UM2814

User manual

STM32WL5x dual-core safety manual

Introduction

This document must be read along with the technical documentation such as reference manual(s) and datasheets for the STM32WL5x dual-core microcontroller devices, available on www.st.com.

It describes how to use the devices in the context of a safety-related system, specifying the user's responsibilities for installation and operation in order to reach the targeted safety integrity level. It also pertains to the X-CUBE-STL software product.

It provides the essential information pertaining to the applicable functional safety standards, which allows system designers to avoid going into unnecessary details.

The document is written in compliance with IEC 61508.

The safety analysis in this manual takes into account the device variation in terms of memory size, available peripherals, and package.

This manual addresses the STM32WL5x dual-core microcontroller, that include two *CPU* cores, the Arm® Cortex®-M0+ and Cortex®-M4.

1 About this document

1.1 Purpose and scope

This document describes how to use STM32WL5x dual-core microcontroller unit (MCU) devices (further also referred to as *Device*(s)) in the context of a safety-related system, specifying the user's responsibilities for installation and operation, in order to reach the desired safety integrity level.

It is useful to system designers willing to evaluate the safety of their solution embedding one or more *Device(s)*.

For terms used, refer to the glossary at the end of the document.

Note: This manual addresses the STM32WL5x dual-core microcontrollers, that are devices with two CPU cores, the Arm® Cortex®-M0+ and Cortex®-M4.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.2 Normative references

This document is written in compliance with the IEC 61508 international norm for functional safety of electrical, electronic and programmable electronic safety-related systems, version IEC 61508-1-7 © IEC:2010. The compliance to other functional safety standards is considered in reference document [3].

The following table maps the document content with respect to the IEC 61508-2 Annex D requirements.

Table 1. **Document sections versus IEC 61508-2 Annex D safety requirements**

Safety requirement	Section number
D2.1 a)	Section 3 Reference safety architecture
D2.1 b)	Section 3.2 Compliant item
D2.1 c)	Section 3.2 Compliant item
D2.2 a)	General information are provided in Section 4.1 Random hardware failure safety results. Detailed information on failure modes and related failure rates are included in other reference documents [1], [2] referred in Section 1.3 Reference documents.
D2.2 b)	
D2.2 c)	
D2.2 d)	
D2.2 e)	
D2.2 f)	Useful information for DTI of each safety mechanisms are provided in related specification tables (filed “Periodicity”) of Section 3.6 Hardware and software diagnostics. General guidance on DTI is included in Section 3.3.1 Safety requirement assumptions.
D2.2 g)	Because of the software-based nature of Device safety concept, the outputs of the Compliant Item triggered by internal diagnostics are decided at application software level, and so they cannot be described in this manual.
D2.2 h)	Periodic proof test is excluded by specific ASR3.1 in Section 3.3.1 Safety requirement assumptions
D2.2 i)	Section 3.7 Conditions of use
D2.2 j)	Section 3.2.3 Reference safety architectures - 1oo1, Section 3.2.4 Reference safety architectures - 1oo2
D2.2 k)	Section 3.2.2 Safety functions performed by Compliant item

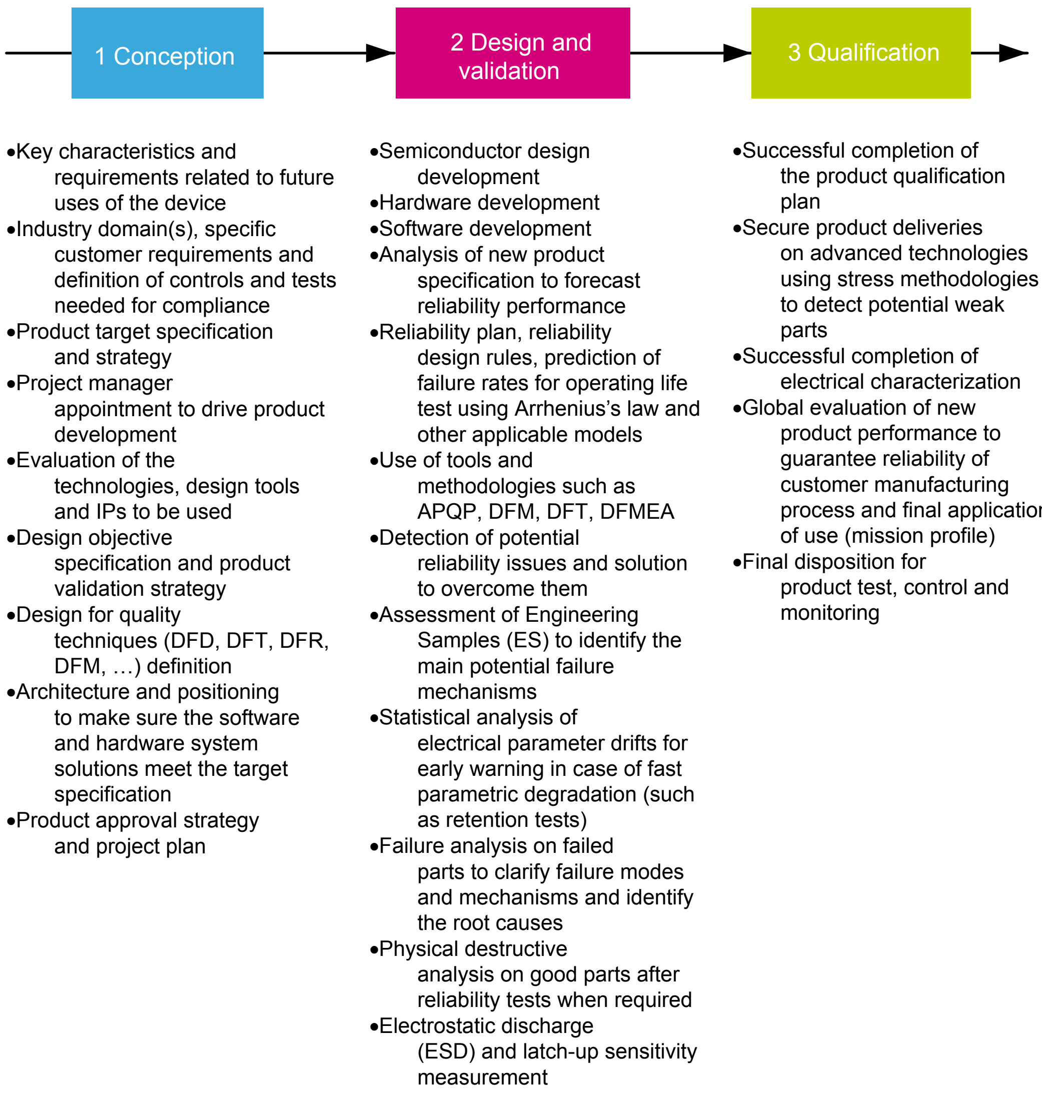
1.3 Reference documents

[1]	AN5663, Results of FMEA on STM32WL5x dual-core microcontrollers.
[2]	AN5662, Results of FMEDA on STM32WL5x dual-core microcontrollers.
[3]	AN5689: Adapting the X-CUBE-STL functional safety package for STM32 (IEC 61508 compliant) to other safety standards
[4]	AN5936 X-CUBE-STL: advanced topics

2 Device development process

STM32 series product development process (see Figure 1), compliant with the IATF 16949 standard, is a set of interrelated activities dedicated to transform customer specification and market or industry domain requirements into a semiconductor device and all its associated elements (package, module, sub-system, hardware, software, and documentation), qualified with ST internal procedures and fitting ST internal or subcontracted manufacturing technologies.

Figure 1. STMicroelectronics product development process



- Key characteristics and requirements related to future uses of the device
- Industry domain(s), specific customer requirements and definition of controls and tests needed for compliance
- Product target specification and strategy
- Project manager appointment to drive product development
- Evaluation of the technologies, design tools and IPs to be used
- Design objective specification and product validation strategy
- Design for quality techniques (DFD, DFT, DFR, DFM, …) definition
- Architecture and positioning to make sure the software and hardware system solutions meet the target specification
- Product approval strategy and project plan

- Semiconductor design development
- Hardware development
- Software development
- Analysis of new product specification to forecast reliability performance
- Reliability plan, reliability design rules, prediction of failure rates for operating life test using Arrhenius’s law and other applicable models
- Use of tools and methodologies such as APQP, DFM, DFT, DFMEA
- Detection of potential reliability issues and solution to overcome them
- Assessment of Engineering Samples (ES) to identify the main potential failure mechanisms
- Statistical analysis of electrical parameter drifts for early warning in case of fast parametric degradation (such as retention tests)
- Failure analysis on failed parts to clarify failure modes and mechanisms and identify the root causes
- Physical destructive analysis on good parts after reliability tests when required
- Electrostatic discharge (ESD) and latch-up sensitivity measurement

- Successful completion of the product qualification plan
- Secure product deliveries on advanced technologies using stress methodologies to detect potential weak parts
- Successful completion of electrical characterization
- Global evaluation of new product performance to guarantee reliability of customer manufacturing process and final application of use (mission profile)
- Final disposition for product test, control and monitoring

3 Reference safety architecture

This section reports details of the STM32WL5x dual-core safety architecture.

Note: *This manual addresses the STM32WL5x dual-core microcontrollers, that include two CPU cores, the Arm® Cortex®-M0+ and Cortex®-M4.*

3.1 Safety architecture introduction

The *Device*(s) analyzed in this document can be used as *Compliant item*(s) within different safety applications.

The aim of this section is to identify such *Compliant item*(s), that is, to define the context of the analysis with respect to a reference concept definition. The concept definition contains reference safety requirements, including design aspects which are outside of the defined *Compliant item*.

As a consequence of a *Compliant item* approach, the goal is to list the system-related information considered during the analysis, rather than to provide an exhaustive hazard and risk analysis of the system around *Device*. Such information includes, among others, application-related assumptions for danger factors, frequency of failures and diagnostic coverage guaranteed by the application.

3.2 Compliant item

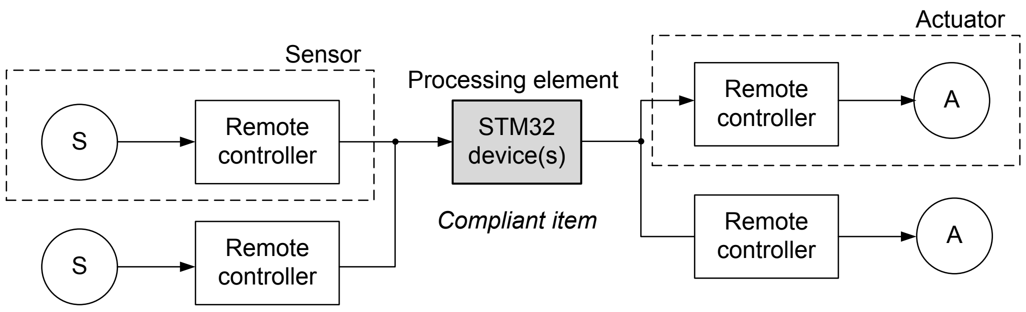
This section defines the *Compliant item* term and provides information on its usage in different safety architecture schemes.

3.2.1 Definition of Compliant item

According to IEC 61508-1 clause 8.2.12, a *Compliant item* is any item (for example an element) on which a claim is being made with respect to the clauses of the IEC 61508 series. Any mature *Compliant item* must be described in a safety manual available to the *End user*.

In this document, *Compliant item* is defined as a system including one or two STM32 devices (see Figure 2). The communication bus is directly or indirectly connected to sensors and actuators.

Figure 2. **STM32 as *Compliant item***



Other components might be related to the *Compliant item*, like the external HW components needed to guarantee either the functionality of the *Device* (external memory, clock quartz and so on) or its safety (for example, the external watchdog or voltage supervisors).

A defined *Compliant item* can be classified as *element* according to IEC 61508-4, 3.4.5.

In summary, claims related to this *Compliant item* are related to the possible use of a *Device* for the implementation of any safety function up to *SIL*2 (for a single *Device*) and up to *SIL*3 (for two destinct *Device*s), with specific architectures and observing all the requirements and indications provided in this manual.

3.2.2 Safety functions performed by Compliant item

In essence, *Compliant item* architecture encompasses the following processes performing the safety function or a part of it:

- input processing elements (PEi) reading safety related data from the remote controller connected to the sensor(s) and transferring them to the following computation elements

- computation processing elements (PEc) performing the algorithm required by the safety function and transferring the results to the following output elements
- output processing elements (PEo) transferring safety related data to the remote controller connected to the actuator
- in 1oo2 architecture, potentially a further voting processing element (PEv) in charge to facilitate the safety function processing by each of the two channels, individually.
- processes external to *Compliant item* ensuring safety integrity, such as watchdog (WDTe) and voltage monitors (VMONe)

The role of the PEv process is clarified in Section 3.2.4 Reference safety architectures - 1oo2. The role of the WDTe and VMONe external processes is clarified under Section 3.6 Hardware and software diagnostics:

- WDTe: refer to External watchdog – CPU_SM_5 and Control flow monitoring in *Application software* – CPUM0+_SM_1 and CPUM4_SM_1,
- VMONe: refer to Supply voltage internal monitoring (PVD) – VSUP_SM_1 and System-level power supply management - VSUP_SM_5.

In summary, *Devices* support the implementation of *End user* safety functions consisting of three operations:

- safe acquisition of safety-related data from input peripheral(s)
- safe execution of *Application software* program and safe computation of related data
- safe transfer of results or decisions to output peripheral(s)

Claims on *Compliant item* and computation of safety metrics are done with respect to these three basic operations.

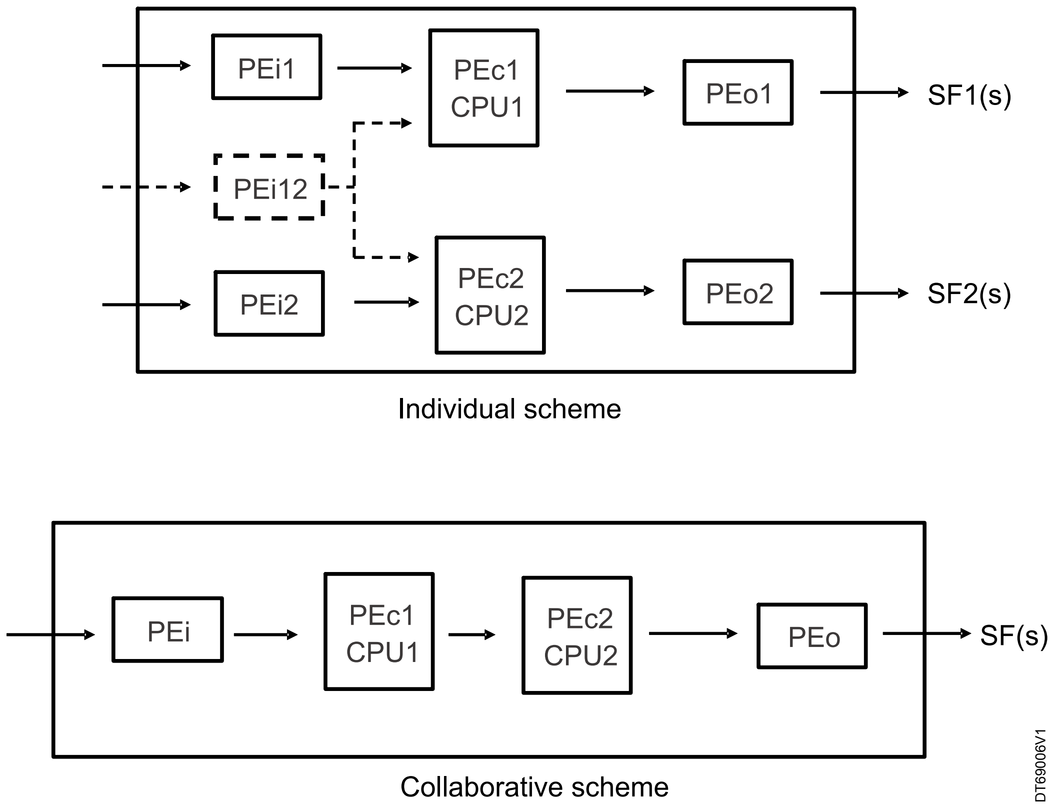
Caution: Due to the general purpose nature of the *Device*, its safety concept is mainly software-based. Accordingly, any following claim related to the possibility of *Device* itself to support the implementation of safety functions up to a certain *SIL* is strongly correlated to the observance of CoUs as requested in Section 3.7 Conditions of use.

As the STM32WL5x dual-core microcontroller includes two separate CPUs (Arm® Cortex®-M0+ and Cortex®-M4), the safety function(s) and so the related three above described suboperations can be performed by different possible schemes. Two possible main schemes are:

- Individual scheme: each CPU may implement a specific safety function, with no collaboration from the other CPU
- Collaborative scheme: the two CPUs collaborate for the implementation of the same safety function(s)

The two schemes are shown in the following figure, where the notation: SF1(s), SF2(s), SF(s), means that each "channel" can be used to implement one or multiple safety functions.

Figure 3. Individual and collaborative schemes



The schemes can be embedded in the reference safety architectures described below.

Consequences related to the coexistence on the two CPUs in the same MCU are embedded in related Assumed Safety Requirements (refer to Section 3.3 Safety analysis assumptions) and Section 3.7 Conditions of use). Related nationals are exposed in Section 3.2.5 The separation concept.

According to the definition for implemented safety functions, *Compliant item* (element) can be regarded as type B (as per IEC 61508-2, 7.4.4.1.3 definition). Despite accurate, exhaustive, and detailed failure analysis, *Device* has to be considered as intrinsically complex. This implies its type B classification.

Two main safety architectures are identified: 1oo1 (using one *Device*) and 1oo2 (using two *Devices*).

3.2.3 Reference safety architectures - 1oo1

1oo1 reference architecture (Figure 4) ensures safety integrity of *Compliant item* through combining *Device* internal processes (implemented safety mechanisms) with external processes WDTe and VMONe. In this architecture, the Device is considered intrinsically having hardware fault tolerance (HFT) equal to 0.

1oo1 reference architecture targets safety integrity level (SIL) *SIL2*. Both individual and collaborative scheme are possible.

3.2.3.1 *Individual scheme*

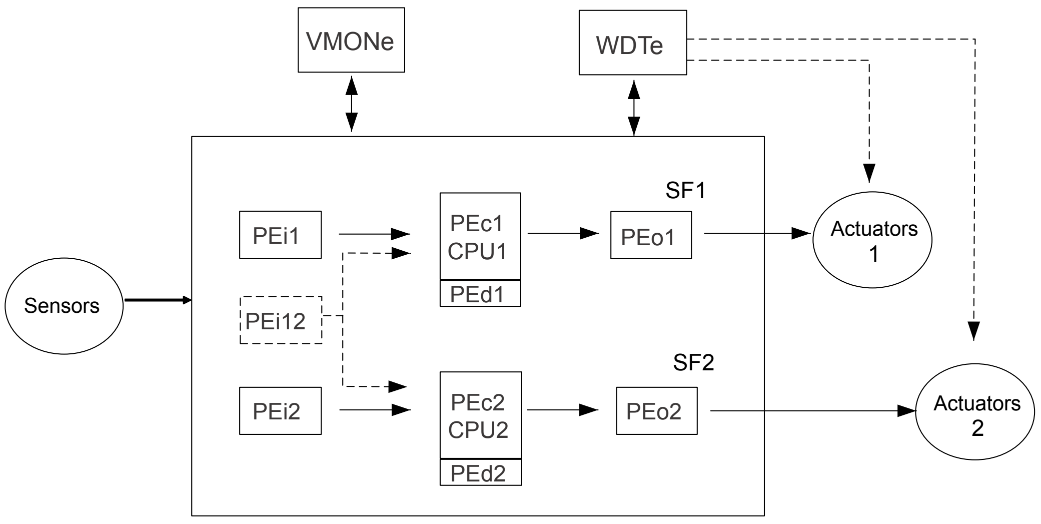


Figure 4. **1oo1 reference architecture - individual scheme**

In the framework of individual scheme, the two chains PEi1/PEc1/PEo1 and PEi2/PEc2/PEo2 implement separate sets of safety function(s). Individual set of external actuators are involved in the safety function(s) implementation. Link between WDTe and actuators show the capability of the external watchdog to force the safe state.

Caution: All implemented safety function(s) share a common management for safe state and PST requirement - refer to Section 3.3 Safety analysis assumptions for related ASR.

3.2.3.2 *Collaborative scheme*

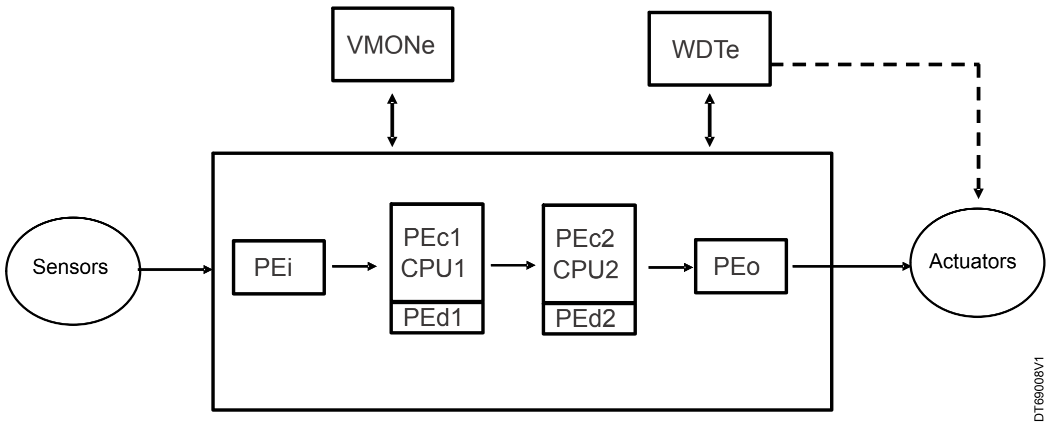


Figure 5. **1oo1 reference architecture - collaborative scheme**

In the collaborative scheme, both CPUs are involved in the implementation of the same safety function(s). The CPUs connection order is just informative and it is not constrained.

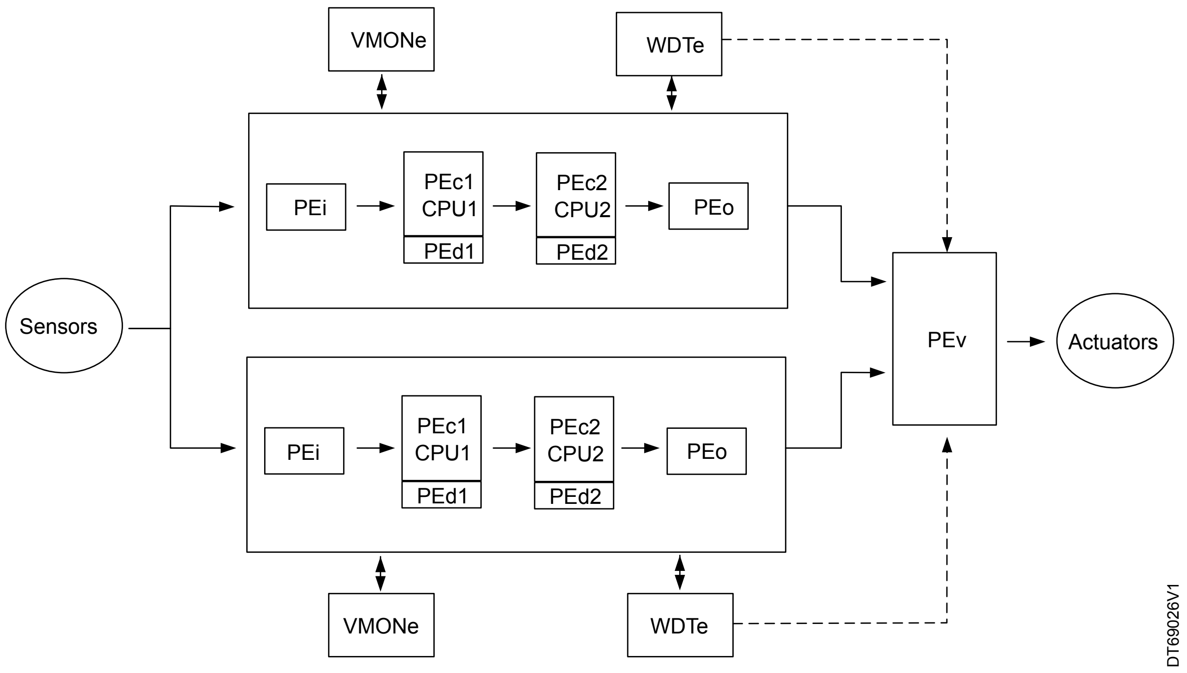
3.2.4 Reference safety architectures - 1oo2

The 1oo2 reference architecture (Figure 6) contains two separate channels, either implemented as a 1oo1 reference architecture ensuring safety integrity of the *Compliant item* through combining *Device* internal processes (implemented safety mechanisms) with external processes: WDTe and VMONe. The overall safety integrity is then ensured by the external voter PEv, which allows claiming hardware fault tolerance (HFT) equal to 1. The PEv role is indeed to facilitate the safety function processing by each of the two individual channels, to allow the correct execution of the safety function even in case one channel is faulty. The complexity of the PEv implementation strongly depends on the nature of the safety function and safe state definitions. Achievement of higher safety integrity levels as per IEC 61508-2 Table 3 is therefore possible. Appropriate separation between the two channels (including power supply separation) must be implemented in order to avoid huge impact of common-cause failures (refer to Section 4.2 Analysis of dependent failures). However, β and βD parameters computation is required.

This architecture targets *SIL3*, under the assumption that each channel follows all requirements indicated for *SIL2* in this manual.

Attention: *According the clause 7.4.3.2 in IEC 61508-2, this architectural scheme may provide benefits to the software applications systematic capability (SC) only in case diverse software is adopted on the two channels.*
For the sake of simplicity, only the collaborative scheme is shown in Figure 6. Note that the 1oo2 architecture is possible with the individual scheme.

Figure 6. **1oo2 reference architecture**



3.2.5 The separation concept

The coexistence of two different *CPU* (Arm® Cortex®-M0+ and M4) on the Device is managed by the means of the STM32WL5x dual-core dedicated separation concept, shown in this section. The separation concept mitigates potential interferences between the two CPUs (and their software) and it is composed by two different aspects: spatial separation and temporal separation.

Spatial separation

Arm® Cortex®-M0+ and M4 coexist on the same device and so they share the control and the access to common resources, like SRAM, flash memory, peripherals. Interferences are therefore possible, but in the lack of hardware-built segregation features for the CPUs, they are mitigated by a combination of requirements driving the implementation of final application:

- CoU_10 and CoU_11 forces the implementation of a kind of “private” SRAMs, one for each CPU, which is quite valuable when individual scheme is adopted. Possible interferences are mitigated by the static allocation of the SRAM and by dynamic checking performed by the two MPUs, as requested by CoU_12.
- Because of CoU_15, each CPU has its “private” watchdog which can be triggered just by the CPU itself – this mitigates potential issues related to the management of the required external watchdog, as it provides an additional layer of protection to the control flow execution for each CPU. Furthermore, CoU_16 guarantees that the software routines responsible for watchdog management and safety data exchange between CPUs are always active because it is developed with the highest systematic capability.

Temporal separation

The two CPUs Arm® Cortex®-M0+ and M4 can boot independently and this makes them not strictly linked from a temporal point of view. The effectiveness of the spatial separation (see above) is correlated to the actual capability of prescribed hardware and software to function correctly. Accordingly, it is *End user* responsibility to guarantee by system-level measures and solutions the safe state during the STM32WL5x dual-core boot. The CPU separation cannot be considered working until at least CoU_17 (startup tests) is satisfied.

3.3 Safety analysis assumptions

This section collects all assumptions made during the safety analysis of the *Devices*.

3.3.1 Safety requirement assumptions

The safety concept specification, the overall safety requirement specification and the consequent allocation determine the assumed requirements for *Compliant item* as further listed. *ASR* stands for assumed safety requirement. Refer to [4] for additional details about following assumptions.

Caution: It is *End user*'s responsibility to check the compliance of the final application with these assumptions.

ASR1: *Compliant item* can be used to implement four kinds of safety function modes of operation according to IEC 61508-4, 3.5.16:

- a continuous mode (CM) or high-demand (HD) *SIL3* safety function (*CM3*), or
- a low-demand (LD) *SIL*3 safety function (*LD3*), or
- a *CM* or *HD SIL2* safety function (*CM2*), or
- a *LD SIL2* safety function (*LD2*).

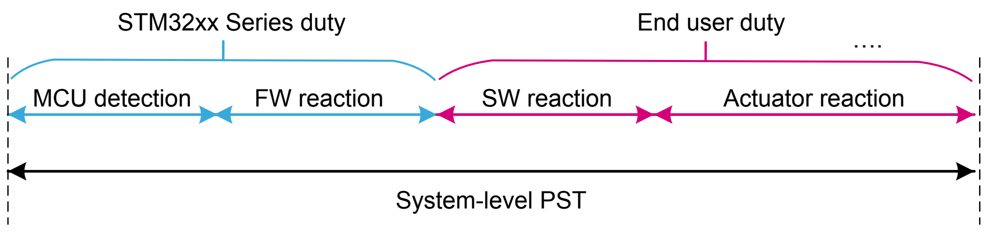
ASR2: *Compliant item* is used to implement safety function(s) allowing a specific worst-case time budget (see note below) for the STM32 *MCU* to detect and react to a failure. That time corresponds to the portion of the process safety time (PST) allocated to *Device* (*STM32xx Series duty* in Figure 7) in error reaction chain at system level.

In case of multiple safety functions implementation leading to multiple different time constraints, the shortest one must be adopted for each safety function.

Note: *As collateral effect, time constraint for the execution of periodical tests are always the same for both CPUs of the device.*

Note: *The computation for time budget mainly depends on the execution speed for periodic tests implemented by software. It is possible that such a duration depends on the actual amount of hardware resources (RAM memory, flash memory, peripherals) actually declared as safety-related. Further constraints and requirements from IEC 61508-2, 7.4.5.3 must be considered.*

Figure 7. Allocation and target for STM32 *PST*



ASR3.1: *Compliant item* is assumed to be operating at constant failure rate and does not intrinsically require any proof tests.

ASR3.2: It is assumed that the Device operates within specified electrical specifications and environment limits. The *End user* is responsible for the compliance to this assumption.

ASR 4.1: It is assumed that both CPUs available in the Device (Arm® Cortex®M0+ and M4) are considered as safety related.

ASR4.2: It is assumed that in case multiple safety functions are implemented in the Compliant item, all functions are classified with the same *SIL* for hardware safety integrity and the same SC for software systematic capability and therefore they are not distinguishable in terms of their safety requirements.

ASR4.3: In case of multiple safety function implementations, it is assumed that *End user* is responsible to duly ensure their mutual independence.

ASR4.4: It is assumed that there are no *non-safety-related* functions implemented in *Application software*, coexisting with safety functions.

ASR5: It is assumed that the implemented safety function(s) does (do) not depend on transition of Device (or one of its CPUs) to and from a low-power state.

ASR6.1: The local safe state of *Compliant item* is the one in which either:

- SS1: *Application software*[1] is informed by the presence of a fault and a reaction by *Application software* itself is possible.
- SS2: *Application software*[1] cannot be informed by the presence of a fault or *Application software* is not able to execute a reaction.

Note: End user must take into account that random hardware failures affecting Device can compromise its operation (for example failure modes affecting the program counter prevent the correct execution of software).

The following table provides details on the SS1 and SS2 safe states.

Table 2. **SS1 and SS2 safe state details**

Safe state	Condition	*Compliant item* action	System transition to safe state - 1oo1 architecture (individual scheme)	System transition to safe state – 1oo1 architecture (collaborative scheme)	System transition to safe state – 1oo2 architecture
SS1	*Application software*[1] is informed by the presence of a fault and a reaction by *Application software* itself is possible.	Fault reporting to *Application software*	Application software[1] drives all implemented safety functions in their safe state, possibly leveraging on inter-CPU communications (CPU_SM_11)	*Application software*[1] drives the overall system in its safe state	*Application software*[1] in one of the two channels drives the overall system in its safe state
SS2	*Application software*[1] cannot be informed by the presence of a fault or *Application software* is not able to execute a reaction.	Reset signal issued by WDTe	WDTe drives all implemented safety functions in their safe state (“safe shutdown”) [2]	WDTe drives the overall system in its safe state (“safe shutdown”)[2]	PEv drives the overall system in its safe state

1. *Any of the application software running on the two different CPUs available on Device, Arm®Cortex® M0+ and M4 (or even both of them).*
2. *The safe state achievement intended here is compliant to Note on IEC 61508-2, 7.4.8.1*

ASR6.2: It is assumed that the safe state(s) defined at system level by *End user* is compatible with the assumed local safe state (SS1, SS2) for *Compliant item*.

ASR6.3: When individual scheme is adopted, it is assumed that the safe state defined at system level is compatible with the causal connection between the individual safe state of each safety functions described in Table 2.

Note: According to the requirements listed on Table 2, the detection of a fault must cause the transition to safe state for each implemented safety functions. Accordingly, even if in principle different safe states can be defined for the different implemented safety functions, their time evolution cannot be kept separated.

ASR7: *Compliant item* is assumed to be analyzed according to routes 1H and 1S of IEC 61508-2.

Note: Refer to Section 3.5 Systematic safety integrity and Section 3.6 Hardware and software diagnostics.

ASR8: *Compliant item* is assumed to be regarded as type B, as per IEC 61508-2, 7.4.4.1.3.

ASR9.1: It is assumed that the STM32WL5x dual-core is not used to build fail-operational solutions based exclusively on the presence of two different CPUs in the Device itself.

ASR9.2: It is assumed that a single STM32WL5x dual-core instance is not used to build a HFT>0 solutions based exclusively on the presence of two different CPUs in the Device itself.

ASR9.3: It is assumed that the architecture 1oo1 with individual scheme (see Section 3.2.3 Reference safety architectures - 1oo1) is not used to artificially implement a 1oo2 scheme with one single STM32WL5x dual-core device.

*Note: **ASR9.3** is placed to avoid the implementation of the same safety function of the two separate CPUs in the individual scheme, managed then with an external voter. That solution cannot be considered a true 1oo2 architecture as described in IEC 61508-6 because of the incomplete separation between the two channels.*

ASR 10.1: Device package is considered full safety related, in the framework of *Device* failure rate computations.

ASR10.2: When adopting individual scheme, is *End user* responsibility to prove the freedom from interferences between physical pins associated to different safety functions (for example, by running a pin-level FMEDA).

ASR11: It is assumed that is *End user*'s responsibility to correct manage the CPUs boot process (including, but not limited to, the handling of BOOT options) in order to implement a boot process compliant to all requirements included in this manual.

ASR12: The Sub-GHz radio (SUBGHZ) module is not used to support the implementation of any of the safety functions.

ASR13: It is assumed that the evaluation of hazards related to human factors (like misuse or security issues) related to the use of the *Compliant item* is under the full responsibility of the *End user*.

3.4 Electrical specifications and environment limits

To ensure safety integrity, the user must operate *Device*(s) within its (their) specified:

- absolute maximum rating
- capacity
- operating conditions

For electrical specifications and environmental limits of *Device*(s), refer to its (their) technical documentation such as datasheet(s) and reference manual(s) available on www.st.com.

Note: *The device operation within specified limits is a prerequisite for the correct implementation of any safety function. This is explicitly assumed within the assumptions (refer to above ASR3.2).*

3.5 Systematic safety integrity

According to the requirements of the IEC 61508-2, 7.4.2.2 clause, the *Route 1S* is considered in the safety analysis of *Device*(s). As authorized by the IEC 61508-2, 7.4.6.1 clause, the STM32 *MCU* products can be considered as standard, mass-produced electronic integrated devices, for which stringent development procedures, rigorous testing and extensive experience of use minimize the likelihood of design faults. However, ST internally assesses the compliance of the *Device* development flow, through techniques and measures suggested in the IEC 61508-2 Annex F. As highly confidential information on ST processes are concerned within the evaluation activity, the *safety case database* (see Section 5 List of evidences) keeps evidences of the current compliance level to the standard.

3.6 Hardware and software diagnostics

This section lists all the safety mechanisms (hardware, software and application-level) considered in the *Device* safety analysis. It is expected that users are familiar with the architecture of *Device*, and that this document is used in conjunction with the related *Device* datasheet, user manual and reference information. To avoid inconsistency and redundancy, this document does not report device functional details. In the following descriptions, the words *safety mechanism*, *method*, and *requirement* are used as synonyms.

As the document provides information relative to the superset of peripherals available on the devices it covers (not all devices have all peripherals), users are supposed to disregard any recommendations not applicable to their *Device* part number of interest.

Information provided for a function or peripheral applies to all instances of such function or peripheral on *Device*. Refer to its reference manual or/and datasheet for related information.

The implementation guidelines reported in the following section are for reference only. The safety verification executed by ST during the *Device* safety analysis and related diagnostic coverage figures reported in this manual (or related documents) are based on such guidelines. For clarity, safety mechanisms are grouped by *Device* function.

Information is organized in form of tables, one per safety mechanism, with the following fields:

SM CODE	Unique safety mechanism code/identifier used also in *FMEA* document. Identifiers use the scheme *mmm_SM_x* where *mmm* is a 3- or 4-letter module (function, peripheral) short name, and *x* is a number. It is possible that the numbering is not sequential (although usually incremental) and/or that the module short name is different from that used in other documents.
Description	Short mnemonic description
Ownership	ST: method is available on silicon. *End user*: method must be implemented by *End user* through *Application software* modification, hardware solutions, or both.

Detailed implementation	Detailed implementation sometimes including notes about the safety concept behind the introduction of the safety mechanism.
Error reporting	Describes how the fault detection is reported to *Application software*.
Fault detection time	Time that the safety mechanism needs to detect the hardware failure.
Addressed fault model	Reports fault model(s) addressed by the diagnostic (permanent, transient, or both), and other information: • If ranked for *Fault avoidance*: method contributes to lower the probability of occurrence of a failure • If ranked for *Systematic*: method is conceived to mitigate systematic errors (bugs) in *Application software* design
Dependency on *Device* configuration	Reports if safety mechanism implementation or characteristics change among different *Device* part numbers.
Initialization	Specific operation to be executed to activate the contribution of the safety mechanism
Periodicity	Continuous : safety mechanism is active in continuous mode. Periodic: safety mechanism is executed periodically[1]. On-demand: safety mechanism is activated in correspondence to a specified event (for instance, reception of a data message). Startup: safety mechanism to be executed only at power-up or during off-line maintenance periods. This is due to functional-only aspects or due to the poor compatibility with the correct execution of the safety function.
Test for the diagnostic	Reports specific procedure (if any and recommended) to allow on-line tests of safety mechanism efficiency. If no specific procedure applies (as for the majority of safety mechanisms), the field indicates *Not applicable*.
Multiple-fault protection	Reports the safety mechanism(s) associated in order to correctly manage a multiple-fault scenario (refer to Section 4.1.3 Notes on multiple-fault scenario).
Recommendations and known limitations	Additional recommendations or limitations (if any) not reported in other fields.

1. *In CM systems, safety mechanism can be accounted for diagnostic coverage contribution only if it is executed at least once per PST. For LD and HD systems, constraints from IEC 61508-2, 7.4.5.3 must be applied.*

3.6.1 Arm® Cortex®-M0+ CPU

Table 3. CPUM0+_SM_0

SM CODE	CPUM0+_SM_0
Description	Periodic core self-test software for Arm® Cortex®-M0+ *CPU*
Ownership	*End user* or ST (X-CUBE-STL, see Appendix A)
Detailed implementation	The software test is built around well-known techniques already addressed by IEC 61508-7, A.3.2 (Self-test by software: walking bit one-channel). To reach the required values of coverage, the self-test software is specified by means of a detailed analysis of all the *CPU* failure modes and related failure modes distribution
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen. The adoption of checksum protection on results variables and defensive programming are recommended.
Multiple-fault protection	CPU_SM_5: external watchdog
Recommendations and known limitations	This method is the main asset in STM32WL5x dual-core safety concept. *CPU* integrity is a key factor because the defined diagnostics for *MCU* peripherals are to major part software-based. Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario for details.

Table 4. **CPUM0+_SM_1**

SM CODE	CPUM0+_SM_1
Description	Control flow monitoring in *Application software*
Ownership	*End user*
Detailed implementation	A significant part of the failure distribution of Arm® Cortex®-M0+ CPU core for permanent faults is related to failure modes directly related to program counter loss of control or hang-up. Due to their intrinsic nature, such failure modes are not addressed by a standard software test method like CPUM0+_SM_0. Therefore it is necessary to implement a run-time control of the *Application software* flow, in order to monitor and detect deviation from the expected behavior due to such faults. Linking this mechanism to watchdog firing assures that severe loss of control (or, in the worst case, a program counter hang-up) is detected. The guidelines for the implementation of the method are the following: • Different internal states of the *Application software* are well documented and described (the use of a dynamic state transition graph is encouraged). • Monitoring of the correctness of each transition between different states of the *Application software* is implemented. • Transition through all expected states during the normal *Application software* program loop is checked. • A function in charge of triggering the system watchdog is implemented in order to constrain the triggering (preventing the issue of *CPU* reset by watchdog) also to the correct execution of the above-described method for program flow monitoring. The use of window feature available on internal window watchdog (WWDG) is recommended. • The use of the independent watchdog (IWDG), or an external one, helps to implement a more robust control flow mechanism fed by a different clock source. In any case, safety metrics do not depend on the kind of watchdog in use (the adoption of independent or external watchdog contributes to the mitigation of dependent failures, see Section 4.2.2 Clock)
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0: periodic core self-test software
Recommendations and known limitations	-

Table 5. **CPUM0+_SM_2**

SM CODE	CPUM0+_SM_2
Description	Double computation in *Application software*
Ownership	*End user*
Detailed implementation	A timing redundancy for safety-related computation is considered to detect transient faults affecting the Arm® Cortex®-M0+ *CPU* subparts devoted to mathematical computations and data access. The guidelines for the implementation of the method are the following: • The requirement needs be applied only to safety-relevant computation, which in case of wrong result could interfere with the system safety functions. Such computation must be therefore carefully identified in the original *Application software* source code • Both mathematical operation and comparison are intended as computation. • The redundant computation for mathematical computation is implemented by using copies of the original data for second computation, and by using an equivalent formula if possible
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0+_SM_0: periodic core self-test software
Recommendations and known limitations	*End user* is responsible to carefully avoid that the intervention of optimization features of the used compiler removes timing redundancies introduced according to this condition of use.

Table 6. **CPUM0+_SM_3**

SM CODE	CPUM0+_SM_3
Description	Arm® Cortex®-M0+ HardFault exceptions
Ownership	ST
Detailed implementation	HardFault exception raise is an intrinsic safety mechanism implemented in Arm® Cortex®-M0+ core, mainly dedicated to intercept systematic faults due to software limitations or error in software design (causing for example execution of undefined operations, unaligned address access). This safety mechanism is also able to detect hardware random faults inside the *CPU* bringing to such described abnormal operations.
Error reporting	High-priority interrupt event
Fault detection time	Depends on implementation. Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Continuous
Test for the diagnostic	It is possible to write a test procedure to verify the generation of the HardFault exception; anyway, given the expected minor contribution in terms of hardware random-failure detection, such implementation is not recommended.
Multiple-fault protection	CPUM0+_SM_0: periodic core self-test software
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 7. **CPUM0+_SM_4**

SM CODE	CPUM0+_SM_4
Description	Stack hardening for *Application software*
Ownership	*End user*
Detailed implementation	The stack hardening method is required to address faults (mainly transient) affecting Arm® Cortex®-M0+ CPU register bank. This method is based on source code modification, introducing information redundancy in register-passed information to called functions. The guidelines for the implementation of the method are the following: • To pass also a redundant copy of the passed parameters values (possibly inverted) and to execute a coherence check in the function. • To pass also a redundant copy of the passed pointers and to execute a coherence check in the function. • For parameters that are not protected by redundancy, to implement defensive programming techniques (plausibility check of passed values). For example enumerated fields are to be checked for consistency.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0+_SM_0: periodic core self-test software
Recommendations and known limitations	This method partially overlaps with defensive programming techniques required by IEC 61508 for software development. Therefore in presence of *Application software* qualified for safety integrity greater or equal to SC2, optimizations are possible.

Table 8. **CPUM0+_SM_7**

SM CODE	CPUM0+_SM_7
Description	Memory protection unit (*MPU*)
Ownership	ST
Detailed implementation	The Arm® Cortex®-M0+ CPU memory protection unit is able to detect illegal access to protected memory areas, according to criteria set by *End user*.
Error reporting	Exception raise (MemManage)
Fault detection time	Refer to functional documentation
Addressed fault model	Systematic (software errors) Permanent/transient (only program counter and memory access failures)
Dependency on *Device* configuration	None
Initialization	*MPU* registers must be programmed at start-up
Periodicity	On line
Test for the diagnostic	Not needed
Multiple-fault protection	MPUM0+_SM_0: Periodic read-back of configuration registers
Recommendations and known limitations	The use of memory partitioning and protection by *MPU* functions is highly recommended when multiple safety functions are implemented in *Application software*. The *MPU* can be indeed used to • enforce privilege rules • separate processes • enforce access rules Hardware random-failure detection capability for *MPU* is restricted to well-selected failure modes, mainly affecting program counter and memory access *CPU* functions. The associated diagnostic coverage is therefore not expected to be relevant for the safety concept of *Device*. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 9. **MPUM0+_SM_0**

SM CODE	MPUM0+_SM_0
Description	Periodic read-back of Arm® Cortex®-M0+ configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to Arm® Cortex®-M0+ MPU configuration registers (also unused by the *End user Application software*). Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 10. **MPUM0+_SM_1**

SM CODE	MPUM0+_SM_1
Description	Arm® Cortex®-M0+ MPU software test.
Ownership	*End user.*
Detailed implementation	This method tests Arm® Cortex®-M0+ MPU capability to detect and report memory accesses violating the policy enforcement implemented by the MPU itself. The implementation is based on intentionally performing memory accesses (in writing and read) to memory areas outside of the allowed by the MPU regions programming, and to collect and verify related generated error exceptions. Test can be executed with the final MPU region programming or with a dedicated one.
Error reporting	Depends on implementation.
Fault detection Time	Depends on implementation.
Addressed Fault Model	Permanent.
Dependency on device configuration	None.
Initialization	Depends on implementation.
Periodicity	On demand.
Test for the diagnostic	Not needed.
Multiple faults protection	CPUM0+_SM_0: Periodic core self test software
Recommendations and known limitations	Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario.

3.6.2 Arm® Cortex®-M4 CPU

Table 11. **CPUM4_SM_0**

SM CODE	CPUM4_SM_0
Description	Periodic core self-test software for Arm® Cortex®-M4 *CPU*
Ownership	*End user* or ST (X-CUBE-STL, see Appendix A)
Detailed implementation	The software test is built around well-known techniques already addressed by IEC 61508-7, A.3.2 (Self-test by software: walking bit one-channel). To reach the required values of coverage, the self-test software is specified by means of a detailed analysis of all the *CPU* failure modes and related failure modes distribution
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen. The adoption of checksum protection on results variables and defensive programming are recommended.
Multiple-fault protection	CPU_SM_5: external watchdog
Recommendations and known limitations	This method is the main asset in STM32WL5x dual-core safety concept. *CPU* integrity is a key factor because the defined diagnostics for *MCU* peripherals are to major part software-based. Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario for details.

Table 12. **CPUM4_SM_1**

SM CODE	CPUM4_SM_1
Description	Control flow monitoring in *Application software*
Ownership	*End user*
Detailed implementation	A significant part of the failure distribution of Arm® Cortex®-M4 CPU core for permanent faults is related to failure modes directly related to program counter loss of control or hang-up. Due to their intrinsic nature, such failure modes are not addressed by a standard software test method like CPUM4_SM_0. Therefore it is necessary to implement a run-time control of the *Application software* flow, in order to monitor and detect deviation from the expected behavior due to such faults. Linking this mechanism to watchdog firing assures that severe loss of control (or, in the worst case, a program counter hang-up) is detected. The guidelines for the implementation of the method are the following: • Different internal states of the *Application software* are well documented and described (the use of a dynamic state transition graph is encouraged). • Monitoring of the correctness of each transition between different states of the *Application software* is implemented. • Transition through all expected states during the normal *Application software* program loop is checked. • A function in charge of triggering the system watchdog is implemented in order to constrain the triggering (preventing the issue of *CPU* reset by watchdog) also to the correct execution of the above-described method for program flow monitoring. The use of window feature available on internal window watchdog (WWDG) is recommended. • The use of the independent watchdog (IWDG), or an external one, helps to implement a more robust control flow mechanism fed by a different clock source. In any case, safety metrics do not depend on the kind of watchdog in use (the adoption of independent or external watchdog contributes to the mitigation of dependent failures, see Section 4.2.2 Clock)
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	-

Table 13. **CPUM4_SM_2**

SM CODE	CPUM4_SM_2
Description	Double computation in *Application software*
Ownership	*End user*
Detailed implementation	A timing redundancy for safety-related computation is considered to detect transient faults affecting the Arm® Cortex®-M4 *CPU* subparts devoted to mathematical computations and data access. The guidelines for the implementation of the method are the following: • The requirement needs be applied only to safety-relevant computation, which in case of wrong result could interfere with the system safety functions. Such computation must be therefore carefully identified in the original *Application software* source code • Both mathematical operation and comparison are intended as computation. • The redundant computation for mathematical computation is implemented by using copies of the original data for second computation, and by using an equivalent formula if possible
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	*End user* is responsible to carefully avoid that the intervention of optimization features of the used compiler removes timing redundancies introduced according to this condition of use.

Table 14. **CPUM4_SM_3**

SM CODE	CPUM4_SM_3
Description	Arm® Cortex®-M4 HardFault exceptions
Ownership	ST
Detailed implementation	HardFault exception raise is an intrinsic safety mechanism implemented in Arm® Cortex®-M4 core, mainly dedicated to intercept systematic faults due to software limitations or error in software design (causing for example execution of undefined operations, unaligned address access). This safety mechanism is also able to detect hardware random faults inside the *CPU* bringing to such described abnormal operations.
Error reporting	High-priority interrupt event
Fault detection time	Depends on implementation. Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Continuous
Test for the diagnostic	It is possible to write a test procedure to verify the generation of the HardFault exception; anyway, given the expected minor contribution in terms of hardware random-failure detection, such implementation is not recommended.
Multiple-fault protection	CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 15. **CPUM4_SM_4**

SM CODE	CPUM4_SM_4
Description	Stack hardening for *Application software*
Ownership	*End user*
Detailed implementation	The stack hardening method is required to address faults (mainly transient) affecting Arm® Cortex®-M4 CPU register bank. This method is based on source code modification, introducing information redundancy in register-passed information to called functions. The guidelines for the implementation of the method are the following: • To pass also a redundant copy of the passed parameters values (possibly inverted) and to execute a coherence check in the function. • To pass also a redundant copy of the passed pointers and to execute a coherence check in the function. • For parameters that are not protected by redundancy, to implement defensive programming techniques (plausibility check of passed values). For example enumerated fields are to be checked for consistency.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method partially overlaps with defensive programming techniques required by IEC 61508 for software development. Therefore in presence of *Application software* qualified for safety integrity greater or equal to SC2, optimizations are possible.

Table 16. **CPUM4_SM_7**

SM CODE	CPUM4_SM_7
Description	Memory protection unit (*MPU*)
Ownership	ST
Detailed implementation	The Arm® Cortex®-M4 CPU memory protection unit is able to detect illegal access to protected memory areas, according to criteria set by *End user*.
Error reporting	Exception raise (MemManage)
Fault detection time	Refer to functional documentation
Addressed fault model	Systematic (software errors) Permanent/transient (only program counter and memory access failures)
Dependency on *Device* configuration	None
Initialization	*MPU* registers must be programmed at start-up
Periodicity	On line
Test for the diagnostic	Not needed
Multiple-fault protection	MPUM4_SM_0: Periodic read-back of configuration registers
Recommendations and known limitations	The use of memory partitioning and protection by *MPU* functions is highly recommended when multiple safety functions are implemented in *Application software*. The *MPU* can be indeed used to • enforce privilege rules • separate processes • enforce access rules Hardware random-failure detection capability for *MPU* is restricted to well-selected failure modes, mainly affecting program counter and memory access *CPU* functions. The associated diagnostic coverage is therefore not expected to be relevant for the safety concept of *Device*. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 17. **MPUM4_SM_0**

SM CODE	MPUM4_SM_0
Description	Periodic read-back of Arm® Cortex®-M4 *MPU* configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to Arm® Cortex®-M4 *MPU* configuration registers (also unused by the *End user Application software*). Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 18. **MPUM4_SM_1**

SM CODE	MPUM4_SM_1
Description	Arm® Cortex®-M4 MPU software test.
Ownership	*End user*.
Detailed implementation	This method tests Arm® Cortex®-M4 MPU capability to detect and report memory accesses violating the policy enforcement implemented by the MPU itself. The implementation is based on intentionally performing memory accesses (in writing and read) to memory areas outside of the allowed by the MPU regions programming, and to collect and verify related generated error exceptions. Test can be executed with the final MPU region programming or with a dedicated one.
Error reporting	Depends on implementation.
Fault detection Time	Depends on implementation.
Addressed Fault Model	Permanent.
Dependency on device configuration	None.
Initialization	Depends on implementation.
Periodicity	On demand.
Test for the diagnostic	Not needed.
Multiple faults protection	CPUM4_SM_0: Periodic core self test software
Recommendations and known limitations	Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario.

3.6.3 CPUs-shared safety mechanisms

Table 19. **CPU_SM_5**

SM CODE	CPU_SM_5
Description	External watchdog
Ownership	*End user*
Detailed implementation	Using an external watchdog linked to control flow monitoring method (refer to CPUM0+_SM_1 and CPUM4_SM_1) addresses failure mode of program counter or control structures of *CPU*. External watchdog can be designed to be able to generate the combination of signals needed on the final system to achieve the safe state. It is recommended to carefully check the assumed requirements about system safe state reported in Safety requirement assumptions. It also contributes to reduce potential common cause failures, because the external watchdog is clocked and supplied independently of *Device*.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	To be defined at system level (outside the scope of *Compliant item* analysis)
Multiple-fault protection	CPUM0+_SM_1, CPUM4_SM_1: control flow monitoring in *Application software* CPU_SM_6: Internal watchdogs IWDG/WWDG
Recommendations and known limitations	In case of usage of windowed watchdog, *End user* must consider possible tolerance in *Application software* execution, to avoid false error reports (affecting system availability). It is worth to note that the use of an external watchdog could be needed anyway when the *Device* is used to trigger final elements, in order to comply at system level with requirements from IEC 61508-2:2010 Table A.1/Table A.14.

Table 20. **CPU_SM_6**

SM CODE	CPU_SM_6
Description	Internal watchdogs IWDG/WWDG
Ownership	ST
Detailed implementation	Using the IDWG/WWDG watchdog linked to control flow monitoring method (refer to CPUM0+_SM_1 and CPUM4_SM_1) addresses failure mode of program counter or control structures of *CPU*.
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	IWDG/WWDG activation. It is recommended to use *hardware watchdog* in Option byte settings (IWDG/WWDG are automatically enabled after reset)
Periodicity	Continuous
Test for the diagnostic	WDG_SM_1: Software test for watchdog at startup
Multiple-fault protection	CPUM0+_SM_1, CPUM4_SM_1: control flow monitoring in *Application software* WDG_SM_0: periodic read-back of configuration registers
Recommendations and known limitations	The IWDG/WWDG intervention is able to achieve a potentially “incomplete” local safe state because it can only guarantee that *CPU* is reset. No guarantee that *Application software* can be still executed to generate combinations of output signals that might be needed by the external system to achieve the final safe state. Internal watchdog use is part of the STM32WL5x dual-core separation concept - refer to Section 3.2.5 The separation concept and to CoU_15 and CoU_16 for further information.

Table 21. CPU_SM_11

SM CODE	CPU_SM_11
Description	Cross-CPU safety information exchange
Ownership	*End user*
Detailed implementation	A communication scheme for safety information between the two CPUs is implemented to allow the exchange of following information: • *CPU* integrity check status (i.e. the correct execution of CPUM0+_SM_0, CPUM4_SM_0 on the related *CPU*) • Successful execution of each implemented software-based periodic safety mechanisms Timestamp/frame counter mechanisms (or other equivalent) must be implemented to detect missing updates of the data and to avoid multiple data consumption. Safety data must be exchanged on the shared SRAM area identified by CoU_11. It is strongly recommended to use the HSEM module to support the implementation of data exchange. Each *CPU* must force the safe state in case of a) failure reporting from the other *CPU* by messages b) wrong, incorrect, or missing message from the other *CPU*.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent/transient
Dependency on MCU configuration	N/A
Initialization	Depends on implementation
Periodicity	Periodical
Test for the diagnostic	Not needed
Multiple faults protection	CPUM0+_SM_0: periodic core self-test software CPUM4_SM_0: periodic core self-test software
Recommendations and known limitations	This method can be used to implement *CPU* communications required by CoU_14, case b) The main target of this method is to improve the separation concept (refer to Section 3.2.5 The separation concept), so its implementation could be recommended or not, depending on the architecture and *CPU* scheme selected. It must not be confused with DUAL_SM_0 which specifies communications between two different STM32WL5x dual-core and not between CPUs of the same MCU.

3.6.4 System bus architecture/BusMatrix/Peripherals interconnect matrix

Table 22. **BUS_SM_0**

SM CODE	BUS_SM_0
Description	Periodic software test for interconnections
Ownership	*End user*
Detailed implementation	The intra-chip connection resources (Bus Matrix, AHB or APB bridges) needs to be periodically tested for permanent faults detection. Note that STM32WL5x dual-core devices have no hardware safety mechanism to protect these structures. The test executes a connectivity test of these shared resources, including the testing of the arbitration mechanisms between peripherals. According to IEC 61508-2 Table A.8, A.7.4 the method is considered able to achieve high levels of coverage.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Implementation can be considered in large part as overlapping with the widely used *Periodic read-back of configuration registers* required for several peripherals.

Table 23. **BUS_SM_1**

SM CODE	BUS_SM_1
Description	Information redundancy in intra-chip data exchanges
Ownership	*End user*
Detailed implementation	This method requires to add some kind of redundancy (for example a *CRC* checksum at packet level) to each data message exchanged inside *Device*. Message integrity is verified using the checksum by the *Application software*, before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Implementation can be in large part overlapping with other safety mechanisms requiring information redundancy on data messages for communication peripherals. Optimizations are therefore possible.

Table 24. **LOCK_SM_0**

SM CODE	LOCK_SM_0
Description	Lock mechanism for configuration options
Ownership	ST
Detailed implementation	The devices feature spread protection to prevent unintended configuration changes for some peripherals and system registers (for example PVD_LOCK, timers); the spread protection detects systematic faults in software application. The use of this method is encouraged to enhance the end application robustness to systematic faults.
Error reporting	Not generated (when locked, register overwrites are just ignored)
Fault detection time	NA
Addressed fault model	None (Systematic only)
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	Not needed
Recommendations and known limitations	No DC associated because this test addresses systematic faults

3.6.5 Global TrustZone® controller (GTZC)

Table 25. **GTZC_SM_0**

SM CODE	GTZC_SM_0
Description	Periodical read-back of GTZC configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to GTZC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 26. **GTZC_SM_1**

SM CODE	GTZC_SM_1
Description	GTZC illegal access detection
Ownership	ST
Detailed implementation	In the framework of the security concept which End user can implement on STM32WL5x dual-core peripherals, the GTZC is able to detect illegal accesses violating the implemented security policy. This GTZC feature, despite mainly conceived to support system security enforcement, may contribute to mitigate the effects of systematic failures of the application software
Error reporting	Depends on peripheral configuration. Refer to functional documentation.
Fault detection time	Depends on peripheral configuration. Refer to functional documentation.
Addressed fault model	Systematic
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.6 Embedded SRAM

Table 27. **RAM_SM_0**

SM CODE	RAM_SM_0
Description	Periodic software test for static random access memory (SRAM)
Ownership	*End user* or ST (X-CUBE-STL, see Appendix A)
Detailed implementation	To enhance the coverage on SRAM data cells and to ensure adequate coverage for permanent faults affecting the address decoder it is required to execute a periodic software test on the system RAM memory. The selection of the algorithm must ensure the target SFF coverage for both the RAM cells and the address decoder. Evidences of the effectiveness of the coverage of the selected method must also be collected
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	RAM size can change according to the part number.
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according to the test implementation design strategy chosen.
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Usage of a March test C- is recommended. Because the nature of this test can be destructive, RAM contents restore must be implemented. Possible interferences with interrupt-serving routines fired during test execution must be also considered (such routines can access to RAM invalid contents). Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario. Unused RAM section can be excluded by the testing, under *End user* responsibility on actual RAM usage by final *Application software*.

Table 28. **RAM_SM_2**

SM CODE	RAM_SM_2
Description	Stack hardening for *Application software*
Ownership	*End user*
Detailed implementation	The stack hardening method is used to enhance the *Application software* robustness to SRAM faults that affect the address decoder. The method is based on source code modification, introducing information redundancy in the stack-passed information to the called functions. Method contribution is relevant in case the combination between the final *Application software* structure and the compiler settings requires a significant use of the stack for passing function parameters. Implementation is the same as method CPUM0+_SM_4, CPUM4_SM_4.
Error reporting	Refer to CPUM0+_SM_4, CPUM4_SM_4
Fault detection time	Refer to CPUM0+_SM_4, CPUM4_SM_4
Addressed fault model	Refer to CPUM0+_SM_4, CPUM4_SM_4
Dependency on *Device* configuration	Refer to CPUM0+_SM_4, CPUM4_SM_4
Initialization	Refer to CPUM0+_SM_4, CPUM4_SM_4
Periodicity	Refer to CPUM0+_SM_4, CPUM4_SM_4
Test for the diagnostic	Refer to CPUM0+_SM_4, CPUM4_SM_4
Multiple-fault protection	Refer to CPUM0+_SM_4, CPUM4_SM_4
Recommendations and known limitations	Refer to CPUM0+_SM_4, CPUM4_SM_4

Table 29. **RAM_SM_3**

SM CODE	RAM_SM_3
Description	Information redundancy for safety-related variables in the *Application software*
Ownership	*End user*
Detailed implementation	To address transient faults affecting the SRAM controller and memory cells, it is required to implement information redundancy on the safety-related system variables stored in the SRAM. The guidelines for the implementation of this method are the following: • The system variables that are safety-related (in the sense that a wrong value due to a failure in reading on the RAM affects the safety functions) are well-identified and documented. • The arithmetic computation or decision based on such variables are executed twice and the two final results are compared. • Safety-related variables are stored and updated in two redundant locations, and comparison is checked before consuming data. • Enumerated fields must use non-trivial values, checked for coherence with the same frequency as for periodically executed diagnostics (see (1) in Section 3.6 Hardware and software diagnostics). • Data vectors stored in SRAM must be protected by an encoding checksum (such as *CRC*).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Implementation of this safety method shows a partial overlap with an already foreseen method for Arm® Cortex®-M0+ (CPUM0+_SM_2), M4 (CPUM4_SM_2) ; optimizations in implementing both methods are therefore possible. Reduction to the application scope for this method is achieved by executing an accurate safety analysis of the software. Refer to [4] for details. However, the scope reduction may not be possible nor desirable.

Table 30. **RAM_SM_4**

SM CODE	RAM_SM_4
Description	Control flow monitoring in *Application software*
Ownership	*End user*
Detailed implementation	In case *End user Application software* is executed from SRAM, permanent and transient faults affecting the memory (cells and address decoder) can interfere with the program execution. The implementation of this method is required to address such failures. For more details on the implementation, refer to CPUM0+_SM_1, CPUM4_SM_1 description.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Needed only in case of *Application software* execution from SRAM. CPUM0+_SM_1, CPUM4_SM_1 correct implementation supersedes this requirement.

Table 31. **RAM_SM_5**

SM CODE	RAM_SM_5
Description	Periodic integrity test for *Application software* in RAM
Ownership	*End user*
Detailed implementation	In case *Application software* or diagnostic libraries are executed in RAM, it is needed to protect the integrity of the code itself against soft-error corruptions and related code mutations. This method must check the integrity of the stored code by checksum computation techniques, on a periodic basis. For implementation details, refer to similar method FLASH_SM_0.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according to the test implementation design strategy chosen.
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software CPUM0+_SM_1, CPUM4_SM_1: Control flow monitoring in Application software
Recommendations and known limitations	This method must only be implemented if Application software or diagnostic libraries are executed from RAM.

Table 32. **RAM_SM_8**

SM CODE	RAM_SM_8
Description	Periodic test by software for SRAM address decoder
Ownership	*End user* or ST
Detailed implementation	Permanent faults affecting the SRAM interfaces address decoder are addressed through a dedicated software test that checks the memory cells contents versus the expected value.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	SRAM size depends on the part number
Initialization	Not required
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Overlaps with RAM_SM_0 implementation are possible.

3.6.7 Embedded flash memory

Table 33. **FLASH_SM_0**

SM CODE	FLASH_SM_0
Description	Periodic software test for flash memory
Ownership	*End user* or ST (X-CUBE-STL, see Appendix A)
Detailed implementation	Permanent faults affecting the system flash memory, memory cells, and address decoder are addressed through a dedicated software test that checks the memory cells contents versus the expected value, using signature-based techniques. According to IEC 61508-2 Table A.5, the effective diagnostic coverage of such techniques depends on the width of the signature in relation to the block length of the information to be protected - therefore the signature computation method is to be carefully selected. Note that the simple signature method (IEC 61508-7 - A.4.2 Modified checksum) is inadequate as it only achieves a low value of coverage. The information block does not need to be addressed with this test as it is not used during normal operation (no data nor program fetch).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	Flash memory size changes according to the part number.
Initialization	Memory signatures must be stored in flash memory as well.
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according to the test implementation design strategy chosen.
Multiple-fault protection	CPUM0+_SM_0: Periodic core self-test software for Arm® Cortex®-M0+ CPUM4_SM_0: Periodic core self-test software for Arm® Cortex®-M4 CPUM0+_SM_1: Control flow monitoring in *Application software* CPUM4_SM_1: Control flow monitoring in *Application software*
Recommendations and known limitations	This test is expected to have a relevant time duration – test integration must therefore consider the impact on *Application software* execution. The use of internal cyclic redundancy check (CRC) module is recommended. In principle direct memory access (DMA) feature for data transfer can be used. Unused flash memory sections can be excluded from testing. Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario for details. As far as ASR2 requirement on PST duration (adoption of the same value for both CPUs) and CoU_14 are correctly implemented, the overall flash memory test can be partitioned among the two CPUs, or even delegated to only one.

Table 34. **FLASH_SM_1**

SM CODE	FLASH_SM_1
Description	Control flow monitoring in *Application software*
Ownership	*End user*
Detailed implementation	Permanent and transient faults affecting the system flash memory, memory cells and address decoder, can interfere with the access operation by the *CPU*, leading to wrong data or instruction fetches. Such failures can be detected by control flow monitoring techniques implemented in *Application software* loaded from flash memory. For more details on the implementation, refer to description CPUM0+_SM_1, CPUM4_SM_1.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	CPUM0+_SM_1, CPUM4_SM_1 correct implementation supersedes this requirement.

Table 35. **FLASH_SM_2**

SM CODE	FLASH_SM_2
Description	Arm®Cortex®-M0+ and M4 HardFault exceptions
Ownership	ST
Detailed implementation	Hardware random faults (both permanent and transient) affecting system flash memory (memory cells, address decoder) can lead to wrong instruction codes fetches, and eventually to the intervention of the Arm®Cortex®-M0+ HardFault exceptions. Refer to CPUM0+_SM_3, CPUM4_SM_3 for detailed description.
Error reporting	Refer to CPUM0+_SM_3, CPUM4_SM_3
Fault detection time	Refer to CPUM0+_SM_3, CPUM4_SM_3
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Refer to CPUM0+_SM_3, CPUM4_SM_3
Periodicity	Continuous
Test for the diagnostic	Refer to CPUM0+_SM_3, CPUM4_SM_3
Multiple-fault protection	Refer to CPUM0+_SM_3, CPUM4_SM_3
Recommendations and known limitations	Refer to CPUM0+_SM_3, CPUM4_SM_3

Table 36. **FLASH_SM_3**

SM CODE	FLASH_SM_3
Description	Option byte write protection
Ownership	ST
Detailed implementation	This safety mechanism prevents unintended writes on the option byte. The use of this method is encouraged to enhance end application robustness for systematic faults.
Error reporting	Write protection exception
Fault detection time	Not applicable
Addressed fault model	None (systematic only)
Dependency on *Device* configuration	None
Initialization	None (always enabled)
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method addresses systematic faults in software application and it have zero efficiency in addressing hardware random faults affecting the option byte value during running time. No DC value is therefore associated.

Table 37. **FLASH_SM_4**

SM CODE	FLASH_SM_4
Description	Static data encapsulation
Ownership	*End user*
Detailed implementation	If static data are stored in flash memory, encapsulation by a checksum field with encoding capability (such as *CRC*) must be implemented. Checksum validity is checked by *Application software* before static data consuming.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 38. **FLASH_SM_6**

SM CODE	FLASH_SM_6
Description	Flash memory unused area filling code
Ownership	*End user*
Detailed implementation	Used flash memory area must be filled with deterministic data. This way in case that the program counter jumps outside the application program area due to a transient fault affecting *CPU*, the system evolves in a deterministic way.
Error reporting	Not applicable
Fault detection time	Not applicable
Addressed fault model	None (fault avoidance)
Dependency on *Device* configuration	None
Initialization	Not applicable
Periodicity	Not applicable
Test for the diagnostic	Not applicable
Multiple-fault protection	Not applicable
Recommendations and known limitations	Filling code can be made of NOP instructions, or an illegal code that leads to a HardFault exception raise.

Table 39. **FLASH_SM_7**

SM CODE	FLASH_SM_7
Description	ECC on flash memory
Ownership	ST
Detailed implementation	Internal Flash memory is protected by ECC (Error Correction Code) redundancy,implementing a protection feature at double-word (64 bit) level: • one-bit fault: correction • two-bit fault: detection This safety mechanism can be associated to the description provided in IEC61508-2, Table A.5 – Invariable memory ranges - Word-protection multibit redundancy. Related achievable DC is indicated in the table as "Medium" (90%). Anyway, considering the correct implementation of all the requirements included in this safety mechanism description, the implementation of the collateral method FLASH_SM_7 as well (for address decoder protection) and the guidance of state-of-the-art safety standard ISO26262-11:2018, Table 32 — Non-volatile memory, related achievable DC can be reasonably assumed as "High" (99%). End user willing to achieve High DC in fully formal compliance to IEC61508-2 Table A.5 indication can rely on implementation of the method FLASH_SM_0
Error reporting	Correction: • ECCC flag (ECC correction) is set in the FLASH_ECCR register. • Interrupt is generated. Detection: • ECCD flag (ECC detection) is set in the FLASH_ECCR register. • NMI is generated. • The address of the failing double word and its associated bank are saved in the ADDR_ECC[20:0] and BK_ECC bitfields of the FLASH_ECCR register.
Fault detection time	ECC bits are checked during a memory reading.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for ECC efficiency is not available. ECC run-time hardware failures leading to the disable of such protection, or to wrong corrections, fall into a *multiple-fault scenario* from IEC61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*. See also the **Recommendations and known limitations** in this table.
Multiple-fault protection	FLASH_SM_0: Periodic software test for flash memory DIAG_SM_0: Periodic read-back of hardware diagnostics configuration registers CPUM0+_SM_3: Arm® Cortex®-M0+ HardFault exceptions CPUM4_SM_3: Arm® Cortex®-M4 HardFault exceptions
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended. Due to the fact that ECC is checked during memory reads, Flash section occupied by safety related program/data which are rarely accessed (for instance, code related to failures/errors management) are potentially exposed to the risk of error accumulation. In such a case, it is recommended to periodically check those locations with FLASH_SM_0 method.

Table 40. **FLASH_SM_8**

SM CODE	FLASH_SM_8
Description	Read protection (RDP), write protection (WRP)
Ownership	ST
Detailed implementation	Flash memory can be protected against illegal read or erase/write accesses by using these protection features. The combination of these techniques and the related different protection levels allows *End user* to build an effective access protection policy.
Error reporting	Refer to functional documentation. In some cases, a HardFault error is generated.
Fault detection time	Refer to functional documentation.
Addressed fault model	Systematic
Dependency on *Device* configuration	None
Initialization	Not required
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	Not required
Recommendations and known limitations	Hardware random-failure detection capability for Flash memory access policy is restricted to well-selected marginal failure modes, mainly affecting program counter and Flash memory interface functions. The associated diagnostic coverage is therefore expected to be irrelevant in the framework of STM32WL5x dual-core safety concept.

Table 41. **FLASH_SM_9**

SM CODE	FLASH_SM_9
Description	Periodic test by software for flash memory address decoder
Ownership	*End user*
Detailed implementation	Permanent faults affecting the system flash memory interface address decoder are addressed through a dedicated software test that checks the memory cells contents versus the expected value.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	flash memory size depends on part number.
Initialization	Not required
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Overlaps with FLASH_SM_0 implementation are possible.

3.6.8 Power controller (PWR)

Table 42. **VSUP_SM_0**

SM CODE	VSUP_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 43. **VSUP_SM_1**

SM CODE	VSUP_SM_1
Description	Supply voltage internal monitoring (PVD)
Ownership	ST
Detailed implementation	The device features an embedded programmable voltage detector (PVD) that monitors the V_{DD} power supply and compares it to the V_{PVD} threshold. An interrupt can be generated when V_{DD} drops below the V_{PVD} threshold or when V_{DD} is higher than the V_{PVD} threshold.
Error reporting	Interrupt event generation
Fault detection time	Depends on threshold programming. Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Protection enable by the PVDE bit and the threshold setting in the Power control register (PWR_CR)
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for PVD efficiency is not available. PVD run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*.
Multiple-fault protection	DIAG_SM_0: Periodic read-back of hardware diagnostics configuration registers
Recommendations and known limitations	Internal monitoring PVD has limited capability to address failures affecting STM32WL5x dual-core internal voltage regulator. Refer to [1] for details. Internal monitoring PVD has limited capability to address failures affecting the internal voltage regulator. Refer to *Device FMEA* for details. In case the hardware option is not available on the chosen partnumbers, its contribution to the overall safety concept is supported by other overlapping methods indicated for the mitigation of failures affecting internal power.

Table 44. **VSUP_SM_2**

SM CODE	VSUP_SM_2
Description	Independent watchdog
Ownership	ST
Detailed implementation	Failures in the power supplies for digital logic (core or peripherals) may lead to alteration of *Application software* timing, which can be detected by IWDG as safety mechanism introduced to monitor the *Application software* control flow. Refer to CPU_SM_1 and CPU_SM_6 for further information.
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	IWDG activation. It is recommended to use *Hardware watchdog* in Option byte settings (IWDG is automatically enabled after reset).
Periodicity	Continuous
Test for the diagnostic	Refer to CPU_SM_6.
Multiple-fault protection	CPUM0+_SM_1, CPUM4_SM_1: Control flow monitoring in Application software
Recommendations and known limitations	In specific part numbers, IWDG can be fed by a power supply independent from the one used for CPU core and main peripherals. Such diversity helps to increase the protection guaranteed by IWDG from main power supply anomalies. The adoption of an external watchdog (refer to CPU_SM_5) adds further diversity.

Table 45. **VSUP_SM_3**

SM CODE	VSUP_SM_3
Description	Internal temperature sensor check
Ownership	*End user*
Detailed implementation	The internal temperature sensor must be periodically tested in order to detect abnormal increase of the die temperature – hardware faults in supply voltage system may cause excessive power consumption and consequent temperature rise.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method also mitigates the probability of common-cause failure due to excessive temperature, affecting the *Device*. Refer to the *Device* datasheet to set the threshold temperature.

Table 46. **VSUP_SM_5**

SM CODE	VSUP_SM_5
Description	System-level power supply management
Ownership	*End user*
Detailed implementation	This method is implemented at system level in order to guarantee the stability of power supply value over time. It can include a combination of different overlapped solutions, some listed here below (but not limited to): • additional voltage monitoring by external components • passive electronics devices able to mitigate overvoltage • specific design of power regulator in order to avoid power supply disturbance in presence of a single failure
Error reporting	Depends on implementation
Fault detection time	Fault avoidance
Addressed fault model	None
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	Not applicable
Recommendations and known limitations	Usually, this method is already required/implemented to guarantee the stability of each component of the final electronic board.

3.6.9 Reset and clock controller (RCC)

Table 47. **CLK_SM_0**

SM CODE	CLK_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to configuration registers for clock and reset system (refer to RCC register map). Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 48. **CLK_SM_1**

SM CODE	CLK_SM_1
Description	Clock security system (CSS)
Ownership	ST
Detailed implementation	The clock security system (CSS) detects the loss of high-speed external (HSE) oscillator clock activity and executes the corresponding recovery action, such as: • switch-off HSE • commutation on the HSI • generation of related NMI
Error reporting	NMI
Fault detection time	Depends on implementation (clock frequency value)
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	CSS protection must be enabled through Clock interrupt register (RCC_CIR) after boot.
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_5: External watchdog CLK_SM_0: Periodic read-back of configuration registers
Recommendations and known limitations	It is recommended to carefully read reference manual instruction on NMI generation, in order to correctly managing the faulty situation by *Application software*. As the test of the diagnostic is not available in the hardware, it must be done at system level during startup or maintenance period. The use of this method to implement fail operational schemes is not recommended.

Table 49. **CLK_SM_2**

SM CODE	CLK_SM_2
Description	Independent watchdog
Ownership	ST
Detailed implementation	The independent watchdog IWDG is able to detect failures in internal main *MCU* clock (lower frequency).
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	IWDG activation. It is recommended to use the *hardware watchdog* in Option byte settings (IWDG is automatically enabled after reset).
Periodicity	Continuous
Test for the diagnostic	Refer to CPU_SM_6.
Multiple-fault protection	CPUM0+_SM_1, CPUM4_SM_1: Control flow monitoring in Application software
Recommendations and known limitations	The adoption of an external watchdog (refer to CPU_SM_5) adds further diversity.

Table 50. **CLK_SM_3**

SM CODE	CLK_SM_3
Description	Internal clock cross-measurement
Ownership	*End user*
Detailed implementation	This method is implemented using general-purpose timers capabilities to be fed by the 32 KHz RTC clock or an external clock source (if available). Timer counter progress is compared with another counter (fed by internal clock). Abnormal values of oscillator frequency can therefore be detected.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_1, CPUM4_SM_1: Control flow monitoring in Application software CPU_SM_5: External watchdog
Recommendations and known limitations	Efficiency versus transient faults is negligible. It provides only medium efficiency in permanent clock-related failure mode coverage.

3.6.10 Hardware semaphore (HSEM)

Table 51. **HSEM_SM_0**

SM CODE	HSEM_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to HSEM configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 52. **HSEM_SM_1**

SM CODE	HSEM_SM_1
Description	Control flow monitoring for concurrent tasks
Ownership	*End user*
Detailed implementation	This method is intended to monitor the correct execution of software tasks that use the HSEM semaphore method for their synchronization. The method is implemented by software, leveraging on the presence of a system watchdog (internal or external). The watchdog periodic reset function must be constrained to the correct timing execution of each software task synchronized by semaphores.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method must be extended to any software task using an HSEM semaphores function for synchronization, regardless task nature (safety relevant or non-safety relevant). Implementation must take into account potential overlaps/optimizations with CPUM0+_SM_1, CPUM4_SM_1.

3.6.11 Inter-processor communication controller (IPCC)

Table 53. **IPCC_SM_0**

SM CODE	IPCC_SM_0
Description	Periodic read-back of configuration registers.
Ownership	End user
Detailed implementation	This method must be applied to IPCC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0.
Fault detection time	
Addressed fault model	
Dependency on device configuration	
Initialization	
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	

Table 54. **IPCC_SM_1**

SM CODE	IPCC_SM_1
Description	End-to-end protection for inter-CPU processor communications
Ownership	End user
Detailed implementation	This method uses the defined end-to-end protection techniques to protect the integrity and the determinism of the messages exchanged by the two CPUs with the IPCC arbitration. Refer to UART_SM_3 description for detailed information.
Error reporting	Depends on implementation.
Fault detection time	
Addressed fault model	Permanent/transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPU1_SM_0: periodic core self-test software for Arm Cortex-M4 CPU CPU2_SM_0: periodic core self-test software for Arm Cortex-M0+ CPU
Recommendations and known limitations	This method protects data exchanged by dedicated RAM mailbox and the sequence/flow of messages data between CPUs (based on IPCC arbitration).

3.6.12 General-purpose input/output (GPIO)

Table 55. **GPIO_SM_0**

SM CODE	GPIO_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to GPIO configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	GPIO availability can differ according to part number
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	The execution of the method before any update on GPIO registers helps to mitigate the possibility of unintended glitches on outputs due to soft errors. For more information refer to [4].

Table 56. **GPIO_SM_1**

SM CODE	GPIO_SM_1
Description	1oo2 for input GPIO lines
Ownership	*End user*
Detailed implementation	This method addresses GPIO lines used as inputs. Implementation is done by connecting the external safety-related signal to two independent GPIO lines. Comparison between the two GPIO values is executed by the *Application software* each time the signal is used to affect *Application software* behavior. This method applies to the single GPIO line used as input.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Permanent/transient
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	To reduce the potential impact of common cause failure, it is recommended to use GPIO lines: • belonging to different I/O ports (for instance port A and B) • with different bit port number (for instance PA1 and PB5) • mapped to non-adjacent pins on the device package As GPIO pins are shared with other *MCU* functions, this method must not be applied to pin connections already used by another peripheral and addressed by related safety mechanisms.

Table 57. **GPIO_SM_2**

SM CODE	GPIO_SM_2
Description	Loopback scheme for output GPIO lines
Ownership	*End user*
Detailed implementation	This method addresses GPIO lines used as outputs. Implementation is done by a loopback scheme, connecting the output to a different GPIO line programmed as input and by using the input line to check the expected value on output port. Comparison is executed by the *Application software* periodically and each time output is updated. This method applies to the single GPIO line used as output.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	To reduce the potential impact of common cause failure, it is recommended to use GPIO lines: • belonging to different I/O ports (for instance port A and B) • with different bit port number (for instance PA1 and PB5) • mapped to non-adjacent pins on the device package Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of GPIO output wrong signal permanence required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm. As GPIO pins are shared with other *MCU* functions, this method must not be applied to pin connections already used by another peripheral and addressed by related safety mechanisms.

Table 58. **GPIO_SM_3**

SM CODE	GPIO_SM_3
Description	GPIO port configuration lock register
Ownership	ST
Detailed implementation	This safety mechanism prevents configuration changes for GPIO registers; it addresses therefore systematic faults in software application. The use of this method is encouraged to enhance the end-application robustness for systematic faults.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	None (Systematic only)
Dependency on *Device* configuration	None
Initialization	*Application software* must apply a correct locking write sequence after writing the final GPIO configuration.
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	Not required
Recommendations and known limitations	This method does not address transient faults (soft errors) that can possibly cause bit-flips on GPIO registers at running time.

3.6.13 Debug system or peripheral control

Table 59. DBG_SM_0

SM CODE	DBG_SM_0
Description	Watchdog protection
Ownership	ST
Detailed implementation	The debug unintentional activation due to hardware random fault results in the massive disturbance of *CPU* operations, leading to an intervention of the independent watchdog or, alternatively, the other system watchdog WWDG or the external one (CPU_SM_5).
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval).
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Refer to CPU_SM_6.
Multiple-fault protection	CPUM0+_SM_1, CPUM4_SM_1: Control flow monitoring in Application software
Recommendations and known limitations	None

3.6.14 System configuration controller (SYSCFG)

Table 60. SYSCFG_SM_0

SM CODE	SYSCFG_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to system configuration controller configuration registers. This method is strongly recommended to protect registers related to hardware diagnostics activation and error reporting chain related features. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	This method is mainly overlapped by several other configuration register read-backs required for other *MCU* peripherals. It is reported here for the sake of completeness.

Table 61. **DIAG_SM_0**

SM CODE	DIAG_SM_0
Description	Periodic read-back of hardware diagnostics configuration registers
Ownership	*End user*
Detailed implementation	In STM32WL5x dual-core, several hardware-based safety mechanisms are available (those with the *Ownership* field set to ST). This method must be applied to any configuration register related to diagnostic measure operations, including error reporting. *End user* must therefore individuate configuration registers related to: • hardware diagnostic enable • interrupt/NMI enable (if used for diagnostic error management)
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

3.6.15 Direct memory access controller (DMA), DMA request multiplexer (DMAMUX)

Table 62. **DMA_SM_0**

SM CODE	DMA_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to *DMA* configuration register and channel address register. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 63. **DMA_SM_1**

SM CODE	DMA_SM_1
Description	Information redundancy on data packet transferred via *DMA*
Ownership	*End user*
Detailed implementation	This method is implemented by adding, to data packets transferred by *DMA*, a redundancy check (such as *CRC* check or similar one) with encoding capability. Full data packet redundancy would be an overkill. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	To give an example about checksum encoding capability, using just a bit-by-bit addition is inappropriate.

Table 64. **DMA_SM_2**

SM CODE	DMA_SM_2
Description	Information redundancy by including sender or receiver identifier on data packet transferred via *DMA*
Ownership	*End user*
Detailed implementation	This method helps to identify inside the MCU the source and the originator of the message exchanged by *DMA*. Implementation is realized by adding an additional field to protected message, with a coding convention for message type identification fixed at *Device* level. Guidelines for the identification fields are: • Identification field value must be different for each possible couple of sender or receiver on *DMA* transactions. • Values chosen must be enumerated and non-trivial. • Coherence between the identification field value and the message type is checked by the *Application software* before consuming data. This method, when implemented in combination with DMA_SM_4, makes available a kind of *virtual channel* between source and destinations entities.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 65. **DMA_SM_3**

SM CODE	DMA_SM_3
Description	Periodic software test for *DMA*
Ownership	*End user*
Detailed implementation	This method requires the periodical testing of the DMA basic functionality, implemented through a deterministic transfer of a data packet from one source to another (for example from memory to memory) and the checking of the correct transfer of the message on the target. Data packets are composed by non-trivial patterns (avoid the use of 0x0000, 0xFFFF values) and organized in order to allow the detection during the check of the following failures: • incomplete packed transfer • errors in single transferred word • wrong order in packed transmitted data
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 66. **DMA_SM_4**

SM CODE	DMA_SM_4
Description	*DMA* transaction awareness
Ownership	*End user*
Detailed implementation	DMA transactions are non-deterministic by nature, because typically driven by external events like communication messages reception. Anyway, well-designed safety systems should keep much control as possible of events – refer for instance to IEC 61508-3 Table 2 item 13 requirements for software architecture. This method is based on system knowledge of frequency and type of expected *DMA* transaction. For instance, an externally connected sensor supposed to send periodically some messages to a STM32 peripheral. Monitoring *DMA* transaction by a dedicated state machine allows the detection of missing or unexpected *DMA* activities.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Because *DMA* transaction termination is often linked to an interrupt generation, implementation of this method can be merged with the safety mechanism NVIC_SM_1: Expected and unexpected interrupt check.

3.6.16 Extended interrupt and events controller (EXTI)

Table 67. NVIC_SM_0

SM CODE	NVIC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This test is implemented by executing a periodic check of the configuration registers for a system peripheral against its expected value. Expected values are previously stored in RAM and adequately updated after each configuration change. The method mainly addresses transient faults affecting the configuration registers, by detecting bit flips in the registers contents. It addresses also permanent faults on registers because it is executed at least once per *PST* (or another timing constraint; refer to [(1)] in Section 3.6 Hardware and software diagnostics) after an update of the peripheral. Method must be implemented to any configuration register whose contents are able to interfere with NVIC or EXTI behavior in case of incorrect settings. Check includes NVIC vector table. According to the state-of-the-art automotive safety standard ISO26262, this method can achieve high levels of diagnostic coverage (DC) (refer to ISO26262-5:2018, Table D.4). An alternative valid implementation requiring less space in SRAM can be realized on the basis of signature concept: • Peripheral registers to be checked are read in a row, computing a *CRC* checksum (use of hardware *CRC* is encouraged). • Obtained signature is compared with the golden value (computed in the same way after each register update, and stored in SRAM). • Coherence between signatures is checked by *Application software* – signature mismatch is considered as failure detection.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method addresses only failures affecting configuration registers, and not peripheral core logic or external interface. Attention must be paid to registers containing mixed combination of configuration and status bits. Mask must be used before saving register contents affecting signature, and related checks done, to avoid false positive detections.

Table 68. **NVIC_SM_1**

SM CODE	NVIC_SM_1
Description	Expected and unexpected interrupt check
Ownership	*End user*
Detailed implementation	According to IEC 61508-2 Table A.1 recommendations, a diagnostic measure for continuous, absence or cross-over of interrupt must be implemented. The method of expected and unexpected interrupt check is implemented at *Application software* level. The guidelines for the implementation of the method are the following: • The interrupts implemented on the *MCU* are well documented, also reporting, when possible, the expected frequency of each request (for example, the interrupts related to ADC conversion completion that come on a regular basis). • Individual counters are maintained for each interrupt request served, in order to detect in a given time frame the cases of a) no interrupt at all b) too many interrupt requests. The control of the time frame duration must be regulated according to the individual interrupt expected frequency. • Interrupt vectors related to unused interrupt source point to a default handler that reports, in case of triggering, a faulty condition (unexpected interrupt). • In case an interrupt service routine is shared between different sources, a plausibility check on the caller identity is implemented. *Important:* *Interrupt requests generated by non-safety-related peripherals must be handled using the same method as all safety related interupts outlined in the list above.*
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	The extension of the method to non-safety related peripherals (see last bullet in "Detailed implementation" box above) is introduced to mitigate interferences between non-safety and safety functions/hardware (FFI).

3.6.17 Cyclic redundancy-check calculation unit (CRC)

Table 69. **CRC_SM_0**

SM CODE	CRC_SM_0
Description	CRC self-coverage
Ownership	ST
Detailed implementation	The *CRC* algorithm implemented in this module (CRC-32 Ethernet polynomial: 0x4C11DB7) offers excellent features in terms of error detection in the message. Therefore permanent and transient faults affecting *CRC* computations are easily detected by any operations using the module to recompute an expected signature.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.18 Analog-to-digital converter (ADC)

Table 70. **ADC_SM_0**

SM CODE	ADC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to the ADC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 71. **ADC_SM_1**

SM CODE	ADC_SM_1
Description	Multiple acquisition by *Application software*
Ownership	*End user*
Detailed implementation	This method implements a timing information redundancy by executing multiple acquisitions on the same input signal. Multiple data acquisitions are then combined by a filter algorithm to determine the signal correct value.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Depends on implementation
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	It is highly probable that this recommendation is satisfied by design by the *End userApplication software*. Usage of multiple acquisitions followed by average operations is a common technique in industrial applications exposed to electromagnetic interference on sensor lines.

Table 72. **ADC_SM_2**

SM CODE	ADC_SM_2
Description	Range check by *Application software*
Ownership	*End user*
Detailed implementation	The guidelines for the implementation of the method are the following: • The expected range of the data to be acquired are investigated and adequately documented. Note that in a well-designed application it is improbable that during normal operation an input signal has a very near or over the upper and lower rail limit (saturation in signal acquisition). • If the *Application software* is aware of the state of the system, this information is to be used in the range check implementation. For example, if the ADC value is the measurement of a current through a power load, reading an abnormal value such as a current flowing in opposite direction versus the load supply may indicate a fault in the acquisition module. • As the ADC module is shared between different possible external sources, the combination of plausibility checks on the different signals acquired can help to cover the whole input range in a very efficient way.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Depends on implementation
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	The implementation and the related diagnostic efficiency of this safety mechanism are strongly application-dependent.

Table 73. **ADC_SM_3**

SM CODE	ADC_SM_3
Description	Periodic software test for ADC
Ownership	*End user*
Detailed implementation	The method is implemented acquiring multiple signals and comparing the read value with the expected one, supposed to be known. Method can be implemented with different level of complexity: • Basic complexity: acquisition and check of upper or lower rails (VDD or VSS) and internal reference voltage • High complexity: in addition to basic complexity tests, acquisition of a DAC output connected to ADC input and checking all voltage excursion and linearity
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Combination of two methods with different complexity can be used to better optimize test frequency in high-demand safety functions.

Table 74. **ADC_SM_4**

SM CODE	ADC_SM_4
Description	1oo2 scheme for ADC inputs
Ownership	*End user*
Detailed implementation	This safety mechanism is implemented using two different SAR ADC channels belonging to separate ADC modules to acquire the same input signal. The *Application software* checks the coherence between the two readings.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	ADC_SM_0: Periodic read-back of configuration registers
Recommendations and known limitations	This method can be used in conjunction with ADC_SM_0 / ADC_SM_2 / ADC_SM_3 to achieve highest level of ADC module diagnostic coverage.

3.6.19 Digital-to-analog converter (DAC)

Table 75. **DAC_SM_0**

SM CODE	DAC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to DAC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 76. **DAC_SM_1**

SM CODE	DAC_SM_1
Description	DAC output loopback on ADC channel
Ownership	*End user*
Detailed implementation	Route the active DAC output to one ADC channel, and check the output current value against the expected one.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous or on demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of DAC wrong signal permanence required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm.

3.6.20 Voltage reference buffer (VREFBUF)

Table 77. VREF_SM_0

SM CODE	VREF_SM_0
Description	Periodic read-back of VREFBUF system configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to VREFBUF configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Refer to NVIC_SM_0

Table 78. VREF_SM_1

SM CODE	VREF_SM_1
Description	VREF cross-check by ADC reading
Ownership	*End user*
Detailed implementation	This method is based on ADC acquisition for VREF generated signal, to crosscheck with the expected value.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Overlaps with ADC_SM_3 are possible.

3.6.21 Comparator (COMP)

Table 79. **COMP_SM_0**

SM CODE	COMP_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to COMP configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 80. **COMP_SM_1**

SM CODE	COMP_SM_1
Description	1oo2 scheme for comparator
Ownership	*End user*
Detailed implementation	This safety mechanism is implemented using the two internal comparators to take the same decision. It requires that the comparator voting is handled accordingly.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method is not compatible with *window* comparator feature.

Table 81. **COMP_SM_2**

SM CODE	COMP_SM_2
Description	Plausibility check on inputs
Ownership	*End user*
Detailed implementation	This method is used to redundantly acquire on dedicated ADC channels the analog inputs that are subjected to comparator function, and to periodically check the coherence of the comparator output on the measured values.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 82. **COMP_SM_3**

SM CODE	COMP_SM_3
Description	Multiple acquisition by *Application software*
Ownership	*End user*
Detailed implementation	This method requires that *Application software* takes a decision not on the basis of a comparator single-shot transition, but after multiple events or after the permanence of comparator trigger conditions for a certain amount of time.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	It is highly probable that this recommendation is satisfied by design on *End user* application - multiple acquisition is a common technique in industrial applications facing electromagnetic interference on sensor lines.

Table 83. **COMP_SM_4**

SM CODE	COMP_SM_4
Description	Comparator lock mechanism
Ownership	ST
Detailed implementation	This safety mechanism prevents configuration changes for comparator control and status registers; it addresses therefore systematic faults in the software application.
Error reporting	Not applicable
Fault detection time	Not applicable
Addressed fault model	None (Fault avoidance)
Dependency on *Device* configuration	None
Initialization	Lock protection must be enabled through the COMPxLOCK bits of the COMP_CSR register.
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	Not applicable
Recommendations and known limitations	This method does not addresses comparator configuration changes due to soft errors.

3.6.22 Public key accelerator (PKA)

Table 84. **PKA_SM_0**

SM CODE	PKA_SM_0
Description	Periodical read-back of PKA configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to PKA configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 85. **PKA_SM_1**

SM CODE	PKA_SM_1
Description	PKA key computation collateral detection
Ownership	ST
Detailed implementation	Key and/or signature computation performed by PKA module is composed by several data manipulations, with different level of complexity according to the selected algorithm. A large part of the hardware random failures affecting PKA module or its private RAM bank leads to wrong key/signature computation. As per CoU_9, security violations must be considered as non-controllable hardware random failures; accordingly, detection of a wrong key/signature must be considered as a valid hardware failure detection by the Application software.

SM CODE	PKA_SM_1
Error reporting	Depends on peripheral configuration. Refer to functional documentation.
Fault detection time	Depends on peripheral configuration. Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.23 True random number generator (RNG)

Table 86. **RNG_SM_0**

SM CODE	RNG_SM_0
Description	Periodic read-back of RNG configuration register
Ownership	*End user*
Detailed implementation	This method must be applied to RNG configuration register RNG_CR. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	RNG module available only on specific part numbers
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 87. **RNG_SM_1**

SM CODE	RNG_SM_1
Description	RNG module entropy on-line tests
Ownership	ST and *End user*
Detailed implementation	RNG module include an internal diagnostic for the analog source entropy that can be used to detect failures on the module itself. Furthermore, the required test on generated random number difference between the previous one (as required by FIPS PUB 140-2) can be exploited as well. Implementation: • Check for RNG error conditions. • Check the difference between generated random number and the previous one.
Error reporting	CEIS, SEIS error bits of the RNG status register (RNG_SR) *Application software* error for FIPS PUB 140-2 test fail
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	RNG module available only on specific part numbers
Initialization	Permanent/transient
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.24 Advanced encryption standard hardware accelerator (AES)

Table 88. **AES_SM_0**

SM CODE	AES_SM_0
Description	Periodic read-back of AES configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to AES configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	AES module available only on specific part numbers
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 89. **AES_SM_1**

SM CODE	AES_SM_1
Description	Encryption/decryption collateral detection
Ownership	ST
Detailed implementation	Encryption and decryption operations performed by AES module are composed by several data manipulations and checks, with different level of complexity according to the selected chaining algorithm. A major part of the hardware random failures affecting AES module leads to algorithm violations/errors. Leading to decoding errors on the receiver side.
Error reporting	Several error conditions can happen, check functional documentation.
Fault detection time	Dependency on *Device* configuration
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	AES module available only on specific part numbers
Initialization	Dependency on *Device* configuration
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for AES efficiency is not available. AES run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*.
Multiple-fault protection	AES_SM_2: Information redundancy techniques on messages, including end-to-end protection
Recommendations and known limitations	This detection capability can be used to implement software-based tests (by processing a predefined message and further checking the expected results) which can be executed periodically to early detect AES failures before its use by application software.

Table 90. **AES_SM_2**

SM CODE	AES_SM_2
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	*End user*
Detailed implementation	This method aim to protect the communication between a peripheral and his external counterpart. It is used in AES local safety concept to address failures not detected by the encryption/decryption features. Refer to UART_SM_3 description for detailed information.
Error reporting	Refer to UART_SM_3
Fault detection time	Refer to UART_SM_3
Addressed fault model	Refer to UART_SM_3
Dependency on *Device* configuration	AES module available only on specific part numbers
Initialization	Refer to UART_SM_3
Periodicity	Refer to UART_SM_3
Test for the diagnostic	Refer to UART_SM_3
Multiple-fault protection	Refer to UART_SM_3
Recommendations and known limitations	Important note: it is assumed that the remote counterpart has an equivalent capability of performing the checks described. Refer to UART_SM_3 for further notice.

Important: *Hardware random failure consequences on potential violations of Device security feature are* ***not*** *detailed in this manual.*

3.6.25 Advanced-control/General-purpose and Low-power timers

As the timers have multiple mutually independent channels possibly used for different functions, the safety mechanism is selected individually for each channel.

Table 91. **ATIM_SM_0**

SM CODE	ATIM_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to advanced, general-purpose and low-power timer configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 92. **ATIM_SM_1**

SM CODE	ATIM_SM_1
Description	1oo2 for counting timers
Ownership	*End user*
Detailed implementation	This method implements via software a 1oo2 scheme between two counting resources. The guidelines for the implementation of the method are the following: • Two timers are programmed with same time base or frequency. • In case of timer use as a time base: use in *Application software* one of the timer as time base source, and the other one just for check. Coherence check for the 1oo2 is done at application level, comparing two counter values each time the timer value is used to affect safety function. • In case of interrupt generation: use the first timer as main interrupt source for the service routines, and the second timer as a “reference” to be checked at the initial of interrupt routine.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic. This method applies to timer channels merely used as elapsed time counters. Events related to timers protected by the safety mechanisms must be monitored inside the routine managing the external watchdog (CPU_SM_5) reset. *Note: One timer may act as a reference for multiple other timers.*

Table 93. **ATIM_SM_2**

SM CODE	ATIM_SM_2
Description	1oo2 for input capture timers
Ownership	*End user*
Detailed implementation	This method is conceived to protect timers used for acquisition and measurement of external signals (input capture, encoder reading). The implementation consists in connecting the external signals also to a redundant timer, and checking the coherence of the measured data at application level. Coherence check between timers is executed each time the reading is used by *Application software*.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	To reduce the potential effect of common cause failures, it is suggested to use for redundant check a channel belonging to a different timer module and mapped to non-adjacent pin on the device package.

Table 94. **ATIM_SM_3**

SM CODE	ATIM_SM_3
Description	Loopback scheme for pulse width modulation (PWM) outputs
Ownership	*End user*
Detailed implementation	This method is implemented by connecting the PWM to a separate timer channel to acquire the generated waveform characteristics. The guidelines are the following: • Both PWM frequency and duty cycle are measured and checked versus the expected value. • To reduce the potential effect of common cause failure, it is suggested to use for the loopback check a channel belonging to a different timer module and mapped to non-adjacent pins on the device package. This measure can be replaced under the end-user responsibility by different loopback schemes already in place in the final application and rated as equivalent. For example if the PWM is used to drive an external power load, the reading of the on-line current value can be used instead of the PWM duty cycle measurement.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Depends on implementation
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of PWM wrong signal permanence (wrong frequency, wrong duty, or both) required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm.

Table 95. **ATIM_SM_4**

SM CODE	ATIM_SM_4
Description	Lock bit protection for timers
Ownership	ST
Detailed implementation	This safety mechanism allows *End user* to lock down specified configuration options, thus avoiding unintended modifications by *Application software*. Therefore, it addresses software development systematic faults.
Error reporting	Not applicable
Fault detection time	Not applicable
Addressed fault model	None (Fault avoidance)
Dependency on *Device* configuration	None
Initialization	Lock protection must be enabled using LOCK bits in the TIMx_BDTR register.
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	Not applicable
Recommendations and known limitations	This method does not address timer configuration changes due to soft errors.

Note: IRTIM is not individually mentioned here as its implementation is mostly based on general-purpose timer functions. Refer to related prescriptions.

3.6.26 Independent and system window watchdogs (IWDG and WWDG)

Table 96. **WDG_SM_0**

SM CODE	WDG_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to IWDG/WWDG configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 97. **WDG_SM_1**

SM CODE	WDG_SM_1
Description	Software test for watchdog at startup
Ownership	*End user*
Detailed implementation	This safety mechanism ensures the right functionality of the internal watchdogs in use. The test implementation allows the application software to induce a watchdog reset for a specific purpose such as at startup, and to determine that the cause of the reset was the test procedure itself, and not a software/hardware malfunction. This is confirmed by reading the associated hardware flag in the RCC status register before and after the test and applying specific SW flag, which stores nontrivial pattern at SRAM, just during the test execution. Both the *HW* and SW flags must be cleared once the test is done. This is essential to avoid repeating the test in a loop, and to correctly manage watchdog resets related to failures.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Startup
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	In a typical *End user* application, this test can be executed only at startup and during maintenance or offline periods. It could be associated to IEC 61508 concept of “proof test” and so it cannot be accounted for a diagnostic coverage contribution during operating time.

3.6.27 Real-time clock module (RTC)

Table 98. **RTC_SM_0**

SM CODE	RTC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to RTC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 99. **RTC_SM_1**

SM CODE	RTC_SM_1
Description	Application check of running RTC
Ownership	*End user*
Detailed implementation	The *Application software* implements some plausibility check on RTC calendar or timing data, mainly after a power-up and further date reading by RTC. The guidelines for the implementation of the method are the following: • RTC backup registers are used to store coded information in order to detect the absence of VBAT during power-off period. • RTC backup registers are used to periodically store compressed information on current date or time • The *Application software* executes minimal consistence checks for date reading after power-on (detecting "past" date or time retrieve). • The *Application software* periodically checks that RTC is actually running, by reading RTC timestamp progress and comparing with an elapsed time measurement based on STM32 internal clock or timers.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method provides a limited diagnostic coverage for RTC failure modes. In case of *End user* application where RTC timestamps accuracy can affect in severe way the safety function (for example, medical data storage devices), it is strongly recommended to adopt more efficient system-level measures.

Table 100. RTC_SM_2

SM CODE	RTC_SM_2
Description	Information redundancy on backup registers
Ownership	*End user*
Detailed implementation	Data stored in RTC backup registers must be protected by a checksum with encoding capability (for instance, CRC). Checksum must be checked by application software before consuming stored data. This method guarantees data versus erases due to backup battery failures.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic/On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 101. RTC_SM_3

SM CODE	RTC_SM_3
Description	Application-level measures to detect failures in timestamps/event capture
Ownership	*End user*
Detailed implementation	This method must detect failures affecting the RTC capability to correct execute the timestamps/event capture functions. Due to the nature strictly application-dependent of this solution, no detailed guidelines for its implementation are given here.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic/On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	This method must be used only if the timestamps/event capture function is used in the safety function implementation. It is worth noting that the use of timestamp / event capture in safety-related applications with the *MCU* in Sleep or Stop mode is prevented by the assumed requirement ASR5 (refer to Section 3.3.1 Safety requirement assumptions).

3.6.28 Tamper and backup registers (TAMP)

Table 102. TAMP_SM_0

SM CODE	TAMP_SM_0
Description	Information redundancy on tamper backup registers
Ownership	*End user*
Detailed implementation	Data stored in tamper backup registers must be protected by a checksum with encoding capability (for instance, *CRC*). Checksum must be checked by *Application software* before consuming stored data. This method guarantees data versus erases due to backup battery failures.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.29 Inter-integrated circuit (I2C)

Table 103. IIC_SM_0

SM CODE	IIC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to I2C configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 104. **IIC_SM_1**

SM CODE	IIC_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	I2C communication module embeds protocol error checks (like overrun, underrun, packet error etc.) conceived to detect network-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Adoption of SMBus option grants the activation of more efficient protocol-level hardware checks such as CRC-8 packet protection. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 105. **IIC_SM_2**

SM CODE	IIC_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by I2C a redundancy check (such as a *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	It is assumed that the remote I2C counterpart has an equivalent capability of performing the check described. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. *Important:* *This method must be considered as a subset of IIC_SM_4. Therefore, the implementation of IIC_SM_4 completely overlap this method. Refer to [4] for additional details.*

Table 106. **IIC_SM_3**

SM CODE	IIC_SM_3
Description	*CRC* packet-level
Ownership	ST
Detailed implementation	I2C communication module allows to activate for specific mode of operation (SMBus) the automatic insertion (and check) of *CRC* checksums to packet data.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*.
Multiple-fault protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	This method can be part of the implementation for IIC_SM_2 or IIC_SM_4. In that case, because of the warning issued in the *Test for the diagnostic* field, this mechanism can not be the only one to guarantee message integrity. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 107. **IIC_SM_4**

SM CODE	IIC_SM_4
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	*End user*
Detailed implementation	This method aims to protect the communication between a I2C peripheral and his external counterpart. Refer to UART_SM_3 description for detailed information.
Error reporting	Refer to UART_SM_3
Fault detection time	Refer to UART_SM_3
Addressed fault model	Refer to UART_SM_3
Dependency on *Device* configuration	Refer to UART_SM_3
Initialization	Refer to UART_SM_3
Periodicity	Refer to UART_SM_3
Test for the diagnostic	Refer to UART_SM_3
Multiple-fault protection	Refer to UART_SM_3
Recommendations and known limitations	It is assumed that the remote I2C counterpart has an equivalent capability of performing the checks described. Refer to UART_SM_3 for further notice.

3.6.30 Universal synchronous/asynchronous receiver transmitter, Low-power universal asynchronous receiver transmitter (USART/UART/LPUART)

Table 108. **UART_SM_0**

SM CODE	UART_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to USART/UART/LPUART configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 109. **UART_SM_1**

SM CODE	UART_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	USART/UART/LPUART communication module embeds protocol error checks (like additional parity bit check, overrun, frame error) conceived to detect network-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself. Error signals connected to these checkers are normally handled in a standard communication software, so the overhead is reduced.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	UART_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	USART/UART/LPUART communication module allows several different configurations. The actual composition of communication error checks depends on the selected configuration. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 110. **UART_SM_2**

SM CODE	UART_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented by adding to data packets transferred by this peripheral a redundancy check (such as a *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	It is assumed that the remote counterpart has an equivalent capability of performing the check described. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. *Important:* *This method must be considered as a subset of UART_SM_3. Therefore, the implementation of UART_SM_3 completely overlap this method. Refer to [4] for additional details.*

Table 111. **UART_SM_3**

SM CODE	UART_SM_3
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	*End user*
Detailed implementation	This method aims to protect the communication between a peripheral and his external counterpart establishing a kind of “protected” channel. The aim is to specifically address communication failure modes as reported in IEC 61508-2, 7.4.11.1. Implementation guidelines are as follows: • Data packet must be protected (encapsulated) by an information redundancy check, like for instance a CRC checksum computed over the packet and added to payload. Checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single-bit flip in the data packet. • Additional field added in payload reporting an unique identification of sender or receiver and an unique increasing sequence packet number. • Timing monitoring of the message exchange (for example check the message arrival within the expected time window), detecting therefore missed message arrival conditions. • *Application software* must verify before consuming data packet its consistency (CRC check), its legitimacy (sender or receiver) and the sequence correctness (sequence number check, no packets lost).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	A major overlap between the requirements of this method and the implementation of complex communication software protocols can exists. Due to large adoption of these protocols in industrial applications, optimizations can be possible. It is assumed that the remote counterpart has an equivalent capability of performing the checks described.

3.6.31 Serial peripheral interface (SPI)

Table 112. SPI_SM_0

SM CODE	SPI_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to SPI configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 113. SPI_SM_1

SM CODE	SPI_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	SPI communication module embeds protocol error checks (like overrun, underrun, timeout and so on) conceived to detect network-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

Table 114. **SPI_SM_2**

SM CODE	SPI_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by SPI a redundancy check (such as a *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software
Recommendations and known limitations	It is assumed that the remote counterpart has an equivalent capability of performing the check described. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. *Important:* *This method must be considered as a subset of SPI_SM_4. Therefore, the implementation of SPI_SM_4 completely overlap this method. Refer to [4] for additional details.*

Table 115. **SPI_SM_3**

SM CODE	SPI_SM_3
Description	*CRC* packet-level
Ownership	ST
Detailed implementation	SPI communication module allows to activate automatic insertion (and check) of CRC-8 or CRC-18 checksums to packet data.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*.
Multiple-fault protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	This method can be part of the implementation for SPI_SM_2 or SPI_SM_4. In that case, because of the warning issued in the *Test for the diagnostic* field, this mechanism can not be the only one to guarantee message integrity.

Table 116. **SPI_SM_4**

SM CODE	SPI_SM_4
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	*End user*
Detailed implementation	This method aims to protect the communication between SPI peripheral and his external counterpart. Refer to UART_SM_3 description for detailed information.
Error reporting	Refer to UART_SM_3
Fault detection time	Refer to UART_SM_3
Addressed fault model	Refer to UART_SM_3
Dependency on *Device* configuration	Refer to UART_SM_3
Initialization	Refer to UART_SM_3
Periodicity	Refer to UART_SM_3
Test for the diagnostic	Refer to UART_SM_3
Multiple-fault protection	Refer to UART_SM_3
Recommendations and known limitations	Refer to UART_SM_3 for further notice. It is assumed that the remote SPI counterpart has an equivalent capability of performing the checks described.

3.6.32 Disable and periodic cross-check of unintentional activation of unused peripherals

This section reports safety mechanisms that address peripherals not used by the safety application, or not used at all.

Table 117. **FFI_SM_0**

SM CODE	FFI_SM_0
Description	Disable of unused peripherals
Ownership	*End user*
Detailed implementation	This method contributes to the reduction of the probability of cross-interferences caused by peripherals not used by the software application, in case a hardware failure causes an unintentional activation. After the system boot, *Application software* must disable all unused peripherals with this procedure: • Enable reset flag on AHB and APB peripheral reset register. • Disable clock distribution on AHB and APB peripheral clock enable register.
Error reporting	Not applicable
Fault detection time	Not applicable
Addressed fault model	Not applicable
Dependency on *Device* configuration	None
Initialization	Not applicable
Periodicity	Startup
Test for the diagnostic	Not applicable
Multiple-fault protection	FFI_SM_1: Periodic read-back of interference avoidance registers
Recommendations and known limitations	None

Table 118. **FFI_SM_1**

SM CODE	FFI_SM_1
Description	Periodic read-back of interference avoidance registers
Ownership	*End user*
Detailed implementation	This method contributes to the reduction of the probability of cross-interferences between peripherals that can potentially conflict on the same input/output pins, including for instance unused peripherals. This diagnostic measure must be applied to following registers: • clock enable and disable registers • alternate function programming registers Detailed information on the implementation of this method can be found in Section 3.6.16 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

3.6.33 System

Table 119. DUAL_SM_0

SM CODE	DUAL_SM_0
Description	Cross-check between two STM32 devices
Ownership	*End user*
Detailed implementation	This method is implemented in the spirit of technique described in IEC 61508-7, A.3.5 “Reciprocal comparison by software”, which is rated in IEC 61508-2 Table A.4 as capable to achieve high level of diagnostic coverage. The two processing units exchange data reciprocally, and a fail in the comparison is considered as a detection of a failure in one of the two unit. The guidelines for the implementation are the following: • Data exchanged include output results, intermediate results(1) and the results (pass/fail) of each software-implemented safety mechanisms executed on periodical basis on both MCUs (for example M0+) • Software routines devoted to data exchange/comparison must be logically separated from the software implementing the safety function(s). • Systematic capability of software implementing this method must be equal or above the one of the software implementing the safety function(s). • Independence and lack of interference between the software implementing the data exchange/ comparison and the one implementing the safety function(s) must be proven. • Frequency of data exchange/comparison is imposed by the system PST (refer to related timing constraints for periodic safety mechanisms), except for output results which needs to be exchanged/ compared at the same rate they are potentially updated.
Error reporting	-
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPUM0+_SM_0, CPUM4_SM_0: Periodic core self-test software (individually executed on both processing units)
Recommendations and known limitations	This method is usually rated as optional because it is not strictly needed in the framework of 1oo2 architecture described in Section 3.2.4 Reference safety architectures - 1oo2. Anyway, it is included here only for its use in such an architecture. This method can provide additional safety margin for systems that need further protection against fault accumulation. Because this method could be a potential source of common cause failure between the two 1oo2 channels (in case of incorrect implementation), *End user* is recommended to closely follow the Detailed implementation guidelines in this table.

1. *the value of each variable able to directly influence the final individual channel output, such as:*
   – *variables included in computation of the final result; for example, of a PWM rate*
   – *variables involved in a decision determining the final result; for example, two variables used in a comparison which determines if a GPIO output is set high or low.*

3.7 Conditions of use

The table below provides a summary of the safety concept recommendations reported in Section 3.6 Hardware and software diagnostics. The conditions of use to be applied to STM32WL5x dual-core devices are reported in form of safety mechanism requirements. Exception is represented by some conditions of use introduced by FMEA analysis in order to correctly address specific failure modes. These conditions of use are reported at the end of the table presented in this section.

Rank column reports how related safety mechanism has been considered during the analysis, with following meaning:

++	The safety mechanism is highly recommended as common practice. It is considered in this document for the computation of safety metrics to allow the use of *Device* in systems implementing safety functions up to *SIL*2 with a single *MCU* or up to *SIL*3 with two *MCU*s in 1oo2 scheme. Missing implementation may lead to invalidate any safety feature claimed in this manual and must be supported by adequate arguments under end user responsibility (refer to Section 4.1.1 for guidance).
+	The safety mechanism is recommended as additional safety measure, but not considered in this document for the computation of safety metrics. The *End user* can skip the implementation in case it is in contradiction with functional requirements or overlapped by another mechanism ranked ++.
o	The safety mechanism is optional. It is not strictly required for the implementation of safety functions up to *SIL*2, or it is related to a specific *MCU* configuration.

The *X* marker in the *Perm* and *Trans* table columns indicates that the related safety mechanism is effective for such fault model. The *X* marker in *Separation* column indicates that related recommendation is effective for the correct implementation of the separation concept described in Section 3.2.5 The separation concept.

Table 120. **List of safety recommendations**

Diagnostic	Description	Rank	Perm	Trans	Separation
Arm® Cortex®-M0+					
CPUM0+_SM_0	Periodic core self-test software for Arm® Cortex®-M0+ *CPU*	++	X	-	X
CPUM0+_SM_1	Control flow monitoring in *Application software*	++	X	X	-
CPUM0+_SM_2	Double computation in *Application software*	++	-	X	-
CPUM0+_SM_3	Arm® Cortex®-M0+ HardFault exceptions	++	X	X	-
CPUM0+_SM_4	Stack hardening for *Application software*	++	X	X	X
CPUM0+_SM_7	Memory protection unit (*MPU*)	++	X	X	X
MPUM0+_SM_0	Periodic read-back of Arm® Cortex®-M0+ configuration registers	++	X	X	X
MPUM0+_SM_1	Arm® Cortex®-M0+ MPU software test.	++	X	-	X
Arm® Cortex®-M4					
CPUM4_SM_0	Periodic core self-test software for Arm® Cortex®-M4 *CPU*	++	X	-	X
CPUM4_SM_1	Control flow monitoring in *Application software*	++	X	X	-
CPUM4_SM_2	Double computation in *Application software*	++	-	X	-
CPUM4_SM_3	Arm® Cortex®-M4 HardFault exceptions	++	X	X	-
CPUM4_SM_4	Stack hardening for *Application software*	++	X	X	X
CPUM4_SM_7	Memory protection unit (*MPU*)	++	X	X	X
MPUM4_SM_0	Periodic read-back of Arm® Cortex®-M4 *MPU* configuration registers	++	X	X	X
MPUM4_SM_1	Arm® Cortex®-M4 MPU software test.	++	X	-	X
CPUs-shared safety mechanisms					
CPU_SM_5	External watchdog	++	X	X	X
CPU_SM_6	Internal watchdogs IWDG/WWDG	++	X	X	X
CPU_SM_11	Cross-CPU safety information exchange	++	-	-	X
System bus architecture/BusMatrix/Peripherals interconnect matrix					
BUS_SM_0	Periodic software test for interconnections	++	X	-	-
BUS_SM_1	Information redundancy in intra-chip data exchanges	++	X	X	-
LOCK_SM_0	Lock mechanism for configuration options	+	-	-	-
Global TrustZone® controller (GTZC)					
GTZC_SM_0	Periodical read-back of GTZC configuration registers	++	X	X	-

Diagnostic	Description	Rank	Perm	Trans	Separation
GTZC_SM_1	GTZC illegal access detection	++	X	X	-
Embedded SRAM					
RAM_SM_0	Periodic software test for static random access memory (SRAM)	++	X	-	-
RAM_SM_2	Stack hardening for *Application software*	++	X	X	-
RAM_SM_3	Information redundancy for safety-related variables in the *Application software*	++	X	X	-
RAM_SM_4	Control flow monitoring in *Application software*	o(1)	X	X	-
RAM_SM_5	Periodic integrity test for *Application software* in RAM	o(1)	X	X	-
RAM_SM_8	Periodic test by software for SRAM address decoder	++	X	-	-
Embedded flash memory					
FLASH_SM_0	Periodic software test for flash memory	++	X	-	-
FLASH_SM_1	Control flow monitoring in *Application software*	++	X	X	-
FLASH_SM_2	Arm®Cortex®-M0+ and M4 HardFault exceptions	++	X	X	-
FLASH_SM_3	Option byte write protection	++	-	-	-
FLASH_SM_4	Static data encapsulation	+	X	X	-
FLASH_SM_6	Flash memory unused area filling code	+	-	-	-
FLASH_SM_7	ECC on flash memory	++	X	X	-
FLASH_SM_8	Read protection (RDP), write protection (WRP)	+	-	-	-
FLASH_SM_9	Periodic test by software for flash memory address decoder	++	X	-	-
Power controller (PWR)					
VSUP_SM_0	Periodic read-back of configuration registers	++	X	X	-
VSUP_SM_1	Supply voltage internal monitoring (PVD)	++	X	-	-
VSUP_SM_2	Independent watchdog	++	X	-	-
VSUP_SM_3	Internal temperature sensor check	++	-	-	X
VSUP_SM_5	System-level power supply management	++	-	-	-
Reset and clock controller (RCC)					
CLK_SM_0	Periodic read-back of configuration registers	++	X	X	-
CLK_SM_1	Clock security system (CSS)	+	X	-	-
CLK_SM_2	Independent watchdog	++	X	-	-
CLK_SM_3	Internal clock cross-measurement	+	X	-	-
Hardware semaphore (HSEM)					
HSEM_SM_0	Periodic read-back of configuration registers	++	X	X	-
HSEM_SM_1	Periodic read-back of configuration registers	++	X	X	-
Inter-processor communication controller (IPCC)					
IPCC_SM_0	Periodic read-back of configuration registers.	++	X	X	X
IPCC_SM_1	End-to-end protection for inter-CPU processor communications	++	X	X	X
General-purpose input/output (GPIO)					
GPIO_SM_0	Periodic read-back of configuration registers	++	X	X	-
GPIO_SM_1	1oo2 for input GPIO lines	++	X	X	-
GPIO_SM_2	Loopback scheme for output GPIO lines	++	X	X	-
GPIO_SM_3	GPIO port configuration lock register	+	-	-	-

Diagnostic	Description	Rank	Perm	Trans	Separation
Debug system or peripheral control					
DBG_SM_0	Watchdog protection	++	X	X	-
System configuration controller (SYSCFG)					
SYSCFG_SM_0	Periodic read-back of configuration registers	++	X	X	-
DIAG_SM_0	Periodic read-back of hardware diagnostics configuration registers	++	X	X	-
Direct memory access controller (DMA), DMA request multiplexer (DMAMUX)					
DMA_SM_0	Periodic read-back of configuration registers	++	X	X	-
DMA_SM_1	Information redundancy on data packet transferred via *DMA*	++	X	X	-
DMA_SM_2	Information redundancy by including sender or receiver identifier on data packet transferred via *DMA*	++	X	X	-
DMA_SM_3	Periodic software test for *DMA*	++	X	-	-
DMA_SM_4	*DMA* transaction awareness	++	X	X	-
Extended interrupt and events controller (EXTI)					
NVIC_SM_0	Periodic read-back of configuration registers	++	X	X	-
NVIC_SM_1	Expected and unexpected interrupt check	++	X	X	-
Cyclic redundancy-check calculation unit (CRC)					
CRC_SM_0	CRC self-coverage	++	X	X	-
Analog-to-digital converter (ADC)					
ADC_SM_0	Periodic read-back of configuration registers	++	X	X	-
ADC_SM_1	Multiple acquisition by *Application software*	++	-	X	-
ADC_SM_2	Range check by *Application software*	++	X	X	-
ADC_SM_3	Periodic software test for ADC	++	X	-	-
ADC_SM_4	1oo2 scheme for ADC inputs	+	X	X	-
Digital-to-analog converter (DAC)					
DAC_SM_0	Periodic read-back of configuration registers	++	X	X	-
DAC_SM_1	DAC output loopback on ADC channel	++	X	X	-
Voltage reference buffer (VREFBUF)					
VREF_SM_0	Periodic read-back of VREFBUF system configuration registers	++	X	X	-
VREF_SM_1	VREF cross-check by ADC reading	+	X	-	-
Comparator (COMP)					
COMP_SM_0	Periodic read-back of configuration registers	++	X	X	-
COMP_SM_1	1oo2 scheme for comparator	++	X	X	-
COMP_SM_2	Plausibility check on inputs	+	X	-	-
COMP_SM_3	Multiple acquisition by *Application software*	+	-	X	-
COMP_SM_4	Comparator lock mechanism	+	-	-	-
True random number generator (RNG)					
RNG_SM_0	Periodic read-back of RNG configuration register	++	X	X	-
RNG_SM_1	RNG module entropy on-line tests	++	X	-	-
Advanced encryption standard hardware accelerator (AES)					
AES_SM_0	Periodic read-back of AES configuration registers	++	X	X	-
AES_SM_1	Encryption/decryption collateral detection	++	X	X	-

Diagnostic	Description	Rank	Perm	Trans	Separation
AES_SM_2	Information redundancy techniques on messages, including end-to-end protection	++	X	X	-
Public key accelerator (PKA)					
PKA_SM_0	Periodical read-back of PKA configuration registers	++	X	X	-
PKA_SM_1	PKA key computation collateral detection	++	X	X	-
Advanced-control/General-purpose and Low-power timers					
ATIM_SM_0	Periodic read-back of configuration registers	++	X	X	-
ATIM_SM_1	1oo2 for counting timers	++	X	X	-
ATIM_SM_2	1oo2 for input capture timers	++	X	X	-
ATIM_SM_3	Loopback scheme for pulse width modulation (PWM) outputs	++	X	X	-
ATIM_SM_4	Lock bit protection for timers	+	-	-	-
Independent and system window watchdogs (IWDG and WWDG)					
WDG_SM_0	Periodic read-back of configuration registers	++	X	X	-
WDG_SM_1	Software test for watchdog at startup	o	X	-	-
Real-time clock module (RTC)					
RTC_SM_0	Periodic read-back of configuration registers	++	X	X	-
RTC_SM_1	Application check of running RTC	++	X	X	-
RTC_SM_2	Information redundancy on backup registers	+	X	X	-
RTC_SM_3	Application-level measures to detect failures in timestamps/event capture	o	X	X	-
Tamper and backup registers (TAMP)					
TAMP_SM_0	Information redundancy on tamper backup registers	+	X	X	-
Inter-integrated circuit (I2C)					
IIC_SM_0	Periodic read-back of configuration registers	++	X	X	-
IIC_SM_1	Protocol error signals	++	X	X	-
IIC_SM_2	Information redundancy techniques on messages	++	X	X	-
IIC_SM_3	*CRC* packet-level	+	X	X	-
IIC_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X	-
Universal synchronous/asynchronous receiver transmitter, Low-power universal asynchronous receiver transmitter (USART/UART/LPUART)					
UART_SM_0	Periodic read-back of configuration registers	++	X	X	-
UART_SM_1	Protocol error signals	++	X	X	-
UART_SM_2	Information redundancy techniques on messages	++	X	X	-
UART_SM_3	Information redundancy techniques on messages, including end-to-end protection	++	X	X	-
Serial peripheral interface (SPI)					
SPI_SM_0	Periodic read-back of configuration registers	++	X	X	-
SPI_SM_1	Protocol error signals	++	X	X	-
SPI_SM_2	Information redundancy techniques on messages	++	X	X	-
SPI_SM_3	*CRC* packet-level	+	X	X	-
SPI_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X	-

Diagnostic	Description	Rank	Perm	Trans	Separation
Disable and periodic cross-check of unintentional activation of unused peripherals					
FFI_SM_0	Disable of unused peripherals	++	-	-	-
FFI_SM_1	Periodic read-back of interference avoidance registers	++	-	-	X
Arm®Cortex®-M0+ and M4 *CPU*					
CoU_1	The reset condition of Arm® Cortex®- M0+ and Arm® Cortex®- M4 CPUs must be compatible as valid safe state at system level	++	-	-	-
Debug					
CoU_2	*Device* debug features must not be used in safety function(s) implementation.	++	-	-	-
Arm®Cortex®-M0+ and M4 / Supply system					
CoU_3	Low-power mode state must not be used in safety function(s) implementation.	++	-	-	-
***Device* peripherals**					
CoU_4	*End user* must implement the required combination of safety mechanism/CoUs for each STM32 peripheral used in implementation of safety function(s).	++	X	X	-
Flash memory subsystem					
CoU_5	During flash memory bank mass erase and reprogramming there must not be safety functions(s) executed by *Device*.	++	-	-	-
***CPU* subsystem**					
CoU_7	In case of multiple safety functions implementations, methods to guarantee their mutual independence must include *MPU* use.	++	-	-	-
Error management					
CoU_9.1	Any security violation event must be considered equivalent to the detection of a non-controllable hardware random failure; accordingly, it must lead to the transition to a safe state (de-energize) at system level.	++	X	X	-
CoU_9.2	For each implemented safety mechanisms, related error/fault detection signal(s)/message(s) must be processed by the application software for the correct management of SS1 safe state. Related software routines must be considered as safety related in the framework of the overall policy for software systematic capability in the final system.(2)	++	X	X	X
Embedded SRAM					
CoU_10	RAM space used by each Device CPU for safety function(3) implementation (including stacks space) must be kept separated, except the area specified by CoU_11. Allocation must be static.	++	-	-	X
CoU_11	Data exchanges between the two CPUs of the Device must use a dedicated shared memory space defined in SRAM.	++	-	-	X
CoU_12	Memory Protection Units (MPU) of each Device CPU must be used to enforce the implementation of CoU_10, CoU_11.	++	-	-	X
Watchdog					
CoU_13	The external watchdog must supervise the correct behavior of both Device CPUs. Communication schemes are ruled by CoU_14.	++	X	X	X
CoU_14	CPUs communications with the external watchdog must be managed either: 1. Both CPUs can communicate with the external watchdog. In this case the watchdog must be able to identify which CPU is sending the message, or	++	-	-	X

Diagnostic	Description	Rank	Perm	Trans	Separation
	2. Only one CPU drives the external watchdog and acts as supervisor for the other CPU. In this case the first CPU must be able to request a reset in case the second CPU is unresponsive or reporting a failure detection.				
CoU_15	In the Device, each of the two internal watchdogs IWDG and WWDG must be statically allocated to only one CPU	++	-	-	X
CoU_16	External and internal watchdog management software routines (implementing requirements of CoU_13, CoU_14, CoU_15) must be developed with software systematic capability (SC) not lower than the highest available on the system, and in any case at least equal to SC2	++	-	-	X
Part separation (no interference)					
CoU_17	During system boot, a combination of the external watchdog and system-level measures must guarantee the overall safe state in case of missed correct application software start on each CPU. Exit from this power-up safe state must be linked also to the successful execution (before the application software start) of following safety measures: • CPUM0+_SM_0 • MPUM0+_SM_0 • MPUM0+_SM_1 • CPUM4_SM_0 • MPUM4_SM_0 • MPUM4_SM_1 • FLASH_SM_0	++	-	-	X
CoU_18	In 1oo2 architecture, the PEv must keep the final system in safe state after the power-up until both devices have performed their exit from the power-up safe state ruled by CoU_17	++	-	-	X
System					
DUAL_SM_0	Cross-check between two STM32 devices	o	X	X	-

1. *Must be considered ranked as "++" if Application software is executed on RAM.*
2. *To provide an example, safety recommendation "CPU_SM_3 Arm® Cortex®- M0+ HardFault exceptions" cannot be considered correctly satisfied if related exception handler is not implemented in software.*
3. *A reminder that any software-based safety mechanisms like CPUM0+_SM_0, CPUM4_SM_0 must be considered "safety functions" and so they must comply to this CoU.*
4. *Therefore, trigger action on each internal watchdog must be allowed to only one CPU, statically selected at software design time.*

The above-described safety mechanism or conditions of use are conceived with different levels of abstraction depending on their nature: the more a safety mechanism is implemented as application-independent, the wider is its possible use on a large range of *End user* applications.

The safety analysis highlights two major partitions inside the *MCU*:

- System-critical *MCU* modules
  Every *End user* application is affected, from safety point of view, by a failure on these modules. Because they are used by every *End user* application, related methods or safety mechanism are mainly conceived to be application-independent. The system-critical modules on *Device* are: CPU, RCC, PWR, bus matrix and interconnect, and flash memory and RAM (including their interfaces).
- Peripheral modules
  Such modules could be not used by the end-user application, or they could be used for non-safety related tasks. Related safety methods are therefore implemented mainly at application level, as *Application software* solutions or architectural solutions.

4 Safety results

This section reports the results of the safety analysis of the STM32WL5x dual-core devices, according to IEC 61508 and to ST methodology flow, related to the hardware random and dependent failures.

4.1 Random hardware failure safety results

The analysis for random hardware failures of STM32WL5x dual-core devices reported in this safety manual is executed according to STMicroelectronics methodology flow for safety analysis of semiconductor devices in compliance with IEC 61508 (refer to [4] for more details). The accuracy of results obtained are guaranteed by three factors:

- STMicroelectronics methodology flow strict adherence to IEC 61508 requirements and prescriptions
- the use, during the analysis, of detailed and reliable information on microcontroller design
- the use, for specific diagnostic coverage evaluation, of state-of-the-art fault injection methods and tools for safety metrics verification

The *Device* safety analysis explored the overall and exhaustive list of *Device* failure modes, to individuate for each of them an adequate mitigation measure (safety mechanism). The overall list of *Device* failure modes is maintained in the related *FMEA* document [1], provided on demand by local STMicroelectronics sales office.

In summary, with the adoption of the safety mechanisms and conditions of use reported in Section 3.7 Conditions of use, it is possible to achieve the integrity levels summarized in the following table.

Table 121. **Overall achievable safety integrity levels**

Note that the achievable integrity levels are the same regardless of the individual or collaborative scheme applied.

Number of *Devices* used	Safety architecture	Target	Safety analysis result
1	1oo1	*SIL2 LD*	Achievable
		SIL2 HD/CM	Achievable with potential performance impact (1)
2	1oo2	*SIL3 LD*	Achievable
		SIL3 HD/CM	Achievable with potential performance impact

1. *Note that the potential performance impact related to some above-reported target achievements is mainly related to the need of execution of periodical software-based diagnostics (refer to safety mechanism description for details). The impact is therefore strictly related to how much "aggressive" the system level PST is (see Section 3.3.1 Safety requirement assumptions).*

The resulting relative safety metrics (DC and safe failure fraction (SFF)) and absolute safety metrics (probability of failure per hour (PFH), probability of dangerous failure on demand (PFD)) are not reported in this section but in the failure mode effect diagnostic analysis (FMEDA) snapshot [2], due to:

- a large number of different STM32WL5x dual-core parts,
- a possibility to declare non-safety-relevant unused peripherals, and
- a possibility to enable or not the different available safety mechanisms.

The *FMEDA* snapshot [2] is a static document reporting the safety metrics computed at different detail levels (at microcontroller level and for microcontroller basic functions) for a given combination of safety mechanisms and for a given part number. If *FMEDA* document is needed, contact the local STMicroelectronics sales representative as early as possible, in order to receive information on expected delivery dates for specific *Device* target part numbers.

Note: *Safety metrics computations are restricted to STM32WL5x dual-core boundary, hence they do not include the WDTe, PEv, and VMONe processes described in Section 3.3.1 Safety requirement assumptions).*

4.1.1 Safety analysis result customization

The safety analysis executed for STM32WL5x dual-core devices documented in this safety manual considers all microcontroller modules to be safety-related, thus able to interfere with the safety function, with no exclusion. This is in line with the conservative approach to be followed during the analysis of a general-purpose microcontroller, in order to be agnostic versus the final application. This means that no microcontroller module has been declared *safe* as per IEC 61508-4, 3.6.8. Therefore, all microcontroller modules are included in *SFF* computations.

In actual *End user* applications, not all the STM32WL5x dual-core parts or modules implement a safety function. That happens if:

- The part is not used at all (disabled), or
- The part implements functions that are not safety-related (for example, a GPIO line driving a power-on signaling light on an electronic board).

Note: Implementation of non-safety-related functions is in principle forbidden by the assumed safety requirement ASR4.4 (see Section 3.3.1 Safety requirement assumptions), hence under End user's entire responsibility. As any other derogation from safety requirements included in this manual, it is End user's responsibility to provide consistent rationales and evidences that the function does not bring additional risks, by following the procedure described in this section. Therefore, it is strongly recommended to reserve such derogation to very simple functions (as the one provided in the example).

Implementing safety mechanisms on such parts would be a useless effort for *End user*. The safety analysis results can therefore be customized.

End user can define a STM32WL5x dual-core part as *non-safety-related* based on:

- Collecting rationales and evidences that the part does not contribute to safety function.
- Collecting rationales and evidences that the part does not interfere with the safety function during normal operation, due to final system design decisions. Mitigation of unused modules is exhaustively addressed in Section 4.1.2 General requirements for freedom from interferences (FFI).
- Fulfilling the general condition for the mitigation of intra-*MCU* interferences (see Section 4.1.2 General requirements for freedom from interferences (FFI)).

For a *non-safety-related* part, *End user* is allowed to:

- Exclude the part from computing metrics to report in *FMEDA*, and
- Not implement safety mechanisms as listed in Table 120. List of safety recommendations.

With regard to *SFF* computation, this section complies with the *no part / no effect* definition as per IEC 61508-4, 3.6.13 / 3.6.14.

4.1.2 General requirements for freedom from interferences (FFI)

A dedicated analysis has highlighted a list of general requirements to be followed in order to mitigate potential interferences between *Device* internal modules in case of internal failures (freedom from interferences, FFI). These precautions are integral part of the *Device* safety concept and they can play a relevant role when multiple microcontroller modules are declared as *non-safety-related* by *End user* as per Section 4.1.1 Safety analysis result customization.

End user must implement the safety mechanisms listed in Table 122 (implementation details in Section 3.6 Hardware and software diagnostics) regardless any evaluation of their contribution to safety metrics.

Table 122. List of general requirements for FFI

Diagnostic	Description
BUS_SM_0	Periodic software test for interconnections
GPIO_SM_0	Periodic read-back of configuration registers
DMA_SM_0	Periodic read-back of configuration registers
DMA_SM_2	Information redundancy by including sender or receiver identifier on data packet transferred via *DMA*(1)
DMA_SM_4	*DMA* transaction awareness(1)
NVIC_SM_0	Periodic read-back of configuration registers
NVIC_SM_1	Expected and unexpected interrupt check
FFI_SM_0	Disable of unused peripherals
FFI_SM_1	Periodic read-back of interference avoidance registers

1. *To be implemented only if DMA is actually used.*

4.1.3 Notes on multiple-fault scenario

According to the requirements of IEC 61508, the safety analysis for STM32WL5x dual-core devices considered multiple-fault scenarios. Furthermore, following the spirit of ISO26262 (the reference and state-of-the-art standard norm for integrated circuit safety analysis), the analysis investigated possible causes preventing the implemented safety mechanisms from being effective, in order to determine appropriate counter-measures. In the *Multiple-fault protection* field, the tables in Section 3.6 Hardware and software diagnostics report the safety mechanisms required to properly manage a multiple-fault scenario, including mitigation measures against failures making safety mechanisms ineffective. It is strongly recommended that the safety concept includes such mitigation measures, and in particular for systems operating during long periods, as they tend to accumulate errors. Indeed, fault accumulation issue has been taken into account during STM32WL5x dual-core devices safety analysis.

Another potential source of multiple error condition is the accumulation of permanent failures during power-off periods. Indeed, if the end system is not powered, no safety mechanism are active and so able to early detect the insurgence of such failures. To mitigate this potential issue, it is strongly recommended to execute all periodic safety mechanism at each system power-up; this measure guarantees a fresh system start with a fault-free hardware. This recommendation is given for periodic safety mechanisms rated as "++" (highly recommended) in the Device safety concept, and mainly for the most relevant ones in term of failure distribution: CPUM0+_SM_0, CPUM4_SM_0, FLASH_SM_0, RAM_SM_0. This startup execution is strongly recommended regardless the safety functions mode of operations and/or the value of PST.

4.2 Analysis of dependent failures

The analysis of dependent failures is important for microcontroller and microprocessor devices. The main subclasses of dependent failures are *CCFs*. Their analysis is ruled by IEC 61508-2 annex E, which lists the design requirements to be verified to allow the use of on-chip redundancy for integrated circuits with one common semiconductor substrate.

As there is no on-chip redundancy on STM32WL5x dual-core devices, the *CCF* quantification through the βIC computation method - as described in Annex E.1, item i - is not required. Note that, in the case of 1oo2 safety architecture implementation, *End user* is required to evaluate the β and βD parameters (used in *PFH* computation) that reflect the common cause factors between the two channels.

The *Device* architecture and structures can be potential sources of dependent failures. These are analyzed in the following sections. The safety mechanisms referred to are described in Section 3.6 Hardware and software diagnostics.

4.2.1 Power supply

Power supply is a potential source of dependent failures, because any alteration can simultaneously affect many modules, leading to not-independent failures. The following safety mechanisms address and mitigate those dependent failures:

- VSUP_SM_1: detection of abnormal value of supply voltage;
- VSUP_SM_2: the independent watchdog is different from the digital core of the *MCU*, and this diversity helps to mitigate dependent failures related to the main supply alterations. As reported in VSUP_SM_2 description, separate power supply for IWDG or/and the adoption of an external watchdog (CPU_SM_5) increase such diversity.
- VSUP_SM_5: power supply stability (guaranteed by system level measures) is an important mitigation factor

The adoption of such safety mechanisms is therefore highly recommended despite their minor contribution to the safety metrics to reach the required safety integrity level. Refer to Section 3.6.8 Power controller (PWR) for the detailed safety mechanism descriptions.

4.2.2 Clock

System clocks are a potential source of dependent failures, because alterations in the clock characteristics (frequency, jitter) can affect many parts, leading to not-independent failures. The following safety mechanisms address and mitigate such dependent failures:

- CLK_SM_1: the clock security system is able to detect hard alterations (stop) of system clock and activate the adequate recovery actions.
- CLK_SM_2: the independent watchdog has a dedicated clock source. The frequency alteration of the system clock leads to the watchdog window violations by the triggering routine on *Application software*, leading to the *MCU* reset by watchdog. The adoption of external watchdog (CPU_SM_5) provides additional diversity and so further mitigation of clock-related common cause failures.

The adoption of such safety mechanism is therefore highly recommended despite their minor contribution to the safety metrics to reach the required safety integrity level. Refer to Section 3.6.9 Reset and clock controller (RCC) for detailed safety mechanisms description.

4.2.3 DMA

The *DMA* function can be involved in data transfers operated by most of the peripherals. Failures of *DMA* can interfere with the behavior of the system peripherals or *Application software*, leading to dependent failures. The adoption of the following safety mechanisms is therefore highly recommended (refer to Section 3.6.15 Direct memory access controller (DMA), DMA request multiplexer (DMAMUX) for description):

- DMA_SM_0
- DMA_SM_1
- DMA_SM_2

Note: Only DMA_SM_0 must be implemented if DMA is not used for data transfer.

4.2.4 Internal temperature

The abnormal increase of the internal temperature is a potential source of dependent failures, as it can affect many *MCU* parts. The following safety mechanism mitigates this potential effect (refer to Section 3.6.8 Power controller (PWR) for description):

VSUP_SM_3: the internal temperature read and check allows the user to quickly detect potential risky conditions before they lead to a series of internal failures.

5 List of evidences

A *safety case database* stores all the information related to the safety analysis performed to derive the results and conclusions reported in this safety manual.

The safety case database is composed of the following:

- safety case with the full list of all safety-analysis-related documents
- STMicroelectronics' internal *FMEDA* tool database for the computation of safety metrics, including estimated and measured values
- safety report, a document that describes in detail the safety analysis executed on STM32WL5x dual-core devices and the compliance to IEC 61508 applicable clauses
- STMicroelectronics' internal fault injection campaign database including tool configuration and settings, fault injection logs and results, related to the *MCU* modules for which fault injection is adopted as verification method.

As these resources contain STMicroelectronics confidential information, they are only available for the purpose of audit and inspection by authorized bodies, without being published, which conforms to Note 2 of IEC 61508-2, 7.4.9.7.

Important: The combination of this document (safety manual), the [1] and [2] documents, the [4] provides per se an exhaustive view of the rationales for the compliance to IEC 61508 requirements of the whole STM32 safety concept. All these documents are available under NDA and they can be shared with certification entities (refer to applicable NDA for details).

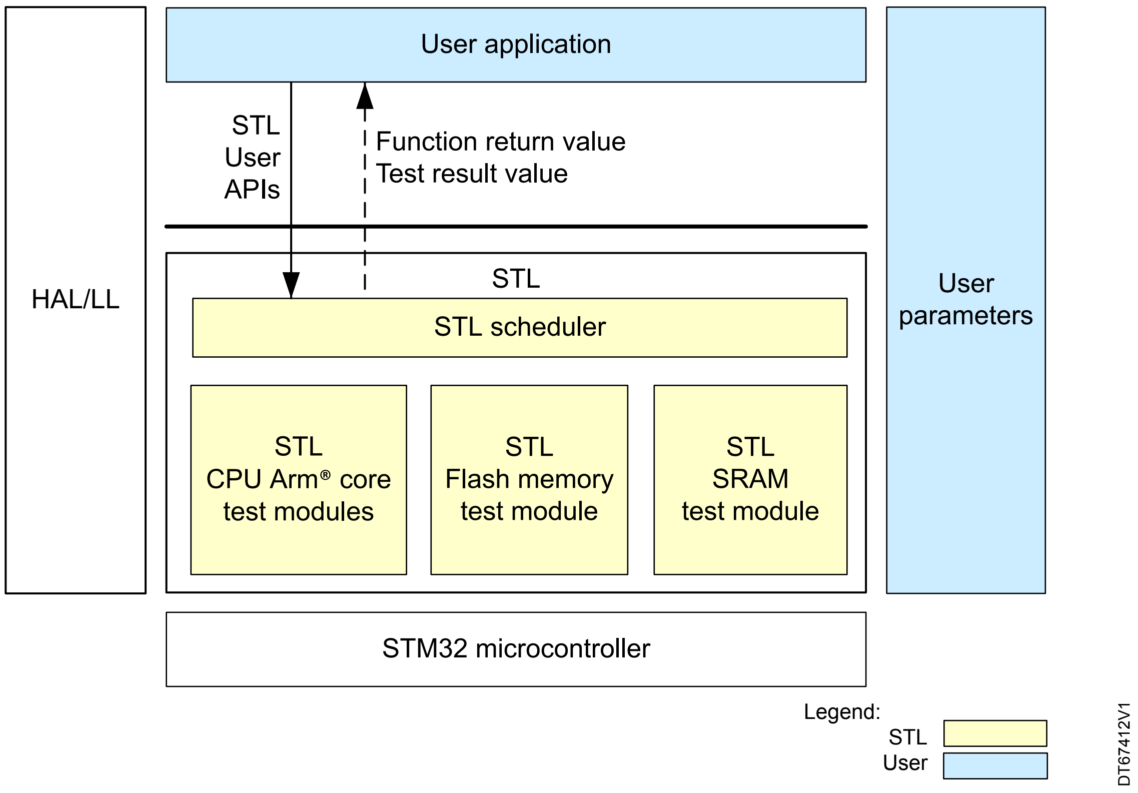
Appendix A X-CUBE-STL self-test software library

The X-CUBE-STL (also referred as "STL" in this document) is a Software-based diagnostic library designed to detect random hardware failures in STM32 safety-critical core components (*CPU* + SRAM + flash memory). It is provided by STMicroelectronics to simplify the implementation of STM32 *MCU* safety concept, by offering a pre-certified brick addressing the most challenging *MCU* functions.

X-CUBE-STL implements a set of of key safety mechanisms described in this Safety Manual:

- Periodic core self-test software for *CPU*.
- FLASH_SM_0 Periodic software test for flash memory
- RAM_SM_0 Periodic software test for static random access memory (SRAM)

Figure 8. **STL architecture**



X-CUBE-STL characteristics:

- Partitioned into Test Modules to ease its coexistence with end user application software
- Provided with a Scheduler function to simplify the periodic execution of the tests
- Flash and SRAM test area can be partitioned in programmable sections to reduce the time for the execution of atomic test sections
- Application independent: can be used in potentially any end-user application.
- It can be interrupted at practically any time by the end user application; the few critical sections are automatically protected by an interrupt disable function.
- Compiler independent: delivered as object code.
- Independence: designed as HAL-, BSP- and CMSIS-agnostic (there are no dependencies from these software packages).
- Compatible with most popular safe RTOS (white papers/application notes on integration with safe RTOS are available)
- Portability: the X-CUBE-STL shares the same APIs set across all the STM32 MCU Series, so projects portability across STM32 portfolio is guaranteed
- Provided with exhaustive end user documentation: safety manual and user guide

- Diagnostic coverage verified by state-of-the-art ST proprietary fault injection methodology
- Development flow compliant to SC3 systematic capability requirements from IEC 61508
- Certified by TÜV Rheinland (certification covers claims related to achieved DC and SC3 development flow)

X-CUBE-STL is available on demand under *NDA* agreement (contact your local ST representative).

Revision history

Table 123. Document revision history

Date	Revision	Changes
23-May-2022	1	Initial release.
10-Nov-2022	2	Added: • Table 32. RAM_SM_1 Updated: • Changed 'periodic" term to "Periodic" in the desription of the tables of required sections • Section Introduction • Section 3.3.1 Safety requirement assumptions • Table 9. MPUM0_SM_0 • Table 114. COMP_SM_1 • Table 115. COMP_SM_2 • Table 116. COMP_SM_3 • Table 121. List of safety recommendations and corrected a typographic mistake Removed: • RAM_SM_6 table
21-Nov-2023	3	Updated: Safety mechanism acronyms updated to CPUM0+_SM_x from CPUM0_SM_x. • Section 3.2.2 Safety functions performed by Compliant item • Section 3.2.4 Reference safety architectures - 1oo2 • Safety requirement assumptions • Section 3.5 Systematic safety integrity • Minor updates of tables and table titles in Section 3.6 Hardware and software diagnostics • Section 4.1 Random hardware failure safety results • Table 121. Overall achievable safety integrity levels • Section 4.1.1 Safety analysis result customization • Section 4.2.1 Power supply • Section 5 List of evidences Added: • Appendix A X-CUBE-STL self-test software library Replacement of Flash by flash in the document.
28-Nov-2023	4	Updated: • Section 3.2.4 Reference safety architectures - 1oo2 • Safety requirement assumptions for incorrect product reference: STM32H7 changed to STM32WL5x dual-core

Glossary

Application software within the software executed by *Device*, the part that ensures functionality of *End user*'s application and integrates safety functions

ASR assumed safety requirement

CCF common cause failure

CM continuous mode

Compliant item any item subject to claim with respect to the clauses of IEC 61508 series of standards

COTS commercial off-the-shelf

CoU conditions of use

CPU central processing unit

CRC cyclic redundancy check

DC diagnostic coverage

Device depending on context, any single or all of the silicon products

DMA direct memory access

DTI diagnostic test interval

End user individual person or company who integrates *Device* in their application, such as an electronic control board

EUC equipment under control

FIT failure in time

FMEA failure mode effect analysis

FMEDA failure mode effect diagnostic analysis

HD high-demand

HFT hardware fault tolerance

HW hardware

ITRS international technology roadmap for semiconductors

LD low-demand

MCU microcontroller unit

MPU memory protection unit

MTBF mean time between failures

MTTFd mean time to dangerous failure

NDA non disclosure agreement

PEc computation processing elements

PEi input processing elements

PEo output processing elements

PEv voting processing element

PFD probability of dangerous failure on demand

PFH probability of failure per hour

PL performance level

PST process safety time

SFF safe failure fraction

SIL safety integrity level

SoC system on chip

VMONe voltage monitors

WDTe watchdog

Contents

List of tables

List of figures