Early Detection of Functional Corner Case Bugs using Methodologies of the ISO 26262

Moonki Jang[1], Sunil Roe[2], Youngsik Kim[3], Seonil Brian Choi[4]
Samsung Electronics, 1-1, Samsungjeonja-ro, Hwaseong-si, Gyeonggi-do 18488, Korea
moonki.jang@samsung.com[1], sunil.roe@samsung.com[2], ys31.kim@samsung.com[3], seonilb.choi@samsung.com[4]

***Abstract*- ISO 26262[1] does not suggest a new verification methodology for systematic failures covered in design verification. However, ISO 26262 requires a persistent impact analysis and risk assessment at every stage. We introduced HARA (Hazard Analysis and Risk Assessment)[2] and safety analysis methodology used in ISO 26262 to the design verification field and used it to measure the risk of each element of the target design. We found risk factors that can represent the level of risk for detailed items. And, through FMEA (Failure Mode and Effect Analysis) and DFA (Dependent Failure Analysis)[3], it was possible to easily see the expected design vulnerabilities by classifying the risk factors that could affect the system level. And to verify the expected design vulnerabilities, a systematic fault model was created using the ML algorithm. Through this, it was possible to verify the time critical corner case, which has been difficult in pre-silicon so far.**

I. INTRODUCTION

The biggest change around us in recent years is probably the automobile sector. This is because the supply of electric vehicles is expanding and high-performance semiconductors and various advanced technologies are being integrated for complete autonomous driving. Driving, steering, and braking systems, which are core functions of automobiles, are already electronically controlled through built-in microcontrollers. In the future autonomous driving era, the driver's role will also be replaced by sensors and artificial intelligence.

However, the number of accidents due to the instability of electronic devices is also steadily increasing. This is because even a minor error that can be solved by simply rebooting in our mobile phone can threaten the life of the driver if it occurs while driving a car.

ISO 26262, an international standard for automotive functional safety, has been published to prevent damage caused by malfunctioning of such automobile functions. ISO 26262 provides clear standards for safety to be considered when developing functions that can be directly related to human life, and provides specific guidelines for effectively reducing risks due to the increase in technical complexity.

ISO 26262 classifies the causes of risk into systematic failure and random hardware failure.

Systematic failure is a failure that arises from the activity itself that develops and produces a system. Human error of personnel participating in development and production activities is the biggest cause. RTL bugs caused by incorrect design in the semiconductor design process are typical systematic failures.

Random HW failure is a term that is limited to HW elements and refers to failures that occur due to physical limitations of HW elements. This occurs mainly in the form of memory cell deterioration and instantaneous state change of logic caused by external environmental factors, and is not covered with in the design verification stage.

In other words, in order to ensure functional safety in ISO 26262, the following two requirements are required in the semiconductor design verification stage.

1) Avoid or prevent systematic failure by checking all errors in the design process.
2) Even if systematic failure occurs, there must be a plan to minimize the impact on the system.

ISO 26262's design verification requirements for systematic failure are not different from other fields. However, in the automotive environment covered by ISO 26262, a higher level of strict reliability is required than before because even a minor error can threaten human life. These high-level requirements of ISO 26262 are also affecting the semiconductor development process in other fields. Semiconductors are already expanding their scope of use from a role for human convenience to a variety of fields where human safety must be ensured. Accordingly, higher reliability than before is required in the entire semiconductor field.

In this paper, we created a methodology called Systematic Failure Analysis (SFA) for semiconductor design verification by referring to the safety analysis methodology of ISO 26262. And to verify the failure mode with complex conditions, the ML based PSS action sequence model [4] announced last year was applied. Our methodologies used the methodologies proposed in ISO 26262, but were made to be used to increase the reliability of the design verification stage in the general semiconductor field. At the end of this paper, we will also show how our results can be used again in ISO 26262.

II. SYSTEMATIC FAILURE ANALYSIS

ISO 26262 relies on the traditional design verification methodology to mitigate the risk due to the systematic failure. However, as system complexity increases, errors caused by unintended actions that occur during interactions between different components are increasing. As a result, fatal systematic failures with complex interaction conditions that are difficult to detect with existing verification methods are often found at the silicon level. In order to prevent this, it is necessary to identify the function and critical sequence in which such a corner case can occur in advance and predict the risk.

For this reason, we created Systematic Failure Analysis (SFA) to expand the functional verification coverage by extracting risk factors from the IP level and predicting risks.

Failure mode definition

Failure mode is created to predict possible failures during the function operation of the semiconductor component and to establish preventive measures against them. The failure mode created during design verification for SFA records the expected failure when a fault occurs due to the following causes during function operation. In SFA, failure mode is used as a verification target to ensure that it does not occur.

- FM1: Integration issues (connection, configuration...)
- FM2: Accessibility issues (access path, access control...)
- FM3: Functionality issues (wrong output, unintended behavior…)
- FM4: State transition issues (power gating, clock gating, reset…)
- FM5: Absence of independence or FFI (Freedom from Interference)

An example of the failure mode of SFA is as follows.

TABLE I
EXAMPLE OF FAILURE MODE

ID	IP	Function	Failure mode
CPUCL_F1	CPU	[CPD] CPU Cluster power down 1) All cores are power down 2) Coherency disconnect 3) L3 goes into power down 4) power gating for CPUCL 5) wakeup by external interrupt	CPU_CPD_FM1: wakeup interrupt can't delivered CPU_CPD_FM2: debugger can't access through debug path during CPD CPU_CPD_FM3: P-ch handshaking has failed CPU_CPD_FM4: core1 does not working after wakeup from CPD CPU_CPD_FM5: ACE interface stalled after snoop arrived between coherency disconnect and coherency disable
CPUCL_F2	CMU	[ACG] Automatic Clock Gating 1) All cores are in idle state 2) enable clock gating 3) blocking all external access 4) CPUCL clock gating	CMU_ACG_FM1: wrong clock pll ratio CMU_ACG_FM2: CMU configuration is no available CMU_ACG_FM3: Q-ch handshaking has failed CMU_ACG_FM4: incoming access occurred after clock down CMU_ACG_FM5: unintended clock gating occurred during CPU is in active state.

Among the types of failure modes above, 'FM5: Absence of independence or FFI' is an item related to functional corner cases that could not be verified in the existing design verification. Because this occurs due to interference access from the outside that occurred at a specific point of weakness in the function, it was difficult to reproduce in the existing verification environment.

And, FM5 failures discovered late at the silicon level lead to a huge cost increase due to revision, so it remains a big problem to be solved in the design verification field. In this study, the biggest reason we introduced the methodology of ISO 26262 and created the design verification methodology called SFA was to find vulnerabilities that could cause FM5 failure through systematic analysis.

To this end, in order to determine what external elements that have a dependency that can affect each function and what kind of interference they cause, and to check whether there is a vulnerable part that can be affected by external interference in the function, as follows I created a correlation table and hazardous function list.

1) Correlation table

The correlation table shows the list of incoming access that can occur during function operation and outgoing access that the function can affect externally. Through this, the dependency relationship with external elements can be identified when each function is operated. And DFA (Dependent Failure Analysis) analyzes whether there is a problem due to dependency between functions that can affect each other by referring to the correlation in the table below.

TABLE II
EXAMPLE OF CORRELATION TABLE

ID	IP	Function	Incoming		Outgoing	
			src	Access type	dst	Access type

CPUCL_F1	CPU	[CPD] CPU Cluster power down 1) All cores are power down 2) Coherency disconnect 3) L3 goes into power down 4) power gating for CPUCL 5) wakeup by external interrupt	CPUCL2 CPUCL2 GPU PCIe GIC Debugger	Snoop transaction DVM transaction Snoop transaction Snoop transaction Interrupt Debug CMD	PMU BUS	Power gating config SYSCOREQ handshaking
CPUCL_F2	CMU	[ACG] Automatic Clock Gating 1) All cores are in idle state 2) enable clock gating 3) blocking all external access 4) CPUCL clock gating	CPUCL2 CPUCL2 GPU PCIe GIC Debugger	Snoop transaction DVM transaction Snoop transaction Snoop transaction Interrupt Debug CMD	CMU	Clock gating config.

2) Hazardous function list

The table below shows the functions in which time critical corner case bugs such as the FM5 failure case occur most empirically so far. In the function list below, it can be seen that functions including state transition have a critical sequence in which external interference is not allowed, and many vulnerabilities have occurred in this part. The critical sequence of hazardous functions with such vulnerabilities should be identified in advance through impact analysis at the design stage and included in verification requirements.

TABLE III
HAZARDOUS FUNCTION LIST

Function	Vulnerable sequences
Power gating	-Interference between gating enable configuration and power gating -Receiving snoop between coherency disconnect phase and coherency disable phase
Clock gating	Interference between gating enable configuration and clock gating
Reset	Abnormal state can’t restored after reset
Buffer overflow	Read/write channel stalled and data lost due to buffer overflow.
Exception occurrence	Interference occurred during exception handling

Accordingly, the FM5 items can define the failure mode by assuming that the incoming access of the correlation table for the above hazardous function is interference occurring in the critical sequence.

Risk assessment process

Risk in ISO 26262 is evaluated by the severity and probability of failure. However, in SFA in the design verification stage, risk refers to the possibility that fatal systematic failure occurs at the silicon level. To evaluate this level of risk, we defined the following risk factors at the architecture and design stage.

1) Proven in use level[5]

If the IP related to the function operation in the failure mode is a new IP that has not been used before or is modified to add a function, systematic faults due to design errors are highly likely to occur. In order to evaluate the risk, the following proven in use level was reflected in each failure mode..

TABLE IV
PROVEN IN USE LEVEL

Proven in use level (Risk Point)	Description
P4 (4)	Implement new In-house IP / logic that has not been used
P3 (3)	Implement new 3rd party IP / logic that commonly used on the industry
P2 (2)	Change version of used IP/logic
P1 (1)	Implement same IP / logic that already used on another product

2) Severity level

In order to indicate the severity of systematic failure defined in the failure mode, it was classified into severity levels as follows. In the case of S4, it means the most severe state in which the entire system falls into a disabled state due to the occurrence of failure. And S3 means that the function falls into an impossible state due to failure, but the rest of the system is operable.

TABLE V
SEVERITY LEVEL CLASSIFICATION

Severity level (Risk Point)	Description
S4 (4)	Entire system will not be available due to critical failure

S3 (3)	Function element will not be available but no system wide impact.
S2 (2)	Performance degradation is expected due to failure
S1 (1)	No impact

3) Known issues in other project

It indicates whether systematic failure defined in failure mode has a history of causing bugs in other projects. This is to check whether the bug has been properly corrected or the avoidance method has been properly applied. This includes the contents of the errata notice provided by the IP provider. In general, the errata notice provided by the IP provider has a very rare condition, but we consider this as one of the risks to be checked in the design verification stage and verify whether there is a risk in our system.

4) Applicable workaround

Indicates whether systematic failure defined in failure mode can be avoided by software workaround. However, SW workaround always affects the performance or function, except when backup IP/logic is prepared or another function can replace the failure occurrence function. This item is determined by the software engineer, and the software engineer records the SW high level function related to the failure mode in the FMEA form for the software DFA.

TABLE VI

APPLICABLE WORKAROUND

W/A level (Risk Point)	Description
W4 (4)	No W/A. H/W revision is required.
W3 (3)	Software W/A is available but function will not be available
W2 (2)	Software W/A is available with performance impact
W1 (1)	Software W/A is available without any performance impact

4) Risk definition of SFA

The risk level for failure mode in SFA is calculated as follows through the risk point of the items shown above.

$$Risk\ Level = \frac{P(4..1) + S(4..1) + H(4,0) + K(4,0)}{W(4..1)}$$

- (P: Proven in use level, S: Severity level, H: Hazardous function, K: Known bug, W: Available SW workaround)

- Figure 1. Calculation of Risk Level

Using the risk level obtained in this way, we created a new SFSL (Systematic Failure Severity Level) as follows. ISO 26262's ASIL (Automotive Safety Integrity Level) [6] indicates the functional safety level guaranteed by the system, but SFSL indicates the risk of fatal failure. We can determine which failure mode to prioritize verification by using the SFSL obtained through this risk assessment process. In addition, additional verification of the effects that may occur due to the failure is also performed by referring to the SFSL.

TABLE VII

DEFINITION OF SFSL LEVEL

SFSL	Level Definition	Description
SFSL_A	Risk Level > 12	Very high risk of critical failures. Detailed verification is required
SFSL_B	Risk Level > 8	High risk of critical failures. Additional verification is required
SFSL_C	Risk Level > 4	Mid risk of critical failures. Impact analysis is required
SFSL_D	Risk Level <= 4	Low risk of critical failures.

FMEA (Failure Mode and Effect Analysis)

FMEA is one of the safety analysis methodologies of ISO 26262 to analyze the effect of failure mode on system function. In SFA, FMEA is used to analyze how critical each failure mode is.

The following shows the results of FMEA conducted by SFA. The systematic failure mode obtained from the function list and the final risk level calculated by the risk factor make it possible to easily identify the expected weakness of the design to be verified.

	FMEA											
Name / Function		Potential Failure Mode(s)	Potential Cause of Failure	Potential Effect of Failures	Risk Assessment						Related high level functions	Occurance Conditions
ID	Requirements				SFSL	P	S	H	K	W		
CPU_CPD_FM3	CPUCL_F1	P-ch handshaking has failed	wrong connection of P-CH interface	Power mode transition does not working	B	3	3	Y	N	1	SYSTEM idle/sleep mode	try power mode transition
CPU_CPD_FM5	CPUCL_F1	ACE interface stalled after snoop arrived between coherency disconnect and coherency disable	reported Errata: 1500609	Deadlock occurred between CPUCL and BUS	A	3	4	Y	Y	1	SYSTEM sleep mode	Generate snoop between SYSCOREQ and SYSCOACK
CMU_ACG_FM1	CPUCL_F2	wrong clock pll ratio	wrong PLL configuration	generate wrong clk out	D	1	2	N	N	3	Normal active mode	Check clk after CMU init
CMU_ACG_FM5	CPUCL_F2	unintended clock gating occurred during CPU is in active state	Interference occurred between clk gating sequence	Deadlock occurred due to incompleted transactions	A	2	4	Y	Y	1	SYSTEM sleep mode throttling enable	Access CPUCL0 register between cpucl0_clk_gating_en and cpucl0_clk_blocking_ext_en

Figure 2. Example of FMEA

DFA (Dependent Failure Analysis)

DFA is performed to investigate the effect of failure modes defined in FMEA on each other. Dependent failure is a failure that is influenced by other failures, rather than failures that occur independently. Dependent failure defined in DFA can be classified into Cascading Failure and Common Cause Failure as shown in the figure below.

-Cascading Failure: Failure of an element, resulting from a root cause, and then causing a dependent failure of another element or elements

-Common Cause Failure: Dependent failure of two or more elements, resulting directly from a single specific event or root cause

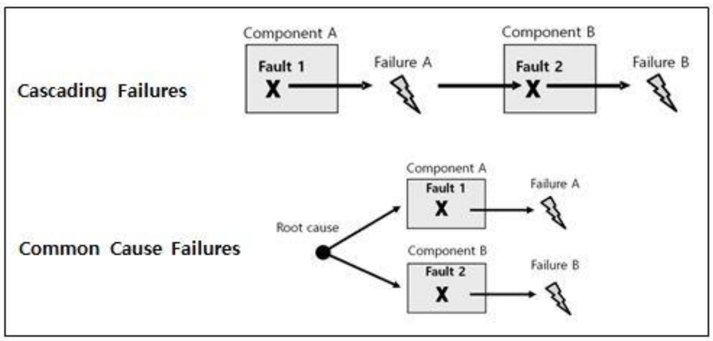

Figure 3. Type of dependent failures

There are two things we want to check through DFA:

-Check whether the failure mode defined in FMEA can be a Dependent Failure Initiator (DFI) that can cause a cascading failure

-Check whether a common cause failure occurs in multiple elements due to a fault generated in a shared resource through fault injection

1) Cascading failure caused by failure mode of FMEA

In order to check whether the failure mode defined in FMEA can act as a DFI that causes cascading failure in other elements, it is necessary to identify the components and functions affected by this failure. To this end, we were able to figure out the dependency relationship between the functions of each component using the correlation table used in FMEA.

2) Common cause failure caused by fault injection

In ISO 26262, fault injection is used for DC (Diagnostic Coverage)[7] measurement, which indicates how much the generated fault can be detected by a safety mechanism. SFA generates a DFI that can occur due to a random hardware fault by generating a fault in the shared resource using the backdoor interface of the memory.

Common cause failures checked here are as follows:

- Whether failure occurs due to DFI in multiple components accessing shared resources and whether normal recovery is possible through error detection logic

- When a component is operating error handling due to a fault occurring in the shared resource, check whether failure occurs in other components due to the coupling factor

This content has already been covered in 'ISO 26262 Dependent Failure Analysis using PSS' presented at DVCon US 2019[8], and we perform fault injection using the backdoor interface to the shared resource below.

- Uncorrectable ECC error injection: Manipulates data in shared region to generate ECC error when accessing

- MMU translation fault generation: Manipulates the valid field of the MMU page descriptor to be invalid, causing a translation fault to occur during table walk for the address.

- RAS error injection for CPU, Interrupt controller, System MMU: The RAS (Reliability, Availability, and Serviceability) extension is ARM's error handling and recovery architecture that is applied from the ARMv8 architecture. RAS has its own fault injection model extension that can inject faults into the L1 / L2 / L3 caches to generate error interrupts

The DFIs generated by fault injection listed here are random hardware faults, but logic to avoid this kind of random hardware fault is generally applied at the design stage. We performed verification using this fault injection to find systematic failures that may occur during fault handling of the corresponding logic.

The coupling factors that can cause failure in other elements in the fault state caused by the above DFI are as follows:

- False sharing access
- DVM (Distributed Virtual Memory) transaction broadcasting
- Exclusive access

We created a fault injection model in the PSS environment to check whether failure can occur in other elements due to the coupling factor in a state where a fault occurred in the shared region and recorded the result in the form of FTR as follows. This fault model is used not only for design verification, but also for verification of recovery operation in SW.

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

Figure 4. Example of FTR (Fault Tolerance Report)

The DFA result for the above cascading failure and common cause failure is included in the SFA result in the form below.

	Dependent Failure Analysis (DFA)						
FMEA_ID	Element	Redundant Element	Functional Dependency	Dependent Failure Initiator(DFI)		DFA	Verification Method
	Short name and description	Short name and description	Description	Systematic fault	Shared resource	Expected Dependent Failure	
CPU_CPD_FM5	BLK_CPUCL0	BLK_GPU	CPU should wake up GPU for requested GPU processing	Stalled ACE interface of CPU		GPU can't wakeup and system hang occurred	PSS_ML_fault_model
	BLK_CPUCL0	BLK_PCIe	PCIe will send posted write request and it will generate snoop to CPUCL0	Stalled ACE interface of CPU		PCIe will not available. Posted write will wait for snoop response from CPU.	PSS_ML_fault_model
MIF_FAULT_1	Memory scheduler:ECC logic	BLK_CPUCL0	False sharing		ECC error generated from shared DRAM region	CPU will access fault address during ECC error state	PSS_fault_injection_model
	Memory scheduler:ECC logic	BLK_CPUCL0	exclusive access		ECC error generated from shared DRAM region	CPU will access fault address during ECC error state	PSS_fault_injection_model

Figure 5. Example of DFA result

III. SYSTEMATIC FAULT MODEL GENERATION USING MACHINE LEARNING

We were able to obtain failure modes with complex conditions with high risk expected through FMEA and DFA. However, it is very difficult to reproduce systematic failure that occurs at the moment when a specific condition of a plurality of elements is satisfied in the design verification stage. This is because this failure mode is a form that cannot be recognized by a method that verifies the expected output for the input. Until now, this type of failure was mainly discovered through long-term random tests at the silicon product level, which increased the semiconductor design cost due to mask revision.

In order to create a fault model to reproduce this failure mode, we needed an environment that could generate the operation of each element at the precise time we wanted.

In DVCon US 2022, we have already presented the process of finding the time when the desired condition occurs by inserting a configurable delay using the ML algorithm.

TB structure of systematic fault model

As shown below, the functions are executed at the same time from the SW point of view, but in the actual HW, they are operated at different times. Configurable delay compensates for this difference.

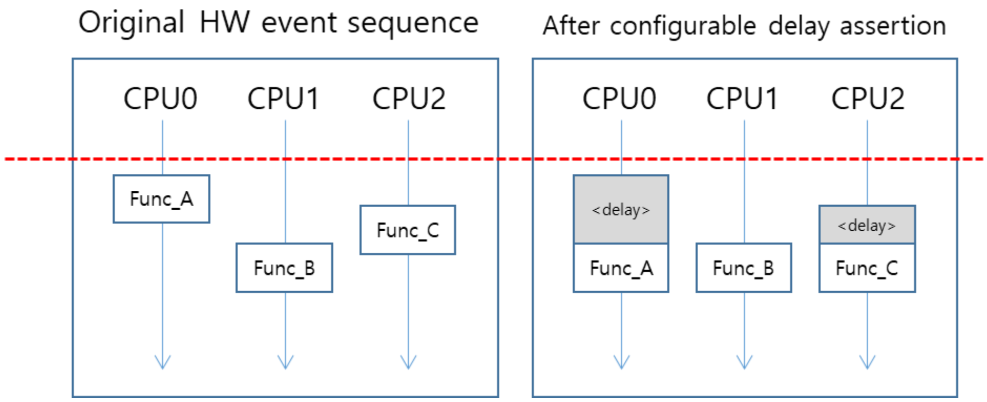

Figure 6. Concept of the ML based PSS action sequence model

The figure below shows the delay counter that counts the configurable delay in testbench.

There are two reasons for not implementing delay in SW. First, the minimum unit of delay that can be generated in SW is microsecond. We need a delay counter with clock cycle resolution because we need to detect the cross condition of HW event. And secondly, in the case of SW delay, the applied delay time may change depending on the state of the CPU cache or instruction pipeline. In order to apply the ML algorithm, which will be discussed later, the applied delay must have a linear characteristic. On the other hand, the delay counter implemented with SV code in testbench performs counting every rising edge of clock. And when the delay count as much as the requested number of clocks is completed, an interrupt is generated to the CPU core that requested the delay, so that the requested delay value and the actual delay have a linear characteristic.

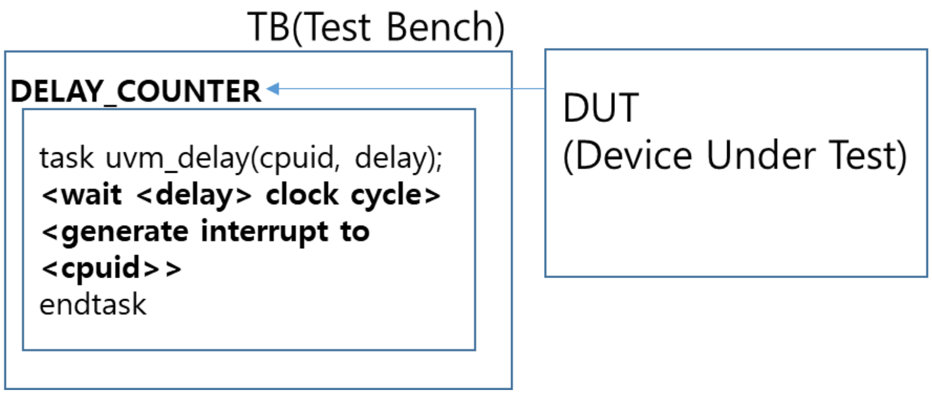

Figure 7. Delay counter

The figure below shows how to detect a target condition and how to store this information. The output monitor checks whether a preset target condition occurs, and when the corresponding condition occurs, the simulation time information and target condition information are exported to the simulation log. After the simulation is finished, the output log goes through the phasing process and is saved in a separate output repository.

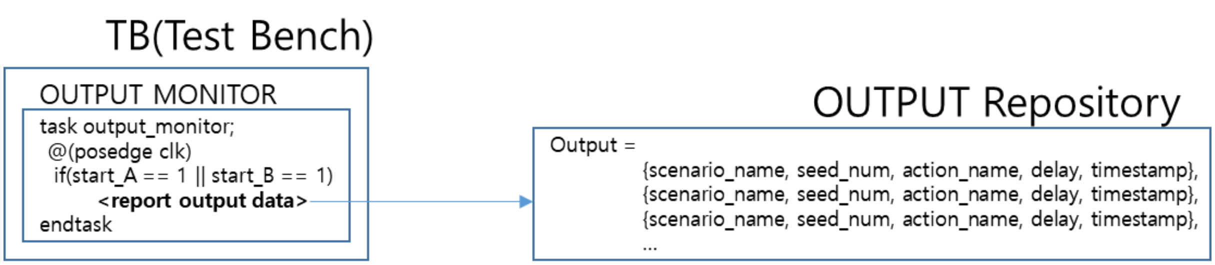

Figure 8. Output monitor and Output repository

The last shows the CPU SW that executes the function. We inserted an action called pss_delay in front of each function using the PSS tool and performed simulation by generating SW codes with different delay values. Pss_delay action transmits the delay request to the delay counter of the testbench through a separate mailbox interface as shown in the figure and waits for the interrupt from the delay counter.

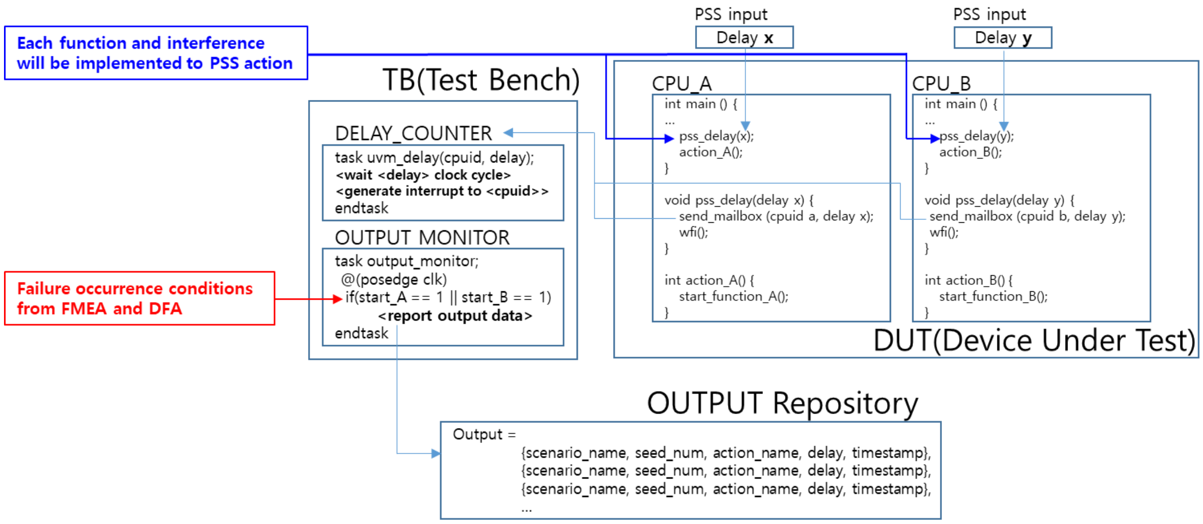

Figure 9. Verification with ML based PSS action sequence model

ML(Machine learning) implementation

Before applying ML, it was repeated to set the delay value, run the simulation, and adjust the value again according to the result. This was an inefficient operation that required a lot of time and human resources. To improve this, we applied the linear regression algorithm, which is the most basic algorithm of ML. First, create a code with a certain range of delay values for function_A and function_B and conduct simulation to examine the simulation timestamp value where the target condition occurred for each. One coordinate can be created by the combination of the input delay value and the simulation timestamp where the actual function is executed, and the linear equations for function_A and function_B can be obtained using the obtained coordinates. Through this, we can predict the simulation time at which the actual target condition occurs according to the delay value. If the extended Euclidean algorithm is applied to these two linear equations, innumerable combinations of delay values of function_A and function_B having a common simulation timestamp value are obtained.

Diophantine equation : ax + by = c

(Euclidean algorithm)

$a = bq_0 + r_1$ (q_x : quotient(=a/b) , r_x : remainder(=a%b))

$b = r_1 q_1 + r_2$

$r1 = r_2 q_2 + r_3$

...

$r_{i-1} = r_i q_i + r_{i+1}$ *(When r_{i+1} is 0, the algorithm terminates and r_i becomes GCD(greatest common divisor))*

-> $r_{i+1} = r_{i-1} - r_i q_i$ *(1)*

(Extended Euclidean algorithm)

In this case, if the coefficient of *a* is s_i and the coefficient of *b* is t_i for any r_i, it can be expressed as follows.

$$r_i = s_i a + t_i b$$

Substituting this into (1), we get the following equation.

$$s_{i+1}a + t_{i+1}b = (s_{i-1}a + t_{i-1}b) - (s_i a + t_i b)q_i$$
$$= s_{i-1}a - s_i a q_i + t_{i-1}b - t_i b q_i$$
$$= (s_{i-1} - s_i q_i)a + (t_{i-1} - t_i q_i)b$$

s and *t* are obtained as follows by increasing *i* until r_{i+1} becomes 0.

$$r_0 = a,\ r_1 = b,\ s_0 = 1,\ s_1 = 0,\ t_0 = 0,\ t_1 = 1$$
$$r_{i+1} = r_{i-1} - r_i q_i$$
$$s_{i+1} = s_{i-1} - s_i q_i$$
$$t_{i+1} = t_{i-1} - t_i q_i$$

Figure 10. Extended Euclidean algorithm for Diophantine equation

The Extended Euclidean algorithm [9] finds the greatest common divisors of *a* and *b*, as well as *x* and *y* values that satisfy the equation. The solution obtained in this way is called a particular solution, and using this, a general solution can be obtained that can find numerous combinations of *x* and *y* depending on the integer *k* value.

- Particular solution (d : GCD of a and b)

$$X_0 = s \text{ x } c/d$$
$$Y_0 = t \text{ x } c/d$$

- General solution (k : integer value)

$$x = x_0 + k * b/d$$
$$y = y_0 - k * a/d$$

Figure 11. Result of Extended Euclidean algorithm

However, unexpected external factors that affect the delay value may occur in the actual system. And this may break the linearity between the input delay and the simulation timestamp where the target condition occurred. To compensate for this, we performed learning only on data with linearity in the analysis stage through the ML flow as shown below and corrected the result value through the ML algorithm.

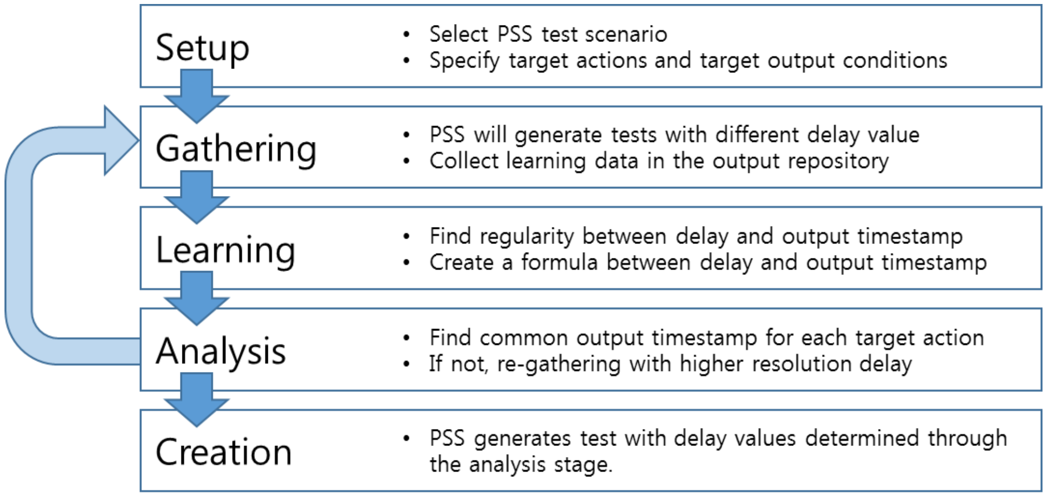

Figure 12. ML sequence modeling flow

As a result, we were able to obtain a verification code that can reproduce the target condition through ML-based regression only by setting the target condition that each element should have.

IV. SFA RESULT EXPORT TO ISO 26262

The results of SFA obtained through the process so far are as follows.

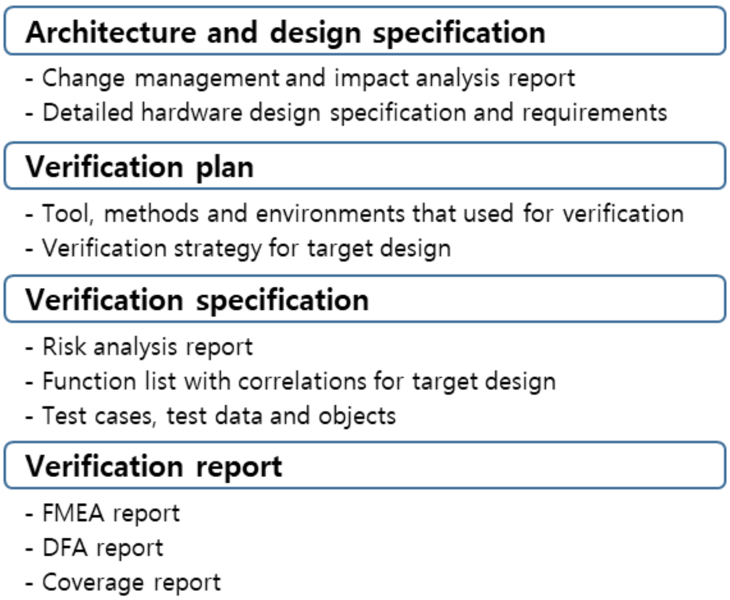

Figure 13. Result of SFA

The above results were classified according to the verification work products required by ISO 26262-8:2018, 9.5.[10] Looking at the contents, it can be seen that most of the contents are covered by the existing design verification.

In particular, in the case of the coverage report, new cross function bins between the dependent elements that were not previously covered were added through correlation table, risk factor analysis, and DFA. In addition, through the ML algorithm-based fault model, the critical sequence condition of the hazardous function could also be covered through simulation. In the case of CPU, it was possible to increase the number of functional coverage bins by about 15% only for high-risk items of SFSL-B or higher through SFA.

The results of this SFA are used in the risk assessment process for systematic faults in the safety analysis process of ISO 26262. And it is used to create the FMEDA report required by ISO 26262 by adding the failure rate and diagonal coverage results for random hardware faults to the FMEA and DFA reports of the SFA.

What we want to show here is that the activities we have been doing in the design verification field to prevent systematic faults can be independently applied and utilized to standards requiring design reliability such as ISO 26262 and IEC 61508[11] through the SFA. This could be the key to simplifying the requirements driven verification process, which has recently been attracting attention.

V. Conclusion

As the density of semiconductors increases and the complexity of their functions increases, systematic failures with complex conditions that were not detected in the design verification stage are more often found at the silicon product level. We thought that in order to detect systematic failure in the pre-silicon design verification stage, an inductive (bottom-up) approach is needed to find and analyze the risk factors from the smallest functional unit and broaden the scope to find system vulnerabilities.

In this paper, we introduced HARA (Hazard Analysis and Risk Assessment) and safety analysis methodology used in ISO 26262 to the design verification field and used it to measure the risk of each element of the target design. We found risk factors that can represent the level of risk for detailed items. And, through FMEA and DFA, it was possible to easily see the expected design vulnerabilities by classifying the risk factors that could affect the system level. To verify the expected design vulnerabilities, a systematic fault model was created using the ML algorithm. Through this, it was possible to verify the time critical corner case, which has been difficult in pre-silicon so far.

ISO 26262 does not suggest a new verification methodology for systematic failures covered in design verification. However, ISO 26262 requires a persistent impact analysis and risk assessment at every stage. This is because all fatal failures start with minor faults, and it is important to predict and prevent them in advance. Systematic failures dealt with in our design verification field are especially caused by human errors in the design process. To prevent this, the establishment of a more rigorous risk assessment and risk prediction process will become an essential element for semiconductor production requiring high reliability in the future

References

[1] ISO-26262:2018 “Road vehicles –Functional safety“
[2] ISO-26262-3:2018 “Road vehicles –Functional safety- part3, 6. Hazard analysis and risk assessment”
[3] ISO-26262-11:2018 “Road vehicles –Functional safety- part11, 4.7 Semiconductor dependent failures analysis”
[4] Jang, M., Hyun, M., Ahn, H., Kim, J., Kim, Y. (DVCon 2022) PSS action sequence modeling using Machine Learning
[5] ISO-26262-8:2018 “Road vehicles –Functional safety- part8, 14 Proven in use argument”
[6] ISO-26262-9:2018 “Road vehicles –Functional safety- part9, Automotive Safety Integrity Level”
[7] ISO-26262-5:2018 “Road vehicles –Functional safety- part5, annex D, Diagnostic coverage
[8] Jang, M., Kim, J., Kim, D. (DVCon 2020) Dependent Failure Analysis using PSS
[9] https://www.math.cmu.edu/~bkell/21110-2010s/extended-euclidean.html
[10] ISO-26262-8:2018 “Road vehicles –Functional safety- part8, 9.5 Verification work product”
[11] IEC 61508-2:2010 “Functional safety of electrical/electronic/programmable electronic safety-related systems.