ISO 26262 Dependent Failure Analysis Using PSS

Moonki Jang – Samsung Electronics Co.,Ltd.

Agenda

- Introduction to ISO 26262
- ISO 26262 functional safety features for semiconductor
- Using PSS for DFA (Dependent Failure Analysis)
- Result and lesson learned

Background of ISO 26262

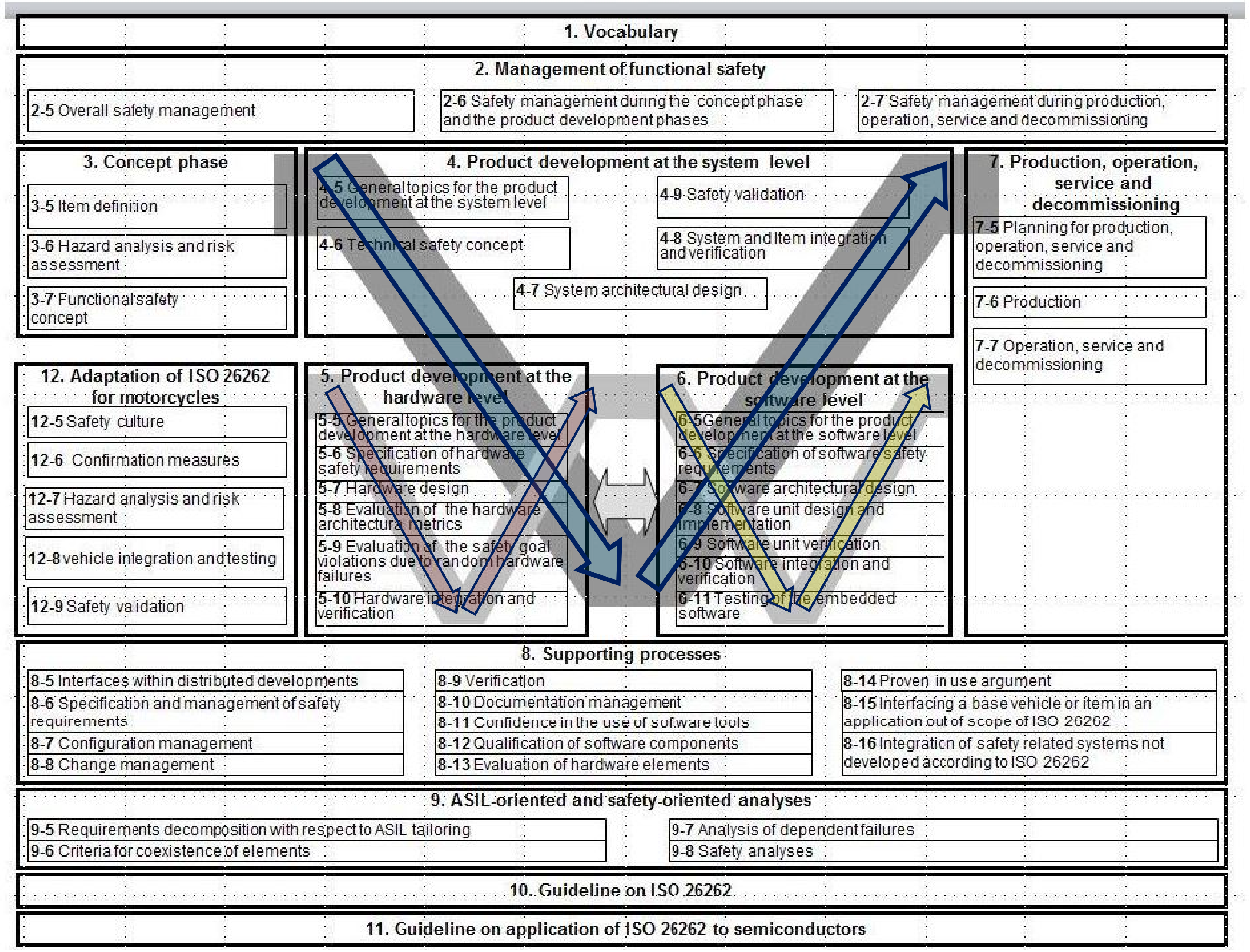
- For a long time electronics were a comfort feature
  - **Now they are a Safety Feature**

EXCLUSIVE
TOYOTA INVESTIGATION
Study: hundreds of reported incidents of sudden acceleration
Anderson Cooper 360
CNN

Functional Safety

- Functional safety (ISO 26262)
  - Absence of unacceptable risk due to hazards caused by malfunctioning or unintended behavior of E/E systems
  - Possible root causes
    - Specification, implementation or realization errors
    - Failure during operation
    - Reasonably foreseeable misuse / operational errors

Overall framework of ISO 26262



1. Vocabulary

2. Management of functional safety
- 2-5 Overall safety management
- 2-6 Safety management during the concept phase and the product development phases
- 2-7 Safety management during production, operation, service and decommissioning

3. Concept phase
- 3-5 Item definition
- 3-6 Hazard analysis and risk assessment
- 3-7 Functional safety concept

4. Product development at the system level
- 4-5 General topics for the product development at the system level
- 4-6 Technical safety concept
- 4-7 System architectural design
- 4-8 System and Item integration and verification
- 4-9 Safety validation

7. Production, operation, service and decommissioning
- 7-5 Planning for production, operation, service and decommissioning
- 7-6 Production
- 7-7 Operation, service and decommissioning

12. Adaptation of ISO 26262 for motorcycles
- 12-5 Safety culture
- 12-6 Confirmation measures
- 12-7 Hazard analysis and risk assessment
- 12-8 vehicle integration and testing
- 12-9 Safety validation

5. Product development at the hardware level
- 5-5 General topics for the product development at the hardware level
- 5-6 Specification of hardware safety requirements
- 5-7 Hardware design
- 5-8 Evaluation of the hardware architectural metrics
- 5-9 Evaluation of the safety goal violations due to random hardware failures
- 5-10 Hardware integration and verification

6. Product development at the software level
- 6-5 General topics for the product development at the software level
- 6-6 Specification of software safety requirements
- 6-7 Software architectural design
- 6-8 Software unit design and implementation
- 6-9 Software unit verification
- 6-10 Software integration and verification
- 6-11 Testing of the embedded software

8. Supporting processes
- 8-5 Interfaces within distributed developments
- 8-6 Specification and management of safety requirements
- 8-7 Configuration management
- 8-8 Change management
- 8-9 Verification
- 8-10 Documentation management
- 8-11 Confidence in the use of software tools
- 8-12 Qualification of software components
- 8-13 Evaluation of hardware elements
- 8-14 Proven in use argument
- 8-15 Interfacing a base vehicle or item in an application out of scope of ISO 26262
- 8-16 Integration of safety related systems not developed according to ISO 26262

9. ASIL-oriented and safety-oriented analyses
- 9-5 Requirements decomposition with respect to ASIL tailoring
- 9-6 Criteria for coexistence of elements
- 9-7 Analysis of dependent failures
- 9-8 Safety analyses

10. Guideline on ISO 26262

11. Guideline on application of ISO 26262 to semiconductors

Agenda

- Introduction to ISO 26262
- ISO 26262 functional safety features for semiconductor
- Using PSS for DFA (Dependent Failure Analysis)
- Result and lesson learned

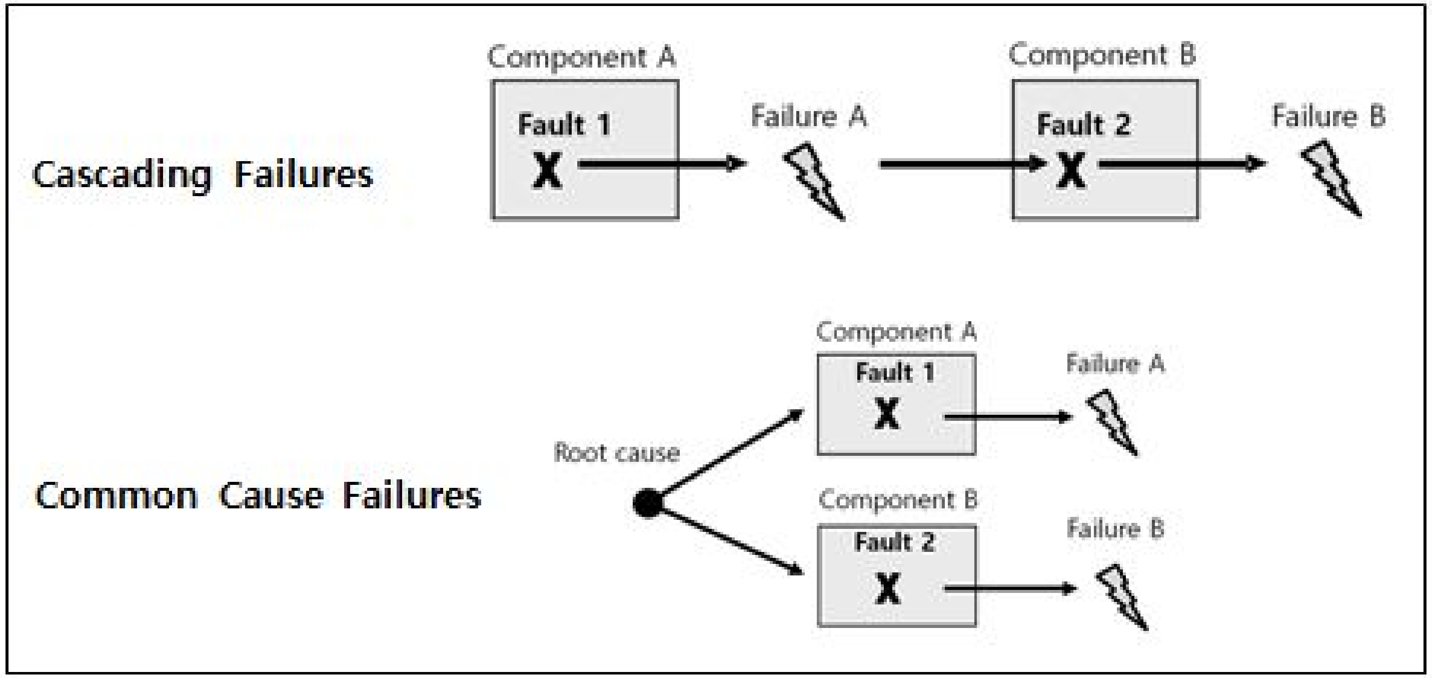
ISO 26262 for Semiconductor

- 2nd revision of ISO 26262 was released in 2018. Part 11 has been modified for semiconductor guideline

Main Agenda	Applicable Items
* Base failure rate estimation - Permanent fault - Transient fault - Component package failure *** Dependent failure analysis (DFA)** *** Fault injection**	- Digital components, memories - Analogue / Mixed signal components - Programmable logic devices **- Multi-core components** - Sensors and transducers

Dependent Failure Analysis (DFA)

- The analysis of dependent failures aims to identify the single events or single causes that could bypass or invalidate a required independence or freedom from interference between elements and violate a safety requirement or a safety goal.



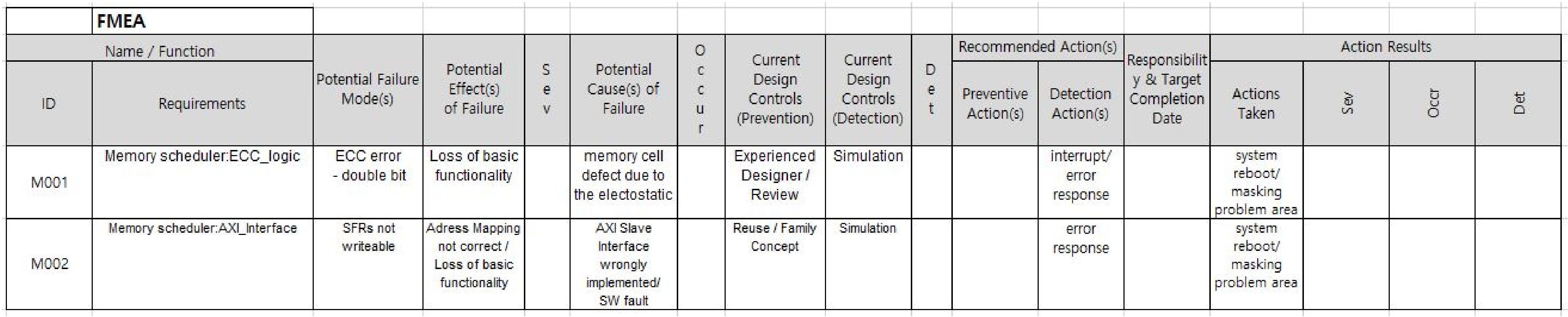
Dependent Failure Initiator (DFI)

- The Dependent Failure Initiator (DFI) represents the root cause of dependent failures in functional safety

- In general, DFI is defined as an item that can threaten the independence required between elements.

Defining DFIs

- Failure Mode and Effects Analysis (FMEA) determines all possible ways a system component can fail and determines the effect of such failures on the system. The DFI is selected based on the pre-defined FMEA items as shown below.



	FMEA															
	Name / Function									Recommended Action(s)			Action Results			
ID	Requirements	Potential Failure Mode(s)	Potential Effect(s) of Failure	Sev	Potential Cause(s) of Failure	Occur	Current Design Controls (Prevention)	Current Design Controls (Detection)	Det	Preventive Action(s)	Detection Action(s)	Responsibility & Target Completion Date	Actions Taken	Sev	Occr	Det
M001	Memory scheduler:ECC_logic	ECC error - double bit	Loss of basic functionality		memory cell defect due to the electostatic		Experienced Designer / Review	Simulation			interrupt/ error response		system reboot/ masking problem area			
M002	Memory scheduler:AXI_Interface	SFRs not writeable	Adress Mapping not correct / Loss of basic functionality		AXI Slave Interface wrongly implemented/ SW fault		Reuse / Family Concept	Simulation			error response		system reboot/ masking problem area			

Fault Injection

- In our experiment, a fault occurring in a shared memory area is defined as the DFI and implemented through fault injection
  - Uncorrectable ECC error injection
  - Memory Management Unit(MMU) translation fault generation
  - RAS error injection for CPU, Interrupt controller, System MMU

Coupling Factor

- A coupling factor is a common characteristic or relationship of elements that leads to dependency in their failure.

- The following coherency interference stimulus for a shared memory region can be a coupling factor
  - False sharing coherency access
  - Distributed Virtual Memory(DVM) transaction broadcasting
  - Exclusive access
  - CPU cluster power down

Agenda

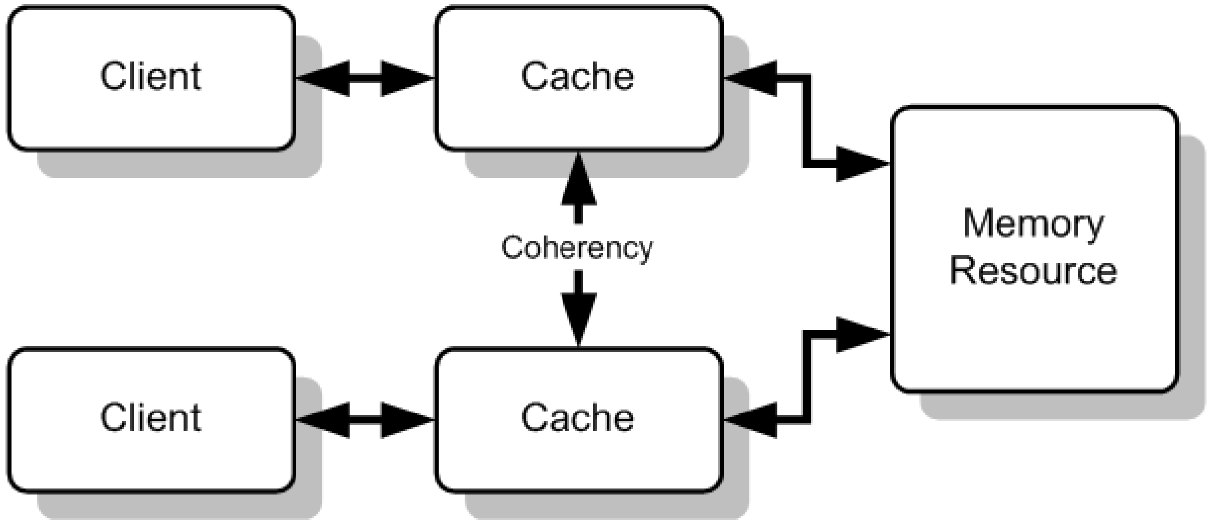
- Introduction to ISO 26262
- ISO 26262 functional safety features for semiconductor
- Using PSS for DFA (Dependent Failure Analysis)
- Result and lesson learned

Why PSS?

- For DFA, we need to create hundreds of scenarios that combine all of the functions that can be used as coupling factors for each DFI
- The PSS model reusability and constrained-random test generation made it easy to generate tests with various conditions defined in safety requirements.

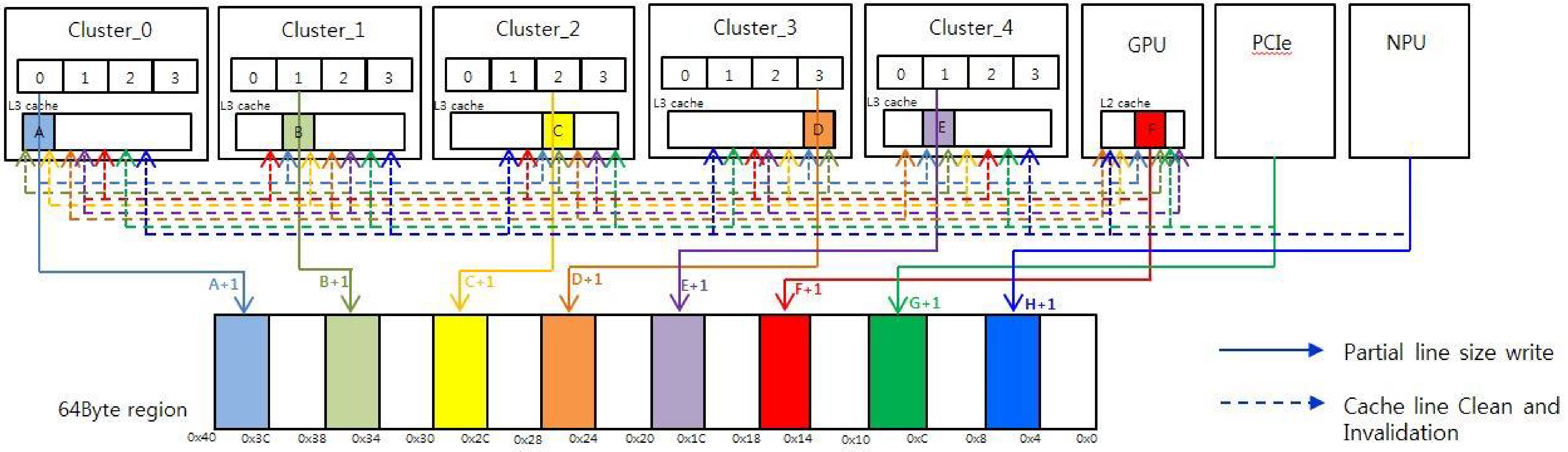
Dependency of Multi-Core System

- Cache coherence is the discipline which ensures that the changes in the values of shared operands (data) are propagated throughout the system in a timely fashion.
- A fault in a shared resource can affect other elements that share that resource



False-Sharing Operation

- Each master uses a unique address-range within the same cache line
- Each time a coherent master writes a value to a block allocated to it, a number of snoop transactions are generated between the coherent masters to clear the caches of all other masters



Fault Injection

- A fault occurring in a shared memory area is defined as the DFI and implemented through fault injection as follows:
  - Uncorrectable ECC error injection
    - Main Memory (DRAM)
    - Unified L3 Data Cache
    - L1/L2 Data cache
  - Memory Management Unit (MMU) translation fault generation
  - RAS (Reliability, Availability, and Serviceability) error injection for CPU, Interrupt controller, System MMU

- If a fault is injected into the 64-byte cache-line, previous coherency operation causes a failure in all coherent masters participating in the false sharing scenario

Fault Generation

- Using PSS, the previous fault injection options are modeled as reusable actions. And it can generate various DFIs with the desired number of faults at any given time.

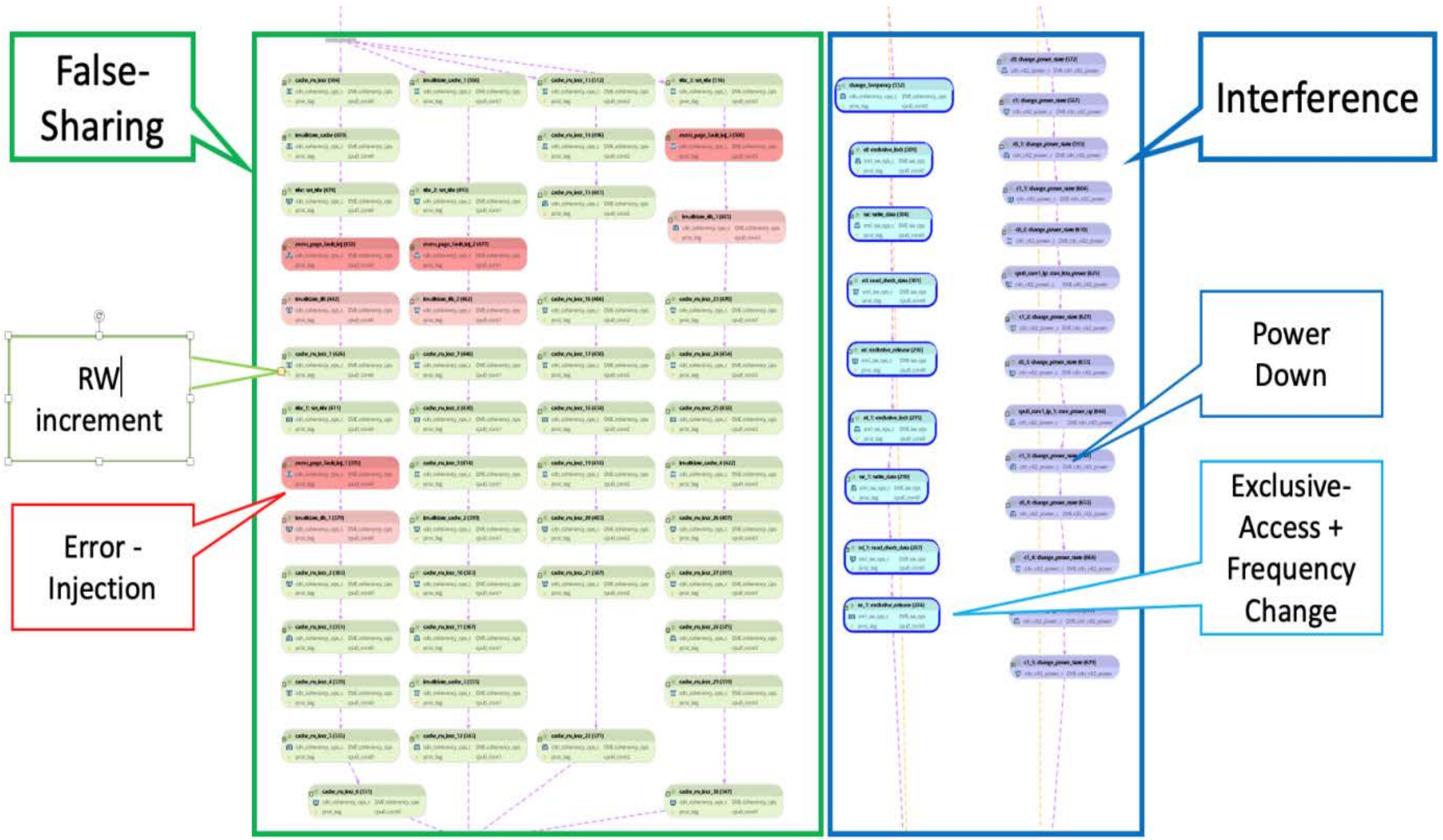
```
action activity_selection {
    activity {
        //Randomly select one of the choices:
        select{
            //Valid coherency actions:
            [85]: do read_increment_write;
            //Error injection options:
            // - RAS error injection (library)
            [5]: do cdn_coherency_ops_c::ras_core_error_inject;
            // - MMU translation fault generation
            [5]: do core_remap_ttbr_error_inject;
            // - Uncorrectable ECC error injection
            [5]: do ecc_memory_error_inject;
        }
    }
}
```

Interference Stimulus Generation

- Once the DFI is determined, the PSS selects an interference stimulus, which can be a coupling factor, to create a dependent failure scenario.

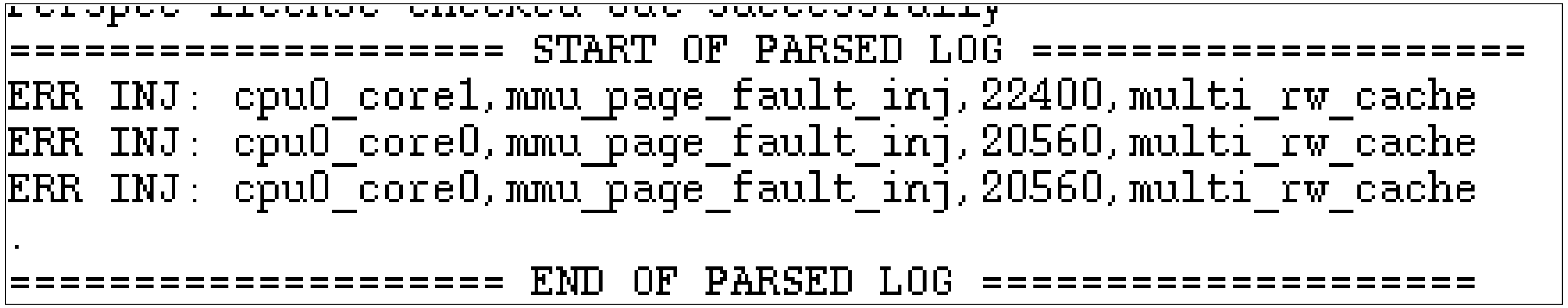
```
action false_sharing_with_err_injection_and_interference {
  activity {
    parallel {
      do false_sharing_with_err_injection;
      repeat (10) {
        select {
          do change_frequency;
          do cdn_coherency_ops::power_activity;
          do cdn_coherency_ops::exclusive_cache_access;
        }
      }
    }
  }
}
```

Generated Dependent Failure Scenario



Interference Reporting

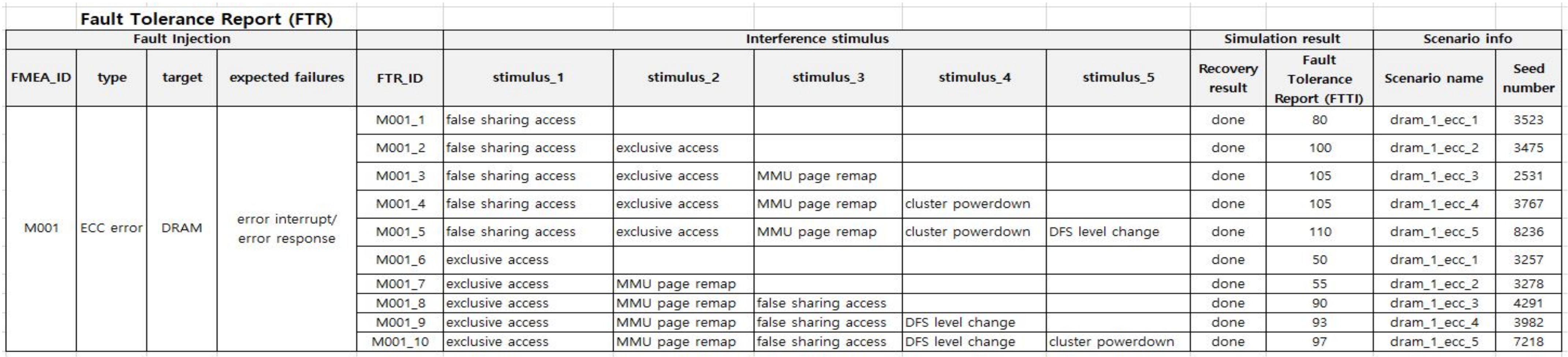
- Each scenario prints out the following information when simulation completes
  - Injected fault information
  - Executed interference action information
  - Maximum Fault Tolerance Time Interval (FTTI) information
  - External recovery monitor



```
Perspec license checked out successfully
===================== START OF PARSED LOG =====================
ERR INJ: cpu0_core1,mmu_page_fault_inj,22400,multi_rw_cache
ERR INJ: cpu0_core0,mmu_page_fault_inj,20560,multi_rw_cache
ERR INJ: cpu0_core0,mmu_page_fault_inj,20560,multi_rw_cache
.
===================== END OF PARSED LOG =====================
```

Fault Tolerance Report (FTR) Generation

- Using scenarios run results, an FTR is generated automatically

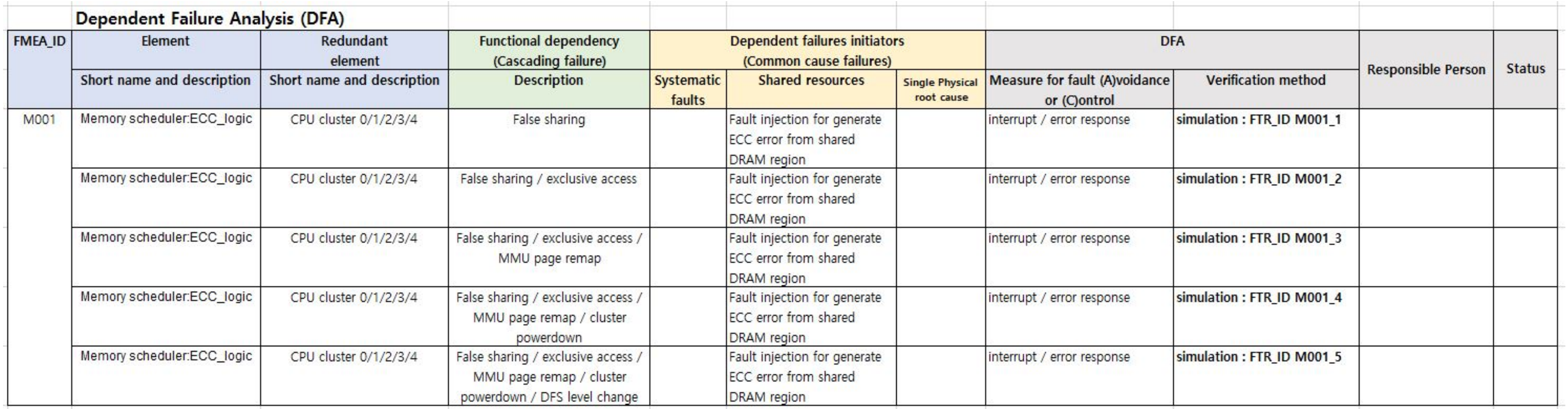


Fault Tolerance Report (FTR)

Fault Injection					Interference stimulus					Simulation result		Scenario info	
FMEA_ID	type	target	expected failures	FTR_ID	stimulus_1	stimulus_2	stimulus_3	stimulus_4	stimulus_5	Recovery result	Fault Tolerance Report (FTTI)	Scenario name	Seed number
M001	ECC error	DRAM	error interrupt/ error response	M001_1	false sharing access					done	80	dram_1_ecc_1	3523
				M001_2	false sharing access	exclusive access				done	100	dram_1_ecc_2	3475
				M001_3	false sharing access	exclusive access	MMU page remap			done	105	dram_1_ecc_3	2531
				M001_4	false sharing access	exclusive access	MMU page remap	cluster powerdown		done	105	dram_1_ecc_4	3767
				M001_5	false sharing access	exclusive access	MMU page remap	cluster powerdown	DFS level change	done	110	dram_1_ecc_5	8236
				M001_6	exclusive access					done	50	dram_1_ecc_1	3257
				M001_7	exclusive access	MMU page remap				done	55	dram_1_ecc_2	3278
				M001_8	exclusive access	MMU page remap	false sharing access			done	90	dram_1_ecc_3	4291
				M001_9	exclusive access	MMU page remap	false sharing access	DFS level change		done	93	dram_1_ecc_4	3982
				M001_10	exclusive access	MMU page remap	false sharing access	DFS level change	cluster powerdown	done	97	dram_1_ecc_5	7218

DFA Result

- The FTRs for each error generated in this way are reflected in the DFA result as shown below, proving that safety is guaranteed under various error conditions.



Dependent Failure Analysis (DFA)

FMEA_ID	Element	Redundant element	Functional dependency (Cascading failure)	Dependent failures initiators (Common cause failures)			DFA		Responsible Person	Status
	Short name and description	Short name and description	Description	Systematic faults	Shared resources	Single Physical root cause	Measure for fault (A)voidance or (C)ontrol	Verification method		
M001	Memory scheduler:ECC_logic	CPU cluster 0/1/2/3/4	False sharing		Fault injection for generate ECC error from shared DRAM region		interrupt / error response	**simulation : FTR_ID M001_1**		
	Memory scheduler:ECC_logic	CPU cluster 0/1/2/3/4	False sharing / exclusive access		Fault injection for generate ECC error from shared DRAM region		interrupt / error response	**simulation : FTR_ID M001_2**		
	Memory scheduler:ECC_logic	CPU cluster 0/1/2/3/4	False sharing / exclusive access / MMU page remap		Fault injection for generate ECC error from shared DRAM region		interrupt / error response	**simulation : FTR_ID M001_3**		
	Memory scheduler:ECC_logic	CPU cluster 0/1/2/3/4	False sharing / exclusive access / MMU page remap / cluster powerdown		Fault injection for generate ECC error from shared DRAM region		interrupt / error response	**simulation : FTR_ID M001_4**		
	Memory scheduler:ECC_logic	CPU cluster 0/1/2/3/4	False sharing / exclusive access / MMU page remap / cluster powerdown / DFS level change		Fault injection for generate ECC error from shared DRAM region		interrupt / error response	**simulation : FTR_ID M001_5**		

Agenda

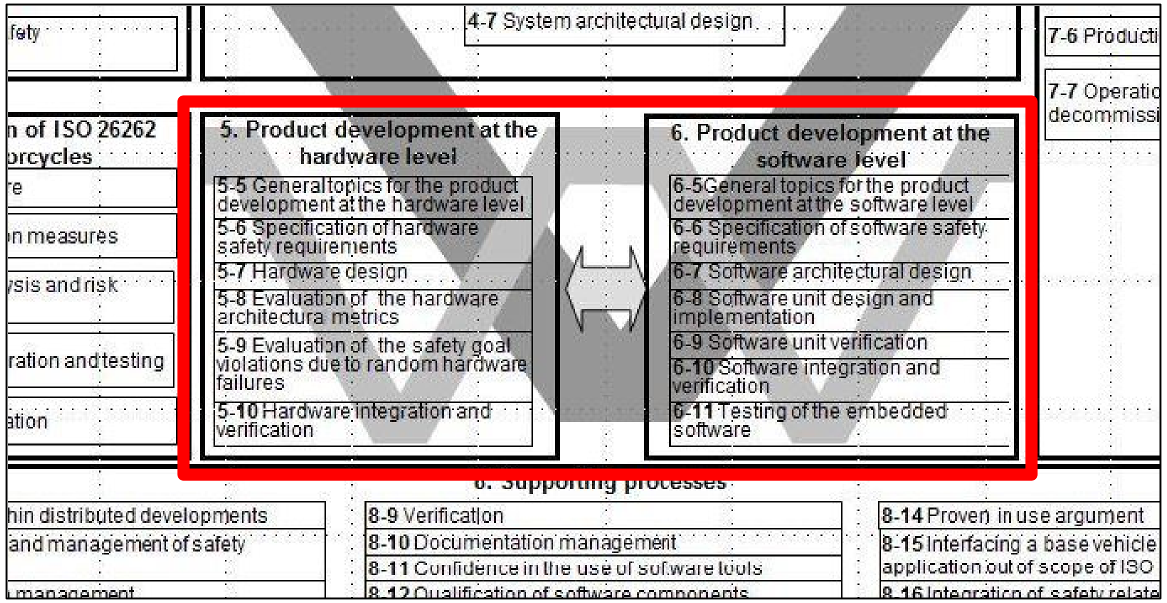
- Introduction to ISO 26262
- ISO 26262 functional safety features for semiconductor
- Using PSS for DFA (Dependent Failure Analysis)
- Conclusion and lesson learned

Conclusion

- Using PSS, we were able to create a number of DFIs, and use random fault injection scenarios to reproduce and prevent a number of dependent failure cases
- Through the DFA results, the verification coverage of our system has increased dramatically.
  - x10 number of additional verification items have been generated from each single FMEA item for shared resource

Lesson learned

- ISO 26262 can be usefully applied to the general SoC verification process as well as functional safety
- The same scenario could be used for SW development as well as HW development through the scenario reusability of PSS.



fety

4-7 System architectural design

7-6 Producti

7-7 Operatio decommissi

n of ISO 26262 orcycles

e

n measures

ysis and risk

ration and testing

ation

5. Product development at the hardware level

5-5 General topics for the product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the hardware architectural metrics
5-9 Evaluation of the safety goal violations due to random hardware failures
5-10 Hardware integration and verification

6. Product development at the software level

6-5 General topics for the product development at the software level
6-6 Specification of software safety requirements
6-7 Software architectural design
6-8 Software unit design and implementation
6-9 Software unit verification
6-10 Software integration and verification
6-11 Testing of the embedded software

8. Supporting processes

hin distributed developments
and management of safety
management

8-9 Verification
8-10 Documentation management
8-11 Confidence in the use of software tools
8-12 Qualification of software components

8-14 Proven in use argument
8-15 Interfacing a base vehicle application out of scope of ISO
8-16 Integration of safety relate

ISO 26262 Dependent Failure Analysis Using PSS

Moonki Jang – Samsung Electronics Co.,Ltd.

DVCon Slide Guidelines

- Use Arial or Helvetica font for slide text
- Use Courier-new or Courier font for code
- First-order bullets should be 24 to 28 point
  - Second-order bullets should be 24 to 26 point
    - Third-order bullets should be 22 to 24 point
    - Code should be at least 18 point
- Your presentation will be shown in a very large room
  - These font guidelines will help ensure everyone can read you slides!

~~No Company Logo except on title slide!~~

Code and Notes

Code should be enclosed in text boxes (using a background color is optional)

Code should be 18pt Courier-bold, or larger

```
module example
(input  logic foo,
 output logic bar
);

  initial begin
    $display (“Hello World!”);

endmodule
```

Informational boxes should be 18pt Arial-bold, or larger (using a background color is optional)