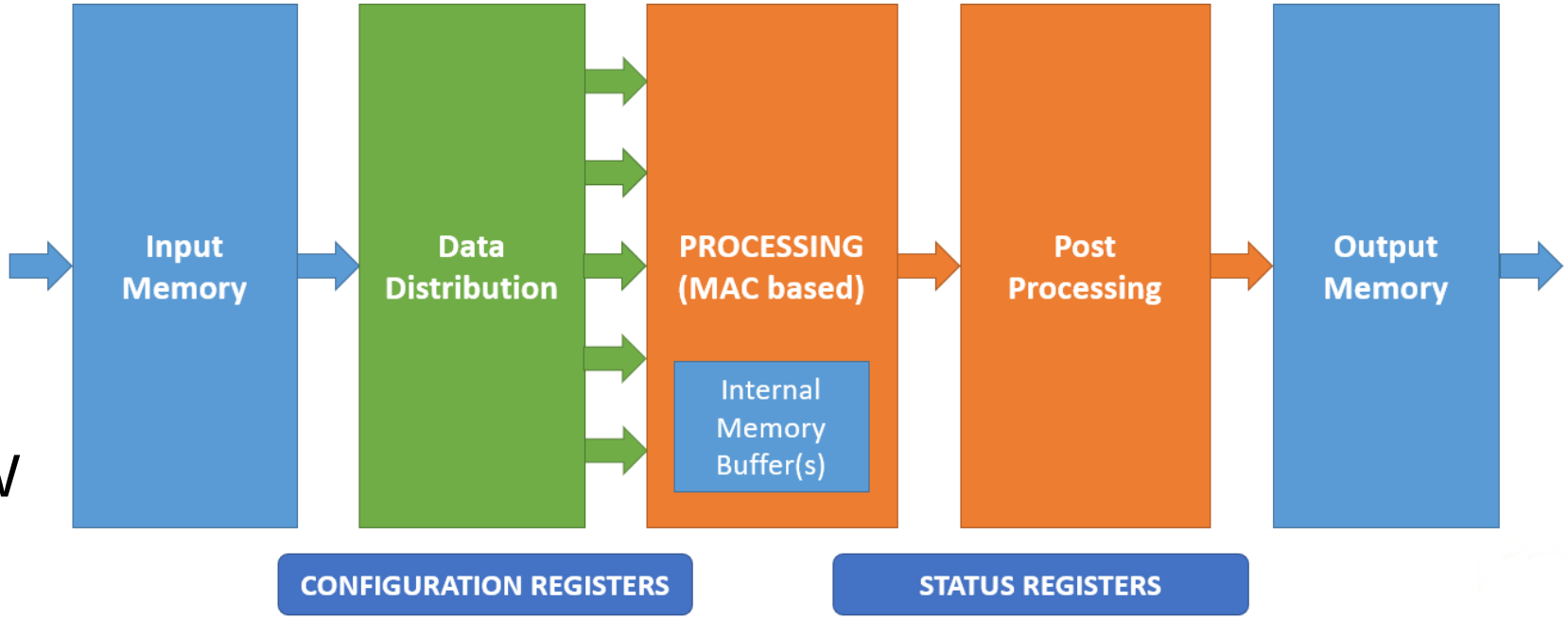
1. **Online BIST**: while circuitry is in normal operational mode (*mission r*
2. **Offline BIST**: when circuitry is not in normal operational mode,
   e.g. during *power on reset* at the engine startup

CHALLENGE

[1] Wang, Laung-Terng, Cheng-Wen Wo, and Xiaoqing Wen.
"VLSI test principles and architectures: design for testability." (2006).

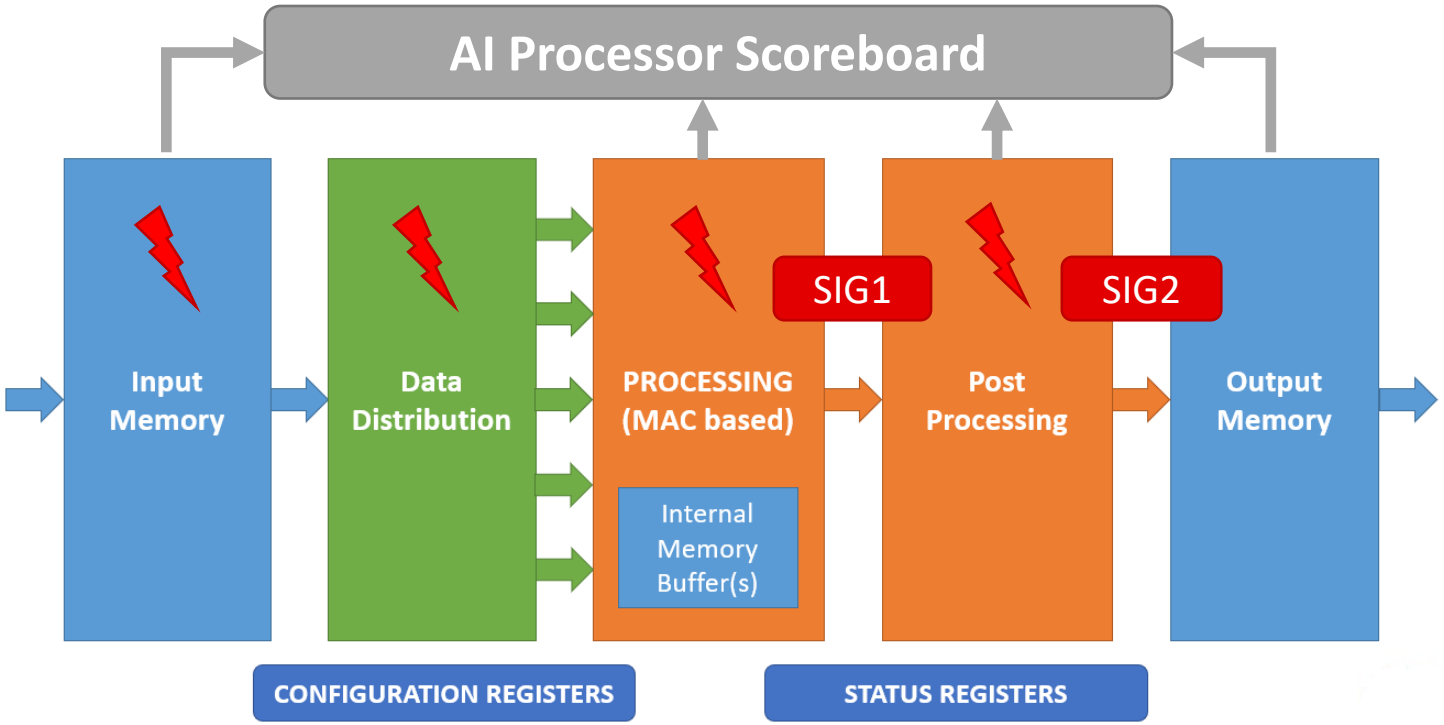
How to Implement BIST like Safety Circuitry Check in AI Processor?

- Data flow in the system
- Input memory content is controllable by the SW
- System configuration is also controllable by the SW
- Transformation function is known and defined
- Meaning with predefined input data and selected HW config we can exactly predict output values from the system



Safety Check of AI Processor

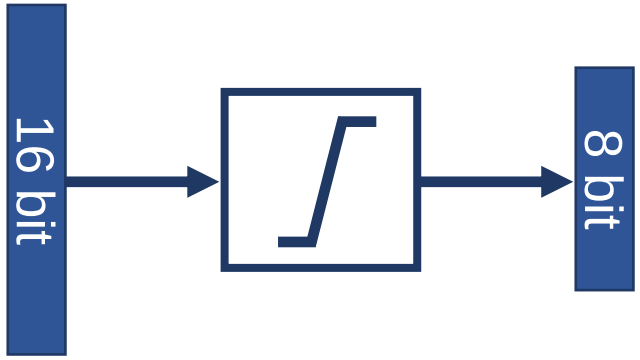
Target is to check if circuitry is operational and there are no stack at faults in the processing pipeline



- Intentional random errors must be injected by hardware itself, and/or from verification environment
- Error confirmation mechanism must be in place in order to verify that ECC logic is functional

Challenges with Error Injection to Data Processing Pipeline

- Corruption of some bits is not having an influence on result and signature



- Due to multi-cycle paths and pipelined organization, it might take several clock cycles for injected error to have an impact on result and signature
  - Signal deposit is much more convenient for error injection in this case. *Be mindful about place where error is injected and impacts of Verilog resolution function !!!*

Agenda

- Functional Safety Introduction
  - Standardization
  - Error Correction Schemes
- Functional Verification and Functional Safety Verification Challenge
- Verification Methodology for Functional Safety Verification
  - Tools
  - Procedures

Verification Procedures for Functional Safety Verification

- Verification ***Process*** must be stricter and more formalized to comply with ISO 26262 requirements

- Verification ***Procedures*** needs to be enforced
  - Safety procedure is stricter and more documented compared to usual functional verification approach

- ***Different mindset is required !!!***
  - Safety procedure is main point, not documents for internal use

Safety Review Requirements – Continuous Review Process

- *The 5-step sign-off process is implemented with following safety review meetings*

1. Test plan review	(verif, design, arch, PM)	Initial phase
2. Coverage plan review	(verif, design, arch, PM)	
3. RTL code review	(design team, verif optional)	Implementation phase
4. Verification code review	(verif team, design optional)	
5. Sign off review	(verif, design, arch, PM)	Finalizing phase
○ Including RTL code coverage review		
○ Special focus on toggle coverage for all external signals		

Meeting Procedures

- All meetings must have recorded meeting reports defining
  - ***When*** was the meeting
  - ***Who*** were the participants
  - ***What*** conclusions are made
  - ***Which*** action items are defined

- Follow up meetings are organized to track implementation of defined action items until all action items are not implemented

Example of status tracking sheet

Area	Feature	Description	Safety Manual Chapter	Design Owner	Verif Owner	RTL status	Verif Status	Release Date	Feature Review				
									Test Plan	Coverage	RTL Code Review	Verif Code Review	Sign-off Review
DMSS	DMSS Register Parity	Parity protection of Key Regs	2.1.1	Jamie	Faris	ready	done	E/O June	15-Jun-20	15-Jun-20	4-Jun-20	15-Jun-20	30-Jun-20
	DMSS Latent Test	SW check for ECC detection and correction logic	3.1.2	Jamie	IvanM	ready	done	E/O June	11-Jun-20	23-Jun-20	5-Jun-20	28-Aug-20	30-Jun-20
	DMSS Pseudo-Fault Injection	Inject errors from safety software in parallel to operation error indicators (shadow registers)	7.18.3 and 7.18.4 in Arch Spec Vol III	Jamie	Faris	ready	done	E/O June	15-Jun-20	15-Jun-20	10-Jun-20	28-Aug-20	30-Jun-20
	DMSS Register Access Protection	Access protection for selected sensitive registers	2.7.4	Jamie	Yinon	ready	done	E/O June	15-Jun-20	15-Jun-20	10-Jun-20	15-Jun-20	30-Jun-20
	Safety IRQs (GVI)	General Violation Indications	2.2, 2.3 and 2.8	Jamie	Ilija	ready	done	E/O June	26-May-20	26-May-20	5-Jun-20	10-Jun-20	30-Jun-20
	Safety Outputs	Safety Interface and Hardware Exceptions	2.2 and 2.3	Jamie	Ilija	done	done	E/O June	26-May-20	26-May-20	5-Jun-20	10-Jun-20	30-Jun-20
PMSS	PMSS Register Parity	Parity protection of Key Regs	2.1.1	Jamie	Faris	ready	done	E/O June	15-Jun-20	15-Jun-20	4-Jun-20	15-Jun-20	30-Jun-20
	PMSS Latent Test	SW check for ECC detection and correction logic	3.1.3	Jamie	Barak	ready	done	E/O June	16-Jun-20	16-Jun-20	5-Jun-20	16-Jun-20	30-Jun-20
	BTB Latent Test	SW check for ECC detection and correction logic	3.1.3	Jamie	Barak	ready	done	E/O June	16-Jun-20	16-Jun-20	5-Jun-20	16-Jun-20	30-Jun-20
	PMSS Pseudo-Fault Injection	Inject errors from safety software in parallel to operation error indicators (shadow registers)	7.17.3 and 7.17.4 in Arch Spec Vol III	Jamie	Faris	ready	done	E/O June	15-Jun-20	15-Jun-20	10-Jun-20	15-Jun-20	30-Jun-20
	PMSS Register Access Protection	Access protection for selected sensitive registers	2.7.4	Jamie	Barak	ready	done	E/O June	16-Jun-20	16-Jun-20	10-Jun-20	16-Jun-20	30-Jun-20
	PMSS 2 DMSS Safety Interface	Safety error indicators from PMSS to DMSS	2.3	Jamie	Barak	ready	done	E/O June	16-Jun-20	16-Jun-20	10-Jun-20	16-Jun-20	30-Jun-20
	PMSS 2 CORE Safety Interface	Safety error interface between PMSS and CORE (reg.pty and BTB latent)	2.3	Jamie	Barak	ready	done	E/O June	16-Jun-20	16-Jun-20	5-Jun-20	16-Jun-20	30-Jun-20
CORE	CORE Register Parity	Parity protection of Key Regs	2.1.1	Tomer	Lior	ready	done	E/O June	22-Jun-20	22-Jun-20	5-Jul-20	22-Jun-20	30-Jun-20
	BTB Latent Test	SW check for ECC detection and correction logic (formal verification)	3.1.3	Jamie	Lior	ready	done	E/O June	22-Jun-20	22-Jun-20	5-Jun-20	22-Jun-20	30-Jun-20
	BTB ECC Error Detection	BTB protection	2.4.4.1	Tomer	Lior	ready	done	E/O June	22-Jun-20	22-Jun-20	5-Jul-20	22-Jun-20	30-Jun-20
	Dispatcher Error Detection	The dispatcher reports on an uncorrectable error	2.4.4.2	Tomer	Lior	ready	done	E/O June	22-Jun-20	22-Jun-20	5-Jul-20	22-Jun-20	30-Jun-20

Functional Verification Results 1/2

- **Memory ECC**
  - 44 test for 14 memory instances (3 different memory types)
    600 runs in single regression
    100 % pass rate
  - Functional coverage
    - 100 % out of 9746 items for P1 configuration
    - 99+ % out of 28940 items for all configurations

- **Hardware error injection scenarios on memory ECC**
  - 18 tests with 98 runs in single regression
    100 % pass rate
  - 99.98 % out of 4076 coverage items (1 bin not hit)

Functional Verification Results 2/2

- **Safety Circuitry of AI processor**
  - 67 tests with 949 runs in single regression
    ~100% pass rate
    - 20 without error injection
      - 9 no output errors expected – data flow checks
      - 12 with expected errors due to data manipulation on input memory content
    - 30 tests with random errors injected to pipeline
    - 17 directed tests with error injection to critical signals
  - Functional coverage
    - 99.72% out of 5645 items
      (not all software parameter combinations reached in regression run)

2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

VIRTUAL | MARCH 1-4, 2021

CEVA®

Thank You !

Veriest

www.VeriestS.com

accellera SYSTEMS INITIATIVE™