Converging on ISO-26262 ASIL-B metrics for Vision AI DSP IP using STL

by Ceva Technologies Ltd: Noam Meser <noam.meser@ceva-ip.com>,
Zvika Melamed <zvika.melamed@ceva-ip.com>,
Optima Design Automation Ltd: Sesha Sai Kumar C V <cvseshu@optima-da.com>,
Jamil Mazzawi jamil@optima-da.com, Ayman Mouallem <ayman@optima-da.com>

***Abstract**: Ceva SensPro AI & Vision DSP is an off the shelf IP used in automotive applications. Hence, it needs to comply with the Automotive Functional Safety standard ISO 26262, ASIL-B level as Safety Element out of Context (SEooC). Software Test Library (STL) is the safety mechanism which allows to bridge between safety metrics achieved using only HW-based mechanisms, and ASIL-B thresholds required from SensPro DSP. Measuring and improving the Diagnostic Coverage (DC) and Single Point Fault Metric (SPFM) of STL until they reach their ASIL (Automotive Safety Integrity Level) requirements, is traditionally a long, iterative process, mostly due to the long time it takes to perform fault-simulation and long debug time. In this paper, we discuss a novel approach of performing fault-analysis in exhaustive manner, using the Optima Safety Platform (OSP) that shortened the schedule, improved the results and allowed the project to meet its ASIL-B safety thresholds.*

Introduction

Ceva SensPro AI & Vision DSP is integrated within automotive and many other market application chips. These DSP cores should achieve the safety metrics set by ISO 26262 certification for relevant ASIL. This certification is critical across these applications as Tier 1s and OEMs move towards running advanced ADAS (Advanced Driver Assistance System) and AD (Autonomous driving) solutions, and safety certification is required for that. Along with hardware safety mechanisms for various types of random faults, STL is used as safety mechanism to quickly self-test the core for safety issues caused by single point random faults. In STL development, multiple teams are involved, viz. Safety, HW design, embedded SW development verification, and Program Management - and it's challenging to collaborate effectively between the teams to finalize the STL.

Safety Metrics requirements for ASIL-B

While the ISO 26262 requirements are very complex, they can be summarized by the key "Safety metrics" defined in fig. 1 that need to be measured for each HW functionality.

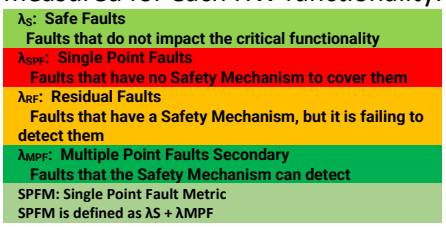


Figure 1: Safety metrics according to ISO 26262

The key ASIL-B requirement, is reaching SPFM of 90%. In this project, other safety mechanisms contribute to the overall SPFM, and the SPFM requirement on STL was 80%.

Common STL development Methodology

To develop STL that meets the SPFM requirement, execution time budget, code size limits and memory budget - involves an iterative process, like the following:

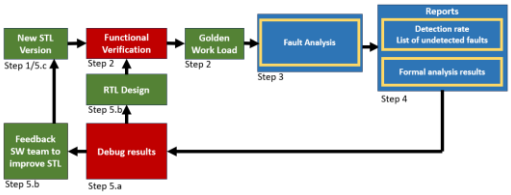


Figure 2: Common STL development cycle

1- **SW team:** Develops and improves a version of the STL to detect random faults in the DSP and avoid failures in the over-all system operation
2- **Verification team:** Simulate the design with the provided STL and existing Test Bench to generate Golden Work Load (waveform file).
3- **Functional Safety (FuSa) team:** performs fault-analysis on the latest version of RTL, STL and using the provided Golden Work-Load (WL) as reference.
4- **All teams:** Results of the fault-analysis are analyzed towards next sprint of SW enhancement.
If Safety metrics goals are met, we're done.
5- Otherwise
a. All teams: Debug why the numbers are low?
b. SW team: Improve the STL
c. Re-iterate (go to step 2)

Converging on ISO-26262 ASIL-B metrics for Vision AI DSP IP using STL

by Ceva Technologies Ltd: Noam Meser <noam.meser@ceva-ip.com>,
Zvika Melamed <zvika.melamed@ceva-ip.com>,
Optima Design Automation Ltd: Sesha Sai Kumar C V <cvseshu@optima-da.com>,
Jamil Mazzawi jamil@optima-da.com, Ayman Mouallem <ayman@optima-da.com>

STL fault-analysis challenges

The challenge in performing the fault-analysis cycle is that it usually takes a lot of time, which limits the number of faults and iterations that can be performed in a reasonable project schedule. Another challenge is that debug is usually very hard and takes a lot of time, as it is hard to correlate lists of "undetected faults" to "how to improve my STL", especially given the multi-disciplinary nature of the team, having both HW, SW and FuSa engineers.

Methodology used in the project

The project used the same methodology as described above; however, fault analysis was performed using the Optima Safety Platform™ which includes three step fault analysis, supported with three dedicated engines that perform analysis to measure and report the Safety metrics mentioned in fig. 1, then provides recommendation or automated debug into how to improve the parameters.

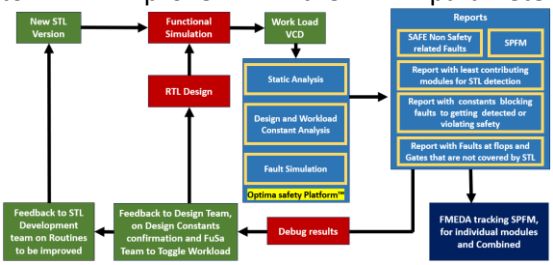


Figure 3: STL development flow and methodology adopted

Fault Injection Analysis using Optima Safety Platform as in fig. 3, provides various insights and recommendations, like which constant signals need to be toggled, which parts of the design need to be activated through STL, or which HW config modes need to be entered to improve coverage etc. More details of the analysis and recommendations are provided in the following sections. The feedback from the fault analysis in terms of debug and recommendations, helps hardware team improve the setup by defining actual design or mode constants, toggle the signals in testbench/workload. This feedback helps Software team to improve STL by toggling the constant signals to activate more design parts by the STL.

Three steps of Fault Analysis

The fault analysis used was three steps:

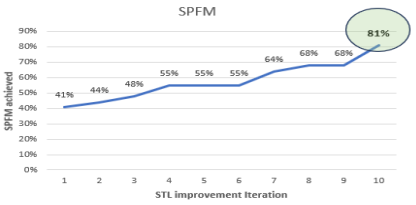
1. **Static Analysis:** The design is analysed in a static semi-formal manner (without workload) to identify faults that can never be detected, hence counted as λspf and faults that has NO impact on the safety goals, hence counted as λs.
2. **Constant Analysis:** In this step, a dedicated engine identifies the non-toggling signals in the Golden-WL, and measures their impact on the detection rate and provide recommendations.
3. **Fault Simulation:** Fault simulation is performed to find which faults will be detected by the STL and which will not.

Note that steps 1 and 2, **reduce** the number of faults that needs fault simulation and provide recommendations to improve the STL.

Reports and Recommendations from OSP

As shown in fig. 3, OSP™ provides advanced reports and recommendations, that enabled the team to achieve the required SPFM for STL, Categorization like SAFE/UNSAFE, Hierarchical fault coverage to know which modules are still not covered by STL, Constants in the design and Work Load, and how they are blocking the faults, SPFM categorization as show in fig. 4 SAFE, Dangerous, Dangerous detected, Dangerous Undetected.

This categorization can later be used to calculate SPFM for transient (Soft) faults as well as PMHF (FIT rate) for the design.



	STL1.13		STL1.14		STL1.15		STL1.16	
Total Faults considered for all 6 instances	2258760	100%	2258760	100.00%	2258760	100.00%	2258760	100.00%
Safe Faults	622407	28%	810918	35.90%	920885	40.77%	1017531	45.05%
Dangerous - SPF(Single Point Fault)	16196	1%	16196	0.72%	16196	0.72%	16196	0.72%
Dangerous Undetected	993443	44%	800073	35.42%	696206	30.82%	420330	18.61%
Dangerous Detected	625614	28%	630473	27.91%	624389	27.64%	803619	35.58%
Undetermined by engines	1100	0%	1100	0.05%	1084	0.05%	1084	0.05%
SPFM for All Six blocks together	0	55%	0	63.81%	0	68.41%	0	80.63%

Fig4. SPFM tracking sheet

Conclusion

Developing STL for the Vision AI DSP was challenging and iterative. However, using the above-described methodology, allowed the project to converge and meet the SPFM goal in 8 weeks and 10 iterations.