Safety Concept of SCST

Author: David Baca

VERSION	DATE	AUTHOR	COMMENT
0.1	03/15/2017	David Baca	Document established
1.0	03/30/2017	David Baca	Document updated per review feedback.

Contents

1 Introduction 4
1.1 Purpose 4
1.2 Acronyms 4
1.3 SCST Roles 4
2 SCST Safety Concept 4
2.1 Safety-Related Faults 4
2.1.1 Hardware Random Faults 4
2.1.2 Software Systematic Faults 5
2.2 Integration Environment 5
2.3 Safety Requirements 6
2.3.1 ASIL Justification 6
2.4 Safety Measures 6
3 References 6
4 Annex – Implicit Safety 6

1 Introduction

1.1 Purpose

This document defines the safety concept of Structural Core Self Test (SCST) product as required to comply with ISO26262 standard (1).

1.2 Acronyms

ASIL	Automotive Safety Integrity Level, see [1]
ES	Explicit Safety
FTE	Fully Trusted Environment
HW	Hardware
IS	Implicit Safety
MCU	Microcontroller Unit
SCST	Structural Core Self Test
SEooC	Safety Element out of Context
SW	Software

1.3 SCST Roles

SCST has the following three roles:

- Functional role:
  - Detection of permanent HW faults in the MCU core; providing the required Diagnostic Coverage.
- Safety role to withstand hardware random faults, chapter 2.1.1, to:
  - prevent failures violating SCST safety requirements, and
  - contribute to meeting hardware architectural safety metrics.
- Safety role to withstand software originating interferences, chapter 2.1.2, leading to violation of safety requirements.

2 SCST Safety Concept

2.1 Safety-Related Faults

There are two kinds of faults recognized by ISO26262 (1) as origins of safety goals violations:

- Hardware random faults
- Software systematic faults

2.1.1 Hardware Random Faults

A hardware random fault is a fault that occurs unpredictably and follows typically exponential probability distribution. A hardware fault is safety-related if it causes a violation of a safety goal, in this case of a safety-related SW requirement. Furthermore, we distinguish permanent and transient (soft) hardware faults.

Faults in the processing unit consisting of memory, core(s), and some surrounding peripherals may be detected before affecting execution. This is achieved by HW measures such as delayed lock-step mode, ECC protection on memory etc. On the other hand, safety-related faults not detected by hardware shall

be detected by application SW including the SCST, which itself represents a safety measure designed for detecting permanent hardware faults in the MCU core.

The effectiveness of detection of hardware random faults is evaluated at hardware architectural level by hardware fault metrics specified in ISO26262 - Part 5. SCST contributes to those metrics by detecting all possible faults violating its safety requirements as well as by its functional role (see chapter 1.3). Note that a fault is covered if a safety mechanism implementation is assumed in the SCST safety manual.

Last but not least, non-safety-related faults do not have to be detected. These faults have no violation effect on the safety requirements.

2.1.2 Software Systematic Faults

There are two sources of software systematic faults – SCST itself and another software element integrated in the application. Interference, a fault caused by a failure of another software element, is considered only from software elements of a lower ASIL. Hence, software elements of the same or higher ASIL are considered “trusted” and cannot cause interference fault.

In order to prevent its own software systematic faults, SCST shall be developed as SEooC according to ISO26262 (1) requirements specified for the highest allocated ASIL.

Faults of other software elements interfering with SCST may cause safety requirement violations. Such faults are either prevented by the integration environment or detected and reacted upon by safety mechanisms implemented either in SCST or in the integrating application.

2.2 Integration Environment

In order to prevent certain faults to cause failures, generic safety measures that isolate faults from causing safety-related requirement violation shall be deployed. As a result, SCST shall be deployed to Fully Trusted Environment (FTE), which is defined by rules below.

The definition of FTE is formulated with respect to a SEooC. SCST is an SEooC.

Definition of Fully Trusted Environment (FTE):

- The SEooC resides in a memory partition dedicated to components with the same or higher ASIL only; this memory partition protects shared resources owned by SEooC (SEooC code, stack, global memory) from being accessed by lower ASIL components.
- The SEooC is called by components with the same or higher ASIL only.
- SEooC calls components with the same or higher ASIL only.

The implications of FTE are:

- Assumed valid arguments when SCST is called.
- Assumed valid return values SCST calls another element, e.g. an MCAL driver.
- Assumed a correct sequence of a call (call protocol) when SCST is called
- Assumed no memory corruption including stack.

2.3 Safety Requirements

The requirements listed in Table 1 are the safety requirements of SCST. Hence, SCST is developed as Explicit Safety (ES) component compared to sMCAL, see Chapter 4. Yet, the same robustness measures are also applied to SCST, as an ES component satisfies the IS requirement implicitly.

Table 1 – SCST safety requirements

ID	Requirement	ASIL
SC_SCST_Req01	SCST shall ascertain the execution of the requested tests and provide correct execution result.	B(D)
SC_SCST_Req02	The tested core shall be brought to an expected state after the SCST execution is completed.	B(D)

2.3.1 ASIL Justification

SCST can be used to detect both single-point faults and latent faults. See [1] for the fault differentiation. In both cases SCST detect permanent faults only. The diagnostic coverage achievable by SCST is just above 90%, e.g. 92%.

If SCST is used to detect latent faults only, then it is run during start-up or shut-down. Hence it is run in a non-safety context. For that SCST does not need to be a SEooC as it loses its safety aspect given the integration environment, which is non-safety. For the unique cases when SCST is run once the safety context was established, ASIL-B(D) shall be sufficient because SCST safety requirements only ensure test execution and robustness and there shall be other system measures to ensure meeting the safety goals by the application. ASIL-B(D) supports the achievement of safety goals up to ASIL-D.

In the case when SCST is used to detect single-point faults, SCST diagnostic coverage is sufficient only for ASIL-B. Hence, SCST can be used to support ASIL-B(D) requirements and other run-time measures shall be in place to detect the remaining single-point faults to achieve ASIL-C and ASIL-D metrics.

2.4 Safety Measures

The safety measures implemented by SCST shall be determined by a safety analysis with respect to the safety requirements allocated to the SCST.

3 References

1. **ISO.** Road vehicles — Functional safety. *Road vehicles — Functional safety.* s.l. : {ISO, Geneva, Switzerland}, 2011. {ISO 26262}.

4 Annex – Implicit Safety

Implicit Safety is a software safety concept that constitutes software SEooC based on properties any safety-related software element possesses. Those properties are captured in Implicit Safety requirement. Furthermore, Implicit Safety concept allocates Implicit Safety requirement to a software element as the only safety requirement.

Implicit Safety (IS) requirement is defined as

A safety-related element shall not corrupt its own integrity and the integrity of other elements. - ASIL-X

The ASIL is freely selected as needed.

Software safety elements that are allocated functional safety requirements fulfill Implicit Safety requirement implicitly.

A software safety element that is allocated the IS requirement as the only requirement is called Implicit Safety element. Conversely, a software safety element with one or more safety functional requirements is called Explicit Safety (ES) element.

An IS software element can be integrated with other software safety elements that have the same or lower highest ASIL without causing interference. An IS element assures stable valid execution even under the presence of HW random faults. However, IS element does not guarantee the correctness of data provided. It means that data provided by IS element obey to the software interface contract, i.e. they are valid, yet they might not be correct. As a result, integration application shall apply measures to detect correctness faults if necessary.