STM32F0 Series safety manual

Introduction

This document must be read along with the technical documentation such as reference manual(s) and datasheets for the STM32F0 Series microcontroller devices, available on www.st.com.

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

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

The document is written in compliance with IEC 61508, and it provides information relative to other functional safety standards.

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

1 About this document

1.1 Purpose and scope

This document describes how to use Arm®Cortex®-M0 -based STM32F0 Series microcontroller unit (MCU) devices (further also referred to as Device(s)) in the context of a safety-related system, specifying the user's responsibilities for installation and operation, in order to reach the desired safety integrity level.

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

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

Note: 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 other functional safety standards considered in this manual are:

- ISO 13849-1:2015, ISO13849-2:2012
- IEC 62061:2005+AMD1:2012+AMD2:2015
- IEC 61800-5-2:2016

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) a functional specification of the functions capable of being performed	3
D2.1 b) identification of the hardware and/or software configuration of the Compliant item	3.2
D2.1 c) constraints on the use of *Compliant item* or assumptions on which analysis of the behavior or failure rates of the item are based	3.2
D2.2 a) the failure modes of *Compliant item* due to random hardware failures, that result in a failure of the function and that are not detected by diagnostics internal to *Compliant item*;	3.7
D2.2 b) for every failure mode in a), an estimated failure rate;	
D2.2 c) the failure modes of *Compliant item* due to random hardware failures, that result in a failure of the function and that are detected by diagnostics internal to *Compliant item*;	
D2.2 d) the failure modes of the diagnostics, internal to *Compliant item* due to random hardware failures, that result in a failure of the diagnostics to detect failures of the function;	
D2.2 e) for every failure mode in c) and d), the estimated failure rate;	
D2.2 f) for every failure mode in c) that is detected by diagnostics internal to *Compliant item*, the diagnostic test interval;	3.2.2
D2.2 g) for every failure mode in c) the outputs of *Compliant item* initiated by the internal diagnostics;	3.6
D2.2 h) any periodic proof test and/or maintenance requirements;	3.7
D2.2 i) for those failure modes, in respect of a specified function, that are capable of being detected by external diagnostics, sufficient information must be provided to facilitate the development of an external diagnostics capability.	
D2.2 j) the hardware fault tolerance;	3
D2.2 k) the classification as type A or type B of that part of *Compliant item* that provides the function (see 7.4.4.1.2 and 7.4.4.1.3);	

1.3 Reference documents

[1] AN5075: Results of FMEA on STM32F0 Series microcontrollers.

[2] AN5076: FMEDA snapshots for STM32F0 series microcontrollers.

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 STM32F0 Series safety architecture.

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 *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

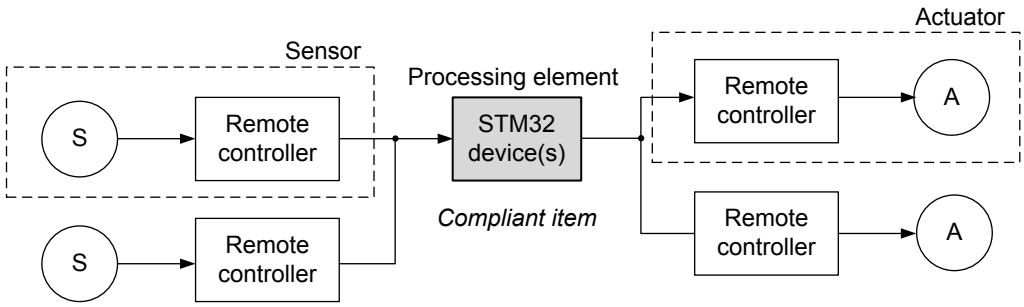
This section defines the *Compliant item* term and provides information on its usage in different safety architecture schemes.

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 *Compliant item*, like the external HW components needed to guarantee either the functionality of *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 IEC61508-4, 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)
- 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 Reference safety architectures - 1oo2. The role of the WDTe and VMONe external processes is clarified under Section 3.6 Hardware and software diagnostics:

- WDTe: refer to External watchdog – CPU_SM_5 and Control flow monitoring in Application software – CPU_SM_1,
- VMONe: refer to Supply voltage internal monitoring (PVD) – VSUP_SM_1 and System-level power supply management - VSUP_SM_5.

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

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

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

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

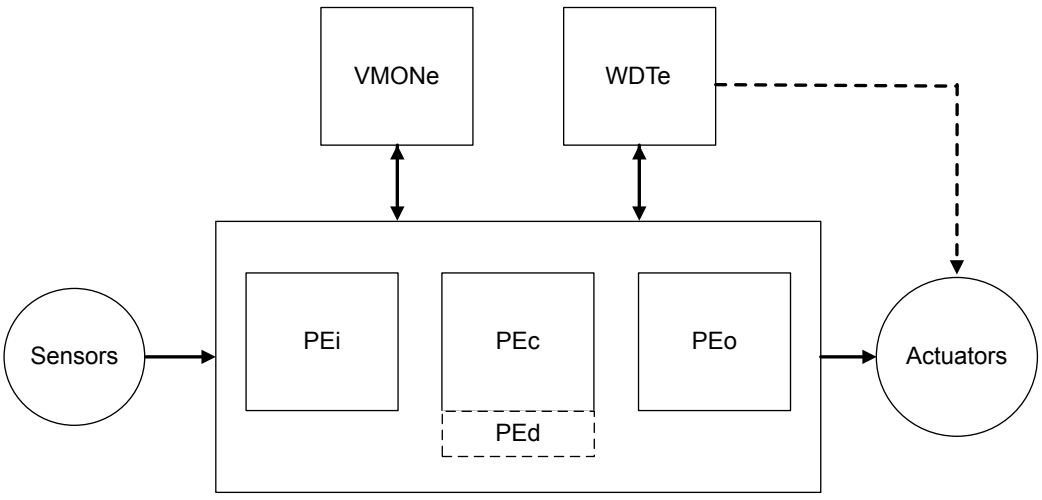
Two main safety architectures are identified: 1oo1 (using one *Device*) and 1oo2 (using two *Devices*).

3.2.3 Reference safety architectures - 1oo1

1oo1 reference architecture (Figure 3) ensures safety integrity of *Compliant item* through combining *Device* internal processes (implemented safety mechanisms) with external processes WDTe and VMONe.

1oo1 reference architecture targets safety integrity level (SIL) *SIL2*.

Figure 3. 1oo1 reference architecture

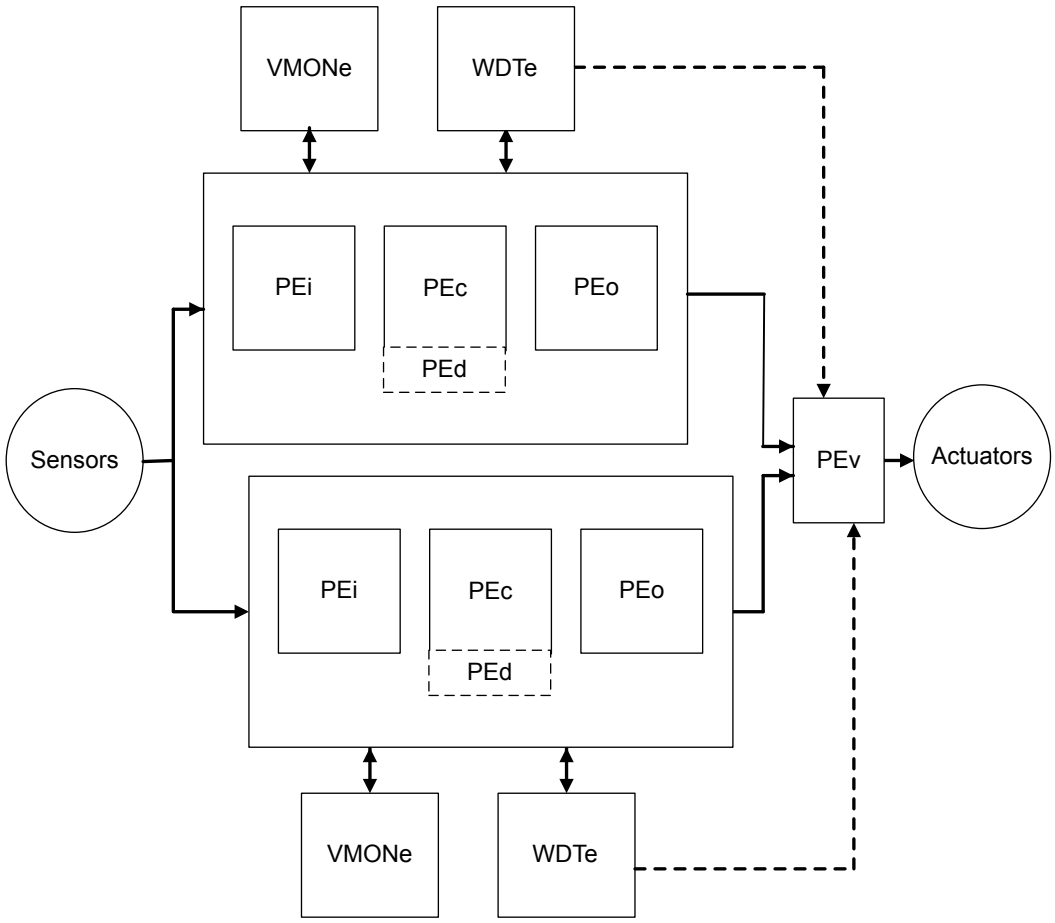


3.2.4 Reference safety architectures - 1oo2

1oo2 reference architecture (Figure 4) contains two separate channels, either 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. 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) should be implemented in order to avoid huge impact of common-cause failures (refer to Section 4.2 Analysis of dependent failures). However, β and βD parameters computation is required.

1oo2 reference architecture targets *SIL3*.

Figure 4. 1oo2 reference architecture



3.3 Safety analysis assumptions

This section collects all assumptions made during the safety analysis of *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*'s responsibility to check the compliance of the final application with these assumptions.

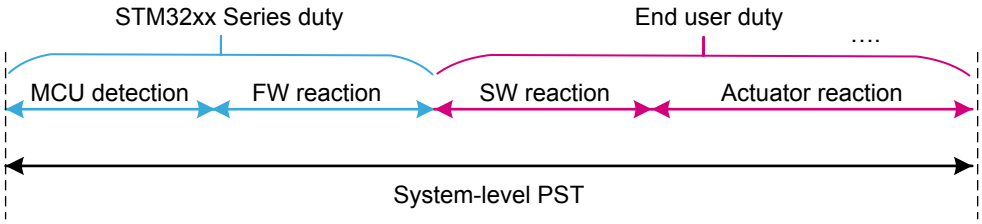
ASR1: *Compliant item* can be used to implement four kinds of safety function modes of operation according to part 4,3.5.16:

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

ASR2: *Compliant item* is used to implement safety function(s) allowing a specific worst-case time budget (see note below) for the STM32 *MCU* to detect and react to a failure. That time corresponds to the portion of the process safety time (PST) allocated to *Device* (*STM32xx Series duty* in Figure 5) in error reaction chain at system level.

Note: The computation for time budget mainly depends on the execution speed for periodic tests implemented by software. Such duration might 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 5. Allocation and target for STM32 *PST*



ASR3: *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.

ASR4: It is assumed that only one safety function is performed or if many, all functions are classified with the same *SIL* and therefore they are not distinguishable in terms of their safety requirements.

ASR5: In case of multiple safety function implementations, it is assumed that *End user* is responsible to duly ensure their mutual independence.

ASR6: It is assumed that there are no *non-safety-related* functions implemented in *Application software*, coexisting with safety functions.

ASR7: It is assumed that the implemented safety function(s) does (do) not depend on transition of *Device* to and from a low-power state.

ASR8: The local safe state of *Compliant item* is the one in which either:

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

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

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

Table 2. SS1 and SS2 safe state details

Safe state	Condition	*Compliant item* action	System transition to safe state – 1oo1 architecture	System transition to safe state – 1oo2 architecture
SS1	*Application software* is informed by the presence of a fault and a reaction by *Application software* itself is possible.	Fault reporting to *Application software*	*Application software* drives the overall system in its safe state	*Application software* in one of the two channels drives the overall system in its safe state
SS2	*Application software* cannot be informed by the presence of a fault or *Application software* is not able to execute a reaction.	Reset signal issued by WDTe	WDTe drives the overall system in its safe state (“safe shut-down”) (1)	PEv drives the overall system in its safe state

1. *Safe state achievement intended here is compliant to Note on IEC 61508-2, 7.4.8.1*

ASR9: It is assumed that the safe state defined at system level by *End user* is compatible with the assumed local safe state (SS1, SS2) for *Compliant item*.

ASR10: *Compliant item* is assumed to be analyzed according to routes 1H and 1S of IEC 61508-2.

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

ASR11: *Compliant item* is assumed to be regarded as type B, as per IEC 61508:2, 7.4.4.1.2.

3.4 Electrical specifications and environment limits

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

- absolute maximum rating
- capacity
- operating conditions

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

3.5 Systematic safety integrity

According to the requirements of IEC 61508 -2, 7.4.2.2, the *Route 1S* is considered in the safety analysis of *Device*(s). As clearly authorized by IEC61508-2, 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 architecture of *Device*, and that this document is used in conjunction with the related *Device* datasheet, user manual and reference information. To avoid inconsistency and redundancy, this document does not report device functional details. In the following descriptions, the words *safety mechanism*, *method*, and *requirement* are used as synonyms.

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

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

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

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

SM CODE	Unique safety mechanism code/identifier used also in *FMEA* document. Identifiers use the scheme *mmm_SM_x* where *mmm* is a 3- or 4-letter module (function, peripheral) short name, and *x* is a number. It is possible that the numbering is not sequential (although usually incremental) and/or that the module short name is different from that used in other documents.
Description	Short mnemonic description
Ownership	ST: method is available on silicon. *End user*: method must be implemented by *End user* through *Application software* modification, hardware solutions, or both.
Detailed implementation	Detailed implementation sometimes including notes about the safety concept behind the introduction of the safety mechanism.
Error reporting	Describes how the fault detection is reported to *Application software*.
Fault detection time	Time that the safety mechanism needs to detect the hardware failure.
Addressed fault model	Reports fault model(s) addressed by the diagnostic (permanent, transient, or both), and other information: • If ranked for *Fault avoidance*: method contributes to lower the probability of occurrence of a failure • If ranked for *Systematic*: method is conceived to mitigate systematic errors (bugs) in *Application software* design
Dependency on *Device* configuration	Reports if safety mechanism implementation or characteristics change among different *Device* part numbers.
Initialization	Specific operation to be executed to activate the contribution of the safety mechanism
Periodicity	Continuous : safety mechanism is active in continuous mode. Periodic: safety mechanism is executed periodically(1). On-demand: safety mechanism is activated in correspondence to a specified event (for instance, reception of a data message). Startup: safety mechanism 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. If no specific procedure applies (as for the majority of safety mechanisms), the field indicates *Not applicable*.
Multiple-fault protection	Reports the safety mechanism(s) associated in order to correctly manage a multiple-fault scenario (refer to Section 4.1.3 Notes on multiple-fault scenario).
Recommendations and known limitations	Additional recommendations or limitations (if any) not reported in other fields.

1. *In CM systems, safety mechanism can be accounted for diagnostic coverage contribution only if it is executed at least once per PST. For LD and HD systems, constraints from IEC61508-2, 7.4.5.3 must be applied.*

3.6.1 Arm® Cortex®-M0 CPU

Table 3. CPU_SM_0

SM CODE	CPU_SM_0
Description	Periodic core self-test software for Arm®Cortex®-M0 *CPU*.
Ownership	*End user* or ST

SM CODE	CPU_SM_0
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 to 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 STM32F0 Series safety concept. Hardware integrity of the *CPU* is a key factor, given that the defined diagnostics for *MCU* peripherals are to major part software-based.

Table 4. CPU_SM_1

SM CODE	CPU_SM_1
Description	Control flow monitoring in *Application software*
Ownership	*End user*
Detailed implementation	A significant part of the failure distribution of *CPU* core for permanent faults is related to failure modes directly related to program counter loss of control or hang-up. Due to their intrinsic nature, such failure modes are not addressed by a standard software test method like SM_CPU_0. Therefore, it is necessary to implement a run-time control of *Application software* flow in order to monitor and detect deviation from the expected behavior due to such faults. Linking this mechanism to watchdog firing assures that severe loss of control (or, in the worst case, a program counter hang-up) is detected. The guidelines for the implementation of the method are the following: • Different internal states of *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 *Application software* is implemented. • Transition through all expected states during the normal *Application software* program loop is checked. • A function in charge of triggering the system watchdog is implemented in order to constrain the triggering (preventing the issue of *CPU* reset by watchdog) also to the correct execution of the above-described method for program flow monitoring. The use of window feature available on internal window watchdog (WWDG) is recommended. • The use of the independent watchdog (IWDG), or an external one, helps to implement a more robust control flow mechanism fed by a different clock source. In any case, safety metrics do not depend on the kind of watchdog in use (the adoption of independent or external watchdog contributes to the mitigation of dependent failures, see Section 4.2.2 Clock).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation

SM CODE	CPU_SM_1
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 5. CPU_SM_2

SM CODE	CPU_SM_2
Description	Double computation in *Application software*
Ownership	*End user*
Detailed implementation	A timing redundancy for safety-related computation is considered to detect transient faults affecting the Arm®Cortex®-M0 *CPU* subparts devoted to mathematical computations and data access. The guidelines for the implementation of the method are the following: • The requirement needs be applied only to safety-relevant computation, which in case of wrong result could interfere with the system safety functions. Such computation must be therefore carefully identified in the original *Application software* source code • Both mathematical operation and comparison are intended as computation. • The redundant computation for mathematical computation is implemented by using copies of the original data for second computation, and by using an equivalent formula if possible
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	*End user* is responsible to carefully avoid that the intervention of optimization features of the used compiler removes timing redundancies introduced according to this condition of use.

Table 6. CPU_SM_3

SM CODE	CPU_SM_3
Description	Arm®Cortex®-M0 HardFault exceptions
Ownership	ST
Detailed implementation	HardFault exception raise is an intrinsic safety mechanism implemented in Arm®Cortex®-M0 core, mainly dedicated to intercept systematic faults due to software limitations or error in software design (causing for example execution of undefined operations, unaligned address access). This safety mechanism is also able to detect hardware random faults inside the *CPU* bringing to such described abnormal operations.
Error reporting	High-priority interrupt event
Fault detection time	Depends on implementation. Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None

SM CODE	CPU_SM_3
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 optional.
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 7. CPU_SM_4

SM CODE	CPU_SM_4
Description	Stack hardening for *Application software*
Ownership	*End user*
Detailed implementation	The stack hardening method is required to address faults (mainly transient) affecting CPU register bank. This method is based on source code modification, introducing information redundancy in register-passed information to called functions. The guidelines for the implementation of the method are the following: • To pass also a redundant copy of the passed parameters values (possibly inverted) and to execute a coherence check in the function. • To pass also a redundant copy of the passed pointers and to execute a coherence check in the function. • For parameters that are not protected by redundancy, to implement defensive programming techniques (plausibility check of passed values). For example enumerated fields are to be checked for consistency.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic 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 *Application software* qualified for safety integrity greater or equal to SC2, optimizations are possible.

Table 8. CPU_SM_5

SM CODE	CPU_SM_5
Description	External watchdog
Ownership	*End user*

SM CODE	CPU_SM_5
Detailed implementation	Using an external watchdog linked to control flow monitoring method (refer to CPU_SM_1) addresses failure mode of program counter or control structures of *CPU*. External watchdog can be designed to be able to generate the combination of signals needed on the final system to achieve the safe state. It is recommended to carefully check the assumed requirements about system safe state reported in 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 *Device*.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	To be defined at system level (outside the scope of *Compliant item* analysis).
Multiple-fault protection	CPU_SM_1: Control flow monitoring in *Application software*
Recommendations and known limitations	In case of usage of windowed watchdog, *End user* must consider possible tolerance in *Application software* execution to avoid false error reports (affecting system availability). It is worth noting that the use of an external watchdog could be needed anyway when *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 9. CPU_SM_6

SM CODE	CPU_SM_6
Description	Independent watchdog
Ownership	ST
Detailed implementation	Using the IDWG watchdog linked to control flow monitoring method (refer to CPU_SM_1) addresses failure mode of program counter or control structures of *CPU*.
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	IWDG activation. It is recommended to use *hardware watchdog* in option byte settings (IWDG is automatically enabled after reset).
Periodicity	Continuous
Test for the diagnostic	WDG_SM_1: Software test for watchdog at startup
Multiple-fault protection	CPU_SM_1: Control flow monitoring in *Application software* WDG_SM_0: Periodic read-back of configuration registers
Recommendations and known limitations	The IWDG intervention is able to achieve a potentially “incomplete” local safe state because it can only guarantee that *CPU* is reset. No guarantee that *Application software* can be still executed to generate combinations of output signals that might be needed by the external system to achieve the final safe state. If this limitation turn out in a blocking point, *End user* must adopt CPU_SM_5.

3.6.2 System bus architecture

Table 10. BUS_SM_0

SM CODE	BUS_SM_0
Description	Periodic software test for interconnections
Ownership	*End user*
Detailed implementation	The intra-chip connection resources (Bus Matrix, AHB or APB bridges) needs to be periodically tested for permanent faults detection. Note that STM32F0 Series devices have no hardware safety mechanism to protect these structures. The test executes a connectivity test of these shared resources, including the testing of the arbitration mechanisms between peripherals. According to IEC 61508:2 Table A.8, A.7.4 the method is considered able to achieve high levels of coverage.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Implementation can be considered in large part as overlapping with the widely used *Periodic read-back of configuration registers* required for several peripherals.

Table 11. BUS_SM_1

SM CODE	BUS_SM_1
Description	Information redundancy in intra-chip data exchanges
Ownership	*End user*
Detailed implementation	This method requires to add some kind of redundancy (for example a *CRC* checksum at packet level) to each data message exchanged inside *Device*. Message integrity is verified using the checksum by *Application software*, before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Implementation can be in large part overlapping with other safety mechanisms requiring information redundancy on data messages for communication peripherals. Optimizations are therefore possible.

3.6.3 Embedded SRAM

Table 12. RAM_SM_0

SM CODE	RAM_SM_0
Description	Periodic software test for static random access memory (SRAM)
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 periodic software test on the system RAM memory. The selection of the algorithm must ensure the target SFF coverage for both the RAM cells and the address decoder. Evidences of the effectiveness of the coverage of the selected method must 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 to the test implementation design strategy chosen.
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Usage of a March test C- is recommended. Because the nature of this test can be destructive, RAM contents restore must be implemented. Possible interferences with interrupt-serving routines fired during test execution must be also considered (such routines can access to RAM invalid contents). Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario. Unused RAM section can be excluded by the testing, under *End user* responsibility on actual RAM usage by final *Application software*.

Table 13. RAM_SM_1

SM CODE	RAM_SM_1
Description	Parity on SRAM2
Ownership	ST
Detailed implementation	Internal SRAM2 is protected by additional parity bits (1 bit per byte). The parity bits are computed and stored when writing into the SRAM2.
Error reporting	Error flag SYSCFG_CFGR2 set NMI raised
Fault detection time	Parity bits are checked during a reading.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	*End user* must enable the parity check using the option bit SYSCFG_CFGR2, after the boot.
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for SRAM parity function is not available. SRAM parity-related 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*.

SM CODE	RAM_SM_1
Multiple-fault protection	DIAG_SM_0: Periodic read-back of hardware diagnostics configuration registers RAM_SM_0: Periodic software test for static random access memory (SRAM)
Recommendations and known limitations	It is advised to initialize by software the whole SRAM2 memory at *Application software* startup, to avoid getting parity errors when reading non-initialized locations. As parity protection is restricted to SRAM2, *End user* is encouraged to store all safety-related data in SRAM2 (if possible), in order to get benefit of such additional hardware-based fast diagnostic.

Table 14. RAM_SM_2

SM CODE	RAM_SM_2
Description	Stack hardening for *Application software*
Ownership	*End user*
Detailed implementation	The stack hardening method is used to enhance *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 CPU_SM_4.
Error reporting	Refer to CPU_SM_4
Fault detection time	Refer to CPU_SM_4
Addressed fault model	Refer to CPU_SM_4
Dependency on *Device* configuration	Refer to CPU_SM_4
Initialization	Refer to CPU_SM_4
Periodicity	Refer to CPU_SM_4
Test for the diagnostic	Refer to CPU_SM_4
Multiple-fault protection	Refer to CPU_SM_4
Recommendations and known limitations	Refer to CPU_SM_4

Table 15. RAM_SM_3

SM CODE	RAM_SM_3
Description	Information redundancy for safety-related variables in *Application software*
Ownership	*End user*

SM CODE	RAM_SM_3
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. The guidelines for the implementation of this method are the following: • The system variables that are safety-related (in the sense that a wrong value due to a failure in reading on the RAM affects the safety functions) are well-identified and documented. • The arithmetic computation or decision based on such variables are executed twice and the two final results are compared. • Safety-related variables are stored and updated in two redundant locations, and comparison is checked before consuming data. • Enumerated fields must use non-trivial values, checked for coherence with the same frequency as for periodically executed diagnostics (see (1) in Section 3.6 Hardware and software diagnostics). • Data vectors stored in SRAM must be protected by a encoding checksum (such as *CRC*).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Implementation of this safety method shows a partial overlap with an already foreseen method for Arm®Cortex®-M0 (CPU_SM_1); optimizations in implementing both methods are therefore possible.

Table 16. RAM_SM_4

SM CODE	RAM_SM_4
Description	Control flow monitoring in *Application software*
Ownership	*End user*
Detailed implementation	In case *End user Application software* is executed from SRAM, permanent and transient faults affecting the memory (cells and address decoder) can interfere with the program execution. The implementation of this method is required to address such failures. For more details on the implementation, refer to CPU_SM_1 description.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Needed only in case of *Application software* execution from SRAM. CPU_SM_1 correct implementation supersedes this requirement.

Table 17. **RAM_SM_5**

SM CODE	RAM_SM_5
Description	Periodic integrity test for *Application software* in RAM
Ownership	*End user*
Detailed implementation	In case *Application software* or diagnostic libraries are executed in RAM, it is needed to protect the integrity of the code itself against soft-error corruptions and related code mutations. This method must check the integrity of the stored code by checksum computation techniques, on a periodic basis. For implementation details, refer to similar method FLASH_SM_0.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according to the test implementation design strategy chosen.
Multiple-fault protection	CPU_SM_0: Periodic core self-test software CPU_SM_1: Control flow monitoring in *Application software*
Recommendations and known limitations	This method must only be implemented if *Application software* or diagnostic libraries are executed from RAM.

3.6.4 Embedded Flash memory

Table 18. **FLASH_SM_0**

SM CODE	FLASH_SM_0
Description	Periodic software test for Flash memory
Ownership	*End user* or ST
Detailed implementation	Permanent faults affecting the system Flash memory interface address decoder are addressed through a dedicated software test that checks the memory cells contents versus the expected value, using signature-based techniques. According to IEC 61508:2 Table A.5, the effective diagnostic coverage of such techniques depends on the width of the signature in relation to the block length of the information to be protected - therefore the signature computation method is to be carefully selected. Note that the simple signature method (IEC 61508:7 - A.4.2 Modified checksum) is inadequate as it only achieves a low value of coverage. The information block does not need to be addressed with this test as it is not used during normal operation (no data nor program fetch).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	Flash memory size changes according to the part number.
Initialization	Memory signatures must be stored in Flash memory as well.
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according to the test implementation design strategy chosen.
Multiple-fault protection	CPU_SM_0: Periodic core self-test software CPU_SM_1: Control flow monitoring in *Application software*

SM CODE	FLASH_SM_0
Recommendations and known limitations	This test is expected to have a relevant time duration – test integration must therefore consider the impact on *Application software* execution. The use of internal cyclic redundancy check (CRC) module is recommended. In principle direct memory access (DMA) feature for data transfer can be used. Unused Flash memory sections can be excluded from testing.

Table 19. FLASH_SM_1

SM CODE	FLASH_SM_1
Description	Control flow monitoring in *Application software*
Ownership	*End user*
Detailed implementation	Permanent and transient faults affecting the system Flash memory, memory cells and address decoder, can interfere with the access operation by the *CPU*, leading to wrong data or instruction fetches. Such failures can be detected by control flow monitoring techniques implemented in *Application software* loaded from Flash memory. For more details on the implementation, refer to description CPU_SM_1.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	CPU_SM_1 correct implementation supersedes this requirement.

Table 20. FLASH_SM_2

SM CODE	FLASH_SM_2
Description	Arm®Cortex®-M0 HardFault exceptions
Ownership	ST
Detailed implementation	Hardware random faults (both permanent and transient) affecting system Flash memory (memory cells, address decoder) can lead to wrong instruction codes fetches, and eventually to the intervention of the Arm®Cortex®-M0 HardFault exceptions. Refer to CPU_SM_3 for detailed description.
Error reporting	Refer to CPU_SM_3
Fault detection time	Refer to CPU_SM_3
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Refer to CPU_SM_3
Periodicity	Continuous
Test for the diagnostic	Refer to CPU_SM_3
Multiple-fault protection	Refer to CPU_SM_3
Recommendations and known limitations	Refer to CPU_SM_3

Table 21. **FLASH_SM_3**

SM CODE	FLASH_SM_3
Description	Option byte write protection
Ownership	ST
Detailed implementation	This safety mechanism prevents unintended writes of the option byte. The use of this method is encouraged to enhance the end application robustness with respect to systematic faults.
Error reporting	Write protection exception
Fault detection time	Not applicable
Addressed fault model	None (systematic only)
Dependency on *Device* configuration	None
Initialization	Not required (enabled by default)
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	This method addresses systematic faults in software applications. It is inefficient for hardware random faults affecting the option byte value in run time. No *DC* value is therefore associated.

Table 22. **FLASH_SM_4**

SM CODE	FLASH_SM_4
Description	Static data encapsulation
Ownership	*End user*
Detailed implementation	If static data are stored in Flash memory, encapsulation by a checksum field with encoding capability (such as *CRC*) must be implemented. Checksum validity is checked by *Application software* before static data consuming.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 23. **FLASH_SM_5**

SM CODE	FLASH_SM_5
Description	Option byte redundancy with load verification
Ownership	ST
Detailed implementation	During option byte loading after each power-on reset, the bit-wise complementarity of the option byte and its corresponding complemented option byte is verified. Mismatches are reported as an error.
Error reporting	Option byte error (OPTVERR) generation

SM CODE	FLASH_SM_5
Fault detection time	Not applicable
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	None (always enabled)
Periodicity	Startup
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 24. FLASH_SM_6

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

3.6.5 Power controller (PWR)

Table 25. VSUP_SM_0

SM CODE	VSUP_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0

SM CODE	VSUP_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 26. VSUP_SM_1

SM CODE	VSUP_SM_1
Description	Supply voltage internal monitoring (PVD)
Ownership	ST
Detailed implementation	The device features an embedded programmable voltage detector (PVD) that monitors the V_{DD} power supply and compares it to the V_{PVD} threshold. An interrupt can be generated when V_{DD} drops below the V_{PVD} threshold or when V_{DD} is higher than the V_{PVD} threshold.
Error reporting	Interrupt event generation
Fault detection time	Depends on threshold programming. Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Protection enable by the PVDE bit and the threshold setting in the Power control register (PWR_CR)
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for PVD efficiency is not available. PVD run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*.
Multiple-fault protection	DIAG_SM_0: Periodic read-back of hardware diagnostics configuration registers
Recommendations and known limitations	Internal monitoring PVD has limited capability to address failures affecting STM32F0 Series internal voltage regulator. Refer to [1] for details. Internal monitoring PVD has limited capability to address failures affecting the internal voltage regulator. Refer to *Device FMEA* for details.

Table 27. VSUP_SM_2

SM CODE	VSUP_SM_2
Description	Independent watchdog
Ownership	ST
Detailed implementation	Failures in the power supplies for digital logic (core or peripherals) may lead to alteration of *Application software* timing, which can be detected by IWDG as safety mechanism introduced to monitor *Application software* control flow. Refer to CPU_SM_1 and CPU_SM_6 for further information.
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	IWDG activation. It is recommended to use *Hardware watchdog* in Option byte settings (IWDG is automatically enabled after reset).
Periodicity	Continuous
Test for the diagnostic	Refer to CPU_SM_6.

SM CODE	VSUP_SM_2
Multiple-fault protection	CPU_SM_1: Control flow monitoring in *Application software*
Recommendations and known limitations	In specific part numbers, IWDG can be fed by a power supply independent from the one used for CPU core and main peripherals. Such diversity helps to increase the protection guaranteed by IWDG from main power supply anomalies. The adoption of an external watchdog (refer to CPU_SM_5) adds further diversity.

Table 28. VSUP_SM_3

SM CODE	VSUP_SM_3
Description	Internal temperature sensor check
Ownership	*End user*
Detailed implementation	The internal temperature sensor must be periodically tested in order to detect abnormal increase of the die temperature – hardware faults in supply voltage system may cause excessive power consumption and consequent temperature rise.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	This method also mitigates the probability of common-cause failure due to excessive temperature, affecting *Device*. Refer to the *Device* datasheet to set the threshold temperature.

Table 29. **VSUP_SM_5**

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

3.6.6 Reset and clock controller (RCC)

Table 30. **CLK_SM_0**

SM CODE	CLK_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to configuration registers for clock and reset system (refer to RCC register map). Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 31. 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 HIS • generation of related NMI
Error reporting	NMI
Fault detection time	Depends on implementation (clock frequency value)
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	CSS protection must be enabled through Clock interrupt register (RCC_CIR) after boot.
Periodicity	Continuous
Test for the diagnostic	CLK_SM_0: Periodic 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 32. CLK_SM_2

SM CODE	CLK_SM_2
Description	Independent watchdog
Ownership	ST
Detailed implementation	The independent watchdog IWDG is able to detect failures in internal main *MCU* clock (lower frequency).
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	IWDG activation. It is recommended to use the *hardware watchdog* in Option byte settings (IWDG is automatically enabled after reset).
Periodicity	Continuous
Test for the diagnostic	Refer to CPU_SM_6.
Multiple-fault protection	CPU_SM_1: Control flow monitoring in *Application software*
Recommendations and known limitations	The adoption of an external watchdog (refer to CPU_SM_5) adds further diversity.

Table 33. CLK_SM_3

SM CODE	CLK_SM_3
Description	Internal clock cross-measurement
Ownership	*End user*

SM CODE	CLK_SM_3
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 applicable
Multiple-fault protection	CPU_SM_1: Control flow monitoring in *Application software* CPU_SM_5: External watchdog
Recommendations and known limitations	Efficiency versus transient faults is negligible. It provides only medium efficiency in permanent clock-related failure mode coverage.

3.6.7 General-purpose input/output (GPIO)

Table 34. GPIO_SM_0

SM CODE	GPIO_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to GPIO configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	GPIO availability can differ according to part number
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 35. 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

SM CODE	GPIO_SM_1
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Permanent/transient
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	To reduce the potential impact of common cause failure, it is recommended to use GPIO lines: • belonging to different I/O ports (for instance port A and B) • with different bit port number (for instance PA1 and PB5) • mapped to non-adjacent pins on the device package

Table 36. 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/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	To reduce the potential impact of common cause failure, it is recommended to use GPIO lines: • belonging to different I/O ports (for instance port A and B) • with different bit port number (for instance PA1 and PB5) • mapped to non-adjacent pins on the device package Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of GPIO output wrong signal permanence required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm.

Table 37. GPIO_SM_3

SM CODE	GPIO_SM_3
Description	GPIO port configuration lock register
Ownership	ST

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

3.6.8 Debug system or peripheral control

Table 38. DBG_SM_0

SM CODE	DBG_SM_0
Description	Watchdog protection
Ownership	ST
Detailed implementation	The debug unintentional activation due to hardware random fault results in the massive disturbance of *CPU* operations, leading to an intervention of the independent watchdog or, alternatively, the other system watchdog WWDG or the external one (CPU_SM_5).
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval).
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Refer to CPU_SM_6.
Multiple-fault protection	CPU_SM_1: Control flow monitoring in *Application software*
Recommendations and known limitations	None

Table 39. LOCK_SM_0

SM CODE	LOCK_SM_0
Description	Lock mechanism for configuration options
Ownership	ST
Detailed implementation	The STM32F0 Series 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 simply ignored).

SM CODE	LOCK_SM_0
Fault detection time	Not applicable
Addressed fault model	None (systematic only)
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	Not required
Recommendations and known limitations	No *DC* associated because this test addresses systematic faults.

3.6.9 System configuration controller (SYSCFG)

Table 40. SYSCFG_SM_0

SM CODE	SYSCFG_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to system configuration controller configuration registers. This method is strongly recommended to protect registers related to hardware diagnostics activation and error reporting chain related features. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	This method is mainly overlapped by several other configuration register read-backs required for other *MCU* peripherals. It is reported here for the sake of completeness.

Table 41. DIAG_SM_0

SM CODE	DIAG_SM_0
Description	Periodic read-back of hardware diagnostics configuration registers
Ownership	*End user*
Detailed implementation	In STM32F0 Series, several hardware-based safety mechanisms are available (those with the *Ownership* field set to ST). This method must be applied to any configuration register related to diagnostic measure operations, including error reporting. *End user* must therefore individuate configuration registers related to: • hardware diagnostic enable • interrupt/NMI enable (if used for diagnostic error management)
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0

SM CODE	DIAG_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

3.6.10 Direct memory access controller (DMA)

Table 42. DMA_SM_0

SM CODE	DMA_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to *DMA* configuration register and channel address register. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 43. DMA_SM_1

SM CODE	DMA_SM_1
Description	Information redundancy on data packet transferred via *DMA*
Ownership	*End user*
Detailed implementation	This method is implemented by adding, to data packets transferred by *DMA*, a redundancy check (such as *CRC* check or similar one) with encoding capability. Full data packet redundancy would be an overkill. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand

SM CODE	DMA_SM_1
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	To give an example about checksum encoding capability, using just a bit-by-bit addition is inappropriate.

Table 44. DMA_SM_2

SM CODE	DMA_SM_2
Description	Information redundancy by including sender or receiver identifier on data packet transferred via *DMA*
Ownership	*End user*
Detailed implementation	This method helps to identify inside the MCU the source and the originator of the message exchanged by *DMA*. Implementation is realized by adding an additional field to protected message, with a coding convention for message type identification fixed at *Device* level. Guidelines for the identification fields are: • Identification field value must be different for each possible couple of sender or receiver on *DMA* transactions. • Values chosen must be enumerated and non-trivial. • Coherence between the identification field value and the message type is checked by *Application software* before consuming data. This method, when implemented in combination with DMA_SM_4, makes available a kind of *virtual channel* between source and destinations entities.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 45. DMA_SM_3

SM CODE	DMA_SM_3
Description	Periodic software test for *DMA*
Ownership	*End user*
Detailed implementation	This method requires the periodical testing of the DMA basic functionality, implemented through a deterministic transfer of a data packet from one source to another (for example from memory to memory) and the checking of the correct transfer of the message on the target. Data packets are composed by non-trivial patterns (avoid the use of 0x0000, 0xFFFF values) and organized in order to allow the detection during the check of the following failures: • incomplete packed transfer • errors in single transferred word • wrong order in packed transmitted data
Error reporting	Depends on implementation
Fault detection time	Depends on implementation

SM CODE	DMA_SM_3
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 46. 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 to detect missing or unexpected *DMA* activities.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Because *DMA* transaction termination is often linked to an interrupt generation, implementation of this method can be merged with the safety mechanism NVIC_SM_1: Expected and unexpected interrupt check.

3.6.11 Extended interrupt and events controller (EXTI)

Table 47. NVIC_SM_0

SM CODE	NVIC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*

SM CODE	NVIC_SM_0
Detailed implementation	This test is implemented by executing a periodic check of the configuration registers for a system peripheral against its expected value. Expected values are previously stored in RAM and adequately updated after each configuration change. The method mainly addresses transient faults affecting the configuration registers, by detecting bit flips in the registers contents. It addresses also permanent faults on registers because it is executed at least once per *PST* (or another timing constraint; refer to (1) in Section 3.6 Hardware and software diagnostics) after an update of the peripheral. Method must be implemented to any configuration register whose contents are able to interfere with NVIC or EXTI behavior in case of incorrect settings. Check includes NVIC vector table. According to the state-of-the-art automotive safety standard ISO26262, this method can achieve high levels of diagnostic coverage (DC) (refer to ISO26262-5:2018, Table D.4). An alternative valid implementation requiring less space in SRAM can be realized on the basis of signature concept: • Peripheral registers to be checked are read in a row, computing a *CRC* checksum (use of hardware *CRC* is encouraged). • Obtained signature is compared with the golden value (computed in the same way after each register update, and stored in SRAM). • Coherence between signatures is checked by *Application software* – signature mismatch is considered as failure detection.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Values of configuration registers must be read after the boot before executing the first check.
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	This method addresses only failures affecting configuration registers, and not peripheral core logic or external interface. Attention must be paid to registers containing mixed combination of configuration and status bits. Mask must be used before saving register contents affecting signature, and related checks done, to avoid false positive detections.

Table 48. **NVIC_SM_1**

SM CODE	NVIC_SM_1
Description	Expected and unexpected interrupt check
Ownership	*End user*
Detailed implementation	According to IEC 61508:2 Table A.1 recommendations, a diagnostic measure for continuous, absence or cross-over of interrupt must be implemented. The method of expected and unexpected interrupt check is implemented at *Application software* level. The guidelines for the implementation of the method are the following: • The interrupts implemented on the *MCU* are well documented, also reporting, when possible, the expected frequency of each request (for example, the interrupts related to ADC conversion completion that come on a regular basis). • Individual counters are maintained for each interrupt request served, in order to detect in a given time frame the cases of a) no interrupt at all b) too many interrupt requests (“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/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	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.

3.6.12 Cyclic redundancy-check calculation unit (CRC)

Table 49. **CRC_SM_0**

SM CODE	CRC_SM_0
Description	CRC self-coverage
Ownership	ST
Detailed implementation	The *CRC* algorithm implemented in this module (CRC-32 Ethernet polynomial: 0x4C11DB7) offers excellent features in terms of error detection in the message. Therefore permanent and transient faults affecting *CRC* computations are easily detected by any operations using the module to recompute an expected signature.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous

SM CODE	CRC_SM_0
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.13 Analog-to-digital converter (ADC)

Table 50. ADC_SM_0

SM CODE	ADC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to ADC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 51. 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	Depends on implementation
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	It is highly probable that this recommendation is satisfied by design by *End userApplication software*. Usage of multiple acquisitions followed by average operations is a common technique in industrial applications exposed to electromagnetic interference on sensor lines.

Table 52. **ADC_SM_2**

SM CODE	ADC_SM_2
Description	Range check by *Application software*
Ownership	*End user*
Detailed implementation	The guidelines for the implementation of the method are the following: • The expected range of the data to be acquired are investigated and adequately documented. Note that in a well-designed application it is improbable that during normal operation an input signal has a very near or over the upper and lower rail limit (saturation in signal acquisition). • If *Application software* is aware of the state of the system, this information is to be used in the range check implementation. For example, if the ADC value is the measurement of a current through a power load, reading an abnormal value such as a current flowing in opposite direction versus the load supply may indicate a fault in the acquisition module. • As the ADC module is shared between different possible external sources, the combination of plausibility checks on the different signals acquired can help to cover the whole input range in a very efficient way.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Depends on implementation
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	The implementation and the related diagnostic efficiency of this safety mechanism are strongly application-dependent.

Table 53. **ADC_SM_3**

SM CODE	ADC_SM_3
Description	Periodic software test for ADC
Ownership	*End user*
Detailed implementation	The method is implemented acquiring multiple signals and comparing the read value with the expected one, supposed to be know. Method can be implemented with different level of complexity: • Basic complexity: acquisition and check of upper or lower rails (VDD or VSS) and internal reference voltage • High complexity: in addition to basic complexity tests, acquisition of a DAC output connected to ADC input and checking all voltage excursion and linearity
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software

SM CODE	ADC_SM_3
Recommendations and known limitations	Combination of two methods with different complexity can be used to better optimize test frequency in high-demand safety functions.

3.6.14 Digital-to-analog converter (DAC)

Table 54. DAC_SM_0

SM CODE	DAC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to DAC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 55. DAC_SM_1

SM CODE	DAC_SM_1
Description	DAC output loopback on ADC channel
Ownership	*End user*
Detailed implementation	Route the active DAC output to one ADC channel, and check the output current value against the expected one.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous or on demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of DAC wrong signal permanence required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm.

3.6.15 Comparator (COMP)

Table 56. COMP_SM_0

SM CODE	COMP_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to COMP configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 57. COMP_SM_1

SM CODE	COMP_SM_1
Description	1oo2 scheme for comparator
Ownership	*End user*
Detailed implementation	This safety mechanism is implemented using the two internal comparators to take the same decision. It requires that the comparator voting is handled accordingly.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	This method is not compatible with *window* comparator feature.

Table 58. 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.

SM CODE	COMP_SM_2
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 59. COMP_SM_3

SM CODE	COMP_SM_3
Description	Multiple acquisition by *Application software*
Ownership	*End user*
Detailed implementation	This method requires that *Application software* takes a decision not on the basis of a comparator single-shot transition, but after multiple events or after the permanence of comparator trigger conditions for a certain amount of time.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	It is highly probable that this recommendation is satisfied by design on *End user* application - multiple acquisition is a common technique in industrial applications facing electromagnetic interference on sensor lines.

Table 60. COMP_SM_4

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

SM CODE	COMP_SM_4
Test for the diagnostic	Not applicable
Multiple-fault protection	Not applicable
Recommendations and known limitations	This method does not addresses comparator configuration changes due to soft errors.

3.6.16 Touch sensing controller (TSC)

Table 61. TSC_SM_0

SM CODE	TSC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to TSC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 62. TSC_SM_1

SM CODE	TSC_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 TSC input data. Multiple acquisition data are then used to determine the acquisition correct state. This method overlaps on the native features of the TSC module of counting events to ensure a stable acquisition against external noise.
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 applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 63. **TSC_SM_2**

SM CODE	TSC_SM_2
Description	Application-level detection of permanent failures of TSC acquisition
Ownership	*End user*
Detailed implementation	This method must detect TSC module permanent failure leading to wrong or missing acquisition of touch sensing events.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Due to the strictly application-dependent nature of this solution, no detailed guidelines for its implementation are given here. As a solution fully based on microcontroller resources is impossible, it is necessary to leverage on the contribution from other components of the final system.

3.6.17 Advanced, general, and low-power timer (TIM1/2/3/14/15/16/17)

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

Table 64. **ATIM_SM_0**

SM CODE	ATIM_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to advanced, general-purpose and low-power timer configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 65. **ATIM_SM_1**

SM CODE	ATIM_SM_1
Description	1oo2 for counting timers

SM CODE	ATIM_SM_1
Ownership	*End user*
Detailed implementation	This method implements via software a 1oo2 scheme between two counting resources. The guidelines for the implementation of the method are the following: • Two timers are programmed with same time base or frequency. • In case of timer use as a time base: use in *Application software* one of the timer as time base source, and the other one just for check. Coherence check for the 1oo2 is done at application level, comparing two counter values each time the timer value is used to affect safety function. • In case of interrupt generation: use the first timer as main interrupt source for the service routines, and the second timer as a “reference” to be checked at the initial of interrupt routine.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic. This method applies to timer channels merely used as elapsed time counters.

Table 66. ATIM_SM_2

SM CODE	ATIM_SM_2
Description	1oo2 for input capture timers
Ownership	*End user*
Detailed implementation	This method is conceived to protect timers used for acquisition and measurement of external signals (input capture, encoder reading). The implementation consists in connecting the external signals also to a redundant timer, and checking the coherence of the measured data at application level. Coherence check between timers is executed each time the reading is used by *Application software*.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	To reduce the potential effect of common cause failures, it is suggested to use for redundant check a channel belonging to a different timer module and mapped to non-adjacent pin on the device package.

Table 67. ATIM_SM_3

SM CODE	ATIM_SM_3
Description	Loopback scheme for pulse width modulation (PWM) outputs
Ownership	*End user*
Detailed implementation	This method is implemented by connecting the PWM to a separate timer channel to acquire the generated waveform characteristics. The guidelines are the following: • Both PWM frequency and duty cycle are measured and checked versus the expected value. • To reduce the potential effect of common cause failure, it is suggested to use for the loopback check a channel belonging to a different timer module and mapped to non-adjacent pins on the device package. This measure can be replaced under the end-user responsibility by different loopback schemes already in place in the final application and rated as equivalent. For example if the PWM is used to drive an external power load, the reading of the on-line current value can be used instead of the PWM duty cycle measurement.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Depends on implementation
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of PWM wrong signal permanence (wrong frequency, wrong duty, or both) required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm.

Table 68. ATIM_SM_4

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

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

3.6.18 Basic timers (TIM6/7)

Table 69. GTIM_SM_0

SM CODE	GTIM_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to basic timer configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 70. 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. The guidelines for the implementation of the method are the following: • Two timers are programmed with same time base or frequency. • In case of timer use as a time base: use in *Application software* one of the timer as time base source, and the other one just for check. Coherence check for the 1oo2 is done at application level, comparing two 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/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic.

3.6.19 Real-time clock module (RTC)

Table 71. RTC_SM_0

SM CODE	RTC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to RTC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 72. RTC_SM_1

SM CODE	RTC_SM_1
Description	Application check of running RTC
Ownership	*End user*
Detailed implementation	*Application software* implements some plausibility check on RTC calendar or timing data, mainly after a power-up and further date reading by RTC. The guidelines for the implementation of the method are the following: • RTC backup registers are used to store coded information in order to detect the absence of VBAT during power-off period. • RTC backup registers are used to periodically store compressed information on current date or time • *Application software* executes minimal consistence checks for date reading after power-on (detecting “past” date or time retrieve). • *Application software* periodically checks that RTC is actually running, by reading RTC timestamp progress and comparing with an elapsed time measurement based on STM32 internal clock or timers.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software

SM CODE	RTC_SM_1
Recommendations and known limitations	This method provides a limited diagnostic coverage for RTC failure modes. In case of *End user* application where RTC timestamps accuracy can affect in severe way the safety function (for example, medical data storage devices), it is strongly recommended to adopt more efficient system-level measures.

Table 73. RTC_SM_2

SM CODE	RTC_SM_2
Description	Application-level measures to detect failures in timestamps/event capture
Ownership	*End user*
Detailed implementation	This method must detect failures affecting the RTC capability to correct execute the timestamps/event capture functions. Due to the nature strictly application-dependent of this solution, no detailed guidelines for its implementation are given here.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic/On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	This method must be used only if the timestamps/event capture function is used in the safety function implementation. It is worth noting that the use of timestamp / event capture in safety-related applications with the *MCU* in Sleep or Stop mode is prevented by the assumed requirement ASR7 (refer to Section 3.3.1 Safety requirement assumptions).

Table 74. RTC_SM_3

SM CODE	RTC_SM_3
Description	Application-level measures to detect failures in timestamps/event capture
Ownership	*End user*
Detailed implementation	This method must detect failures affecting the RTC capability to correct execute the timestamps/event capture functions. Due to the nature strictly application-dependent of this solution, no detailed guidelines for its implementation are given here.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic/On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	This method must be used only if the timestamps/event capture function is used in the safety function implementation. It is worth noting that the use of timestamp / event capture in safety-related applications with the *MCU* in Sleep or Stop mode is prevented by the assumed requirement ASR7 (refer to Section 3.3.1 Safety requirement assumptions).

3.6.20 Inter-integrated circuit (I2C)

Table 75. IIC_SM_0

SM CODE	IIC_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to I2C configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 76. IIC_SM_1

SM CODE	IIC_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	I2C communication module embeds protocol error checks (like overrun, underrun, packet error etc.) conceived to detect network-related abnormal conditions. These mechanisms are 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/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Adoption of SMBus option grants the activation of more efficient protocol-level hardware checks such as CRC-8 packet protection. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 77. IIC_SM_2

SM CODE	IIC_SM_2
Description	Information redundancy techniques on messages

SM CODE	IIC_SM_2
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by I2C a redundancy check (such as a *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	It is assumed that the remote I2C counterpart has an equivalent capability of performing the check described. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. This method is superseded by IIC_SM_3 if hardware handled *CRC* insertion is possible.

Table 78. IIC_SM_3

SM CODE	IIC_SM_3
Description	*CRC* packet-level
Ownership	ST
Detailed implementation	I2C communication module allows to activate for specific mode of operation (SMBus) the automatic insertion (and check) of *CRC* checksums to packet data.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*.
Multiple-fault protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	This method can be part of the implementation for IIC_SM_2 or IIC_SM_4. In that case, because of the warning issued in the *Test for the diagnostic* field, this mechanism can not be the only one to guarantee message integrity. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 79. **IIC_SM_4**

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

3.6.21 Universal synchronous/asynchronous and low-power universal asynchronous receiver/ transmitter (USART)

Table 80. **UART_SM_0**

SM CODE	UART_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to USART configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 81. **UART_SM_1**

SM CODE	UART_SM_1
Description	Protocol error signals
Ownership	ST

SM CODE	UART_SM_1
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/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	UART_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	USART communication module allows several different configurations. The actual composition of communication error checks depends on the selected configuration. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 82. UART_SM_2

SM CODE	UART_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented by adding to data packets transferred by USART a redundancy check (such as a *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	It is assumed that the remote USART counterpart has an equivalent capability of performing the check described. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated.

Table 83. UART_SM_3

SM CODE	UART_SM_3
Description	Information redundancy techniques on messages, including end-to-end protection

SM CODE	UART_SM_3
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 as follows: • Data packet must be protected (encapsulated) by an information redundancy check, like for instance a CRC checksum computed over the packet and added to payload. Checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single-bit flip in the data packet. • Additional field added in payload reporting an unique identification of sender or receiver and an unique increasing sequence packet number. • Timing monitoring of the message exchange (for example check the message arrival within the expected time window), detecting therefore missed message arrival conditions. • *Application software* must verify before consuming data packet its consistency (CRC check), its legitimacy (sender or receiver) and the sequence correctness (sequence number check, no packets lost).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	A major overlap between the requirements of this method and the implementation of complex communication software protocols can exists. Due to large adoption of these protocols in industrial applications, optimizations can be possible. It is assumed that the remote counterpart has an equivalent capability of performing the checks described.

3.6.22 Serial peripheral interface (SPI)

Table 84. SPI_SM_0

SM CODE	SPI_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to SPI configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0

SM CODE	SPI_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 85. 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.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

Table 86. SPI_SM_2

SM CODE	SPI_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by SPI a redundancy check (such as a *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic 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. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated.

Table 87. SPI_SM_3

SM CODE	SPI_SM_3
Description	*CRC* packet-level
Ownership	ST
Detailed implementation	SPI communication module allows to activate automatic insertion (and check) of CRC-8 or CRC-18 checksums to packet data.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field *Multiple-fault protection*.
Multiple-fault protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	This method can be part of the implementation for SPI_SM_2 or SPI_SM_4. In that case, because of the warning issued in the *Test for the diagnostic* field, this mechanism can not be the only one to guarantee message integrity.

Table 88. SPI_SM_4

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

3.6.23 Controller area network (CAN)

Table 89. CAN_SM_0

SM CODE	CAN_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to CAN configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 90. CAN_SM_1

SM CODE	CAN_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	CAN communication module embeds protocol error checks (like error counters) 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	Several error condition are reported by flag bits in related CAN registers.
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CAN_SM_2: Information redundancy techniques on messages, including end-to-end protection.
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 91. CAN_SM_2

SM CODE	CAN_SM_2
Description	Information redundancy techniques on messages, including end-to-end protection.

SM CODE	CAN_SM_2
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 as follows: • Data packet must be protected (encapsulated) by an information redundancy check, like for instance a CRC checksum computed over the packet and added to payload. Checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single-bit flip in the data packet. • Additional field added in payload reporting an unique identification of sender or receiver and an unique increasing sequence packet number. • Timing monitoring of the message exchange (for example check the message arrival within the expected time window), detecting therefore missed message arrival conditions. • *Application software* must verify before consuming data packet its consistency (CRC check), its legitimacy (sender or receiver) and the sequence correctness (sequence number check, no packets lost).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	A major overlap between the requirements of this method and the implementation of complex communication software protocols can exists. Due to large adoption of these protocols in industrial applications, optimizations can be possible. It is assumed that the remote counterpart has an equivalent capability of performing the checks described.

3.6.24 USB on-the-go full-speed (OTG_FS)

Table 92. USB_SM_0

SM CODE	USB_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to USB OTG_FS configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0

SM CODE	USB_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 93. USB_SM_1

SM CODE	USB_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	USB communication module embeds protocol error checks (like overrun, underrun, NRZI, bit stuffing etc.) 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/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	USB_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 94. USB_SM_2

SM CODE	USB_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user* or ST
Detailed implementation	The implementation of required information redundancy on messages, USB communication module is fitted by hardware capability. It basically allows to activate 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/transient
Dependency on *Device* configuration	None
Initialization	Error reporting configuration, if interrupt events are planned
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

Table 95. USB_SM_3

SM CODE	USB_SM_3
Description	Information redundancy techniques on messages, including end-to-end protection.

SM CODE	USB_SM_3
Ownership	*End user*
Detailed implementation	This method aims to protect the communication between the USB OTG_FS peripheral and its external counterpart. Refer to UART_SM_3 description for detailed information.
Error reporting	Refer to UART_SM_3
Fault detection time	Refer to UART_SM_3
Addressed fault model	Refer to UART_SM_3
Dependency on *Device* configuration	Refer to UART_SM_3
Initialization	Refer to UART_SM_3
Periodicity	Refer to UART_SM_3
Test for the diagnostic	Refer to UART_SM_3
Multiple-fault protection	Refer to UART_SM_3
Recommendations and known limitations	This method applies in case USB bulk or isochronous transfers are used. For other transfers modes the USB hardware protocol already implements several features of this requirement. Refer to UART_SM_3 for further notice.

3.6.25 HDMI-CEC (CEC)

Table 96. HDMI_SM_0

SM CODE	HDMI_SM_0
Description	Periodic read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to CEC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 97. HDMI_SM_1

SM CODE	HDMI_SM_1
Description	Protocol error signals
Ownership	ST

SM CODE	HDMI_SM_1
Detailed implementation	CEC communication module embeds protocol error checks (such as 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 instance baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	HDMI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 98. HDMI_SM_2

SM CODE	HDMI_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by CEC a redundancy check (such as *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by *Application software* before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	It is assumed that the remote HDMI-CEC counterpart has an equivalent capability of performing the check described. To give an example on checksum encoding capability, using just a bit-by-bit addition is inappropriate.

3.6.26 Part separation (no interference)

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

Table 99. **FFI_SM_0**

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

Table 100. **FFI_SM_1**

SM CODE	FFI_SM_1
Description	Periodic read-back of interference avoidance registers
Ownership	*End user*
Detailed implementation	This method contributes to the reduction of the probability of cross-interferences between peripherals that can potentially conflict on the same input/output pins, including for instance unused peripherals. This diagnostic measure must be applied to following registers: • clock enable and disable registers • alternate function programming registers Detailed information on the implementation of this method can be found in Section 3.6.11 Extended interrupt and events controller (EXTI).
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on *Device* configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

3.7 Conditions of use

The table below provides a summary of the safety concept recommendations reported in Section 3.6: Description of hardware and software diagnostics. The conditions of use to be applied to STM32F0 Series 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:

- **M** The safety mechanism is always active during normal operation, with no possibility for *End user* to deactivate it.
- **++** The safety mechanism is highly recommended as common practice. It is considered in this document for the computation of safety metrics to allow the use of *Device* in systems implementing safety functions up to *SIL*2 with a single *MCU* or up to *SIL*3 with two *MCU*s in 1oo2 scheme.
- **+** The safety mechanism is recommended as additional safety measure, but not considered in this document for the computation of safety metrics. STM32F0 Series users can skip the implementation in case it is in contradiction with functional requirements or overlapped by another mechanism ranked ++.
- **o** The safety mechanism is optional. It is not strictly required for the implementation of safety functions up to *SIL*2, or it is related to a specific *MCU* configuration.

The *X* marker in the *Perm* and *Trans* table columns indicates that the related safety mechanism is effective for such fault model.

Table 101. List of safety mechanisms

Diagnostic	Description	Rank	Perm	Trans
Arm® Cortex®-M0				
CPU_SM_0	Periodic core self-test software for Arm®Cortex®-M0 *CPU*.	++	X	-
CPU_SM_1	Control flow monitoring in *Application software*	++	X	X
CPU_SM_2	Double computation in *Application software*	++	-	X
CPU_SM_3	Arm®Cortex®-M0 HardFault exceptions	M	X	X
CPU_SM_4	Stack hardening for *Application software*	+	X	X
CPU_SM_5	External watchdog	++(1)	X	X
CPU_SM_6	Independent watchdog	++(1)	X	X
System bus architecture				
BUS_SM_0	Periodic software test for interconnections	++	X	-
BUS_SM_1	Information redundancy in intra-chip data exchanges	++	X	X
Embedded SRAM				
RAM_SM_0	Periodic software test for static random access memory (SRAM)	++	X	-
RAM_SM_1	Parity on SRAM2	++	X	X
RAM_SM_2	Stack hardening for *Application software*	+	X	X
RAM_SM_3	Information redundancy for safety-related variables in *Application software*	++	X	X
RAM_SM_4	Control flow monitoring in *Application software*	o(2)	X	X

Diagnostic	Description	Rank	Perm	Trans
RAM_SM_5	Periodic integrity test for *Application software* in RAM	o(2)	X	X
Embedded Flash memory				
FLASH_SM_0	Periodic software test for Flash memory	++	X	-
FLASH_SM_1	Control flow monitoring in *Application software*	++	X	X
FLASH_SM_2	Arm®Cortex®-M0 HardFault exceptions	M	X	X
FLASH_SM_3	Option byte write protection	M	-	-
FLASH_SM_4	Static data encapsulation	+	X	X
FLASH_SM_5	Option byte redundancy with load verification	M	X	X
FLASH_SM_6	Flash memory unused area filling code	+	-	-
Power controller (PWR)				
VSUP_SM_0	Periodic read-back of configuration registers	++	X	X
VSUP_SM_1	Supply voltage internal monitoring (PVD)	++	X	-
VSUP_SM_2	Independent watchdog	++	X	-
VSUP_SM_3	Internal temperature sensor check	o	-	-
VSUP_SM_5	System-level power supply management	+	-	-
Reset and clock controller (RCC)				
CLK_SM_0	Periodic read-back of configuration registers	++	X	X
CLK_SM_1	Clock security system (CSS)	++	X	-
CLK_SM_2	Independent watchdog	++	X	-
CLK_SM_3	Internal clock cross-measurement	+	X	-
General-purpose input/output (GPIO)				
GPIO_SM_0	Periodic read-back of configuration registers	++	X	X
GPIO_SM_1	1oo2 for input GPIO lines	++	X	X
GPIO_SM_2	Loopback scheme for output GPIO lines	++	X	X
GPIO_SM_3	GPIO port configuration lock register	+	-	-
Debug system or peripheral control				
DBG_SM_0	Watchdog protection	++	X	X
LOCK_SM_0	Lock mechanism for configuration options	+	-	-
System configuration controller (SYSCFG)				
SYSCFG_SM_0	Periodic read-back of configuration registers	++	X	X
DIAG_SM_0	Periodic read-back of hardware diagnostics configuration registers	++	X	X
Direct memory access controller (DMA)				
DMA_SM_0	Periodic read-back of configuration registers	++	X	X
DMA_SM_1	Information redundancy on data packet transferred via *DMA*	++	X	X
DMA_SM_2	Information redundancy by including sender or receiver identifier on data packet transferred via *DMA*	++	X	X
DMA_SM_3	Periodic software test for *DMA*	++	X	-
DMA_SM_4	*DMA* transaction awareness	++	X	X

Diagnostic	Description	Rank	Perm	Trans
Extended interrupt and events controller (EXTI)				
NVIC_SM_0	Periodic read-back of configuration registers	++	X	X
NVIC_SM_1	Expected and unexpected interrupt check	++	X	X
Cyclic redundancy-check calculation unit (CRC)				
CRC_SM_0	CRC self-coverage	++	X	X
Analog-to-digital converter (ADC)				
ADC_SM_0	Periodic read-back of configuration registers	++	X	X
ADC_SM_1	Multiple acquisition by *Application software*	++	-	X
ADC_SM_2	Range check by *Application software*	++	X	X
ADC_SM_3	Periodic software test for ADC	++	X	-
Digital-to-analog converter (DAC)				
DAC_SM_0	Periodic read-back of configuration registers	++	X	X
DAC_SM_1	DAC output loopback on ADC channel	++	X	X
Comparator (COMP)				
COMP_SM_0	Periodic read-back of configuration registers	++	X	X
COMP_SM_1	1oo2 scheme for comparator	++	X	X
COMP_SM_2	Plausibility check on inputs	+	X	-
COMP_SM_3	Multiple acquisition by *Application software*	+	-	X
COMP_SM_4	Comparator lock mechanism	+	-	-
Touch sensing controller (TSC)				
TSC_SM_0	Periodic read-back of configuration registers	++	X	X
TSC_SM_1	Multiple acquisition by *Application software*	++	-	X
TSC_SM_2	Application-level detection of permanent failures of TSC acquisition	+	X	-
Advanced, general, and low-power timer (TIM1/2/3/14/15/16/17)				
ATIM_SM_0	Periodic read-back of configuration registers	++	X	X
ATIM_SM_1	1oo2 for counting timers	++	X	X
ATIM_SM_2	1oo2 for input capture timers	++	X	X
ATIM_SM_3	Loopback scheme for pulse width modulation (PWM) outputs	++	X	X
ATIM_SM_4	Lock bit protection for timers	+	-	-
Basic timers (TIM6/7)				
GTIM_SM_0	Periodic read-back of configuration registers	++	X	X
GTIM_SM_1	1oo2 for counting timers	++	X	X
Real-time clock module (RTC)				
RTC_SM_0	Periodic read-back of configuration registers	++	X	X
RTC_SM_1	Application check of running RTC	++	X	X
RTC_SM_2	Application-level measures to detect failures in timestamps/event capture	o	X	X
RTC_SM_3	Application-level measures to detect failures in timestamps/event capture	o	X	X

Diagnostic	Description	Rank	Perm	Trans
Inter-integrated circuit (I2C)				
IIC_SM_0	Periodic read-back of configuration registers	++	X	X
IIC_SM_1	Protocol error signals	++	X	X
IIC_SM_2	Information redundancy techniques on messages	++	X	X
IIC_SM_3	*CRC* packet-level	+	X	X
IIC_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X
Universal synchronous/asynchronous and low-power universal asynchronous receiver/transmitter (USART)				
UART_SM_0	Periodic read-back of configuration registers	++	X	X
UART_SM_1	Protocol error signals	++	X	X
UART_SM_2	Information redundancy techniques on messages	++	X	X
UART_SM_3	Information redundancy techniques on messages, including end-to-end protection	++	X	X
Serial peripheral interface (SPI)				
SPI_SM_0	Periodic read-back of configuration registers	++	X	X
SPI_SM_1	Protocol error signals	++	X	X
SPI_SM_2	Information redundancy techniques on messages	++	X	X
SPI_SM_3	*CRC* packet-level	+	X	X
SPI_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X
Controller area network (CAN)				
CAN_SM_0	Periodic read-back of configuration registers	++	X	X
CAN_SM_1	Protocol error signals	++	X	X
CAN_SM_2	Information redundancy techniques on messages, including end-to-end protection.	++	X	X
USB on-the-go full-speed (OTG_FS)				
USB_SM_0	Periodic read-back of configuration registers	++	X	X
USB_SM_1	Protocol error signals	++	X	X
USB_SM_2	Information redundancy techniques on messages	++	X	X
USB_SM_3	Information redundancy techniques on messages, including end-to-end protection.	+	X	X
HDMI-CEC (CEC)				
HDMI_SM_0	Periodic read-back of configuration registers	++	X	X
HDMI_SM_1	Protocol error signals	+	X	X
HDMI_SM_2	Information redundancy techniques on messages	++	X	X
Part separation (no interference)				
FFI_SM_0	Disable of unused peripherals	++	-	-
FFI_SM_1	Periodic read-back of interference avoidance registers	++	-	-
Arm®Cortex®-M0 *CPU*				
CoU_1	Disable of unused peripherals	++	-	-
Debug				
CoU_2	*Device* debug features must not be used in safety function(s) implementation.	++	-	-

Diagnostic	Description	Rank	Perm	Trans
Arm®Cortex®-M0 / Supply system				
CoU_3	Low-power mode state must not be used in safety function(s) implementation.	++	-	-
***Device* peripherals**				
CoU_4	*End user* must implement the required combination of safety mechanism/CoUs for each STM32 peripheral used in implementation of safety function(s).	++	X	X
Flash memory subsystem				
CoU_5	During Flash memory bank mass erase and reprogramming there must not be safety functions(s) executed by *Device*.	++	-	-
CRS				
CoU_8	CRS features must not be used in safety function(s) implementation.	++	-	-
Device				
DUAL_SM_0	Cross-check between two STM32 MCUs	o	X	X

1. *To achieve on the single MCU local safety metrics compatible with SIL2 target , method CPU_SM_6 could be sufficient. Anyway, to understand the rationale behind "++" classification for both methods, refer to the "Recommendations" row of related description in Section 3.6 Hardware and software diagnostics for more details.*
2. *Must be considered ranked as "++" if Application software is executed on RAM.*

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

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

- System-critical *MCU* modules

  Every *End user* application is affected, from safety point of view, by a failure on these modules. Because they are used by every *End user* application, related methods or safety mechanism are mainly conceived to be application-independent. The system-critical modules on *Device* are: CPU, RCC, PWR, bus matrix and interconnect, and Flash memory and RAM (including their interfaces).

- Peripheral modules

  Such modules could be not used by the end-user application, or they could be used for non-safety related tasks. Related safety methods are therefore implemented mainly at application level, as *Application software* solutions or architectural solutions.

4 Safety results

This section reports the results of the safety analysis of the STM32F0 Series 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 STM32F0 Series 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 *Device* failure modes is maintained in the related *FMEA* document [1], provided on demand by local STMicroelectronics sales office.

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

Table 102. 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. *Note that the potential performance impact related to some above-reported target achievements is mainly related to the need of execution of periodical software-based diagnostics (refer to safety mechanism description for details). The impact is therefore strictly related to how much "aggressive" the system level PST is (see Section 3.3.1 Safety requirement assumptions).*

The resulting relative safety metrics (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 [2], due to:

- a large number of different STM32F0 Series parts,
- a possibility to declare non-safety-relevant unused peripherals, and
- a possibility to enable or not the different available safety mechanisms.

The *FMEDA* snapshot [2] is a static document reporting the safety metrics computed at different detail levels (at microcontroller level and for microcontroller basic functions) for a given combination of safety mechanisms and for a given part number. If *FMEDA* 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 STM32F0 Series boundary, hence they do not include the WDTe, PEv, and VMONe processes described in Section 3.3.1 Safety requirement assumptions).

4.1.1 Safety analysis result customization

The safety analysis executed for STM32F0 Series devices documented in this safety manual considers all microcontroller modules to be safety-related, thus able to interfere with the safety function, with no exclusion. This is in line with the conservative approach to be followed during the analysis of a general-purpose microcontroller, in order to be agnostic versus the final application. This means that no microcontroller module has been declared *safe* as per IEC61508-4, 3.6.8. Therefore, all microcontroller modules are included in *SFF* computations.

In actual *End user* applications, not all the STM32F0 Series parts or modules implement a safety function. That happens if:

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

Note: Implementation of non-safety-related functions is in principle forbidden by the assumed safety requirement ASR6 (see Section 3.3.1 Safety requirement assumptions), hence under End user's entire responsibility. As any other derogation from safety requirements included in this manual, it is End user's responsibility to provide consistent rationales and evidences that the function does not bring additional risks, by following the procedure described in this section. Therefore, it is strongly recommended to reserve such derogation to very simple functions (as the one provided in the example).

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

End user can define a STM32F0 Series part as *non-safety-related* based on:

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

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

- Exclude the part from computing metrics to report in *FMEDA*, and
- Not implement safety mechanisms as listed in Table 101. List of safety mechanisms.

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

4.1.2 General requirements for freedom from interferences (FFI)

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

End user must implement the safety mechanisms listed in Table 103 (implementation details in Section 3.6 Hardware and software diagnostics) regardless any evaluation of their contribution to safety metrics.

Table 103. List of general requirements for FFI

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

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

4.1.3 Notes on multiple-fault scenario

According to the requirements of IEC61508, the safety analysis for STM32F0 Series 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 STM32F0 Series 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: CPU_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 STM32F0 Series 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, *End user* is required to evaluate the β and βD parameters (used in *PFH* computation) that reflect the common cause factors between the two channels.

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

4.2.1 Power supply

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

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

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 3.6.5 Power controller (PWR) for the detailed safety mechanism descriptions.

4.2.2 Clock

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

- CLK_SM_1: the clock security system is able to detect hard alterations (stop) of system clock and activate the adequate recovery actions.
- CLK_SM_2: the independent watchdog has a dedicated clock source. The frequency alteration of the system clock leads to the watchdog window violations by the triggering routine on *Application software*, leading to the *MCU* reset by watchdog.

The adoption of 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 3.6.6 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 3.6.10 Direct memory access controller (DMA) for description):

- DMA_SM_0
- DMA_SM_1
- DMA_SM_2

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

4.2.4 Internal temperature

The abnormal increase of the internal temperature is a potential source of dependent failures, as it can affect many *MCU* parts. The following safety mechanism mitigates this potential effect (refer to Section 3.6.5 Power controller (PWR) for description):

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

5 List of evidences

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

The safety case database is composed of the following:

- safety case with the full list of all safety-analysis-related documents
- STMicroelectronics' internal *FMEDA* tool database for the computation of safety metrics, including estimated and measured values
- safety report, a document that describes in detail the safety analysis executed on STM32F0 Series 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.

6 Change impact analysis for other safety standards

The safety analysis reported in this safety manual is executed according to the IEC 61508 safety norm. This section reports the outcome of a change impact analysis with respect to different safety standards. For each new safety standard addressed, the following items are considered:

- Differences in the suggested hardware architecture (architectural categories), and how to map to safety architectures of IEC 61508.
- Differences in the safety integrity level definitions and metrics computation methods, and how to recompute and judge the safety performances of the devices according to the new standard.

The safety standards examined within this change impact analysis are:

- ISO 13849-1:2015, ISO13849-2:2012 – *Safety of machinery and Safety-related parts of control systems*,
- IEC 62061:2005+AMD1:2012+AMD2:2015 – *Safety of machinery and Functional safety of safety-related electrical, electronic and programmable electronic control systems*,
- IEC 61800-5-2:2016 –*Adjustable speed electrical power drive systems – Part 5-2: Safety requirements – Functional*

6.1 ISO 13849-1:2015, ISO 13849-2:2012

ISO 13849-1 is a type B1 standard. It provides a guideline for the development of Safety-related parts of machinery control systems (SRP/CS) including programmable electronics, hardware and software.

6.1.1 ISO 13849 architectural categories

ISO 13849-1:2015 reports in section 4.4, Figure 4 a typical safety function diagrammatic presentation. Under the assumption that *Compliant item* as defined in section is used to implement the *b* (logic), the equivalence of the ISO 13849 representation with the one in Section 3.2.1 is evident. The mapping of ISO 13849 architectures with the one described in Section 3 is possible.

ISO 13849-1:2015 in section §6 defines in details five different categories. The following table lists for each category the possible implementation by one of the IEC 61508 compliant architectures described in this manual in Section 3 . It is worth to note that for each category, the achievable *PL* is decided by the specific values of diagnostic coverage $(DC)_{avg}$ and mean time to dangerous failure (MTTFd) (refer to Section 6.1.2 for details on computations).

Table 104. ISO 13849 architectural categories

ISO13849-1:2015		Link to IEC61508-compliant safety architectures	Notes/constraints
Category	Clause		
B	6.2.3	Possible with 1oo1 architecture	No requirements for *MTTFd* and DC_{avg} are given for category B, anyway it is recommended to follow safety manual recommendation.
1	6.2.4	Not recommended	Category not recommended because of the NOTE1 in IEC13849-1, section §6.2.4.
2	6.2.5	Possible with 1oo1 architecture (external WDT is mandatory)	The adoption of external WDT (CPU_SM_5) acting as TE is mandatory. Constraints on DC_{avg} and *MTTFd* can be satisfied but computations are needed(1). Constraints on *CCF* are satisfied(2).
3	6.2.6	Possible with 1oo2 architecture + DUAL_SM_0	Constraints on DC_{avg} and *MTTFd* can be satisfied but computations are needed(1) Constraints on *CCF* are satisfied(2).
4	6.2.7	Possible with 1oo2 architecture + DUAL_SM_0	Implementation of DUAL_SM_0 scheme is mandatory to mitigate fault accumulation. Constraints on DC_{avg} and *MTTFd* can be satisfied but computations are needed(1) Constraints on *CCF* are satisfied(2).

1. *Computations related to DC_{avg} and MTTFd can involve also other components than Device because used in the safety function implementation (sensors, actuators, etc). The figures need therefore to be evaluated at system level – refer to Section 6.1.2 for the correct interpretation of Device data in such a computation.*
2. *CCF additional requirements expressed in ISO13849-1, Annex F table F.1 are basically enforcing the system implementation and therefore outside the scope of this manual. It is worth to note that the complete safety analysis resulting as output of the IEC61508 compliance activity (this manual) helps to claim the score for item #4 in Table F.1.*

6.1.2 ISO 13849 safety metrics computation

Appendix C of ISO 13849 presents tables of standardized mean time to dangerous failure (MTTFd) for the various electric or electronics components. However, table C.3 in ISO 13849 points to ICs manufacturer's data while attempting to classify *MTTFd* for programmable ICs. As a consequence, safety analysis results of this Safety Manual can be re-mapped in ISO 13849 domain, because even computed for IEC 61508 they are definitely more and more accurate in the definition of dangerous failures identification.

When for a certain component *PFH* << 1 it can be assumed that *MTTFd* = 1 / *PFH*.

It is worth to note that according ST methodology, FMEDA data includes failure rate related to transient faults without any assumption about their potential partial safeness. Because of this assumption, *PFH* values in Device FMEDA leads to very conservative values for computed *MTTFd*.

In ISO 13849-1 the *DC* for each single component has the same meaning of the IEC 61508 metric; results of this safety manual and related FMEA/FMEDA can therefore be reused. However, this standard defines the concept of DC_{avg} applicable to the whole *SRP/CS* in the form of the equation defined in Annex E, formula E.1, where the contribution of each part of the control system is weighted with respect to *MTTFd* of the various subsystems of the channel. *End user* is therefore responsible for the computations of the overall DC_{avg}.

The standard denies any possibility of fault exclusion while calculating DC_{avg} (ISO13849-2 Tab.D.21 no exclusion allowed), which is also the assumption of *Device* analysis documented in this safety manual.

Note: Each architectural solution analyzed in this safety manual results in PFH values producing high MTTFd.

6.2 IEC 62061:2005+AMD1:2012+AMD2:2015

This standard is applicable in the specification, design and verification or validation of safety-related electrical control systems (SRECS) of machines. *SRECS* is the electrical or electronics control system of the machine which failure could lead to reduction or loss of safety. *SRECS* implements a safety-related control function (SRCF) to prevent any increase of the risk.

Because STM32xx has been classified as Type B according IEC61508 (refer to Section 3.2.2), it must be considered as a “complex component” in IEC62061 definition.

6.2.1 IEC 62061 architectural categories

IEC 62061 defines a set of basic system architectures to be used for the design of safety-related electrical control systems (safety-related electrical control systems (SRECS)) implementing their *SRCF*s. The following table lists for each system architecture the possible implementation/mapping by/to one of the IEC 61508 compliant architectures described in this manual in Section 3 .

Safety metrics related to STM32xx *MCU* can be reused from IEC61508 analysis (refer to *Device* FMEDA), while their combination with the ones related to other devices included in the system is full responsibility of *End user*.

Table 105. IEC 62061 architectural categories

IEC 62061		Link to IEC61508-compliant safety architectures	Notes/constraints
Architecture	Clause		
A	6.7.8.2.2	Equivalent of 1oo1, with *HFT* = 0, no diagnostic function(s) implemented.	-
B	6.7.8.2.3	Equivalent to 1oo2 with *HFT* = 1, a single failure does not lead to the loss of *SRCF*. No diagnostic function(s) implemented.	-
C	6.7.8.2.4	Equivalent of 1oo1 architecture.	All requirements related to 1oo1 architecture must be implemented.
D	6.7.8.2.5	Equivalent of 1oo2 architecture.	All requirements related to 1oo2 architecture must be implemented.

6.2.2 IEC 62061 safety metrics computation

The failure rate (λ) in T is the smaller proof test interval or the life time of the subsystem.

As seen in ISO 13849, the approximation §6.7.8.2.1 NOTE2 is still considered valid, hence

λ = 1 / *MTTFd*, where it is assumed that 1 >> λ x T.

So, as $PFH_D = \lambda_D$ x 1h, so *PFD* = 1 / *MTTFd*.

Safety analysis executed for STM32F0 Series devices according to IEC 61508 is more and more accurate for the definition of dangerous failure identifications that can be re-mapped in IEC 62061 domain. Thus, values of λ, *PFH* and *SFF* that are reported in the *FMEDA* (refer to Section 4 Safety results), are still valid and can be reused.

For evaluation of *CCF* in basic architectures with *HFT* = 1, *End user* can rely to what reported in Section 4.2 Analysis of dependent failures, and to the guidelines included in IEC 61508:2010-6 Annex D.

Alternatively, *End user* can apply the simplified approach from the standard (refer to Annex F) to calculate the β factor value to be used in formulas for *PFD*.

6.3 IEC 61800-5-2:2016

The scope of this standard is the functional safety of adjustable speed electric drive systems.

6.3.1 IEC 61800 architectural categories

Because IEC 61800 definitions for *HFT* and for architectures are equivalent to the ones of IEC61508, the remapping is straightforward.

The STM32xx *MCU* is considered as Type B for the consideration reported in Section 3.2.2 .

6.3.2 IEC 61800 safety metrics computation

The PFH of a safety function performed by *PDS(SR)* is evaluated by the application of IEC 61508-2. The strong link with the norm IEC 61508 is reflected also by the adoption in IEC 61800-5-2 of the same relevant metrics *PFH*, and SFF. So, results of this safety manual (and related *FMEA* and *FMEDA*) can be re-mapped in IEC 61800 domain.

Revision history

Table 106. **Document revision history**

Date	Version	Changes
19-Jun-2014	1	Initial release.
30-Jan-2015	2	Extended the user manual applicability to STM32F0 Series and to STM32-SafeSIL part number. Updated: • Figure 1: STMicroelectronics product development process • Figure 16: Block diagram for IEC 62061 Cat. B • Figure 18: Block diagram for IEC 62061 Cat. D
03-Mar-2015	3	Replaced all NVC occurrences with NVIC in Table 3: List of safety mechanisms and in Table 17: List of STM32F0 Series safety mechanism overlapped by fRSTL_STM32F0_SIL2(3).
05-Oct-2017	4	• Removed: former fR Methodology, Dual MCU architecture, Latent Fault detection, examples of safety architecture, fRSTL_STM32F0_SIL2(3) • Added: Figure 3: 1oo1 reference architecture, Figure 4: 1oo2 reference architecture, Section 3.6.26: System configuration controller (SYSCFG), Section 3.6.28: Notes on multiple faults scenario, Table 5 to Table 102 for description of hardware and software diagnostics • Updated: Section 3.3.1: Assumed safety requirements, Section A.4.1: Architectural categories, Table 104: Overall achievable safety integrity levels, Table 112: IEC 60730 required safety mechanism for Class B/C compliance
24-Apr-2018	5	• Updated: Reference of “IEC 13849” was updated to “ISO 13549” in the whole document, including titles of Figure 6, Figure 7, Figure 8, Table 106 and Table 107 Table 103: List of safety mechanisms Name of Section A.2: IEC 62061:2005/AMD1:2012 Section 1.3: Reference normative Section 4.1.1: Safety analysis results customization Section Appendix A: Change impact analysis for other safety standards Section A.2.2: Safety metrics computation Figure 9: SRECS high-level diagram • Deleted: Section A.4: IEC 60730-1:2010
08-Aug-2018	6	• *Notes on multiple fault scenarios* section moved from *Description of hardware and software diagnostics* section to *Safety results* section. • Updated *Assumed safety requirements* section

Date	Version	Changes
25-Jun-2019	7	Updated: • Functional safety documentation framework • Updated *Reference normative* section • Updated Change impact analysis for other safety standards Deleted: • Section A.4 ISO 26262:2010
01-Jul-2020	8	General update. Former appendix changed into Section 6 Change impact analysis for other safety standards, with the removal of *ISO 13849 work products*, *IEC 62061 work products*, and *IEC 61800 work products* subsections.

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 STM32F0 Series 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 – PLEASE 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 please 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, please 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.