STM32WL5x dual-core safety manual

Introduction

This document must be read along with the technical documentation such as reference manual and datasheet for the STM32WL5x dual-core microcontrollers, available on www.st.com.

This user manual 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.

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

The safety manual 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 microcontrollers, 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 Arm® Cortex®-based STM32WL5x dual-core 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 devices.

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

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

arm

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 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
D2.1 b)	Section 3.2
D2.1 c)	
D2.2 a)	General information in Section 4.1 See document [1] and 'section 1.3' of document [2] for detailed information on failure modes and related failure rates
D2.2 b)	
D2.2 c)	
D2.2 d)	
D2.2 e)	
D2.2 f)	Section 3.6 for useful information on DTI of each safety mechanisms ('Periodicity' in specification tables) Section 3.3.1 for general guidance on DTI
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 (cannot be described in this document).
D2.2 h)	Periodic proof test excluded by specific ASR3.1 in Section 3.3.1
D2.2 i)	Section 3.7
D2.2 j)	Section 3.2.3 and Section 3.2.4
D2.2 k)	Section 3.2.2

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

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

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 external to the defined *Compliant item*.

As a consequence of *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 the device. Such information includes, among others, application-related assumptions for danger factors, frequency of failures and diagnostic coverage already guaranteed by the application.

3.2 Compliant item

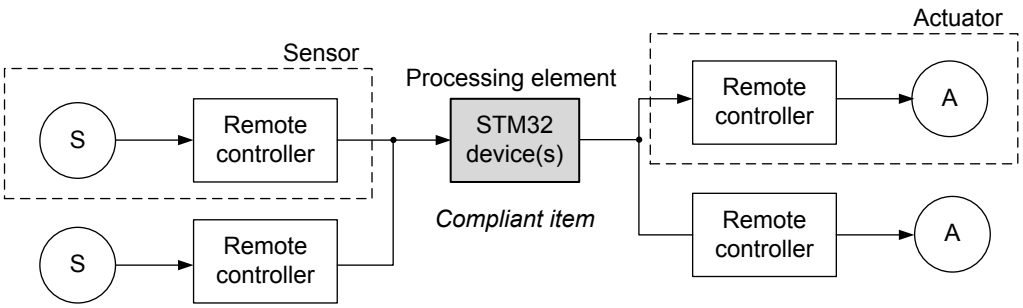
This section defines the *Compliant item* term and provides information on its usage in different safety architecture schemes.

3.2.1 Definition of Compliant item

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

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.
- the computation processing elements can be involved (to the extent depending to the target safety integrity) in the implementation of local software-based diagnostic functions; this is represented by the block PEd
- 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 . The role of the WDTe and VMONe external processes is clarified in Section 3.6 .

- WDTe: refer to *External watchdog* – CPU_SM_5 and *Control flow monitoring* in the application software – CPUM0_SM_1, CPUM4_SM_1
- VMONe: refer to *Supply voltage monitoring* – VSUP_SM_1 and *System-level power supply management* - VSUP_SM_5.

In summary, the 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.

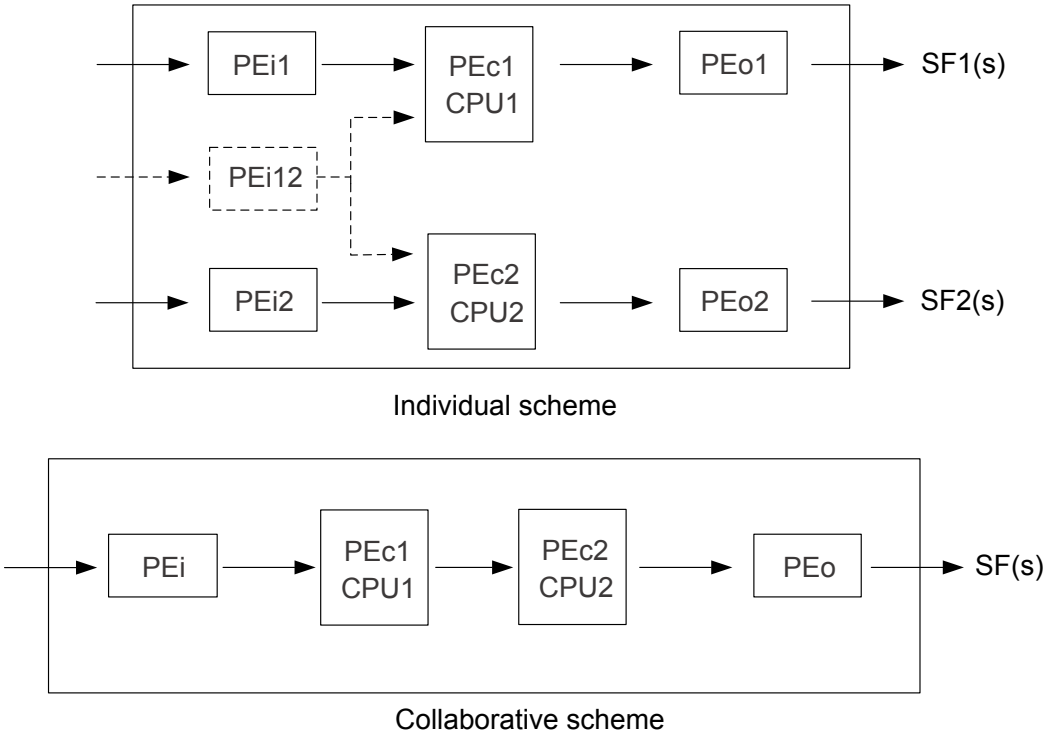
Warning: *Because of general-purpose nature of the device, its safety concept is mainly software-based. Accordingly, any following claim related to the device possibility to support itself the implementation of safety functions up to a certain SIL, is strongly correlated to the observance of CoUs as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

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

- Individual scheme: each CPU may implement a specific safety function, with no collaboration from the other*Compliant item*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) and condition of use (refer to Section 3.7). Related rationales are exposed in Section 3.2.5 Separation concept.

According to the definition for implemented safety functions, a *Compliant item* (element) can be classified as type B (as per IEC61508-2, 7.4.4.1.3 definition). Despite accurate, exhaustive and detailed failure analysis, the device has to be considered as intrinsically complex. This implies its type B classification.

Two main safety architectures are identified:

- architecture 1oo1 (using one device)
- architecture 1oo2 (using two devices)

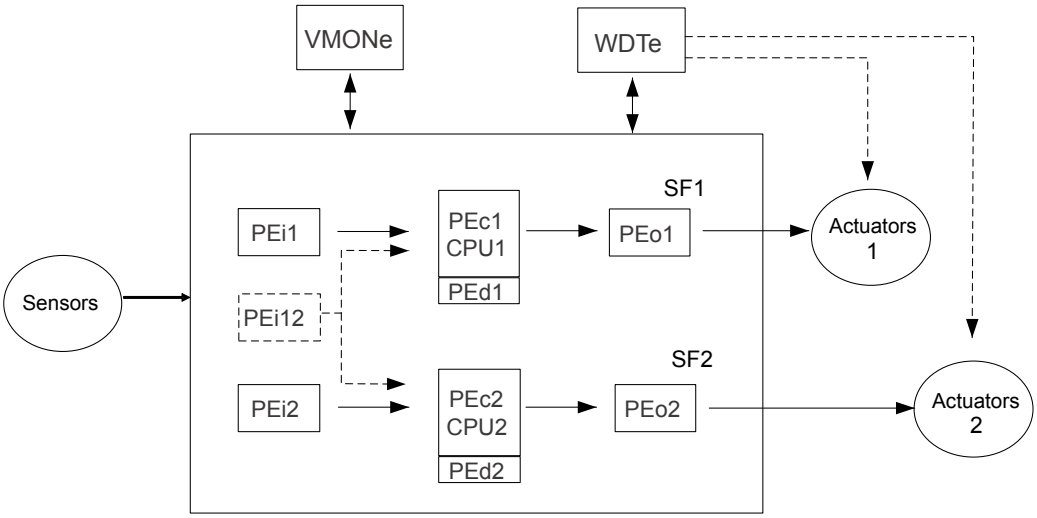
3.2.3 Reference safety architectures - 1oo1

The 1oo1 reference architecture (Section 3.2.3) 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.

This architecture targets safety integrity level (SIL) *SIL2*. Both individual and collaborative schemes are possible.

3.2.3.1 *Individual scheme*

Figure 4. 1oo1 reference architecture - individual scheme

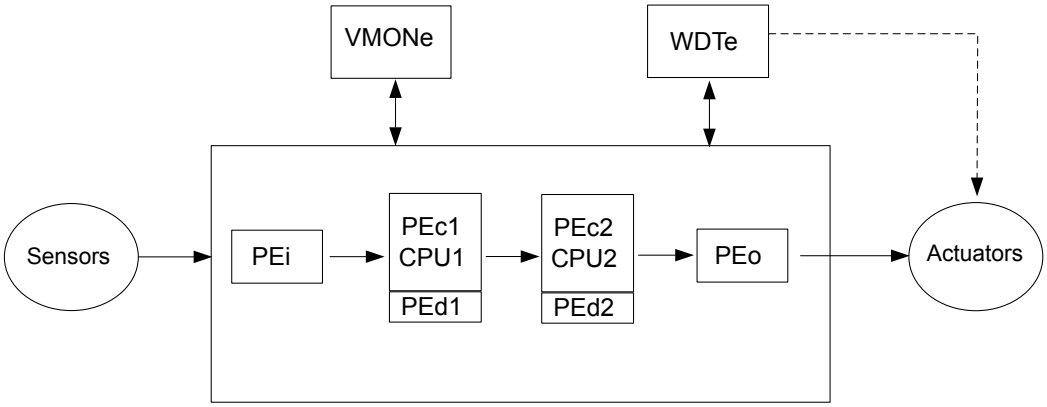


In the framework of individual scheme, the two chains PEi1/PEc1/PEo1 and PEi2/PEc2/PEo2 implement separate sets of safety function(s). Individual sets of external actuators are involved in the safety function(s) implementation. Links between WDTe and actuators show the capability of the external watchdog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

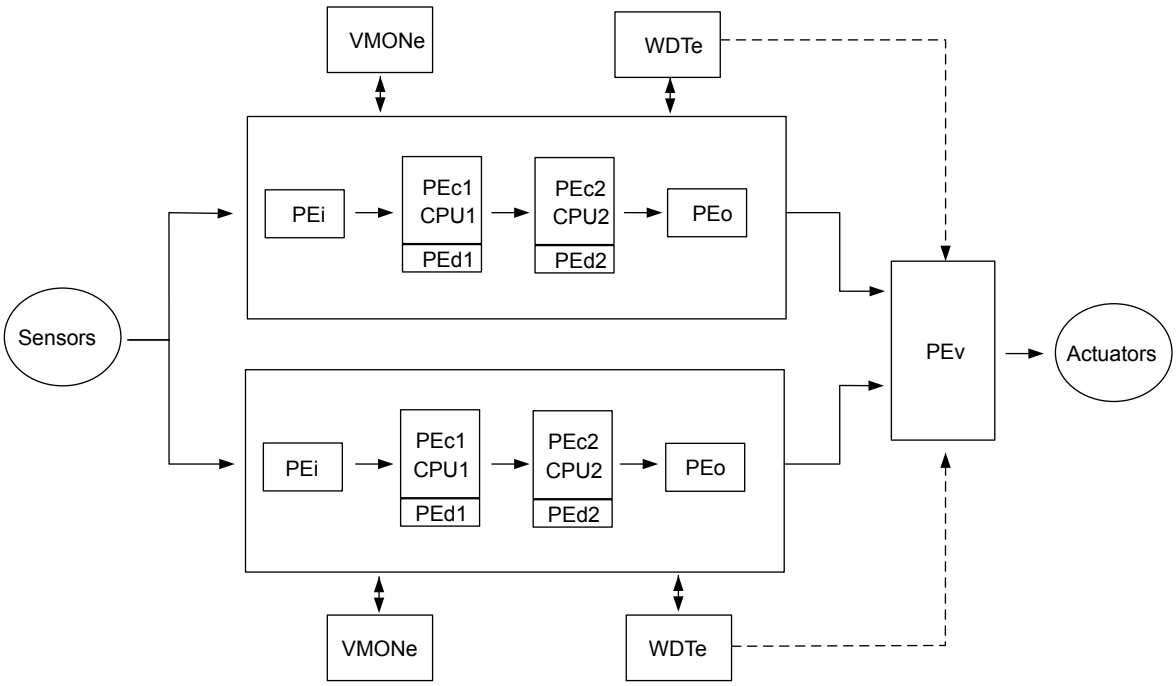
The 1oo2 reference architecture (Figure 6) contains two separate channels, both implemented as 1oo1 reference architecture ensuring safety integrity of *Compliant item* through combining device internal processes (implemented safety mechanisms) with external processes WDTe and VMONe. The overall safety integrity is then ensured by the external voter PEv, which allows claiming hardware fault tolerance (HFT) equal to 1. PEv role is to facilitate the safety function processing by each of the two individual channels, to allow the safety function implementation even when one channel is faulty. PEv complexity implementation is strongly dependent on the nature of the safety function and safe state definitions. Achievement of higher safety integrity levels as per IEC61508-2 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 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 to the clause 7.4.3.2 in IEC61508-2, this architectural scheme may provide benefits to the software applications systematic capability (SC) only when diverse software is adopted in the two channels.*

For the sake of simplicity, only collaborative scheme is adopted for 1oo2 architecture.

Figure 6. 1oo2 reference architecture



3.2.5 Separation concept

The coexistence of two different CPUs (Arm® Cortex®-M0+ and Cortex®-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 Cortex®-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 force 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 any potential issues related to the management of the required external watchdog. Furthermore, CoU_16 guarantees that the software routines in charge of watchdog management and safety data exchange between CPUs, are always active because developed with highest systematic capability.

Temporal separation

The effectiveness of the spatial separation (see above) is correlated to the actual capability of prescribed hardware and software to correctly functioning. 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 requirements for *Compliant item* as further listed. *ASR* stands for assumed safety requirement.

Caution: It is end-user responsibility to check the compliance of the final application with these assumptions.

ASR1: *Compliant item* can be used to implement one of following kinds of safety function modes of operation according to IEC61508-4,3.5.16:

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

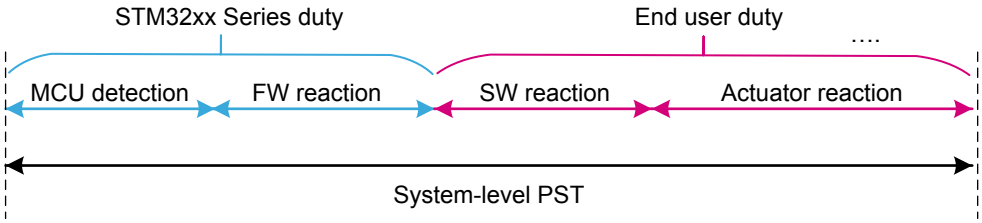
ASR2: *Compliant item* is used to implement safety function(s) allowing a specific worst-case time budget (see note below) for the STM32 MCU to detect and react to a failure. That time corresponds to the portion of the process safety time (PST) allocated to the device (*STM32xx Series duty* in Figure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

The computation for time budget mainly depends on the execution speed for periodic tests implemented by software. Such duration depends on the actual amount of hardware resources (RAM memory, Flash memory, peripherals) actually declared as safety related. Further constraints and requirements from IEC61508-2, 7.4.5.3 must be considered.

Figure 7. Allocation and target for STM32 *PST*



ASR3.1: *Compliant item* is used to implement safety function(s) that can be continuously powered on for a period over eight hours. It is assumed to not require any proof test, and the lifetime of the product is considered to be no less than 10 years.

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 (Arm® Cortex®-M0+ and Cortex®-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. Therefore they are not distinguishable in terms of their safety requirements.

ASR4.3: In case of multiple safety function implementations, it is assumed that the 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 the application software, coexisting with safety functions.

ASR5: It is assumed that the implemented safety function(s) does (do) not depend on transition of the 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: The application software(1) is informed by the presence of a fault and a reaction by the application software itself is possible.
- SS2: The application software(1) cannot be informed by the presence of a fault or is not able to execute a reaction.

Note: The end user must take into account that random hardware failures affecting the 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	The application software(1) is informed by the presence of a fault. A reaction by the application software itself is possible.	Fault reporting to the application software	The application software(1) drives all implemented safety functions in their safe state, possibly leveraging on inter-CPU communications (CPU_SM_11).	The application software(1) drives the overall system in its safe state.	The application software(1) in one of the two channels drives the overall system in its safe state.
SS2	The application software(1) cannot be informed by the presence of a fault or 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 the device (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 the 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 (refer to Section 3.5 and Section 3.6 .

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

ASR9.1: It is assumed that 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 device 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) is not used to artificially implement a 1oo2 scheme with one single STM32WL5x dual-core device.

Note: This assumption 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. This solution cannot be considered a true 1oo2 architecture as described in IEC61508-6, because of the incomplete separation between the two channels.

ASR10: When adopting individual scheme, it is the 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).

3.4 Electrical specifications and environment limits

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

- absolute maximum rating
- capacity
- operating conditions

For electrical specifications and environmental limits of the 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 IEC 61508-2, 7.4.2.2, the *Route 1S* is considered in the analysis of *Device*(s). As clearly authorized by IEC 61508-2, 7.4.6.1, 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. A *safety case database* (see Section 5 List of evidences) keeps evidences of the current compliance level to the norm.

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 device architecture, 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 this document provides information relative to the superset of peripherals available on the device 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 the device. Refer to its reference manual 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: means that the method is available on silicon. End user: the method must be implemented by the end user through the 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 the 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*: the method contributes to lower the probability of occurrence of a failure. • If ranked for *Systematic*: the method is conceived to mitigate systematic errors (bugs) in the 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* : the safety mechanism is active in continuous mode. *Periodic*: the safety mechanism is executed periodically.(1) *On-demand*: the safety mechanism is activated in correspondence to a specified event (for instance, reception of a data message). *Startup*: the safety mechanism is supposed to be executed only at power-up or during off-line maintenance periods.
Test for the diagnostic	Reports specific procedure (if any and recommended) to allow on-line tests of safety mechanism efficiency.
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 IEC61508-2, 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	Periodical core self-test software for Arm® Cortex®-M0+
Ownership	End user or ST
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 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. The 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 (see 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 the application software
Ownership	End user
Detailed implementation	A significant part of the failure distribution of the Arm® Cortex®-M0+ 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 SM_CPU_0. Therefore it is necessary to implement a run-time control of the application-software flow, in order to monitor and detect a 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. Guidelines for the implementation of this 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 a 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 the 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 4.1.3 Notes on multiple-fault scenario for details).

SM CODE	CPUM0_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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM0_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 5. CPUM0_SM_2

SM CODE	CPUM0_SM_2
Description	Double computation in the 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+ subparts devoted to mathematical computations and data access. Guidelines for the implementation of this method are the following: • The requirement needs be applied only to safety-relevant computation, which in case of wrong result may 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: periodical core self-test software
Recommendations and known limitations	The 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.

SM CODE	CPUM0_SM_3
Error reporting	High-priority interrupt event
Fault detection Time	Depends on implementation. Refer to functional documentation.
Addressed Fault Model	Permanent and 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 faults protection	CPUM0_SM_0: periodical 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 the application software
Ownership	End user
Detailed implementation	The stack hardening method is required to address faults (mainly transient) affecting Arm® Cortex®-M0+ register bank. This method is based on source code modification, introducing information redundancy in register-passed information to called functions. Guidelines for the implementation of this 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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple faults protection	CPUM0_SM_0: periodical core self-test software
Recommendations and known limitations	This method partially overlaps with defensive programming techniques required by IEC61508 for software development. Therefore in presence of an 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

SM CODE	CPUM0_SM_7
Detailed implementation	The Arm® Cortex®-M0+ memory protection unit is able to detect illegal access to protected memory areas, according to criteria set by the end user.
Error reporting	Exception raise (MemManage)
Fault detection time	Refer to functional documentation
Addressed fault model	Systematic (software errors) Permanent and 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: periodical 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 the application software. The MPU can be indeed used: • to enforce privilege rules • to separate processes • to 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 the device. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 9. MPUM0_SM_0

SM CODE	MPUM0_SM_0
Description	Periodical 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 application software). Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 10. MPUM0_SM_1

SM CODE	MPUM0_SM_1
Description	Arm® Cortex®-M0+ MPU software test
Ownership	End user

SM CODE	MPUM0_SM_1
Detailed implementation	This method tests the 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-fault protection	CPUM0_SM_0: periodical core self test software
Recommendations and known limitations	Startup execution of this safety mechanism is recommended for multiple fault mitigations (see Section 4.1.3 Notes on multiple-fault scenario for details) .

3.6.2 Arm® Cortex®-M4 CPU

Table 11. CPUM4_SM_0

SM CODE	CPUM4_SM_0
Description	Periodical core self-test software for Arm® Cortex®-M4 CPU
Ownership	End user or ST
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. The 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 (see 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 the application software
Ownership	End user
Detailed implementation	A significant part of the failure distribution of the Arm® Cortex®-M4 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 SM_CPU_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. Guidelines for the implementation of this 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 4.1.3 Notes on multiple-fault scenario for details).
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 13. CPUM4_SM_2

SM CODE	CPUM4_SM_2
Description	Double computation in the 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 subparts devoted to mathematical computations and data access. Guidelines for the implementation of this 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.

SM CODE	CPUM4_SM_2
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: periodical core self-test software
Recommendations and known limitations	The 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 and 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: periodical 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 the application software
Ownership	End user
Detailed implementation	The stack hardening method is required to address faults (mainly transient) affecting Arm® Cortex®-M4 register bank. This method is based on source code modification, introducing information redundancy in register-passed information to called functions.

SM CODE	CPUM4_SM_4
	Guidelines for the implementation of this 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 and 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: periodical core self-test software
Recommendations and known limitations	This method partially overlaps with defensive programming techniques required by IEC61508 for software development. Therefore in presence of an 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 MPU is able to detect illegal access to protected memory areas, according to criteria set by the end user.
Error reporting	Exception raise (MemManage)
Fault detection time	Refer to functional documentation.
Addressed fault model	Systematic (software errors) Permanent and transient (only program counter and memory access failures)
Dependency on device configuration	None
Initialization	MPU registers must be programmed at startup.
Periodicity	On line
Test for the diagnostic	Not needed
Multiple-fault protection	MPU_SM_0: periodical 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 the application software. The MPU can be indeed used: • to enforce privilege rules • to separate processes • to 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 the device. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 17. MPUM4_SM_0

SM CODE	MPUM4_SM_0
Description	Periodical read-back of Arm® Cortex®-M4 MPU configuration registers
Ownership	End user
Detailed implementation	This method must be applied to the Arm® Cortex®-M4 MPU configuration registers (also unused by the application software). Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. NVIC_SM_0.
Fault detection time	
Addressed fault model	
Dependency on device configuration	
Initialization	
Periodicity	
Test for the diagnostic	
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 the 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-fault protection	CPUM4_SM_0: periodical core self test software
Recommendations and known limitations	Startup execution of this safety mechanism is recommended for multiple fault mitigations (see Section 4.1.3 Notes on multiple-fault scenario for details).

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) addresses failure mode of program counter or control structures of the 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 Section 3.3.1 Safety requirement assumptions. It also contributes to dramatically reduce potential common cause failures, because the external watchdog is clocked and supplied independently of the device.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation (watchdog timeout interval).
Addressed fault model	Permanent and 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 the application software CPU_SM_6: Internal watchdogs IWDG/WWDG
Recommendations and known limitations	In case of window watchdog use, the end user must consider possible tolerance in the application-software execution, to avoid false error reports (affecting system availability). *Note: An external watchdog may be needed anyway when the device is used to trigger final elements, in order to comply at system level with requirements from IEC61508-2:2010 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 or WWDG watchdog linked to control flow monitoring method (refer to CPUM0_SM_1) addresses the failure mode of program counter or control structures of the 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 the application software WDG_SM_0: periodical read-back of configuration registers

SM CODE	CPU_SM_6
Recommendations and known limitations	The IWDG/WWDG intervention is able to achieve a potentially “incomplete” local safe state because it can only guarantee that the CPU is reset. No guarantee that the application software can be still executed to generate combinations of output signals that may be needed by the external system to achieve the final safe state. Internal watchdog use is part of the STM32WL5x dual-core separation concept (see Section 3.2.5 Separation concept and CoU_15,16 for details).

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 the following information: • CPU integrity check status (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 and transient
Dependency on device configuration	NA
Initialization	Depends on implementation.
Periodicity	Periodical
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0: periodical core self-test software CPUM4_SM_0: periodical 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). Its implementation can be recommended or not, depending on the architecture and CPU scheme selected. It must not be confused with DUAL_SM_0 that specifies communications between two different STM32WL5x dual-core devices and not between CPUs of the same device.

3.6.4 Embedded Flash memory

Table 22. FLASH_SM_0

SM CODE	FLASH_SM_0
Description	Periodical software test for the Flash memory
Ownership	End user or ST
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 cell 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 must 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 the Flash memory as well.
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen.
Multiple-fault protection	CPUM0_SM_0: periodical core self-test software for Arm® Cortex®-M0+ CPU CPUM4_SM_0: periodical core self-test software for Arm® Cortex®-M4 CPU CPUM0_SM_1: control flow monitoring in the application software CPUM4_SM_1: control flow monitoring in the application software
Recommendations and known limitations	This test is expected to have a relevant time duration. Test integration must therefore consider the impact on the application software execution. The use of internal cyclic redundancy check (CRC) module is recommended. In principle, the 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 (see 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 23. 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 the application software loaded from the Flash memory. For more details on the implementation, refer to description CPUM0_SM_1 and CPUM4_SM_1.
Error reporting	Depends on implementation.

SM CODE	FLASH_SM_1
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	CPUM0_SM_1 and CPUM4_SM_1 correct implementation supersedes this requirement.

Table 24. FLASH_SM_2

SM CODE	FLASH_SM_2
Description	Arm® Cortex®-M0+ and Arm® Cortex®-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+ and Arm® Cortex®-M4 HardFault exceptions. Refer to CPUM0_SM_3 and CPUM4_SM_3 for detailed description.
Error reporting	Refer to CPUM0_SM_3 and CPUM4_SM_3.
Fault detection time	Refer to CPUM0_SM_3 and CPUM4_SM_3.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Refer to CPUM0_SM_3 and CPUM4_SM_3.
Periodicity	Continuous
Test for the diagnostic	None
Multiple-fault protection	CPUM0_SM_1, CPUM4_SM_1: control flow monitoring in application software
Recommendations and known limitations	None

Table 25. 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	NA
Addressed fault model	None (systematic only)
Dependency on device configuration	None
Initialization	Not needed (enabled by default)
Periodicity	Continuous

SM CODE	FLASH_SM_3
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	This method addresses systematic faults in software application and it has zero efficiency in addressing hardware random faults affecting the option-byte value during running time. No DC value is therefore associated.

Table 26. FLASH_SM_4

SM CODE	FLASH_SM_4
Description	Static data encapsulation
Ownership	End user
Detailed implementation	If static data are stored in the Flash memory, the 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 and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 27. 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. In case the program counter jumps outside the application program area due to a transient fault affecting the CPU, the system evolves in a deterministic way.
Error reporting	NA
Fault detection time	
Addressed fault model	None (fault avoidance)
Dependency on device configuration	None
Initialization	NA
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	Filling code can be made of NOP instructions, or an illegal code that leads to a HardFault exception raise.

Table 28. FLASH_SM_7

SM CODE	FLASH_SM_7
Description	ECC on Flash memory
Ownership	ST
Detailed implementation	The internal Flash memory is protected by an ECC (error correction code) redundancy, implementing a protection feature at double-word (64 bit) level: • one bit fault: correction • two bits fault: detection
Error reporting	Refer to functional documentation.
Fault detection time	ECC bits are checked during a memory reading.
Addressed fault model	Permanent and 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". Read also the "Recommendations and known limitations" below.
Multiple-fault protection	• FLASH_SM_0: periodical software test for Flash memory(1) • DIAG_SM_0: periodical 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. Important note: because of the lack of ECC direct test procedure, single-fault failures leading to unintended ECC correction may cause an incorrect data read from Flash memory. This is why STM32WL5x dual-core safety concept strongly recommends the adoption of multiple layers of overlapped safety mechanisms which collaborate to mitigate such kind of ECC failures. Refer to [1] for further detailed information on ECC failures mitigation strategy.

1. *The FMEDA snapshot document includes information on recommended frequency for FLASH_SM_0.*

Table 29. FLASH_SM_8

SM CODE	FLASH_SM_8
Description	Readout protection (RDP), write protection (WRP)
Ownership	ST
Detailed implementation	Flash memory can be protected against illegal reads or erase/write by using these protection features. The combination of these techniques and the related different protection level allows the end user to build an effective access protection policy.
Error reporting	Refer to functional documentation. In some cases, an HardFault error is generated.
Fault detection time	Refer to functional documentation.
Addressed fault model	Systematic
Dependency on device configuration	None
Initialization	Not needed
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	Not needed

SM CODE	FLASH_SM_8
Recommendations and known limitations	Hardware random-failure detection capability for the 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 not relevant in the framework of STM32WL5x dual-core safety concept.

Table 30. FLASH_SM_9

SM CODE	FLASH_SM_9
Description	Periodic test by software for the Flash memory address decoder
Ownership	ST or end user
Detailed implementation	Permanent faults affecting the system Flash memory 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 density depends upon the part number.
Initialization	Not needed
Periodicity	Periodical
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Overlaps with FLASH_SM_0 implementation are possible.

3.6.5 Embedded SRAM

Table 31. RAM_SM_0

SM CODE	RAM_SM_0
Description	Periodical software test for static random access memory (SRAM or RAM)
Ownership	End user or ST
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 periodical 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 be also 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 the test implementation design strategy chosen.
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical 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). *Note: Unused RAM section can be excluded by the testing, under end user responsibility on actual RAM usage by final application software.* Startup execution of this safety mechanism is recommended for multiple fault mitigations (see Section 4.1.3 Notes on multiple-fault scenario for details). Test implementation must consider the constraints introduced by CoU_10. Accordingly, it is expected that each CPU tests its "private" SRAM partition and not the one allocated to the other CPU. In case a destructive test is selected (such as March C-), the execution of the test on the shared memory defined by CoU_12 is not recommended, to avoid conflicts between the two CPUs.

Table 32. RAM_SM_2

SM CODE	RAM_SM_2
Description	Stack hardening for the 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 Table 7. CPUM0_SM_4 and Table 15. CPUM4_SM_4.
Fault detection time	
Addressed fault model	
Dependency on device configuration	

SM CODE	RAM_SM_2
Initialization	Refer to Table 7. CPUM0_SM_4 and Table 15. CPUM4_SM_4.
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	

Table 33. 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 SRAM controller, it is required to implement information redundancy on the safety-related system variables stored in the RAM. 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 at least one time per *PST*. • Data vectors stored in SRAM must be protected by a encoding checksum (such as *CRC*).
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Implementation of this safety method shows a partial overlap with an already foreseen method (CPUM0_SM_2, CPUM4_SM_2); optimizations in implementing both methods are therefore possible.

Table 34. RAM_SM_4

SM CODE	RAM_SM_4
Description	Control flow monitoring in the application software
Ownership	End user
Detailed implementation	In case the 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. To address such failures it is needed to implement this method. 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 and transient

SM CODE	RAM_SM_4
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Needed just in case of application-software execution from SRAM. CPUM0_SM_1, CPUM4_SM_1 correct implementation supersedes this requirement.

Table 35. RAM_SM_5

SM CODE	RAM_SM_5
Description	Periodical integrity test for the 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 (at least once per *PST*). 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 and 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 the test implementation design strategy chosen.
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self test software CPUM0_SM_1, CPUM4_SM_1: control flow monitoring in application software
Recommendations and known limitations	This method must be implemented only in case of application software or diagnostic libraries are executed from RAM.

Table 36. RAM_SM_6

SM CODE	RAM_SM_6
Description	Read protection (RDP), Write protection (WRP)
Ownership	ST
Detailed implementation	The SRAM2 memory can be protected against illegal reads or erase/write by using these protection features. The combination of these techniques and the related different protection level allows the end user to build an effective access protection policy.
Error reporting	Refer to functional documentation. In some cases, an HardFault error is generated.
Fault detection time	Refer to functional documentation.
Addressed fault model	Systematic
Dependency on device configuration	SRAM2 size can vary depending on the part number.
Initialization	Not needed
Periodicity	Continuous

SM CODE	RAM_SM_6
Test for the diagnostic	Not needed
Multiple-fault protection	Not needed
Recommendations and known limitations	Hardware random-failure detection capability for SRAM2 access policy is restricted to well-selected marginal failure modes, mainly affecting program counter and SRAM2 interface functions. The associated diagnostic coverage is therefore expected to be irrelevant in the framework of STM32WL5x dual-core safety concept.

Table 37. RAM_SM_8

SM CODE	RAM_SM_8
Description	Periodic test by software for SRAM address decoder
Ownership	ST or end user
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 varies according to the part number.
Initialization	Not needed
Periodicity	Periodical
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self test software
Recommendations and known limitations	Overlaps with RAM_SM_0 implementation are possible.

3.6.6 System bus architecture/peripherals interconnect matrix

Table 38. BUS_SM_0

SM CODE	BUS_SM_0
Description	Periodical 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 needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Implementation can be considered in large part as overlapping with the widely used periodical read-back of configuration registers required for several peripherals.

Table 39. 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 the 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 and 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, CPUM4_SM_0: periodical 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 40. LOCK_SM_0

SM CODE	LOCK_SM_0
Description	Lock mechanism for configuration options

SM CODE	LOCK_SM_0
Ownership	ST
Detailed implementation	The STM32WL5x dual-core 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

3.6.7 EXTI controller

Table 41. NVIC_SM_0

SM CODE	NVIC_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This test is implemented by executing a periodical 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 one time within *PST* after a peripheral update. 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, 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 the application software. Signature mismatch is considered as failure detection.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Values of configuration registers must be read after the boot before executing the first check.
Periodicity	Periodic
Test for the diagnostic	NA
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, to avoid false positive detections.

Table 42. NVIC_SM_1

SM CODE	NVIC_SM_1
Description	Expected and unexpected interrupt check
Ownership	End user

SM CODE	NVIC_SM_1
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. Guidelines for the implementation of the method are the following: • The interrupts implemented on the device 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 (“babbling idiot” interrupt source). 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. • Interrupt requests related to non-safety-related peripherals are handled with the same method here described, despite their originator safety classification
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodic core self-test software
Recommendations and known limitations	In order to decrease the complexity of method implementation, it is suggested to use polling technique (when possible) instead of interrupt for end system implementation

3.6.8 Global TrustZone® controller (GTZC)

Table 43. GTZC_SM_0

SM CODE	GTZC_SM_0
Description	Periodical read-back of 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 3.6.7 .
Error reporting	Refer to Table 41. 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 44. **GTZC_SM_1**

SM CODE	GTZC_SM_1
Description	GTZC illegal access detection
Ownership	ST
Detailed implementation	In the framework of the security concept that the 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	
Addressed fault model	Systematic
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPU_SM_0: periodic core self-test software
Recommendations and known limitations	None

3.6.9 Direct memory access controllers (DMA1/2 and DMAMUX)

Table 45. **DMA_SM_0**

SM CODE	DMA_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to DMA configuration register and channel addresses register as well. Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 46. **DMA_SM_1**

SM CODE	DMA_SM_1
Description	Information redundancy on data packet transferred via the DMA
Ownership	End user
Detailed implementation	This method is implemented adding to data packets transferred by the DMA, a redundancy check (such as *CRC* check, or similar one) with encoding capability. Full data packet redundancy is overkilling.

SM CODE	DMA_SM_1
	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 the application software before consuming data.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	To give an example about checksum encoding capability, using just a bit-by-bit addition is unappropriated.

Table 47. DMA_SM_2

SM CODE	DMA_SM_2
Description	Information redundancy by including sender or receiver identifier on data packet transferred via the DMA
Ownership	End user
Detailed implementation	This method helps to identify inside the MCU the source and the originator of the message exchanged by the 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 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 and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 48. DMA_SM_3

SM CODE	DMA_SM_3
Description	Periodical software test for the DMA

SM CODE	DMA_SM_3
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 needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 49. 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 IEC61508:3 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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical 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.

3.6.10 Hardware semaphore (HSEM)

Table 50. HSEM_SM_0

SM CODE	HSEM_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 51. 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. This method is implemented by software, leveraging on the presence of a system watchdog (internal or external). 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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self test software
Recommendations and known limitations	Implementation must take into account potential overlaps/optimizations with CPUM0_SM_1, CPUM4_SM_1.

3.6.11 Inter-processor communication controller (IPCC)

Table 52. IPCC_SM_0

SM CODE	IPCC_SM_0
Description	Periodical 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 3.6.7 .
Error reporting	Refer to Table 41. 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 53. 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: periodical core self-test software for Arm Cortex-M4 CPU CPU2_SM_0: periodical 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 Universal synchronous/asynchronous receiver/transmitter (USART1/2)

Table 54. UART_SM_0

SM CODE	UART_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to UART configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 55. UART_SM_1

SM CODE	UART_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	USART communication module embeds protocol error checks (like additional parity bit check, overrun, frame error) conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a marginal percentage 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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not required
Multiple-fault protection	UART_SM_2: information redundancy techniques on messages
Recommendations and known limitations	USART communication module is fitted by several different configurations. The actual composition of communication error checks depends on selected configuration. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 56. UART_SM_2

SM CODE	UART_SM_2
Description	Information redundancy techniques on messages

SM CODE	UART_SM_2
Ownership	End user
Detailed implementation	This method is implemented adding to data packets transferred by UART a redundancy check (like 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 the application software before consuming data.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	It is assumed that the remote UART counterpart has an equivalent capability of performing the check described. Transmission full redundancy (message repetition) should not be used because its detection capability is limited to a subset of communication unit failure modes. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated.

Table 57. 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 IEC61508:2, 7.4.11.1. Implementation guidelines are the following: • 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 • The 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 packet lost).
Error reporting	Depends on implementation.
Fault detection time	
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical core self-test software

SM CODE	UART_SM_3
Recommendations and known limitations	Important note: it is assumed that the remote UART counterpart has an equivalent capability of performing the checks described. 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.

3.6.13 Inter-integrated circuit (I2C1/2)

Table 58. IIC_SM_0

SM CODE	IIC_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 59. IIC_SM_1

SM CODE	IIC_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	The I^2C communication module embeds protocol error checks (such as overrun, underrun, packet error) conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a marginal percentage 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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
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 60. **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 the application software before consuming data.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical 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. Transmission full redundancy (message repetition) should not be used because its detection capability is limited to a subset of communication unit failure modes. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. This method is overlapped with IIC_SM_3 if hardware handled *CRC* insertion is possible.

Table 61. **IIC_SM_3**

SM CODE	IIC_SM_3
Description	*CRC* packet-level
Ownership	ST
Detailed implementation	The I^2C communication module allows the activation 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 peripheral configuration (for example baud rate), refer to functional documentation.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
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. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 62. **IIC_SM_4**

SM CODE	IIC_SM_4
Description	Information redundancy techniques on messages, including end-to-end protection.
Ownership	End user

SM CODE	IIC_SM_4
Detailed implementation	This method aims to protect the communication between an I2C peripheral and its external counterpart. Refer to USART_SM_3 description for detailed information.
Error reporting	Refer to Table 57. UART_SM_3.
Fault detection time	
Addressed fault model	
Dependency on device configuration	
Initialization	
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	Important note: it is assumed that the remote I2C counterpart has an equivalent capability of performing the checks described. Refer to USART_SM_3 for further notice.

3.6.14 Serial peripheral interface (SPI1/2)

Table 63. SPI_SM_0

SM CODE	SPI_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 64. 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 able anyway to detect a marginal percentage 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.

SM CODE	SPI_SM_1
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	SPI_SM_2: information redundancy techniques on messages
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 65. 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 the application software before consuming data.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	It is assumed that the remote SPI counterpart has an equivalent capability of performing the check described. Transmission full redundancy (message repetition) should not be used because its detection capability is limited to a subset of communication unit failure modes. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. This method is overlapped with SSP_SM_3 if hardware handled *CRC* insertion is possible.

Table 66. 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 and transient
Dependency on device configuration	None

SM CODE	SPI_SM_3
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
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. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 67. 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 USART_SM_3 description for detailed information.
Error reporting	Refer to Table 57. UART_SM_3.
Fault detection time	
Addressed fault model	
Dependency on device configuration	
Initialization	
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	Important note: it is assumed that the remote SPI counterpart has an equivalent capability of performing the checks described. Refer to USART_SM_3 for further notice.

3.6.15 Analog-to-digital converters (ADC)

Table 68. ADC_SM_0

SM CODE	ADC_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to ADC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 69. 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 acquisition data 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	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	It is highly probable that this recommendation is satisfied by design by the end-user application software. Use of multiple acquisitions followed by average operations is a common technique in industrial applications where it is needed to survive with spurious EMI disturbs on sensor lines.

Table 70. ADC_SM_2

SM CODE	ADC_SM_2
Description	Range check by application software
Ownership	End user
Detailed implementation	Guidelines for the implementation of this method are the following:

SM CODE	ADC_SM_2
	• 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	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	The implementation (and the related diagnostic efficiency) of this safety mechanism are strongly application-dependent.

Table 71. ADC_SM_3

SM CODE	ADC_SM_3
Description	Periodical 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 know. Method can be implemented with different level of complexity: • Basic complexity: acquisition and check of upper or lower rails (V_{DD} or V_{SS}) 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 needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Combination of two different complexity method can be used to better optimize test frequency in high demand safety functions.

Table 72. ADC_SM_4

SM CODE	ADC_SM_4
Description	1oo2 scheme for ADC inputs

SM CODE	ADC_SM_4
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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple-fault protection	ADC_SM_0: periodical read-back of ADC 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.

3.6.16 Digital-to-analog converter (DAC)

Table 73. DAC_SM_0

SM CODE	DAC_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 74. DAC_SM_1

SM CODE	DAC_SM_1
Description	DAC output loopback on ADC channel
Ownership	End user
Detailed implementation	Implementation is realized by routing the active DAC output to one ADC channel, and by checking the output current value with his expected one.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.

SM CODE	DAC_SM_1
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous or on demand
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Efficiency versus transient failures is linked to final application characteristics. Tm is defined as 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.

3.6.17 Advanced encryption standard hardware accelerator (AES)

Table 75. AES_SM_0

SM CODE	AES_SM_0
Description	Periodical read-back of 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 3.6.7 .
Error reporting	Refer to Table 41. 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 76. AES_SM_1

SM CODE	AES_SM_1
Description	Encryption/decryption collateral detection
Ownership	ST
Detailed implementation	Encryption and decryption operations performed by the AES are composed of 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 the AES 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	Depends on device configuration.
Addressed fault model	Permanent/transient
Dependency on device configuration	None
Initialization	Depends 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 IEC61508 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), that can be executed periodically to early detect AES failures before its use by the application software.

Table 77. 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 its 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 Table 57. UART_SM_3.
Fault detection time	
Addressed fault model	
Dependency on device configuration	None
Initialization	Refer to Table 57. UART_SM_3.
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
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 the device security feature are **not** detailed in this document.*

3.6.18 Basic timers TIM 6/7

Table 78. GTIM_SM_0

SM CODE	GTIM_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to basic counter timer TIM6 or TIM7 configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 79. GTIM_SM_1

SM CODE	GTIM_SM_1
Description	1oo2 for counting timers
Ownership	End user
Detailed implementation	This method implements via software a 1oo2 scheme between two counting resources. Guidelines for the implementation of this 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 the 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 counters values each time the timer value is used to affect safety function. • In case of interrupt generation usage: use the first timer as main interrupt source for the service routines, and use 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 and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic.

3.6.19 Advanced, general, and low-power timers (TIM1/2/16/17 and LPTIM2/3)

Note: As the timers are equipped with many different channels, each independent from the others, and possibly programmed to realize different features, the safety mechanism is selected individually for each channel.

Table 80. ATIM_SM_0

SM CODE	ATIM_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to advanced, general, high resolution and low-power timers (TIM1/2/16/17 and LPTIM2/3) configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 81. 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. Guidelines for the implementation of this 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 the 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 counters values each time the timer value is used to affect safety function. • In case of interrupt generation usage: use the first timer as main interrupt source for the service routines, and use 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 and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic.

SM CODE	ATIM_SM_1
	This method applies to timer channels that are merely used as elapsed time counters.

Table 82. 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 external signal acquisition and measurement, like “input capture” and “encoder reading”. Implementation requires to connect the external signals also to a redundant timer, and to perform a coherence check on the measured data at application level. Coherence check between timers is executed each time the reading is used by the application software
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical 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 83. ATIM_SM_3

SM CODE	ATIM_SM_3
Description	Loopback scheme for 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. Guidelines for the implementation of this method 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	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software

SM CODE	ATIM_SM_3
Recommendations and known limitations	Efficiency versus transient failures is linked to final application characteristics. Tm is defined as 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 84. ATIM_SM_4

SM CODE	ATIM_SM_4
Description	Lock bit protection for timers
Ownership	ST
Detailed implementation	This safety mechanism allows the end user to lock down specified configuration options, avoiding unintended modifications by application software. It addresses therefore software development systematic faults
Error reporting	NA
Fault detection time	
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	NA
Multiple-fault protection	
Recommendations and known limitations	This method does not addresses timer configuration changes due to soft-errors.

Note: IRTIM is not individually mentioned here, being mainly implemented by TIM16 and TIM17 functions. Refer therefore to related prescriptions.

3.6.20 General-purpose input/output (GPIO)

Table 85. GPIO_SM_0

SM CODE	GPIO_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. NVIC_SM_0.
Fault detection time	
Addressed fault model	
Dependency on device configuration	GPIO availability can differ according to the part number.
Initialization	Refer to Table 41. NVIC_SM_0.
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	

Table 86. 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 application software each time the signal is used to affect application software behavior.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and 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, CPUM4_SM_0: periodical 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

Table 87. 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 application software periodically and each time output is updated.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical 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. Tm is defined as 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.

Table 88. **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 and 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	The application software must apply a correct write sequence to LCKK bit (bit 16 of the GPIOx_LCKR register) after writing the final GPIO configuration.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	Not needed
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.

3.6.21 Real-time clock module (RTC)

Table 89. **RTC_SM_0**

SM CODE	RTC_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 90. **RTC_SM_1**

SM CODE	RTC_SM_1
Description	Application check of running RTC
Ownership	End user

SM CODE	RTC_SM_1
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. Guidelines for the implementation of this 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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Periodical
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical 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 a severe way the safety function (for example, medical data storage devices), it is strongly recommended to adopt more efficient system-level measures.

Table 91. 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 and 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, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 92. RTC_SM_3

SM CODE	RTC_SM_3
Description	Application-level measures to detect failures in timestamps/event capture

SM CODE	RTC_SM_3
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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Periodic and on demand
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical 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 device in Sleep or Stop mode is prevented by the assumed requirement ASR7 (see Section 3.3.1 Safety requirement assumptions).

3.6.22 Tamper and backup registers (TAMP)

Table 93. 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 the 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	
Addressed fault model	Permanent/transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	One demand
Test for the diagnostic	NA
Multiple-fault protection	CPU_SM_0: periodic core self-test software
Recommendations and known limitations	None

3.6.23 Power controller (PWR)

Table 94. VSUP_SM_0

SM CODE	VSUP_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 95. 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 $V_{PV}D$ 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 and transient
Dependency on device configuration	None
Initialization	Protection enabled by PVDE bit and threshold programming in power control register (PWR_CR)
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	DIAG_SM_0: periodical 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 device FMEA for details. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 96. VSUP_SM_2

SM CODE	VSUP_SM_2
Description	External watchdog
Ownership	End user

SM CODE	VSUP_SM_2
Detailed implementation	Failures in the power supplies for digital logic (core or peripherals) may lead to alteration of the application software timing, which can be detected by the external watchdog as safety mechanism introduced to monitor the application software control flow. Refer to CPU_SM_5 for further information.
Error reporting	Refer to Table 19. CPU_SM_5.
Fault detection time	
Addressed fault model	
Dependency on device configuration	
Initialization	
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	

Table 97. 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 needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	This method also mitigates the eventuality of common-cause affecting the MCU and due to too high temperature. Refer to the device datasheet to set the threshold temperature.

Table 98. 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 of them are listed 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 perturbation in presence of a single failure

SM CODE	VSUP_SM_5
Error reporting	Depends on implementation.
Fault detection time	Fault avoidance
Addressed fault model	None
Dependency on device configuration	Depends on implementation.
Initialization	Continuous
Periodicity	NA
Test for the diagnostic	
Multiple-fault protection	Usually this method is already required/implemented to guarantee the stability of each component of the final electronic board.
Recommendations and known limitations	NA

3.6.24 Reset and clock controller (RCC)

Table 99. CLK_SM_0

SM CODE	CLK_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 100. 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).

SM CODE	CLK_SM_1
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	CSS protection must be enabled through the clock interrupt register (RCC_CIR) after boot stabilization.
Periodicity	Continuous
Test for the diagnostic	CLK_SM_0: periodical read-back of configuration registers
Multiple-fault protection	CPU_SM_5: external watchdog
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.

Table 101. CLK_SM_2

SM CODE	CLK_SM_2
Description	External watchdog
Ownership	ST
Detailed implementation	The external watchdog is able to detect failures in the device internal main clock (lower frequency). Refer to CPU_SM_5 for further details.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation (watchdog timeout interval).
Addressed fault model	Permanent
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_1, CPUM4_SM_1: control flow monitoring in application software
Recommendations and known limitations	If a time window function option is used, the end user must consider possible tolerance in application software execution, to avoid false error reports (affecting system availability).

Table 102. CLK_SM_3

SM CODE	CLK_SM_3
Description	Internal clock cross-measure
Ownership	End user
Detailed implementation	This method is implemented using TIM14 capabilities to be fed by the 32 kHz RTC clock or an external clock source (if available). TIM14 counter progresses are 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 needed
Multiple-fault protection	CPUM0_SM_1, CPUM4_SM_1: control flow monitoring in application software

SM CODE	CLK_SM_3
	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.

3.6.25 Independent watchdog (IWDG), system window watchdog (WWDG)

Table 103. WDG_SM_0

SM CODE	WDG_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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 104. WDG_SM_1

SM CODE	WDG_SM_1
Description	Software test for IWDG/WWDG watchdogs at startup
Ownership	End user
Detailed implementation	This safety mechanism ensures the right functionality of the internal watchdogs in use. At startup, the software test programs the watchdog with the required expiration timeout, stores a specific non-trivial code in SRAM and waits for the reset signal. After the watchdog reset, the software understands that the watchdog has correctly triggered, and does not execute the procedure again. IWDG and WWDG must be individually tested.
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 (see below)
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical 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 can be associated to IEC61508 concept of “proof test” and so it cannot be accounted for a diagnostic coverage contribution during operating time.

3.6.26 Clock recovery system (CRS)

No safety mechanisms are defined for CRS because of the consequences of CoU_8 (refer to Table 121. List of safety recommendations). CRS inactivation is guaranteed by Section 3.6.33 Disable and periodic cross-check of unintentional activation of unused peripherals.

3.6.27 Debug support (DBG)

Table 105. DBG_SM_0

SM CODE	DBG_SM_0
Description	Independent watchdog
Ownership	ST
Detailed implementation	The debug unintentional activation due to hardware random fault results in the massive disturbance of CPU operations, leading to intervention of the independent watchdog or alternately, the other system watchdog (WWGDG) or an external one.
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval).
Addressed fault model	Permanent
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_1, CPUM4_SM_1: control flow monitoring in application software
Recommendations and known limitations	None

3.6.28 Cyclic redundancy-check module (CRC)

Table 106. 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 operation using the module to recompute an expected signature.
Error reporting	Depends on implementation.
Fault detection time	Depends on implementation.
Addressed fault model	Permanent and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software
Recommendations and known limitations	None

3.6.29 System configuration controller (SYSCFG)

Table 107. SYSCFG_SM_0

SM CODE	SYSCFG_SM_0
Description	Periodical 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 the activation of hardware diagnostics and error reporting chain related features. Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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	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 108. DIAG_SM_0

SM CODE	DIAG_SM_0
Description	Periodical read-back of hardware diagnostic configuration registers
Ownership	End user
Detailed implementation	In STM32WL5x dual-core, several hardware-based safety mechanisms are available (they are reported in this manual with the wording ownership = ST). This method must be applied to any configuration register related to diagnostic measure operations, including error reporting. The 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 Table 41. 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	

3.6.30 True random number generator (RNG)

Table 109. RNG_SM_0

SM CODE	RNG_SM_0
Description	Periodical read-back of RNG configuration register RNG_CR
Ownership	End user
Detailed implementation	This method must be applied to RNG configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. NVIC_SM_0.
Fault detection time	
Addressed fault model	
Dependency on device configuration	None
Initialization	Refer to Table 41. NVIC_SM_0.
Periodicity	
Test for the diagnostic	
Multiple-fault protection	
Recommendations and known limitations	

Table 110. RNG_SM_1

SM CODE	RNG_SM_1
Description	RNG module entropy on-line tests
Ownership	ST and end user
Detailed implementation	The RNG module includes 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 in 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 and transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self test software
Recommendations and known limitations	NA

3.6.31 Voltage reference buffer (VREFBUF)

Table 111. VREF_SM_0

SM CODE	VREF_SM_0
Description	Periodical 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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. NVIC_SM_0.
Fault detection time	
Addressed fault model	
Dependency on device configuration	
Initialization	
Periodicity	
Test for the diagnostic	
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self test software
Recommendations and known limitations	Refer to Table 41. NVIC_SM_0.

Table 112. 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	NA
Multiple-fault protection	CPUM0_SM_0, CPUM4_SM_0: periodical core self test software
Recommendations and known limitations	Overlaps with ADC_SM_3 are possible.

3.6.32 Comparator (COMP)

Table 113. COMP_SM_0

SM CODE	COMP_SM_0
Description	Periodical 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 3.6.7 .
Error reporting	Refer to Table 41. 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 114. 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	
Addressed fault model	Permanent/transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	On demand
Test for the diagnostic	NA
Multiple-fault protection	CPU_SM_0: periodic core self-test software
Recommendations and known limitations	This method is not compatible with window comparator feature.

Table 115. **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	
Addressed fault model	Permanent
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	Periodic
Test for the diagnostic	NA
Multiple-fault protection	CPU_SM_0: periodic core self-test software
Recommendations and known limitations	None

Table 116. **COMP_SM_3**

SM CODE	COMP_SM_3
Description	Multiple acquisition by the application software
Ownership	End user
Detailed implementation	This method requires that the 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	
Addressed fault model	Transient
Dependency on device configuration	None
Initialization	Depends on implementation.
Periodicity	On demand
Test for the diagnostic	NA
Multiple-fault protection	CPU_SM_0: periodic core self-test software
Recommendations and known limitations	This recommendation is most probably satisfied by design on end-user application. The multiple acquisition is a common technique in industrial applications facing electromagnetic interference on sensor lines.

Table 117. **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	
Addressed fault model	None (fault avoidance)
Dependency on device configuration	None
Initialization	The 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	
Recommendations and known limitations	This method does not addresses comparator configuration changes due to soft errors.

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

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

Table 118. **FFI_SM_0**

SM CODE	FFI_SM_0
Description	Unused peripherals disable
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, the 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	NA
Fault detection time	
Addressed fault model	
Dependency on device configuration	None
Initialization	NA
Periodicity	Startup
Test for the diagnostic	Not needed
Multiple-fault protection	FFI_SM_1: periodical read-back of interference avoidance registers
Recommendations and known limitations	None

Table 119. **FFI_SM_1**

SM CODE	FFI_SM_1
Description	Periodical read-back of interference avoidance registers

SM CODE	FFI_SM_1
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 3.6.7 EXTI controller.
Error reporting	Refer to Table 41. 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	

3.6.34 System

Table 120. DUAL_SM_0

SM CODE	DUAL_SM_0
Description	Cross-check between two STM32 MCUs
Ownership	End user
Detailed implementation	This method is implemented in the spirit of technique described in IEC61508-7, A.3.5 “Reciprocal comparison by software”, which is rated in IEC61508-2 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 of each software-implemented safety mechanisms executed on periodical basis on both MCUs (for example CPUM0_SM_0, CPUM4_SM_0) • Software routines devoted to data exchange/comparison must be logically separated from the software implementing the safety function(s). • Systematic capability of data exchange/comparison software 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 (1) 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	Depends on implementation.
Fault detection time	Permanent and transient
Addressed fault model	None
Dependency on device configuration	Depends on implementation.
Initialization	Periodic

SM CODE	DUAL_SM_0
Periodicity	NA
Test for the diagnostic	CPUM0_SM_0, CPUM4_SM_0: periodical core self-test software (individually executed on both processing units)
Multiple-fault protection	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 . Anyway, it is included here only for its use in such an architecture. This method can provide additional safety margin for systems needing further protection against fault accumulation. Because this method can be a potential source of common cause failure between the two 1oo2 channels (in case of incorrect implementation), the end user is recommended to carefully follow above the reported guidelines (box “Detailed implementation”).
Recommendations and known limitations	None

1. *It is defined here “intermediate result” the value of each variable able to directly influence the final individual channel output. To give some examples:*
   - *If the final output is a value resulting from some computation (for example a PWM rate), “intermediate results” are the values of each variable included in such computation.*
   - *If the final output is the result of a decision (for example GPIO value decided on the basis of the comparison between values), “intermediate results” are the values of each variable involved in such decision.*

3.7 Conditions of use

Table 121 provides a summary of the safety concept recommendations reported in Section 3.6 . 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:

- ++ = Highly recommended because considered, in this safety manual, as required to allow the implementation of safety functions up to SIL2 on a single device. Missing implementation must be supported by adequate arguments under end-user responsibility (refer to Section 4.1.1 for guidance).
- + = Recommended as additional safety measure, but not considered in this safety manual for the computation of safety metrics. End users can skip the implementation in case it is in contradiction with functional requirements or overlapped by another mechanism marked as “++”.
- o = Optional, not needed or related to specific MCU configuration

The “X” marker in the “Perm” and “Trans” 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 .

Table 121. List of safety recommendations

STM32WL5x dual-core function	Diagnostic	Description	Rank	Perm	Trans	Separation
Arm® Cortex®-M0+	CPUM0_SM_0	Periodical software test addressing permanent faults in Arm® Cortex®-M0+ core	++	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	MPU – Memory protection unit	++	X	X	X
	MPUM0_SM_0	Periodical read-back of configuration registers	++	X	X	X
	MPUM0_SM_1	MPU software tests	++	X	-	X

STM32WL5x dual-core function	Diagnostic	Description	Rank	Perm	Trans	Separation
Arm® Cortex®-M4	CPUM4_SM_0	Periodical software test addressing permanent faults in Arm® Cortex®-M4 core	++	X	-	X
	CPUM4_SM_1	Control flow monitoring in application software	++	X	X	-
	CPUM4_SM_2	Double computation in the application software	++	-	X	-
	CPUM4_SM_3	Arm® Cortex®-M0+ HardFault exceptions	++	X	X	-
	CPUM4_SM_4	Stack hardening for application software	++	X	X	X
	CPUM4_SM_7	MPU – Memory protection unit	++	X	X	X
	MPUM4_SM_0	Periodical read-back of configuration registers	++	X	X	X
	MPUM4_SM_1	MPU software tests	++	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
Embedded Flash memory	FLASH_SM_0	Periodical software test for Flash memory	++(1)	X	-	-
	FLASH_SM_1	Control flow monitoring in application software	++	X	X	-
	FLASH_SM_2	Arm® Cortex®- and Cortex®-M4 HardFault exceptions	++	X	X	-
	FLASH_SM_3	Option-byte write protection	++	-	-	-
	FLASH_SM_4	Static data encapsulation	+	X	X	-
	FLASH_SM_6	Flash unused area filling code	+	-	-	-
	FLASH_SM_7	ECC on Flash memory	++	X	X	-
	FLASH_SM_8	Read/write/proprietary code protection	+	-	-	-
	FLASH_SM_9	Periodic test by software for Flash memory address decoder	++	X	-	-
Embedded SRAM	RAM_SM_0	Periodical software test for SRAM	++	X	-	-
	RAM_SM_2	Stack hardening for application software	++	X	X	-
	RAM_SM_3	Information redundancy for system variables in application software	++	X	X	-
	RAM_SM_4	Control flow monitoring in application software	o (2)	X	X	-
	RAM_SM_5	Periodical integrity test for application software in RAM	o(2)	X	X	-
	RAM_SM_6	Readout protection (RDP), write protection (WRP)	+	-	-	-
	RAM_SM_8	Periodic test by software for SRAM address decoder	++	X	-	-
System architecture	BUS_SM_0	Periodical software test for interconnections	++	X	-	-
	BUS_SM_1	Information redundancy in intra-chip data exchanges	++	X	X	-
EXTI controller	NVIC_SM_0	Periodical read-back of configuration registers	++	X	X	-
	NVIC_SM_1	Expected and unexpected interrupt check by application software	++	X	X	-
GTZC	GTZC_SM_0	Periodical read-back of configuration registers	++	X	X	-
	GTZC_SM_1	GTZC illegal access detection	++	X	X	-
DMA, DMAMUX	DMA_SM_0	Periodical 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	-

STM32WL5x dual-core function	Diagnostic	Description	Rank	Perm	Trans	Separation
DMA, DMAMUX	DMA_SM_3	Periodical software test for DMA	++	X	-	-
	DMA_SM_4	DMA transaction awareness	++	X	X	-
HSEM	HSEM_SM_0	Periodical read-back of configuration registers	++	X	X	-
	HSEM_SM_1	Control flow monitoring for concurrent tasks	++	X	X	-
IPCC	IPCC_SM_0	Periodical read-back of configuration registers	++	X	X	X
	IPCC_SM_1	End-to-end protection for inter-CPU processor communications	++	X	X	X
USART, LPUART	UART_SM_0	Periodical 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 operation	++	X	X	-
I2C	IIC_SM_0	Periodical 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 operation	+	X	X	-
SPI	SPI_SM_0	Periodical 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 operation	+	X	X	-
ADC	ADC_SM_0	Periodical 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	Periodical software test for ADC	++	X	-	-
	ADC_SM_4	1oo2 scheme for ADC inputs	+	X	X	-
DAC	DAC_SM_0	Periodical read-back of configuration registers	++	X	X	-
	DAC_SM_1	DAC output loopback on ADC channel	++	X	X	-
AES	AES_SM_0	Periodical read-back of configuration registers	++	X	X	-
	AES_SM_1	Encryption/decryption collateral detection	++	X	X	-
	AES_SM_2	Information redundancy techniques on messages, including end-to-end protection	++	X	X	-
Basic timers (TIM6/7)	GTIM_SM_0	Periodical read-back of configuration registers	++	X	X	-
	GTIM_SM_1	1oo2 for counting timers	++	X	X	-
Advanced, general and low-power timers (TIM1/2/16/17)	ATIM_SM_0	Periodical 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 PWM outputs	++	X	X	-
	ATIM_SM_4	Lock bit protection for timers	+	-	-	-
GPIO	GPIO_SM_0	Periodical read-back of configuration registers	++	X	X	-

STM32WL5x dual-core function	Diagnostic	Description	Rank	Perm	Trans	Separation
GPIO	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	+	-	-	-
RTC	RTC_SM_0	Periodical 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 or event capture	o	X	X	-
TAMP	TAMP_SM_0	Periodical read-back of configuration registers	++	X	X	-
Power control	VSUP_SM_0	Periodical 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	CLK_SM_0	Periodical read-back of configuration registers	++	X	X	-
	CLK_SM_1	Clock security system (CSS)	++	X	-	-
	CLK_SM_2	External watchdog	++	X	-	-
	CLK_SM_3	Internal clock cross-measure	+	X	-	-
IWDG/WWDG	WDG_SM_0	Periodical read-back of configuration registers	++	X	X	-
	WDG_SM_1	Software test for IWDG/WWDG watchdogs at startup	o	X	-	-
Debug	DBG_SM_0	Independent watchdog	++	X	X	-
CRC	CRC_SM_0	CRC self-coverage	++	X	X	-
System or peripheral control	LOCK_SM_0	Lock mechanism for configuration options	+	-	-	-
	SYSCFG_SM_0	Periodical read-back of configuration registers	++	X	X	-
	DIAG_SM_0	Periodical read-back of hardware diagnostic configuration registers	++	X	X	-
RNG	RNG_SM_0	Periodical read-back of configuration register	++	X	X	-
	RNG_SM_1	RNG module entropy on-line tests	++	X	-	-
VREFBUF	VREF_SM_0	Periodical read-back of VREFBUF system configuration registers	++	X	X	-
	VREF_SM_1	VREF cross-check by ADC reading	+	X	-	-
Part separation (no interference)	FFI_SM_0	Unused peripherals disable	++	-	-	-
	FFI_SM_1	Periodical read-back of interference avoidance registers	++	-	-	X
Arm® Cortex®-M0+ and Cortex®-M4	CoU_1	The reset condition of Arm® Cortex®-M0+ and Cortex®-M4 must be compatible as valid safe state at system level.	++	-	-	-
Debug	CoU_2	STM32WL5x dual-core debug features must not be used in safety function(s) implementation.	++	-	-	-
Supply system for Arm® Cortex®-M0+ and Cortex®-M4	CoU_3	Low-power mode state must not be used in safety function(s) implementation.	++	-	-	-

STM32WL5x dual-core function	Diagnostic	Description	Rank	Perm	Trans	Separation
STM32WL5x dual-core peripherals	CoU_4	The end user must implement the required combination of safety mechanism/CoUs for each STM32 peripheral used in safety function(s) implementation.	++	X	X	-
Flash memory subsystem	CoU_5	During the Flash memory bank mass erase and reprogramming, there must not be safety function(s) executed by the STM32WL5x dual-core.	++	-	-	-
	CoU_6	On-field application software live update by the dual-Flash system must include the execution of code/data integrity check by methods like FLASH_SM_0.	++	X	X	-
CPUs subsystems	CoU_7	In case of multiple safety function implementations, methods to guarantee their mutual independence must include MPU use.	++	-	-	-
CRS	CoU_8	CRS features must not be used in safety function implementation.	++	-	-	-
Embedded SRAM	CoU_9	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. (3)	++	-	-	-
	CoU_10	RAM space used by each device CPU for safety function(4) 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	The MPU of each device CPU must be used to enforce the implementation of CoU_10, CoU_11.	++	-	-	X
Watchdogs	CoU_13	The external watchdog must supervise the correct behavior of both CPUs. Communication schemes are ruled by CoU_14.	++	X	X	X
	CoU_14	CPU communications with the external watchdog must be managed in one of the following ways: • Both CPUs can communicate with the external watchdog. The watchdog must be able to identify which CPU sends the message. • Only one CPU drives the external watchdog and acts as supervisor for the other CPU. The first CPU must be able to request a reset in case the second CPU is unresponsive or reporting a failure detection.	++	-	-	X
	CoU_15	Each of the two internal watchdogs IWDG and WWDG must be statically allocated to only one CPU.(5).	++	-	-	X
	CoU_16	External and internal watchdog management software routines (implementing requirements of CoU_13,14,15) 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

STM32WL5x dual-core function	Diagnostic	Description	Rank	Perm	Trans	Separation
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
1oo2 architecture	CoU_18	In 1oo2 architecture, the PEv must keep the final system in safe state after the power-up until both STM32WL5x dual-core CPUs perform 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. *Ranked ++ because considered as mitigation measure for multiple-fault protection for FLASH_SM_7 (refer to related description) .*
2. *Must be considered ranked as "++" if the application software is executed on RAM.*
3. *To provide an example, safety recommendation 'CPUM7_SM_3 Arm Cortex-M7 HardFault exceptions' cannot be considered correctly satisfied if related exception handler is not implemented in software.*
4. *Reminder: 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.*
5. *Therefore, trigger action on each internal watchdog must be possible for 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 a safety point of view by a failure on these modules. Because these modules are used by every end-user application, related methods or safety mechanism are mainly conceived to be application-independent. For STM32WL5x dual-core microcontroller, system critical modules are: the two CPUs, RCC, PWR, clocks, bus matrix and interconnect, Flash and RAM memories (including their interfaces).
- Peripheral modules. Such modules cannot be used by the end-user application, or they can 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 IEC61508. The accuracy of results obtained are guaranteed by three factors:

- STMicroelectronics methodology flow strict adherence to IEC61508 requirements and prescriptions
- the use, during the analysis, of detailed and reliable information on microcontroller design
- the use 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 failure modes is maintained in the related FMEA document, provided on demand by local STMicroelectronics sales office.

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

Table 122. Overall achievable safety integrity levels

Number of devices used	Safety architecture	Target	Safety analysis result
1	1oo1/1oo1D	*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. *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 Safety requirement assumptions).*

The resulting relative safety metrics (diagnostic coverage - 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, due to:

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

The FMEDA snapshot 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 the FMEDA computation sheet is needed, early contact the local STMicroelectronics sales representative, in order to receive information on expected delivery dates for specific device target part number.

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

4.1.1 Safety analysis result customization

The safety analysis executed for STM32WL5x dual-core devices documented in this safety manual, consider 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 IEC61508-4, 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 only in one of the following conditions:

- The part is not used at all (disabled).
- 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 ASR6 (see Section 3.3.1), under end-user entire responsibility. As any other derogation from safety requirements included in this document, it is end-user responsibility to provide consistent rationales and evidences that the function does not bring additional risks, by following the procedure described in this section. 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 is a useless effort for the end user. The safety analysis results can therefore be customized.

The 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 .
- Fulfilling the general condition for the mitigation of intra-MCU interferences (see Section 4.1.2).

For a non-safety-related part, the end user can:

- exclude the part from computing metrics to report in FMEDA
- not implement safety mechanisms as listed in Table 1

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 module 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 Safety analysis result customization.

End user must implement the safety mechanisms listed in Table 123 (implementation details in Hardware and software diagnostics) regardless any evaluation of their contribution to safety metrics.

Table 123. List of general requirements for FFI

Diagnostic	Description
FFI_SM_0	Unused peripheral disable
FFI_SM_1	Periodical read-back of interference avoidance registers
BUS_SM_0	Periodical software test for interconnections
NVIC_SM_0	Periodical read-back of configuration registers
NVIC_SM_1	Expected and unexpected interrupt check by application software
DMA_SM_0	Periodical read-back of configuration registers
DMA_SM_2	Information redundancy including sender or receiver identifier on data packet transferred via DMA (1)
DMA_SM_4	DMA transactions awareness(1)
GPIO_SM_0	Periodical read-back of configuration registers

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

4.1.3 Notes on multiple-fault scenario

According to the requirements of IEC61508, 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 required by Annex E.1, item i - is not required. Note that, in the case of 1oo2 safety architecture implementation, the 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 referred safety mechanisms 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 external 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.

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 3.6.23 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 external 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 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 3.6.24 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 3.6.9 Direct memory access controllers (DMA1/2 and 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 device parts. The following safety mechanism mitigates this potential effect:

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 (see VSUP_SM_3) .

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 clause-by-clause compliance to IEC 61508
- STMicroelectronics' internal fault injection campaign database including tool configuration and settings, fault injection logs and results

As these materials 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.

Revision history

Table 124. **Document revision history**

Date	Revision	Changes
23-May-2022	1	Initial release.

Glossary

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

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 STM32WL5x dual-core silicon products

DMA direct memory access

DTI diagnostic test interval

ECM engine control module

ECU electronic control unit

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

PDS(SR) safety-related power drive system

PEc programmable electronics - core

PEd programmable electronics - diagnostic

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

SILCL safety integrity level claim limit

SRCF safety-related control function

SRECS safety-related electrical control systems

SRP/CS safety-related parts of machinery control systems

Contents

List of tables

List of figures

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics International NV and its affiliates (“ST”) reserve the right to make changes corrections, enhancements, modifications, and improvements to ST products and/or to this document any time without notice.

This document is provided solely for the purpose of obtaining general information relating to an ST product. Accordingly, you hereby agree to make use of this document solely for the purpose of obtaining general information relating to the ST product. You further acknowledge and agree that this document may not be used in or in connection with any legal or administrative proceeding in any court, arbitration, agency, commission or other tribunal or in connection with any action, cause of action, litigation, claim, allegation, demand or dispute of any kind. You further acknowledge and agree that this document shall not be construed as an admission, acknowledgment or evidence of any kind, including, without limitation, as to the liability, fault or responsibility whatsoever of ST or any of its affiliates, or as to the accuracy or validity of the information contained herein, or concerning any alleged product issue, failure, or defect. ST does not promise that this document is accurate or error free and specifically disclaims all warranties, express or implied, as to the accuracy of the information contained herein. Accordingly, you agree that in no event will ST or its affiliates be liable to you for any direct, indirect, consequential, exemplary, incidental, punitive, or other damages, including lost profits, arising from or relating to your reliance upon or use of this document.

Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment, including, without limitation, the warranty provisions thereunder.

In that respect, note that ST products are not designed for use in some specific applications or environments described in above mentioned terms and conditions.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

Information furnished is believed to be accurate and reliable. However, ST assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.