Functional Safety Workflow of Internal IP (NPU) Within Large Automotive IC Through Analysis and Emulation Usage for SW-based Safety Mechanisms

Andrey Likhopoy, Samsung Electronics, South Korea, a.likhopoy@samsung.com
Sangkyu Park, Samsung Electronics, South Korea, sangq.park@samsung.com
Hyeonuk Noh, Samsung Electronics, South Korea, hyeonuk.noh@samsung.com
Wonil Cho, Samsung Electronics, South Korea, wonil.cho@samsung.com
Inhwan Kim, Siemens EDA, South Korea, inhwan.kim@siemens.com
Robert Serphillips, Siemens EDA, United States, robert.serphillips@siemens.com
Chanjin Kim, Siemens EDA, South Korea, chanjin.kim@siemens.com
Justin Lee, Siemens EDA, South Korea, justinlee@siemens.com
James Kim, Siemens EDA, South Korea, james.kim@siemens.com
Sougata Bhattacharjee, Samsung Semiconductor India Research (SSIR), India, sougata.b@samsung.com
Gulshan Kumar Sharma, Samsung Semiconductor India Research (SSIR), India, gks.92@samsung.com
Akshaya Kumar Jain, Samsung Semiconductor India Research (SSIR), India, akshaya.jain@samsung.com

Abstract – As the number of semiconductors used in modern cars dramatically increases, safety is becoming the dominating factor in the automotive semiconductors' lifetime. Achieving the desired safety level is burdened by the runtime of fault injection and data management over the duration of the fault campaign. In this paper we describe the process of data management and fault injection runtime optimizations that reduce the time to achieve the desired safety level and the product certification. The paper gives a case study of the fault campaign flow for an internal IP (NPU) in a large automotive IC by using the emulation for software-based (SW-based) safety mechanisms.

I. Introduction

According to the ISO 26262 standard functional safety is the absence of unreasonable risk due to hazards caused by malfunctioning behavior of electronic systems. Safety manager handles hazard analysis and risk assessment (HARA). At this stage each hazard event (HE) is assigned with the Automotive Safety Integrity Level (ASIL) based on severity of classification such as potential harm (S), probability of the operational situation (E), controllability by the driver or other traffic participant (C) [1]. Then for each HE, Safety manager defines the safety goals and sets the functional safety requirements needed to fulfill those safety goals. During the product development the functional safety requirements comes down to the technical safety requirements and further to the hardware (HW) safety requirements and the software (SW) safety requirements.

Based on the assumption of use, a certain IP inside an automotive SoC, which contributes to a certain safety goal is assigned with the needed safety requirements. The safety mechanisms (SM) are integrated into IP and are intended to verify the associated safety requirements.

To prove the effectiveness of SMs, the diagnostic coverage (DC) is measured. The DC expresses the capability of SM to discover the faults and is calculated by dividing the number of failures detected by SM by the total number of failures in the electronic systems. ASIL determines single-point fault metric (SPFM) and the level of coverage provided by SM for single-point fault and residual fault. For example, to satisfy ASIL B, SPFM greater than or equal to 90% is required (Table I).

TABLE I
SPFM VALUE FOR EACH ASIL

	ASIL A	ASIL B	ASIL C	ASIL D
SPFM	Not Applicable	≥90%	≥97%	≥99%

After the SM information with the expected DC and the expected protection area is defined the fault campaign (FC) process continues to the identification of the alarm net and the observe net. The next step is to determine and generate the test scenarios followed by fault injection. Workflow for fault injection has three steps: 1) generate the fault list, 2) inject the fault list, and 3) report the metrics.

Siemens provides a smooth flow for all the above steps and the single Functional Safety Database (FuSa DB) is used throughout [2].

II. NPU SAFETY VERIFICATION WITH SAFETY MECHANISMS

The NPU IP (Fig.1) is a part of an automotive SoC and contributes to safe video sensing.

NPU provides fast and energy-efficient hardware accelerations for AI and ML applications in computer vision, image processing and other domains where state-of-the-art deep learning networks are used. NPU consists of several processing blocks (PU0, PU1, and PU2), memory access block (MEM) and controller block (CTRL).

Safety Island is designed to manage and monitor all of the safety content within the automotive SoC. In particular, Fault Management Unit (FMU) in Safety Island collects faults and reports related safety event information.

The list of SMs (Table II) is defined by Safety Manager in the Statement of Request for Verification.

Typically memory parts are protected by parity or ECC (SM9-SM13). Related fault signals are connected to FMU directly. While logic parts often require SW-based SMs like self-test (SM2, SM3), where a firmware is needed to control the intended function, so the firmware checks the behavior and asserts fault signals.

Majorly FC for SMs protecting memory parts was done using a simulation flow. This paper deals with FC for SW-based SMs which was conducted using an emulation flow.

The goal was to verify the DC defined at the level of 90% (ASIL B). Considering the margin of error of 1.21%, the expected DC should be greater than or equal to 91.21%.

Failures In Time (FIT) rate metric proportional to the number of transistors in the die was used. Based on FIT rate distribution for logic and memory parts of each block inside NPU, the expected DC for each block was estimated. So, it was decided to use SM3 (Table III), SM1, SM2 (Table IV), as mandatory SMs and other SMs as additional as their contribution to the overall DC score is insignificant. For SM3 which protects all blocks the expected DC was set to 91.21%, and for SM1, SM2 which protect CPU inside CTRL block the expected DC set to 60% was sufficient. The final DC for NPU can be calculated by the weighted average of the DCs for all logic and memory parts.

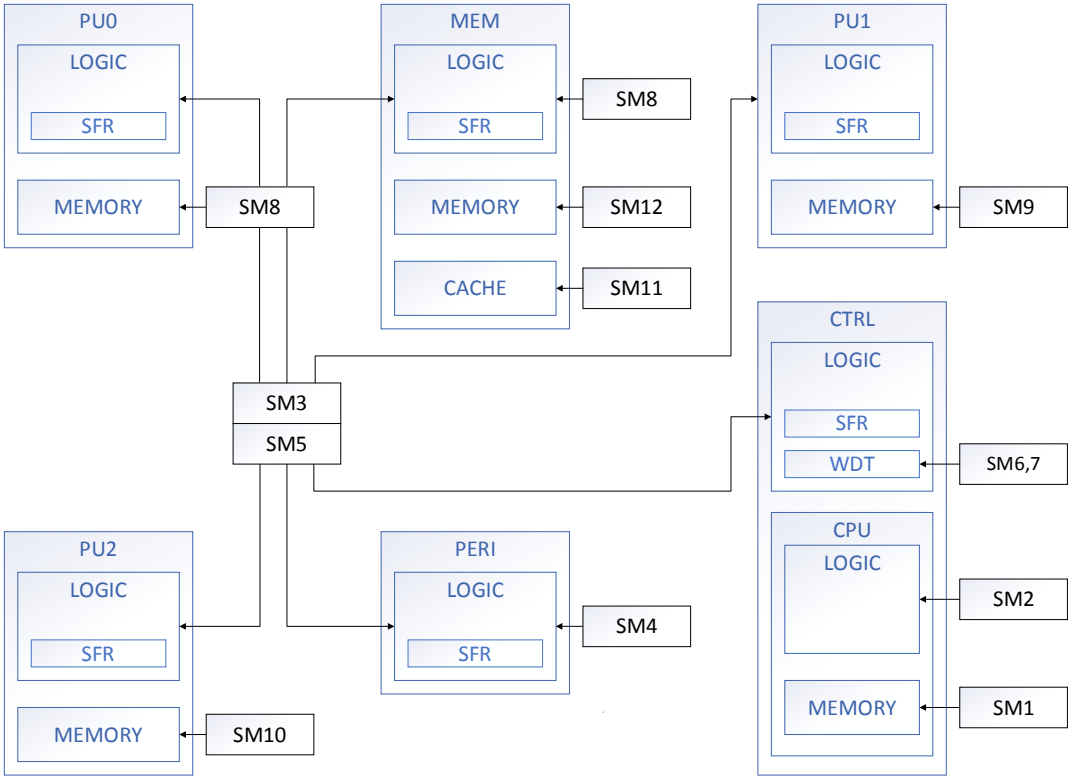

Figure 1. Block diagram of NPU with related SMs.

TABLE II
SMS TARGETED FOR EMULATION AND SIMULATION

Type	SM	Part	Description
emulation	SM1	CPU memory	ECC
	SM2	CPU logic	STL
	SM3	PU0/1/3, MEM, CTRL, PERI	Self-test
	SM4	CTRL, PERI (bus)	Ext mem access
	SM5	SFR of all parts	SFR access
	SM6	CTRL	WDT
	SM7	CTRL	WDT SFR
	SM8	MEM	CRC
simulation	SM9	PU0 memory	Parity
	SM10	PU1 memory	Parity
	SM11	PU2 memory	Parity
	SM12	MEM cache memory	Parity
	SM13	MEM memory	Parity

TABLE III
EXPECTED DC FOR SM3 (SELF-TEST)

Block	FIT rate distribution (%)	Expected DC (%) (≥91.21%)
PU0	31.35	96
PU1	31.35	96
MEM	17.9	90
PU2	10.86	80
CTRL	2.04	60
PERI	6.48	80
Total	100	91.41

TABLE IV
EXPECTED DC FOR SM1, SM2 (CPU)

Part	FIT rate distribution (%)	Expected DC (%) (≥60%)
CPU memory	1.02	90
CPU logic	98.98	60
Total	100	60.31

III. EMULATION METHODOLOGY

To facilitate the FC for complex SW-related SMs and to make DC verification feasible within achievable time HW assisted verification tools, like an Emulator, are required. Since emulation was one of the options to be used for functional verification of NPU, it is easy to add additional scenarios for the target DC with the existing emulation verification environment. If the target design is big, and the scenarios needs to be long, the emulation FC is the best option in terms of runtime. To set up the environment of the FC, the fault list generation, the fault injection run, and the database for fault list and the results are required. As a few separated steps for the FC are required, Siemens closed-loop tool chain is chosen for the NPU FC.

The fault list generated by SafetyScope, the RTL, and the fault detection nets (FDNs) are the inputs to Compile DB [3]. The generated fault list is stored into the FuSa DB. Compilation by Veloce refers the FuSa DB for the fault list, and the file for FDNs. FDNs are those nets whose values are stored during a Golden Run and compared during a Check Run. There are two types of FDNs: alarm and observe net. Alarm net is a signal that SM can recognize when there is a safety violation. Observe net is a signal that Veloce can recognize when there is a safety violation. Veloce Fault App is used to inject fault on the fault list. Finally, the result of the fault injection run is stored into the FuSa DB. Workflow is presented in Fig. 2.

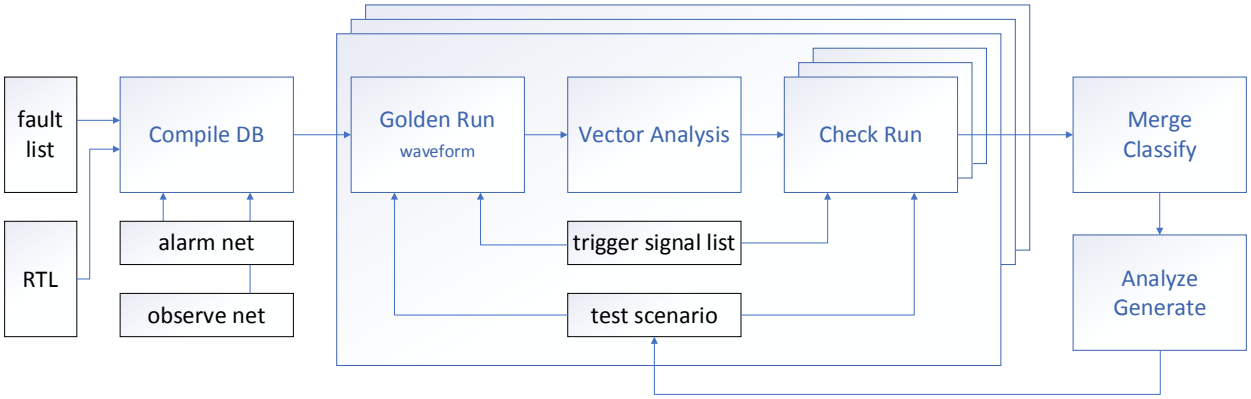

Figure 2. Fault campaign using an emulation flow.

The outputs of the compilation are the fault injection net (FIN), and the blocks synthesized for Veloce emulator. FIN which is an output of compilation will be double of fault nets in FuSa DB because Veloce assigns stuck at zero (SA0) and stuck at one (SA1) on a fault net. Only permanent faults are considered (SA0, SA1). Other fault types like transient faults, etc. are also supported but beyond the scope of this FC. If FuSa DB has N of fault net, the output of fault list from Compile DB will be 2N due to SA0 and SA1 for each fault net. Some faults can be optimized out at this stage.

There are several types that do not require fault injection run. Veloce provides optimization method to increase runtime performance by skipping fault injection run.

Deadlogic

If the fanout of a FIN does not have any reader, it means fault impact cannot be propagated further. This logic can be optimized as "Deadlogic."

No port in memory

In a design if there is a memory that no address is read anywhere, there would be no way that a design is affected by injecting fault. Memory which is not read anywhere will be optimized as "Memory:No Read Ports in the memory."

FinAndFdn

If a fault net is defined as FIN and FDN both by user, this fault net will be optimized as "FinAndFdn Fault."

IDs are assigned to each FINs after compilation not including the optimized nets. This ID based FIN provides easy fault injection run environment setup.

The FDNs are captured during the execution of the test scenario on the emulator, and the results are saved as the reference values of Golden Run [4]. Golden Run means normal run without a fault impact, so that this run will record all the behavior of FDNs to be reference. Then each fault from the FIN is injected during the execution of the same test scenario and the FDNs for this Check Run are compared against the reference (Fig. 3). As a result of Check Run, the alarm can be Detected or Undetected, the fault can be Observed or Unobserved thus giving four cases: DO, DU, UO and UU. Check Run means run with a fault impact can be observed or detected by comparing them against Golden Run.

As the number of faults to be injected can be huge, Check Runs are executed on the emulator in parallel. At Vector Analysis the fault nodes are checked and nodes with constant values are optimized out from the injected list for this test scenario.

To save runtime more it can achieve the fast fault injection runtime by excluding unnecessary time such as system initialization time. The initial state of DUT before Trigger Signal can be saved for Golden Run and then restored for each Check Run.

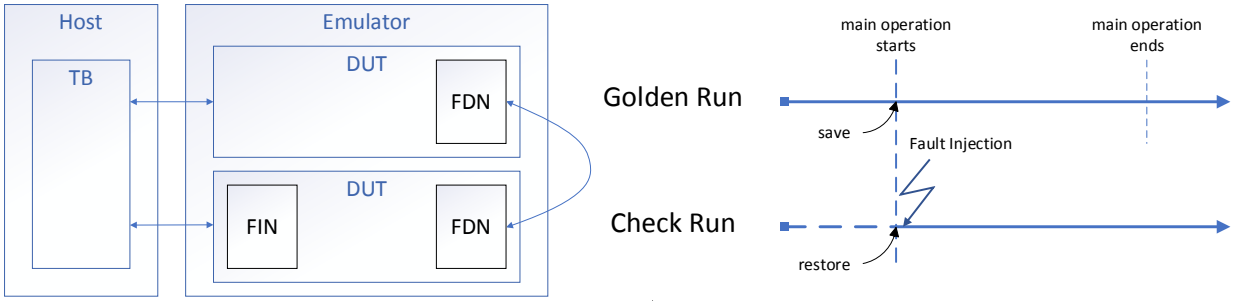

Figure 3. Golden Run and Check Run.

During the fault injection run, no additional runtime is necessary after the alarm is detected and the observe net is observed. By applying early termination feature, fast runtime can be also achieved based on the number of DO cases.

Finally, the results are merged for all test scenarios for the given SM, then they are classified and used for calculation of the ASIL and FMEDA metrics. If the score is not enough then test scenarios are regenerated and the FC is restarted.

A. Generation of Test Scenarios for FC

Typical test scenario for FC includes the following steps.

1) Host boots CPU.
2) CPU activates Watchdog Timer WDT.
3) CPU triggers Command Scheduler (CMNS) to start operation.
4) CMNS configures internal blocks.
5) CMNS controls internal blocks which read input, process data, write output.
6) CMNS waits internal blocks end.
7) CPU gets interrupt from CMNS about operation end.
8) CPU triggers MEM operation to generate CRC and/or reads CRC generated by internal blocks at step 5.
9) CPU compares CRC with the reference.
10) If the comparison fails, CPU sends interrupt (alarm_fail).

Otherwise, steps 3-9 can be repeated.

11) If all operations end, CPU sends interrupt (alarm_pass).
12) CPU stops WDT.

If WDT triggers before CPU sends pass or fail interrupt, CPU sends interrupt (alarm_hang).

It is also possible that CPU makes CRC comparison once after all operations end.

Test scenarios for FC must meet two conditions, both high quality and short runtime.

High quality means that if a test scenario does not satisfy with the above metrics then it should be regenerated. To get DC greater than or equal to 90% required for ASIL B one should make the test scenarios with the quality above the sign-off level of test scenarios used for functional verification.

On the other hand, test scenarios used for function verification often cover each function with redundancy, so it takes a long time to run all test scenarios. However, test scenarios for FC should also be used for error detection in the actual field tests. And those scenarios should be completed in a short time because they should not affect the actual SW operation.

There is no simple method to make an ideal test scenario. But there is an accurate and confident but slow method through analyzing IP specifications and algorithms and adding features one by one.

The following points can be considered to generate high quality test scenarios more efficiently.

1) Test scenarios used in real SW does not have DC as high as expected.

- In general, SW safety function scenarios do not utilize all IP functions. Depending on the scenario required by the system, IP functionality may be active/inactive.

- If certain functions of IP continue to be inactive, those functions can be excluded from the DC measurement range. However, SW engineers generally do not guarantee that certain IP features will continue to be inactive. This is because the function can be activated again depending on the scenario.

- So one should provide a test scenario that satisfies higher DC compared to test scenario used in SW. In general, to obtain the desired DC and achieve ASIL B, it is needed to utilize all minor functions in addition to the main function of IP. So, test scenarios used in functional verification can have higher DC.

2) As a starting point it is advisable to measure DC for test scenarios used for functional verification.

- In a lucky case it is possible to achieve ASIL B with only those scenarios. However, this case is rare.

- If you achieved ASIL B with that test scenarios to get shorter runtime it is possible to rank test scenarios by toggle coverage and remove duplicated test scenarios. It is good if the contributing test scenario satisfies DC and execution time, otherwise its optimization is required.

3) If a test scenario for FC is newly generated, it should be able to detect both system hang and data mismatch by fault.

- In particular, if the IP has a chain of multiple sub-IPs and the fault is injected into the sub-IP in the front part of the chain and the failure is not propagated then it is usually cannot be changed by configuring the sub-IP in the back part of the chain.

- To solve this issue, one should configure each sub-IP so that they can react sensitively to small fluctuations caused by faults.

4) As the change by one bit is required, intermediate values (like 'h8000 and 'h7fff) are more advantageous than extreme values (like 'h0000 and 'hffff').

- The fault is injected into register or port by one bit. So, the test scenario should be able to detect the fault caused by this change in a single bit.

- If the values are extreme, the fault may not be observed. Extreme values are likely to cause data saturation while changing by one bit due to their small variable range. On the contrary, configuring the values to be intermediate is effective in preventing data saturation while changing values by one bit because the variable range is larger than for extreme values.

5) It is often safe if the system's performance is degraded due to the fault.

- In general, performance improvement IPs such as cache may only reduce performance without system hang and data mismatch when the fault is injected.

- If the performance is important to the system, one should define a test scenario at the system level that should catch the difference when the performance is not satisfied. Therefore, faults that cause only performance difference can be found at the system level and the IP itself is considered to be safe.

B. Debug Methodology

The most important part in the FC is the result analysis to meet the target DC. For UU cases, the engineer must use the expert judgement to define whether it is a Safe fault or additional scenarios are required. An SW and/or HW methods can be used to find how a fault propagates in the system. Using an SW method in Codelink, the expectation is that the monitor of the core (CPU) may capture the reasons why the fault propagation is hidden. Also, comparing the waveform (Fast Triage) between Golden Run and Check Run can be one of the approaches based on HW to figure out where the fault propagation is masked. By having these debug methodologies, the engineer can find the debug point and add more scenarios to get the expected DC.

UU Analysis

Non-injectable nets based on synthesized information and vector quality are already filtered out by compiler at Compile DB (optimized nets) and by applying Vector Analysis (VQC). The classification of fault injection run with injectable nets is needed based on the resolution report. The following three methods are used for this FC: Fast Triage, Configuration check, and Read transition check.

Fast Triage (logic)

The main concept of Fast Triage is to find out how the propagation by injected fault cannot be spread out and where the propagation is reached out to the alarm net and the observe net. The result of Fast Triage can be classified as a Safe fault with the Expert Judgement based on the evidence why the fault effect is not propagated. Since Veloce supports the utility to compare two waveforms easily, the deviation between Golden Run and Check Run can be found as long as two waveforms are ready. By comparing the port of hierarchical modules from the fault injection net (FIN) to the alarm net and the observe net (FDN), the deviation can be found. If the deviation between Golden Run and Check Run is found on the output of “module 2” in Fig. 4 while no deviation is found on the input of the module which has the alarm net, it can be a Safe fault because the fault effect is not propagated to the alarm net. Then, it can be assumed that there is the logic which blocks the effect of fault propagation in “module 2” or “module 1.”

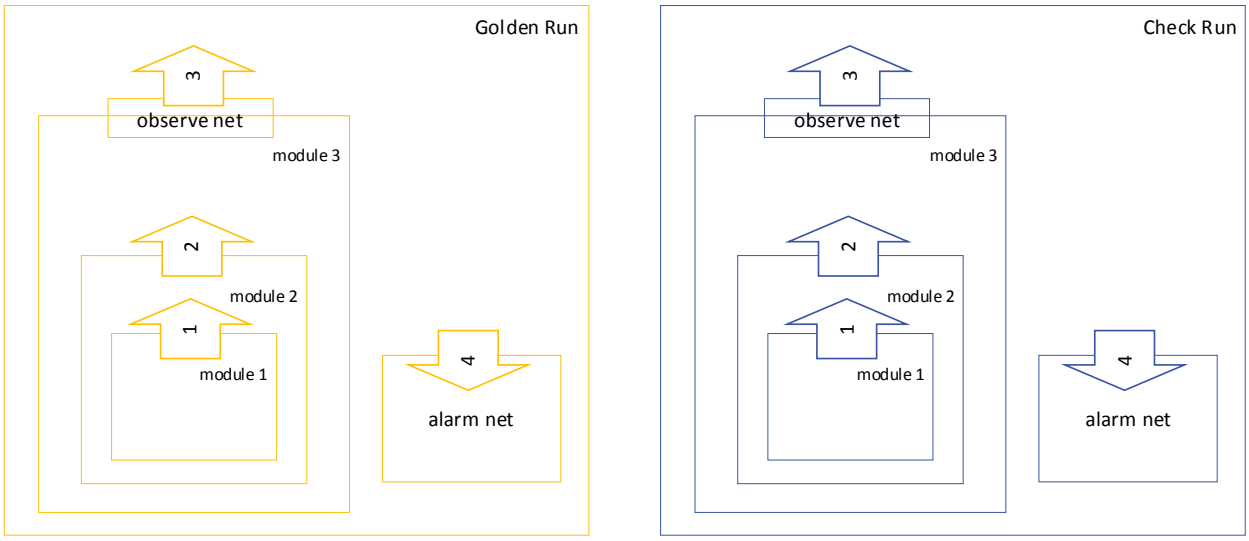

Figure 4. Fast Triage Concept.

This is one of the case why a deviation by fault propagation cannot be found. The deviation on "out_A" was not found because "value_A[x][y]" in Golden Run and Check Run were same with current test scenario (Fig. 5). While there were some toggled values on "value_A[x][y]" in Golden Run, "value_A[x][y]" was stuck at one in Check Run. However, when value of the "value_A[x][y]" in Golden Run was "1" which is same as the value in Check Run. So, it can be a Safe fault because there was no deviation between Golden Run and Check Run.

```
assign x = x_val[xxx:yyy];
assign y = y_val;
assign out_A = value_A[x][y]; // fault net is value_A[x][y] with SA1
```

Figure 5. Snippy Code for UU case

Configuration Check (STL with ARM Core)

If the core has a safety mechanism such as ECC or parity and a fault is injected on the cell, the configuration needs to be checked whether this safety mechanism is enabled or not. If the ARM core is not familiar, it is hard to debug on the waveform or SW code. If a safety engineer can have both access to SW code and synchronized waveform at the same time, it is helpful to check the configuration environment such as the enable of ECC, parity, and watchdog. If a safety engineer is familiar with waveform, the engineer can trace back SW by moving waveform time frame back and forth. If a safety engineer is familiar with SW, the engineer can use break point on SW, so that waveform shows the current status of HW. Codelink which is Veloce application can support this methodology.

Read Transition Check (Memory)

For the case of injecting a fault on a memory cell, the fault cannot be propagated toward several receivers unless the system read the data from the cell. To prove whether the fault's propagation effects on receivers or not, the address and read address needs to be verified. In the case of Fig. 6, read transaction can be checked with the value of "enable" and "A." If there is no transaction with current test scenario, it can be a Safe fault because the injected fault cannot be propagated further.

```
always@(posedge clk) begin
  if (enable = 1'b1)
    Q <= mem[A];
  end
end
```

Figure 6. Snippy Code for Read Transaction.

IV. RESULTS AND ANALISYS

The FC required several iterations before the expected DC was achieved. After each iteration one should analyze UU cases, generate test scenarios and rerun the FC.

C. Analysis of Results and Updates of Test Scenarios for PU0, PU1, MEM

NPU architecture, data flow and algorithms were analyzed and the above mentioned methods were applied at each iteration (Table V). Namely, at least those steps were taken.

1) After checking the bit width processed by sub-IP, two special data patterns were created: for the former MSB = "1," others = all "0," and for the latter MSB = "0," others = all "1."

2) If there were two or more inputs that affect the output of sub-IP, then special data pattern was applied only to one input. The data for other inputs was simplified so that the values changed by one bit can propagate to the end of the IP chain. Creating input data in this way increases the data sensitivity to the fault injected into special data patterns, making it easier to generate output failures.

3) The NPU IP has an activation feature using LUT for non-linear functions, and operations using LUT have a chain structure. If several LUTs are used in one test scenario, then due to injected faults the values of each LUT of the chain can be very variable, resulting in considerable data saturation. To prevent this the target LUT was set for each test scenario, and all other LUTs were bypassed, so test scenarios were created in a way that minimizes the influence of LUT.

4) The result of MAC is a combination of multiple multipliers and adders. If the result is configured to be only positive or only negative, then data saturation is often expected due to clipping. To avoid this the input data was created so that the output data also generated intermediate values. For example, if the output data is a signed value, zero value was created as the sum of a positive value and a negative value. This output data can improve the data sensitivity by the fault because it can increase the range of outputs that can be changed by the fault.

5) To cover register logic additional test scenario was created. Value = "0" and value = all "1" were consecutively written, read, and compared, mismatch indicated an error due to the fault. The scenario was more efficient in terms runtime compared to normal test scenario for registers where random values were used.

TABLE V
PERCENTAGE OF DO FOR SM3 (PU0 BLOCK)

		DO for iteration, %				
Block	Valid net (after Compile), %	1	2	3	4	5
PU0	100	36.70	56.70	87.15	89.08	90.26

After updating test scenarios, formal analysis was used to check the Cone-of-influence of the remaining UU cases. The faults which were outside of the COI were marked as Safe faults.

Then, to increase DC more Expert Judgement was applied to the remaining UU cases.

1) The fault of the spec-out feature is a Safe fault. If the HW feature not included in SW scenario existed in the IP then the faults injected in the related module, register, port, etc. were marked as Safe faults (spec-out).

2) The fault of the debug feature is a Safe fault. Debug features such as performance monitors and internal counters for debug do not affect the output of IP, so the related faults were marked as Safe faults (debug).

3) The fault of clock gating disable feature is a Safe fault. The faults which disables the clock gating logic intended for power reduction does not affect the outputs, so they were marked as Safe faults (power).

4) The fault of invalid range bit is a Safe fault. The valid range of the register or port may be limited depending on the scenario. The faults of invalid range do not affect the output, so they were marked as Safe faults (configuration).

5) The fault of the performance feature is a Safe fault. Features that only increase or decrease performance do not affect system hang and output data mismatch, so they were marked as Safe faults (performance).

D. Analysis of Results and Updates of Test Scenarios for PU2

PU2 itself consists of several parts. To get the high score, test scenario should cover all of them including computing core, memory, command scheduler, bus, and memory access blocks.

1) At the first iteration real SW test scenario was used but it only touches arithmetic logics partially some related registers and almost none of other register like control and other logics (Table VI).

2) Then test scenario from functional verification was applied. It included all arithmetic logics.

3) But memory operations were not enough, so they were added also.

4) After further analysis of UU case it was found that access to some registers has specific pattern and it was added also.

TABLE VI
PERCENTAGE OF DO FOR SM3 (PU2 BLOCK)

		DO for iteration, %							
Block	Valid net (after Compile), %	1	2	3	4	5	6	7	8
PU2	100	38.29	59.15	63.04	63.68	76.21	77.12	81.30	81.96

E. Results of FC

The final DC was calculated as the sum of DO, DU, Compile (optimized: Dead Logic, No port in mem), COI, and Safe) divided by the difference of total fault list and Compile (optimized: FIN=FDN).

$$DC\ ⑨ = \frac{DU\ ② + DO\ ③ + Compile\left(Dead\ Logic\ ④ + No\ port\ in\ mem\ ⑤ + \ FIN = FDN\ ⑥\right) + COI\ ⑦ + Safe\ ⑧}{Fault\ list\ ①} \quad (1)$$

The results of FC for SM3 are shown in Table VII. The final suite for SM3 includes 30 optimized test scenarios with high efficiency and low runtime.

TABLE VII
RESULTS OF FC FOR SM3 (SELF-TEST)

	FC steps						Result analysis					Result		
	FuSa DB	Compile	FI run				Compile (optimized)			Formal	Expert	Expected		Final
Block	Fault list ①	Valid net (after Compile)	DU ②	DO ③	UU (incl. VQC)	UO	Dead Logic ④	No port in mem ⑤	FIN = FDN ⑥	COI ⑦	Safe ⑧	FIT rate distr. (%)	Exp. DC (%)	DC (%) ⑨
PU0	4694	4220	0	3827	281	132	438	16	0	19	227	31.35	96	96.44
PU1	4694	4220	0	3827	282	131	438	16	0	19	227	31.35	96	96.44
MEM	4688	3472	0	2392	1022	58	1148	0	0	78	316	17.9	90	83.92
PU2	4658	4064	0	3331	711	22	522	72	0	21	80	10.86	80	86.43
CTRL	4582	2050	6	235	1716	93	1430	1102	0	131	0	2.04	60	63.38
PERI	4576	3058	0	2151	895	12	1518	0	0	75	0	6.48	80	81.82
Total												100	91.41	91.49

The results of FC for SM1, SM2 (CPU part) are shown in Table VIII. For SM1 software test library (STL) and custom test scenarios were used, for SM2 only STL was used. Proper configuration settings were checked with Codelink. As the number of test scenarios provided by ARM is limited, additional analysis of the remaining UU cases using debug methodology with Codelink mentioned earlier is planned for our future work.

TABLE VIII
RESULTS OF FC FOR SM1, SM2 (CPU)

	FC steps						Result analysis					Result		
	FuSa DB	Compile	FI run				Compile (optimized)			Formal	Expert	Expected		Final
Part	Fault list ①	Valid net (after Compile)	DU ②	DO ③	UU (incl. VQC)	UO	Dead Logic ④	No port in mem ⑤	FIN = FDN ⑥	COI ⑦	Safe ⑧	FIT rate distr. (%)	Exp. DC (%)	DC (%) ⑨
CPU memory	4696	4642	320	3835	486	1	54	0	0	0	0	1.02	90	89.63
CPU logic	4554	4174	0	2303	1818	53	364	0	16	94	0	98.98	60	60.98
Total												100	60.31	61.13

V. SUMMARY

Functional safety workflow of NPU IP with SW-based SMs is presented. The internal blocks and related SMs were selected based on FIT rate distribution. The FC process including fault injection run by emulation was applied. Debug methodology for efficient UU analysis was introduced. At each FC iteration after the detailed analysis of the IP architecture and faults, the test scenarios were updated, and finally some remaining faults were classified as Safe. The target DC score of 91.21% necessary to meet ASIL B metrics was achieved. Further optimization of test scenario runtime for SW safety verification can be included in future work.

REFERENCES

[1] ISO 26262 Road Vehicles – Functional Safety – Part 3: Concept phase, Second edition 2018-12.
[2] D. Cha et al., "Complex Safety Mechanisms Require Interoperability and Automation For Validation And Metric Closure," DVCon US, 2023.
[3] Siemens, "Veloce User Guide," Release v22.0.2, Document Revision 7.1, 2022.
[4] Siemens, "Veloce Fault Application User Guide," Release v22.0.2, Document Revision 7.1, 2022.