
STM32MP1 series safety manual

Introduction

This document must be read along with the technical documentation such as reference manual(s) and datasheets for the STM32MP1 series microprocessor devices, available on www.st.com.

This manual addresses the STM32MP1 series microprocessors, that include two CPU cores, the Arm® Cortex®-A7 and Cortex®-M4. 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. The safety concept described in this manual is based on the possible implementation of safety function(s) on the Arm®Cortex®-M4 CPU (and associated peripherals) included in STM32MP1 series.

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

The document is written in compliance with IEC 61508.

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



1 About this document

1.1 Purpose and scope

This document describes how to use STM32MP1 series [microprocessor unit \(MPU\)](#) 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 compliance to other functional safety standards is considered in reference document [3].

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

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

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

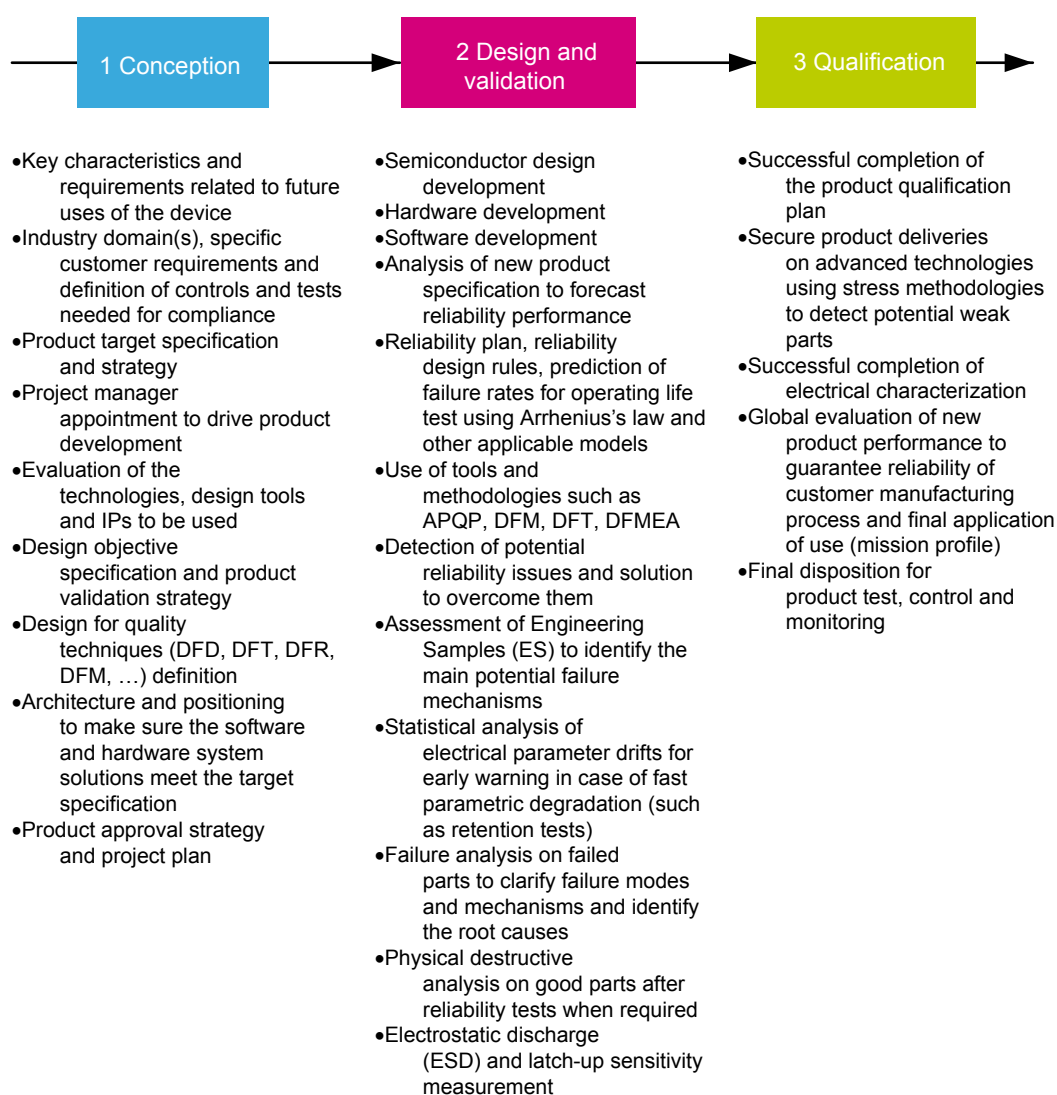
1.3 Reference documents

- [1] AN5459: FMEDA snapshots for STM32MP1 microprocessor series.
- [2] AN5460: Results of FMEA on STM32MP1 series microprocessor.
- [3] AN5689: Adapting the X-CUBE-STL functional safety package for STM32 (IEC 61508 compliant) to other safety standards
- [4] AN5936 X-CUBE-STL: advanced topics

2 Device development process

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

Figure 1. STMicroelectronics product development process



3 Reference safety architecture

This section reports details of the STM32MP1 series safety architecture.

3.1 Safety architecture introduction

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

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

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

3.2 Compliant item

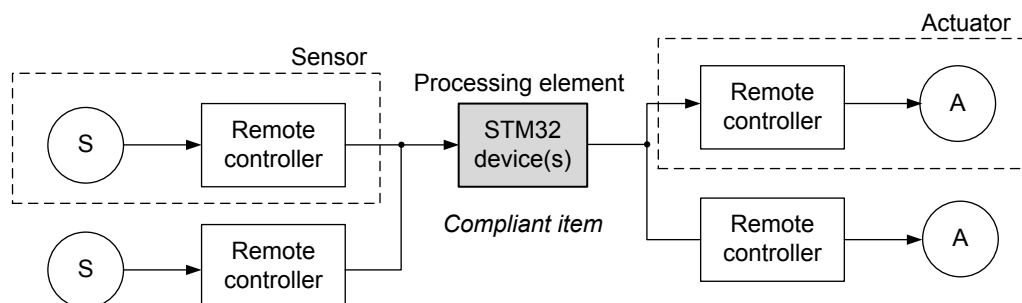
This section defines the *Compliant item* term and provides information on its usage in different safety architecture schemes.

3.2.1 Definition of Compliant item

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

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

Figure 2. STM32 as *Compliant item*



Compliant item partition

In the framework of STM32MP1 safety concept, the *Compliant item* is assumed to be divided in two partitions:

- **Safe Partition**, including all STM32MP1 logic considered as safety related and suitable for the implementation of End User safety functions. This partition includes also the STM32MP1 logic implementing (in collaboration with application software) the separation (freedom from interference) between itself and the Non-Safe Partition.
The complete list of STM32MP1 internal modules belonging to Safe Partition is reported in [Section 3.7 Conditions of use](#). The Arm® Cortex®-M4 CPU belongs to Safe Partition.
- **Non-Safe Partition**, including all STM32MP1 logic considered as non-safety related and therefore not suitable for the implementation of End User safety functions and so out of the safety scope.
The STM32MP1 modules not listed explicitly in [Section 3.7 Conditions of use](#) are considered belonging to Non-Safe Partition. The Arm®-A7 CPU belongs to Non-Safe Partition.

Accordingly, the implementation of the End User safety function is restricted to the STM32MP1 logic belonging to Safe Partition.

Caution: According to Non-Safe Partition definition, the implementation of safety function(s) based on Arm®-A7 CPU is not allowed.

Interference due to the presence in STM32MP1 device of logic belonging to Non-Safe Partition are managed by the help of the separation concept, which is described in detail in [Section 3.2.5 The separation concept](#).

It is worth to note that because of the effective isolation between Safe and Non Safe Partition, the presence of non-safety related application software running on A7 CPU is allowed. [Section 4.2 Analysis of dependent failures](#) of this document provides more information on this specific case.

Other components might be related to the *Compliant item*, like the external HW components needed to guarantee either the functionality of the device (external memory, clock quartz and so on) or its safety (for example, the external watchdog or voltage supervisors). These components are not analyzed in this safety manual.

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
- computation processing elements (PEd, see [Note](#)) executing hardware and software-based safety functions devoted to:
 1. detect/prevent hardware random failures affecting all considered processing elements (PEi/PEc/PEo/PEd)
 2. prevent/mitigate systematic failures in the software executed on PEd/PEd
 3. prevent/detect interferences on the safety related hardware and software caused by the non-safety related hardware and software
- in 1oo2 architecture, potentially a further voting processing element (PEv)
- processes external to *Compliant item* ensuring safety integrity, such as watchdog (WDTe) and voltage monitors (VMONE)

Note: *Because the large part of safety mechanisms included in STM32MP1 are software-based, PEd and PEd hardware are mainly coincident.*

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 and to the PEd processes, which are integral parts of the STM32MP1 series safety concept.

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

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

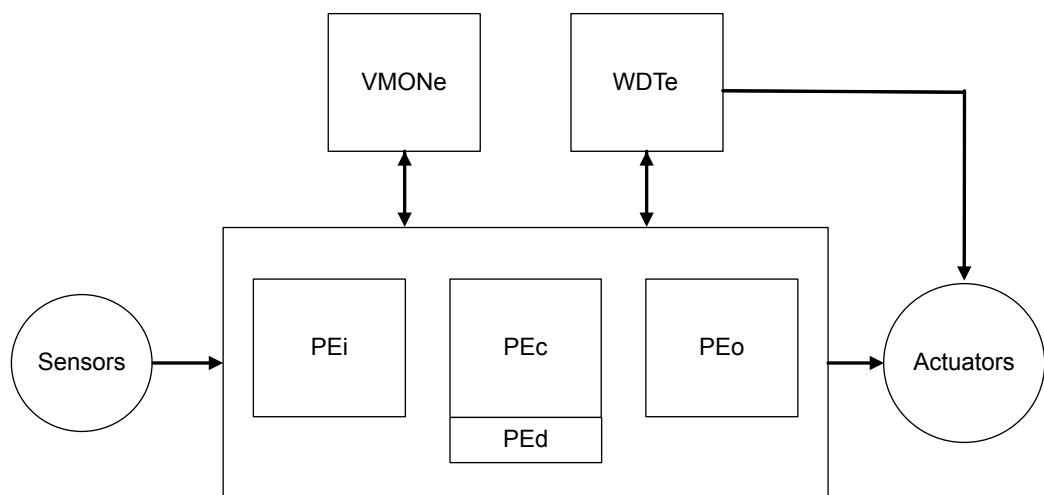
Two main safety architectures are identified: 1oo1 (using one *Device*) and 1oo2 (using two *Devices*).

3.2.3 Reference safety architectures - 1oo1

1oo1 reference architecture (Figure 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



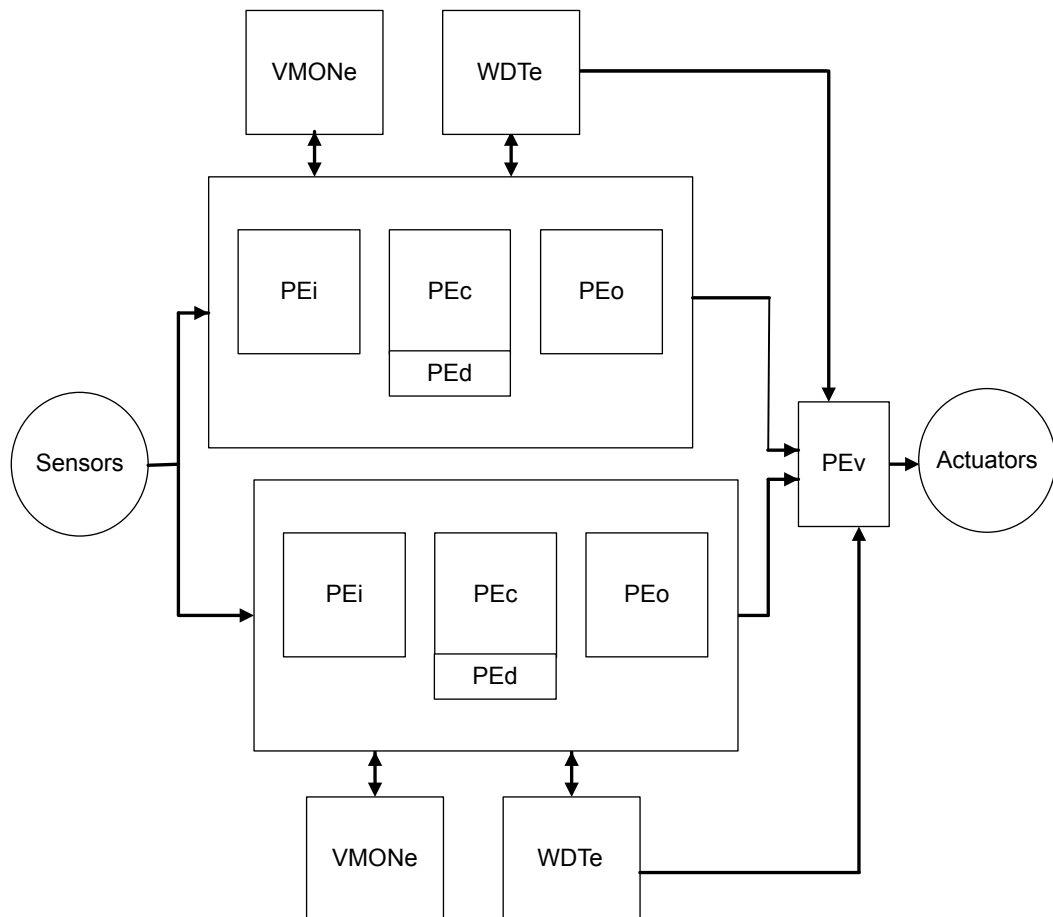
3.2.4 Reference safety architectures - 1oo2

The 1oo2 reference architecture (Figure 4) contains two separate channels, either implemented as a 1oo1 reference architecture ensuring safety integrity of the *Compliant item* through combining *Device* internal processes (implemented safety mechanisms) with external processes: WDT_e and VMON_e. The overall safety integrity is then ensured by the external voter P_{Ev}, which allows claiming **hardware fault tolerance (HFT)** equal to 1. The P_{Ev} role is indeed to facilitate the safety function processing by each of the two individual channels, to allow the correct execution of the safety function even in case one channel is faulty. The complexity of the P_{Ev} implementation strongly depends on the nature of the safety function and safe state definitions. Achievement of higher safety integrity levels as per IEC 61508-2 Table 3 is therefore possible. Appropriate separation between the two channels (including power supply separation) 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 β_D parameters computation is required.

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

Attention: According the clause 7.4.3.2 in IEC 61508-2, this architectural scheme may provide benefits to the software applications systematic capability (SC) only in case diverse software is adopted on the two channels.

Figure 4. 1oo2 reference architecture



3.2.5 The separation concept

The Safe Partition and the Non-Safe Partition are isolated by the separation concept, which is composed by two different aspects, spatial separation and temporal separation.

Spatial separation

Safe and Non-Safe Partitions share the same device (STM32MP1), therefore interferences are possible. The protection against spatial interferences is built by two successive layers of protection:

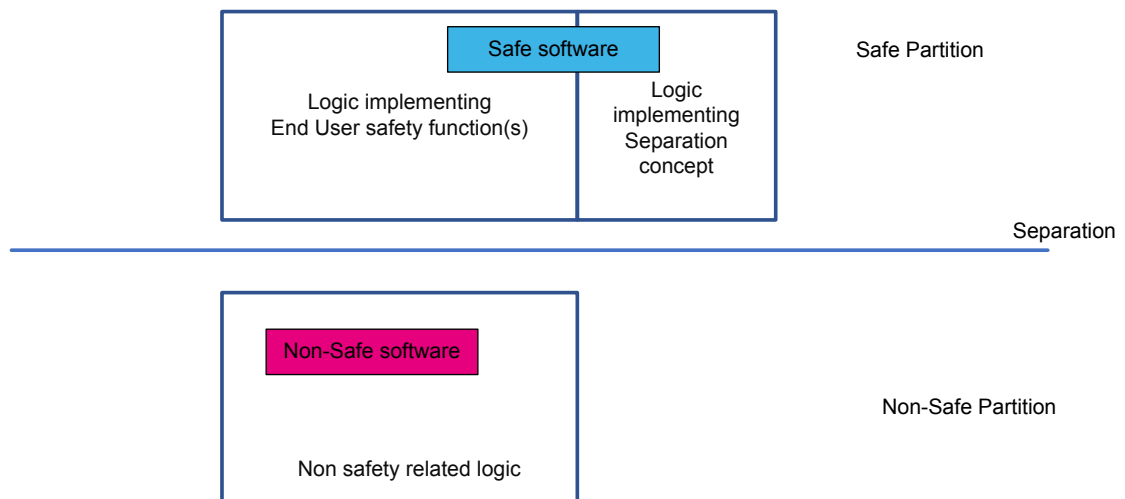
- eTZPC protection: the major part of STM32MP1 peripherals and DMA masters belonging to Safe Partition are "isolable" and so can be allocated to Cortex®-M4 CPU domain accesses exclusively by proper setting of MCU resource isolation system of the eTZPE controller. Then any access tried by a Non-Safe partition AXI master like the A7 CPU is detected and stopped by eTZPC hardware module. Note that all the peripherals which are securable to A7 CPU exclusively are excluded from the Safe Partition and safety concept by definition.
- The local safety concept for each peripheral in Safe Partition, which is composed by several overlapped safety mechanism defined at application level, is able to detect in efficient way unintended perturbing accesses coming from Non Safe Partition. All those safety mechanisms are declared as "highly recommended" in Conditions of use [Section 3.7 Conditions of use](#), so their presence in the final system is guaranteed. This protection is quite valuable for the few common resources belonging to Safe Partitions which cannot be isolated by eTZPC protection, because fully shared with A7 CPU (power and clock configuration interfaces).

The two layers are overlapped, so the second one acts as second line of defense in case of failures of the main protection by eTZPC.

The separation concept protects the Safe Partition from unintended accesses regardless their nature on the Non Safe Partition side, so because of random hardware failures in the hardware or systematic errors in the software.

Figure 5 represents the separation concept:

Figure 5. Spatial separation concept



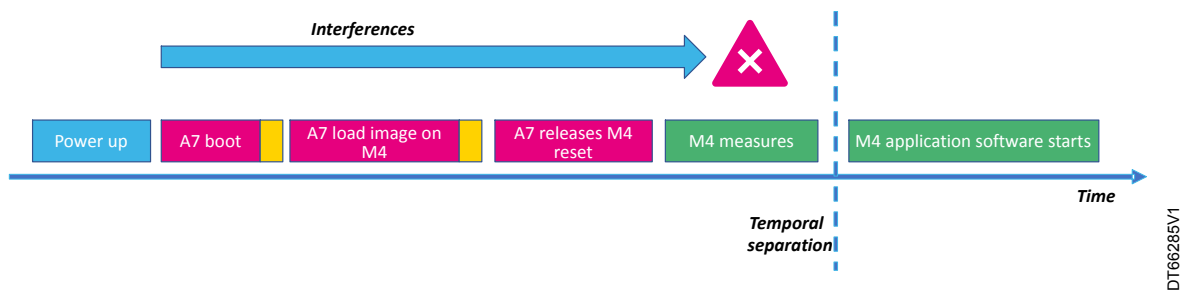
DT65248V1

It is worth to highlight that despite the M4 isolation feature belongs primarily to STM32MP1 security, it is applied for safety purposes in this specific case.

Temporal separation

The temporal separation is required because of the specific boot structure of STM32MP1, where the A7 CPU is the key player during the boot sequence because it is in charge to load the software image for M4 CPU. The temporal separation guarantees that any possible interference from non-safety related hardware and/or software on A7 CPU side during the boot sequence is detected/mitigated by adequate measures implemented on M4 CPU side (and therefore built over safety related hardware and software). Following picture provides a graphical representation (red boxes: events on non-safety related resources, green boxes: events on safety related resources)

Figure 6. Temporal separation concept



The box marked “M4 measures” includes both hardware and software measures. Hardware measures can be implemented by final system characteristics helping to keep the safe state despite issues occurred during the boot (for example the presence of an intelligent external watchdog), while software measures can be implemented by additional checks executed on M4 CPU just after its startup and before the execution of the end user safety application software. These measures are detailed in [Section 3.7 Conditions of use](#) in form of CoU.

It is worth to note that the authentication features implemented in first stage boot loader (FSBL) and that can be activated in second stage boot loader (U-Boot), despite mainly conceived for security reasons, can be considered as part of the temporal separation implementation because they protect the integrity of the software images loaded during the boot. That protection represented in [Figure 6. Temporal separation concept](#) by the yellow boxes, can be considered valid as per the outcomes of CoU_17 application.

3.3 Safety analysis assumptions

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

3.3.1 Safety requirement assumptions

The concept specification, the hazard and risk analysis, the overall safety requirement specification and the consequent allocation determine the requirements for *Compliant item* as further listed. *ASR* stands for assumed safety requirements.

Caution: It is the *End user's* responsibility to check the compliance of the final application with these assumptions. Furthermore, beside the recommendation included in this Safety Manual, it is also *End user's* responsibility to guarantee the compliance of the final application with the requirements of the IEC61508.

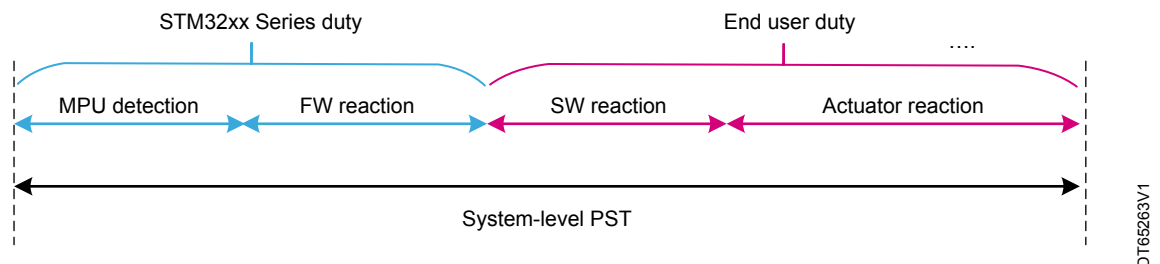
ASR1: *Compliant item* can be used to implement four kinds of safety function modes of operation according to part 4,3.5.16:

- a continuous mode (CM) or high-demand (HD) SIL3 safety function (CM3), or
- a low-demand (LD) SIL3 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 MPU to detect and react to a failure. That time corresponds to the portion of the process safety time (PST) allocated to the device (STM32xx Series duty in Figure 7) in error reaction chain at system level.

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 and peripherals) actually declared as safety-related. Further constraints and requirements from IEC61508-2, 7.4.5.3 must be considered.

Figure 7. Allocation and target for STM32 PST



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

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

ASR4.1: It is assumed that End User safety functions are implemented only on Device logic belonging to the Safe Partition.

ASR4.2: 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.

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

ASR4.4: It is assumed that logic and resources belonging to Safe Partition are not used for the implementation of non-safety related functions, coexisting with safety functions.

ASR4.5: It is assumed that the software functions running on A7 CPU have been developed according some specific quality standards.

Note: This Assumption is not placed to require a formal compliance of A7 software development flow to IEC61508-3 model, but to guarantee a minimum quality of the software running on the Non-Safe partition. Related CoU_17 provides more specific insights.

ASR5: It is assumed that the implemented safety function(s) does (do) not depend on transition of the overall STM32MP1 or Arm® Cortex®-M4 CPU to and from a low-power or suspended state.

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

- SS1: the application software⁽¹⁾ is informed by the presence of a fault and a reaction by the application software⁽¹⁾ itself is possible.
- SS2: the application software⁽¹⁾ cannot be informed by the presence of a fault or the application software⁽¹⁾ is not able to execute a reaction.

Note: *End user must consider that random hardware failures affecting the Device can compromise its operations (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	The application software ⁽¹⁾ is informed by the presence of a fault and a reaction by the application software itself is possible.	Fault reporting to application software ⁽¹⁾	Application software ⁽¹⁾ drives the overall system in its safe state	Application software ⁽¹⁾ in one of the two channels drives the overall system in its safe state
SS2	The application software ⁽¹⁾ cannot be informed by the presence of a fault or the 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") ⁽²⁾	PEv drives the overall system in its safe state

1. Referred to application software executed on Arm® Cortex®-M4 CPU.

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

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

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

Note: *Refer to Systematic safety integrity and Hardware and software diagnostics.*

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

ASR9.1: it is assumed that STM32MP1 is not used to build fail-operational solutions based exclusively on the resources of a unique STM32MP1 device itself.

ASR10.1: STM32MP1 package is considered full safety related, in the framework of Device failure rate computations.

ASR10.2: is End User responsibility to prove the freedom from interferences between safety related and non-safety related physical pins (e.g. by running a pin-level FMEDA).

ASR11: it is assumed that glitches in GPIO output values with a duration lower than the adopted PST are not able to cause violations of the implemented safety function(s)

Note: *This assumption can be fulfilled in two ways, either:*

- *Final application robustness against GPIO glitches, as required by the assumption text*
or
- *The execution frequency of the prescribed method GPIO_SM_2 and GPIO_SM_0 is higher than 1/Tm (refer to related "Recommendations and known limitations" fields for details)*

ASR12: It is assumed that data exchanges between the Safe Partition and the Non-Safe Partition are implemented by using statically allocated locations in SYSRAM bank and restricted to non-safety related data.

3.4

Electrical specifications and environment limits

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

- absolute maximum rating
- capacity

- operating conditions

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

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

3.5 Systematic safety integrity

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

3.6 Hardware and software diagnostics

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

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

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

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

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

SM CODE	Unique safety mechanism code/identifier used also in <i>FMEA</i> document. Identifiers use the scheme <i>mmm_SM_x</i> where <i>mmm</i> is a 3- or 4-letter module (function, peripheral) short name, and <i>x</i> 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. <i>End user</i> : method must be implemented by <i>End user</i> through <i>Application software</i> 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 <i>Application software</i> .
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: <ul style="list-style-type: none"> • If ranked for <i>Fault avoidance</i>: method contributes to lower the probability of occurrence of a failure • If ranked for <i>Systematic</i>: method is conceived to mitigate systematic errors (bugs) in <i>Application software</i> design
Dependency on Device configuration	Reports if safety mechanism implementation or characteristics change among different <i>Device</i> part numbers.
Initialization	Specific operation to be executed to activate the contribution of the safety mechanism
Periodicity	Continuous : safety mechanism is active in continuous mode.

Periodic: safety mechanism is executed periodically⁽¹⁾.

On-demand: safety mechanism is activated in correspondence to a specified event (for instance, reception of a data message).

Startup: safety mechanism to be executed only at power-up or during off-line maintenance periods. This is due to functional-only aspects or due to the poor compatibility with the correct execution of the safety function.

Test for the diagnostic

Reports specific procedure (if any and recommended) to allow on-line tests of safety mechanism efficiency. If no specific procedure applies (as for the majority of safety mechanisms), the field indicates *Not applicable*.

Multiple-fault protection

Reports the safety mechanism(s) associated in order to correctly manage a multiple-fault scenario (refer to [Section 4.1.3 Notes on multiple-fault scenario](#)).

Recommendations and known limitations

Additional recommendations or limitations (if any) not reported in other fields.

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

3.6.1

Arm® Cortex®-M4 CPU

Table 3. CPU_SM_0

SM CODE	CPU_SM_0
Description	Periodic core self-test software for Arm® Cortex®-M4 CPU.
Ownership	End user or ST (X-CUBE-STL, see Appendix A)
Detailed implementation	The software test is built around well-known techniques already addressed by IEC 61508-7, A.3.2 (Self-test by software: walking bit one-channel). To reach the required values of coverage, the self-test software is specified by means of a detailed analysis of all the CPU failure modes and related failure modes distribution.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on Device configuration	None
Initialization	None
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according 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	<p>This method is the main asset in STM32MP1 series safety concept. Hardware integrity of the CPU is a key factor, given that the defined diagnostics for MPU peripherals are to major part software-based.</p> <p>Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario for details.</p>

Table 4. CPU_SM_1

SM CODE	CPU_SM_1
Description	Control flow monitoring in Application software.
Ownership	End user
Detailed implementation	<p>A significant part of the failure distribution of <i>CPU</i> 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 <i>Application software</i> 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.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> • Different internal states of <i>Application software</i> 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 <i>Application software</i> is implemented. • Transition through all expected states during the normal <i>Application software</i> 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 <i>CPU</i> 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. <p>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).</p>
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 <i>Device</i> 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 5. CPU_SM_2

SM CODE	CPU_SM_2
Description	Double computation in <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	<p>A timing redundancy for safety-related computation is considered to detect transient faults affecting the Arm®Cortex®-M4 CPU subparts devoted to mathematical computations and data access.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> 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 <i>Application software</i> 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 <i>Device</i> 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	<p><i>End user</i> 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.</p> <p>Reduction to the application scope for this method is achieved by executing an accurate safety analysis of the software. Refer to [4] for details. However, the scope reduction may not be possible nor desirable.</p>

Table 6. CPU_SM_3

SM CODE	CPU_SM_3
Description	Arm®Cortex®-M4 HardFault exceptions
Ownership	ST
Detailed implementation	<p>HardFault exception raise is an intrinsic safety mechanism implemented in Arm®Cortex®-M4 core, mainly dedicated to intercept systematic faults due to software limitations or error in software design (causing for example execution of undefined operations, unaligned address access). This safety mechanism is also able to detect hardware random faults inside the CPU bringing to such described abnormal operations.</p>
Error reporting	High-priority interrupt event
Fault detection time	Depends on implementation. Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	None
Periodicity	Continuous
Test for the diagnostic	It is possible to write a test procedure to verify the generation of the HardFault exception; anyway, given the expected minor contribution in terms of hardware random-failure detection, such implementation is 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 <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	<p>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.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> 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 <i>Device</i> 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 IEC 61508 for software development. Therefore in presence of <i>Application software</i> 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	<i>End user</i>
Detailed implementation	<p>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 <i>CPU</i>.</p> <p>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.</p> <p>Compared to the <i>MPU</i> internal watchdogs, it is not affected by potential common cause failures, because the external watchdog is clocked and supplied independently of <i>Device</i>.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	To be defined at system level (outside the scope of <i>Compliant item</i> analysis).
Multiple-fault protection	CPU_SM_1: Control flow monitoring in Application software
Recommendations and known limitations	<p>In case of usage of windowed watchdog, <i>End user</i> must consider possible tolerance in <i>Application software</i> execution to avoid false error reports (affecting system availability).</p> <p>It is worth noting that the use of an external watchdog is needed when <i>Device</i> is used to trigger final elements, in order to comply at system level with requirements from IEC 61508-2:2010 Table A.1/Table A.14.</p>

Table 9. CPU_SM_6

SM CODE	CPU_SM_6
Description	MCU window 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 <i>CPU</i> .
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on <i>Device</i> configuration	None
Initialization	IWDG activation. It is recommended to use <i>hardware watchdog</i> 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	<p>CPU_SM_1: Control flow monitoring in Application software</p> <p>WDG_SM_0: Periodic read-back of configuration registers</p>
Recommendations and known limitations	<p>The IWDG intervention is able to achieve a potentially “incomplete” local safe state because it can only guarantee that <i>CPU</i> is reset. No guarantee that <i>Application software</i> 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, <i>End user</i> must adopt CPU_SM_5.</p>

Table 10. CPU_SM_7

SM CODE	CPU_SM_7
Description	Memory protection unit (<i>MPU</i>).
Ownership	ST
Detailed implementation	The <i>CPU</i> memory protection unit is able to detect illegal access to protected memory areas, according to criteria set by <i>End user</i> .
Error reporting	Exception raise (MemManage).
Fault detection time	Refer to functional documentation
Addressed fault model	Systematic (software errors) Permanent/transient (only program counter and memory access failures)
Dependency on <i>Device</i> configuration	None
Initialization	<i>MPU</i> registers must be programmed at start-up.
Periodicity	On line
Test for the diagnostic	MPU_SM_1: <i>MPU</i> software test
Multiple-fault protection	MPU_SM_0: Periodic read-back of <i>MPU</i> configuration registers
Recommendations and known limitations	<p>The use of memory partitioning and protection by <i>MPU</i> functions is highly recommended when multiple safety functions are implemented in <i>Application software</i>. The <i>MPU</i> can be indeed used to</p> <ul style="list-style-type: none"> enforce privilege rules separate processes enforce access rules <p>Hardware random-failure detection capability for <i>MPU</i> is restricted to well-selected failure modes, mainly affecting program counter and memory access <i>CPU</i> functions. The associated diagnostic coverage is therefore not expected to be relevant for the safety concept of <i>Device</i>.</p>

Table 11. MPU_SM_0

SM CODE	MPU_SM_0
Description	Periodic read-back of <i>MPU</i> configuration registers
Ownership	<i>End user</i>
Detailed implementation	<p>This method must be applied to <i>MPU</i> configuration registers (also unused by <i>End user</i> <i>Application software</i>).</p> <p>Detailed information on the implementation of this method can be found in Section 3.6.10 Extended interrupt and events controller (EXTI).</p>
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 <i>Device</i> 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 12. MPU_SM_1

SM CODE	MPU_SM_1
Description	MPU software test
Ownership	End user
Detailed implementation	<p>This method tests MPU capability to detect and report memory accesses violating the policy enforcement implemented by the MPU itself.</p> <p>The implementation is based on intentionally performing read and write accesses outside the memory areas allowed by the MPU region programming, and collecting and verifying related generated error exceptions.</p> <p>Test can be executed with the final MPU region programming or with a dedicated one.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on Device configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario .

3.6.2 System bus architecture/BusMatrix

Table 13. BUS_SM_0

SM CODE	BUS_SM_0
Description	Periodic software test for interconnections
Ownership	End user
Detailed implementation	<p>The intra-chip connection resources (Bus Matrix, AHB or APB bridges) needs to be periodically tested for permanent faults detection. Note that STM32MP1 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.</p> <p>According to IEC 61508-2 Table A.8, A.7.4 the method is considered able to achieve high levels of coverage.</p>
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 <i>Periodic read-back of configuration registers</i> required for several peripherals.

Table 14. BUS_SM_1

SM CODE	BUS_SM_1
Description	Information redundancy in intra-chip data exchanges
Ownership	<i>End user</i>
Detailed implementation	<p>This method requires to add some kind of redundancy (for example a <i>CRC</i> checksum at packet level) to each data message exchanged inside <i>Device</i>.</p> <p>Message integrity is verified using the checksum by the <i>Application software</i>, before consuming data.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 15. RAM_SM_0

SM CODE	RAM_SM_0
Description	Periodic software test for static random access memory (SRAM)
Ownership	<i>End user</i> or ST (X-CUBE-STL, see Appendix A)
Detailed implementation	To enhance the coverage on SRAM data cells and to ensure adequate coverage for permanent faults affecting the address decoder it is required to execute a periodic software test on the system RAM memory. The selection of the algorithm must ensure the target SFF coverage for both the RAM cells and the address decoder. Evidences of the effectiveness of the coverage of the selected method must also be collected
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on <i>Device</i> 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	<p>Usage of a March test C- is recommended.</p> <p>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).</p> <p>Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 4.1.3 Notes on multiple-fault scenario.</p> <p>Unused RAM section can be excluded by the testing, under <i>End user</i> responsibility on actual RAM usage by final <i>Application software</i>.</p> <p>RAM sections hosting application software or diagnostic libraries can be excluded by the testing with this method, if correctly protected by the dedicated safety mechanism RAM_SM_5. Indeed, the diagnostic coverage given by RAM_SM_5 permits avoiding the overlap of the two methods on the same RAM area.</p>

Table 16. RAM_SM_2

SM CODE	RAM_SM_2
Description	Stack hardening for <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	<p>The stack hardening method is used to enhance the <i>Application software</i> 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 <i>Application software</i> structure and the compiler settings requires a significant use of the stack for passing function parameters.</p> <p>Implementation is the same as method CPU_SM_4.</p>
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 <i>Device</i> 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 17. RAM_SM_3

SM CODE	RAM_SM_3
Description	Information redundancy for safety-related variables in the <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	<p>To address transient faults affecting the SRAM controller and memory cells, it is required to implement information redundancy on the safety-related system variables stored in the SRAM.</p> <p>The guidelines for the implementation of this method are the following:</p> <ul style="list-style-type: none"> • The system variables that are safety-related (in the sense that a wrong value due to a failure in reading on the RAM affects the safety functions) are well-identified and documented. • The arithmetic computation or decision based on such variables are executed twice and the two final results are compared. • Safety-related variables are stored and updated in two redundant locations, and comparison is checked before consuming data. • Enumerated fields must use non-trivial values, checked for coherence with the same frequency as for periodically executed diagnostics (see ⁽¹⁾ in Section 3.6 Hardware and software diagnostics). • Data vectors stored in SRAM must be protected by an encoding checksum (such as CRC).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	<p>Implementation of this safety method shows a partial overlap with an already foreseen method for Arm® Cortex®-M4 (CPU_SM_2) ; optimizations in implementing both methods are therefore possible.</p> <p>Reduction to the application scope for this method is achieved by executing an accurate safety analysis of the software. Refer to [4] for details. However, the scope reduction may not be possible nor desirable.</p>

Table 18. RAM_SM_4

SM CODE	RAM_SM_4
Description	Control flow monitoring in <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	In case <i>End user Application software</i> 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 <i>Device</i> 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 <i>Application software</i> execution from SRAM. CPU_SM_1 correct implementation supersedes this requirement.

Table 19. RAM_SM_5

SM CODE	RAM_SM_5
Description	Periodic integrity test for <i>Application software</i> in RAM
Ownership	<i>End user</i>
Detailed implementation	Because application software and diagnostic libraries are executed in RAM, it is needed to protect the integrity of the code itself against soft-error corruptions and related code mutations. This method must check the integrity of the stored code by checksum computation techniques, on a periodic basis (at least once per PST), or another timing constraint; refer to ⁽¹⁾ in Section 3.6 Hardware and software diagnostics. RAM memory cell contents are then checked versus the expected values, using signature-based techniques. According to IEC 61508:2 Table A.5, the effective diagnostic coverage of such techniques depends on the width of the signature in relation to the block length of the information to be protected - therefore the signature computation method must be carefully selected. Note that the simple signature method (IEC 61508:7 - A.4.2 Modified checksum) is inadequate as it only achieves a low value of coverage. The use of internal hardware CRC module is therefore recommended.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 <i>Application software</i>
Recommendations and known limitations	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen.

Table 20. RAM_SM_9

SM CODE	RAM_SM_9
Description	SRAM static data encapsulation
Ownership	End user
Detailed implementation	If static data are stored in SRAM, encapsulation by a checksum field with encoding capability (such as CRC) must be implemented. Checksum validity is checked by application software before static data consuming.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple-fault protection	CPU_SM_0: periodical core self test software
Recommendations and known limitations	None

3.6.4 Power controller (PWR)

Table 21. VSUP_SM_0

SM CODE	VSUP_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 22. 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 <i>Device</i> configuration	None
Initialization	Protection enable by the PVDE bit and the threshold setting in the Power control register (PWR_CR)
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for PVD efficiency is not available. PVD run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field <i>Multiple-fault protection</i> .
Multiple-fault protection	DIAG_SM_0: Periodic read-back of hardware diagnostics configuration registers
Recommendations and known limitations	<p>Internal monitoring PVD has limited capability to address failures affecting STM32MP1 series internal voltage regulator. Refer to [1] for details.</p> <p>Internal monitoring PVD has limited capability to address failures affecting the internal voltage regulator. Refer to <i>Device FMEA</i> for details.</p> <p>In case the hardware option is not available on the chosen partnumbers, its contribution to the overall safety concept is supported by other overlapping methods indicated for the mitigation of failures affecting internal power.</p>

Table 23. VSUP_SM_2

SM CODE	VSUP_SM_2
Description	External watchdog
Ownership	ST
Detailed implementation	Failures in the power supplies for digital logic (core or peripherals) may lead to alteration of <i>Application software</i> timing, which can be detected by IWDG as safety mechanism introduced to monitor the <i>Application software</i> 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 <i>Device</i> configuration	None
Initialization	IWDG activation. It is recommended to use <i>Hardware watchdog</i> 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	<p>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.</p> <p>The adoption of an external watchdog (refer to CPU_SM_5) adds further diversity.</p>

Table 24. VSUP_SM_3

SM CODE	VSUP_SM_3
Description	Internal temperature sensor check
Ownership	<i>End user</i>
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 <i>Device</i> 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 the <i>Device</i> . Refer to the <i>Device</i> datasheet to set the threshold temperature.

Table 25. VSUP_SM_5

SM CODE	VSUP_SM_5
Description	System-level power supply management
Ownership	<i>End user</i>
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): <ul style="list-style-type: none"> • 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 <i>Device</i> 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.

Table 26. VSUP_SM_6

SM CODE	VSUP_SM_6
Description	PWR TrustZone® security
Ownership	ST
Detailed implementation	<p>This method is able to secure PWR configuration registers from being modified by non-secure accesses.</p> <p>This safety mechanism prevents configuration changes for PWR registers; it addresses therefore systematic faults in software application.</p> <p>The use of this method is encouraged to enhance the end-application robustness for systematic faults in software, and to contribute to the isolation concept implementation.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	None (systematic only)
Dependency on <i>Device</i> configuration	None
Initialization	NA
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	Not needed
Recommendations and known limitations	None

3.6.5 Reset and clock controller (RCC)

Table 27. CLK_SM_0

SM CODE	CLK_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	<p>This method must be applied to configuration registers for clock and reset system (refer to RCC register map).</p> <p>Detailed information on the implementation of this method can be found in Section 3.6.10 Extended interrupt and events controller (EXTI).</p>
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 <i>Device</i> 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 28. CLK_SM_1

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

Table 29. CLK_SM_2

SM CODE	CLK_SM_2
Description	External watchdog
Ownership	ST
Detailed implementation	The independent watchdog IWDG is able to detect failures in internal main <i>MPU</i> clock (lower frequency).
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on <i>Device</i> configuration	None
Initialization	IWDG activation. It is recommended to use the <i>hardware watchdog</i> 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 30. CLK_SM_3

SM CODE	CLK_SM_3
Description	Internal clock cross-measurement
Ownership	End user
Detailed implementation	This method is implemented using general-purpose timers capabilities to be fed by the 32 KHz RTC clock or an external clock source (if available). Timer counter progress is compared with another counter (fed by internal clock). Abnormal values of oscillator frequency can therefore be detected.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on <i>Device</i> 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.

Table 31. CLK_SM_4

SM CODE	CLK_SM_4
Description	RCC TrustZone® function
Ownership	ST
Detailed implementation	This method is able to secure RCC configuration registers from being modified by non-secure accesses. This safety mechanism prevents configuration changes for RCC 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 in software, and to contribute to the isolation concept implementation.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	None (systematic only)
Dependency on <i>Device</i> configuration	None
Initialization	NA
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	Not needed
Recommendations and known limitations	None

3.6.6 General-purpose input/output (GPIO)

Table 32. GPIO_SM_0

SM CODE	GPIO_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to GPIO configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> configuration	GPIO availability can differ according to part number
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	The execution of the method before any update on GPIO registers helps to mitigate the possibility of unintended glitches on outputs due to soft errors. For more information refer to [4].

Table 33. GPIO_SM_1

SM CODE	GPIO_SM_1
Description	1002 for input GPIO lines
Ownership	<i>End user</i>
Detailed implementation	This method addresses GPIO lines used as inputs. Implementation is done by connecting the external safety-related signal to two independent GPIO lines. Comparison between the two GPIO values is executed by the <i>Application software</i> each time the signal is used to affect <i>Application software</i> behavior. This method applies to the single GPIO line used as input.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	<p>To reduce the potential impact of common cause failure, it is recommended to use GPIO lines:</p> <ul style="list-style-type: none"> • 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 <p>As GPIO pins are shared with other <i>MPU</i> functions, this method must not be applied to pin connections already used by another peripheral and addressed by related safety mechanisms.</p>

Table 34. GPIO_SM_2

SM CODE	GPIO_SM_2
Description	Loopback scheme for output GPIO lines
Ownership	End user
Detailed implementation	This method addresses GPIO lines used as outputs. Implementation is done by a loopback scheme, connecting the output to a different GPIO line programmed as input and by using the input line to check the expected value on output port. Comparison is executed by the <i>Application software</i> periodically and each time output is updated. This method applies to the single GPIO line used as output.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	<p>To reduce the potential impact of common cause failure, it is recommended to use GPIO lines:</p> <ul style="list-style-type: none"> • 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 <p>Efficiency versus transient failures is linked to final application characteristics. We define as T_m 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 $1/T_m$.</p> <p>As GPIO pins are shared with other <i>MPU</i> functions, this method must not be applied to pin connections already used by another peripheral and addressed by related safety mechanisms.</p>

Table 35. GPIO_SM_3

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

3.6.7 Debug system or peripheral control

Table 36. 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 <i>CPU</i> 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 <i>Device</i> 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 37. LOCK_SM_0

SM CODE	LOCK_SM_0
Description	Lock mechanism for configuration options
Ownership	ST
Detailed implementation	The STM32MP1 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).
Fault detection time	Not applicable
Addressed fault model	None (systematic only)
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	Not required
Recommendations and known limitations	No <i>DC</i> associated because this test addresses systematic faults.

3.6.8 System configuration controller (SYSCFG)

Table 38. SYSCFG_SM_0

SM CODE	SYSCFG_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
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 3.6.10 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 <i>Device</i> 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 <i>MPU</i> peripherals. It is reported here for the sake of completeness.

Table 39. DIAG_SM_0

SM CODE	DIAG_SM_0
Description	Periodic read-back of hardware diagnostics configuration registers
Ownership	<i>End user</i>
Detailed implementation	In STM32MP1 series, several hardware-based safety mechanisms are available (those with the <i>Ownership</i> field set to ST). This method must be applied to any configuration register related to diagnostic measure operations, including error reporting. <i>End user</i> must therefore individuate configuration registers related to: <ul style="list-style-type: none"> hardware diagnostic enable interrupt/NMI enable (if used for diagnostic error management)
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed fault model	Refer to NVIC_SM_0
Dependency on <i>Device</i> 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

3.6.9 Direct memory access controller (DMA/ DMA2D/ DMAMUX)

Table 40. DMA_SM_0

SM CODE	DMA_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to <i>DMA</i> configuration register and channel address register. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 41. DMA_SM_1

SM CODE	DMA_SM_1
Description	Information redundancy on data packet transferred via <i>DMA</i>
Ownership	<i>End user</i>
Detailed implementation	This method is implemented by adding, to data packets transferred by <i>DMA</i> , a redundancy check (such as <i>CRC</i> 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 <i>Application software</i> before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 give an example about checksum encoding capability, using just a bit-by-bit addition is inappropriate.

Table 42. DMA_SM_2

SM CODE	DMA_SM_2
Description	Information redundancy by including sender or receiver identifier on data packet transferred via <i>DMA</i>
Ownership	<i>End user</i>
Detailed implementation	<p>This method helps to identify inside the MCU the source and the originator of the message exchanged by <i>DMA</i>.</p> <p>Implementation is realized by adding an additional field to protected message, with a coding convention for message type identification fixed at <i>Device</i> level. Guidelines for the identification fields are:</p> <ul style="list-style-type: none"> • Identification field value must be different for each possible couple of sender or receiver on <i>DMA</i> transactions. • Values chosen must be enumerated and non-trivial. • Coherence between the identification field value and the message type is checked by the <i>Application software</i> before consuming data. <p>This method, when implemented in combination with DMA_SM_4, makes available a kind of <i>virtual channel</i> between source and destinations entities.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 43. DMA_SM_3

SM CODE	DMA_SM_3
Description	Periodic software test for <i>DMA</i>
Ownership	<i>End user</i>
Detailed implementation	<p>This method requires the periodical testing of the <i>DMA</i> 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:</p> <ul style="list-style-type: none"> • incomplete packed transfer • errors in single transferred word • wrong order in packed transmitted data
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on <i>Device</i> 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 44. DMA_SM_4

SM CODE	DMA_SM_4
Description	DMA transaction awareness
Ownership	End user
Detailed implementation	<p>DMA transactions are non-deterministic by nature, because typically driven by external events like communication messages reception. Anyway, well-designed safety systems should keep much control as possible of events – refer for instance to IEC 61508-3 Table 2 item 13 requirements for software architecture.</p> <p>This method is based on system knowledge of frequency and type of expected DMA transaction. For instance, an externally connected sensor supposed to send periodically some messages to a STM32 peripheral. Monitoring DMA transaction by a dedicated state machine allows the detection of missing or unexpected DMA activities.</p>
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.

3.6.10 Extended interrupt and events controller (EXTI)

Table 45. NVIC_SM_0

SM CODE	NVIC_SM_0
Description	Periodic read-back of configuration registers
Ownership	End user
Detailed implementation	<p>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 <i>PST</i> (or another timing constraint; refer to ⁽¹⁾ in Section 3.6 Hardware and software diagnostics) after an update of the peripheral.</p> <p>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.</p> <p>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).</p> <p>An alternative valid implementation requiring less space in SRAM can be realized on the basis of signature concept:</p> <ul style="list-style-type: none"> Peripheral registers to be checked are read in a row, computing a <i>CRC</i> checksum (use of hardware <i>CRC</i> 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 <i>Application software</i> – 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 <i>Device</i> 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	<p>This method addresses only failures affecting configuration registers, and not peripheral core logic or external interface.</p> <p>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.</p>

Table 46. NVIC_SM_1

SM CODE	NVIC_SM_1
Description	Expected and unexpected interrupt check
Ownership	End user
Detailed implementation	<p>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 <i>Application software</i> level.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> • The interrupts implemented on the <i>MPU</i> are well documented, also reporting, when possible, the expected frequency of each request (for example, the interrupts related to ADC conversion completion that come on a regular basis). • Individual counters are maintained for each interrupt request served, in order to detect in a given time frame the cases of a) no interrupt at all b) too many interrupt requests. The control of the time frame duration must be regulated according to the individual interrupt expected frequency. • Interrupt vectors related to unused interrupt source point to a default handler that reports, in case of triggering, a faulty condition (unexpected interrupt). • In case an interrupt service routine is shared between different sources, a plausibility check on the caller identity is implemented. <p><i>Important:</i> <i>Interrupt requests generated by non-safety-related peripherals must be handled using the same method as all safety related interrupts outlined in the list above.</i></p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 extension of the method to non-safety related peripherals (see last bullet in "Detailed implementation" box above) is introduced to mitigate interferences between non-safety and safety functions/hardware (FFI).

3.6.11 Cyclic redundancy-check calculation unit (CRC2)

Table 47. CRC_SM_0

SM CODE	CRC_SM_0
Description	CRC2 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 <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.12 Flexible static memory controller (FSMC)

Table 48. FSMC_SM_0

SM CODE	FSMC_SM_0
Description	Control flow monitoring in <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	<p>If FSMC is used to connect an external memory containing software code to be executed by the CPU, permanent and transient faults affecting the FSMC memory controller are able to interfere with the access operation by the CPU, leading to wrong data or instruction fetches. A strong control flow mechanism linked to a system watchdog is able to detect such failures, in case they interfere with the expected flow of <i>Application software</i>.</p> <p>The implementation of this method is identical to the one reported for CPU_SM_1, refer there for details.</p>
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 <i>Device</i> configuration	FSMC interface is available only on selected part numbers.
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	This mechanism must only be used if FSMC external memory is used to store executable programs.

Table 49. FSMC_SM_1

SM CODE	FSMC_SM_1
Description	Information redundancy on external memory connected to FSMC
Ownership	<i>End user</i>
Detailed implementation	<p>If FSMC interface is used to connect an external memory where safety-relevant data are stored, information redundancy techniques for stored data are able to address faults affecting the FSMC interface. The possible techniques are:</p> <ul style="list-style-type: none"> • using redundant copies of safety-relevant data and performing coherence check before consuming • organizing data in arrays and computing the checksum field to check before use
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	FSMC interface is available only on selected part numbers.
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	<p>This mechanism must be used just if FSMC external memory is used to store safety-related data.</p> <p>This safety mechanism can overlap with information redundancy techniques implemented at system level to address failure of physical device connected to FSMC port.</p>

Table 50. FSMC_SM_2

SM CODE	FSMC_SM_2
Description	Periodic read-back of FSMC configuration registers
Ownership	<i>End user</i>
Detailed implementation	<p>This method must be applied to FSMC configuration registers.</p> <p>Detailed information on the implementation of this method can be found in Section 3.6.10 Extended interrupt and events controller (EXTI).</p>
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 <i>Device</i> configuration	FSMC interface is available only on selected part numbers.
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 51. FSMC_SM_3

SM CODE	FSMC_SM_3
Description	ECC engine on NAND interface in FSMC module
Ownership	ST
Detailed implementation	The FMC NAND Card controller includes two error correction code computation hardware blocks, one per memory bank. They reduce the host CPU workload when processing the ECC by software. ECC mechanism protects data integrity on the external memory connected to NAND port.
Error reporting	Refer to functional documentation
Fault detection time	ECC bits are checked during memory reading.
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	FSMC interface is available only on selected part numbers.
Initialization	None
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	FSMC_SM_2: Periodic read-back of FSMC configuration registers
Recommendations and known limitations	This method has negligible efficiency in detecting hardware random failures affecting the FSMC interface. It can be part of <i>End user</i> safety concept because addressing memories outside STM32MP1 series MCU.

3.6.13 Quad-SPI interface and Octo-SPI interface (QUADSPI/OCTOSPI)

Table 52. QSPI_SM_0

SM CODE	QSPI_SM_0
Description	Periodic read-back of QUADSPI/OCTOSPI configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to QUADSPI/OCTOSPI configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 53. QSPI_SM_1

SM CODE	QSPI_SM_1
Description	Protocol error signals including hardware CRC
Ownership	ST
Detailed implementation	QUADSPI/OCTOSPI communication module embeds protocol error checks (like overrun, underrun, timeout and so on), conceived to detect communication-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field <i>Multiple-fault protection</i> .
Multiple-fault protection	QSPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 54. QSPI_SM_2

SM CODE	QSPI_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i>
Detailed implementation	This method is implemented adding to data packets (not commands) transferred by QUADSPI/OCTOSPI interface a redundancy check (like a CRC check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by <i>Application software</i> before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. This safety mechanism can overlap with information redundancy techniques implemented at system level to address failure of physical device connected to QUADSPI/OCTOSPI port.

3.6.14 Delay block (DLYB)

Table 55. DLB_SM_0

SM CODE	DLB_SM_0
Description	Periodic read-back of DLYB configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to DLYB configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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

Note: *It is assumed that DLYB output, if used, will feed STM32MP1 internal communication peripherals (like, for instance, QUADSPI). It is also assumed that for the connected peripherals all prescript safety mechanisms (rated as ++ and +) are correctly implemented.*

3.6.15 Analog-to-digital converter (ADC)

Table 56. ADC_SM_0

SM CODE	ADC_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to the ADC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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. ADC_SM_1

SM CODE	ADC_SM_1
Description	Multiple acquisition by <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	This method implements a timing information redundancy by executing multiple acquisitions on the same input signal. Multiple data acquisitions are then combined by a filter algorithm to determine the signal correct value.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Depends on implementation
Dependency on <i>Device</i> 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 the <i>End userApplication software</i> . Usage of multiple acquisitions followed by average operations is a common technique in industrial applications exposed to electromagnetic interference on sensor lines.

Table 58. ADC_SM_2

SM CODE	ADC_SM_2
Description	Range check by <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	<p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> The expected range of the data to be acquired are investigated and adequately documented. Note that in a well-designed application it is improbable that during normal operation an input signal has a very near or over the upper and lower rail limit (saturation in signal acquisition). If the <i>Application software</i> 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 <i>Device</i> 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 59. ADC_SM_3

SM CODE	ADC_SM_3
Description	Periodic software test for ADC
Ownership	<i>End user</i>
Detailed implementation	<p>The method is implemented acquiring multiple signals and comparing the read value with the expected one, supposed to be known. Method can be implemented with different level of complexity:</p> <ul style="list-style-type: none"> 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 <i>Device</i> 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	Combination of two methods with different complexity can be used to better optimize test frequency in high-demand safety functions.

Table 60. ADC_SM_4

SM CODE	ADC_SM_4
Description	1002 scheme for ADC inputs
Ownership	<i>End user</i>
Detailed implementation	This safety mechanism is implemented using two different SAR ADC channels belonging to separate ADC modules to acquire the same input signal. The <i>Application software</i> checks the coherence between the two readings.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	ADC_SM_0: Periodic read-back of configuration registers
Recommendations and known limitations	This method can be used in conjunction with ADC_SM_0 / ADC_SM_2 / ADC_SM_3 to achieve highest level of ADC module diagnostic coverage.

3.6.16 Digital-to-analog converter (DAC)

Table 61. DAC_SM_0

SM CODE	DAC_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to DAC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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. DAC_SM_1

SM CODE	DAC_SM_1
Description	DAC output loopback on ADC channel
Ownership	<i>End user</i>
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 <i>Device</i> 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 T_m 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 $1/T_m$.

3.6.17 Voltage reference buffer (VREFBUF)

Table 63. VREF_SM_0

SM CODE	VREF_SM_0
Description	Periodic read-back of VREFBUF system configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to VREFBUF configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	Refer to NVIC_SM_0

Table 64. VREF_SM_1

SM CODE	VREF_SM_1
Description	VREF cross-check by ADC reading
Ownership	<i>End user</i>
Detailed implementation	This method is based on ADC acquisition for VREF generated signal, to crosscheck with the expected value.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on <i>Device</i> 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	Overlaps with ADC_SM_3 are possible.

3.6.18 Digital filter for sigma delta modulators (DFSDM)

Table 65. DFS_SM_0

SM CODE	DFS_SM_0
Description	Periodic read-back of DFSDM configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to DFSDM configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 66. DFS_SM_1

SM CODE	DFS_SM_1
Description	Multiple acquisition by <i>Application software</i>
Ownership	<i>End user</i>
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	Transient
Dependency on <i>Device</i> 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 <i>End userApplication software</i> . Usage of multiple acquisitions followed by average operations is a common technique in industrial applications where it is needed to survive with spurious EMI disturbs on sensor lines.

Table 67. DFS_SM_2

SM CODE	DFS_SM_2
Description	Range check by <i>Application software</i>
Ownership	<i>End user</i>
Detailed implementation	This method is implemented as described in ADC_SM_2: Range check by <i>Application software</i> .
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Not applicable
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	The implementation of this safety mechanism is strongly application-dependent.

Table 68. DFS_SM_3

SM CODE	DFS_SM_3
Description	1oo2 scheme for DFSDM inputs
Ownership	<i>End user</i>
Detailed implementation	This safety mechanism is implemented using two different DFSDM modules to acquire the same input signal. The <i>Application software</i> checks the coherence between the two readings.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	DFS_SM_0: Periodic read-back of DFSDM configuration registers
Recommendations and known limitations	This method can be used in conjunction with DFS_SM_0 to achieve highest level of DFSDM module diagnostic coverage (as an alternative to DFS_SM_1 and DFS_SM_2).

3.6.19 Digital camera interface (DCMI)

Table 69. DCMI_SM_0

SM CODE	DCMI_SM_0
Description	Periodic read-back of DCMI configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to DCMI configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> configuration	DCMI interface is available only on selected part numbers.
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 70. DCMI_SM_1

SM CODE	DCMI_SM_1
Description	DCMI video input data synchronization
Ownership	ST
Detailed implementation	According to the nature of video data stream received, DCMI module implements synchronization controls, from the simplest one (hardware synchronization) to the most complex (e.g. embedded data synchronization mode). DCMI internal failures leading to the incapability of correcting synchronizing the data stream can be therefore detected.
Error reporting	No explicit error signal/message generation is provided (*).
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on <i>Device</i> configuration	DCMI interface is available only on selected part numbers.
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	DCMI_SM_0: Periodic read-back of DCMI configuration registers
Recommendations and known limitations	(*) For its nature, the detection of an actual hardware failure by this safety mechanism can be confused with functional-related scenarios (e.g. camera device disconnected or powered-off). It is responsibility of <i>Application software</i> to discriminate, as far as it is technically possible, among different events.

3.6.20 HASH processor (HASH)

Table 71. HASH_SM_0

SM CODE	HASH_SM_0
Description	Periodic read-back of HASH configuration registers
Ownership	End user
Detailed implementation	This method must be applied to HASH configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> configuration	HASH module available only on specific part numbers
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 72. HASH_SM_1

SM CODE	HASH_SM_1
Description	HASH processing collateral detection
Ownership	ST
Detailed implementation	Message digest computation performed by HASH module is composed by several data manipulations and checks. A major part of the hardware random failures affecting HASH module leads to algorithm violations/errors, and so to decoding errors on the receiver side.
Error reporting	Several error condition can happens, check functional documentation.
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	HASH module available only on specific part numbers
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for HASH efficiency is not available. HASH run-time hardware failures leading to disabling related collateral protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field Multiple-fault protection.
Multiple-fault protection	HASH_SM_0: Periodic read-back of HASH configuration registers CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	This detection capability can be used to implement software-based tests (by processing a predefined message and further checking the expected results) which can be executed periodically to early detect HASH failures before its use by application software.

Note: Hardware random failures consequences on potential security features violations are **not** analyzed in this manual.

3.6.21 True random number generator (RNG)

Table 73. RNG_SM_0

SM CODE	RNG_SM_0
Description	Periodic read-back of RNG configuration register
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to RNG configuration register RNG_CR. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> configuration	RNG module available only on specific part numbers
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 74. RNG_SM_1

SM CODE	RNG_SM_1
Description	RNG module entropy on-line tests
Ownership	ST and <i>End user</i>
Detailed implementation	RNG module include an internal diagnostic for the analog source entropy that can be used to detect failures on the module itself. Furthermore, the required test on generated random number difference between the previous one (as required by FIPS PUB 140-2) can be exploited as well. Implementation: <ul style="list-style-type: none"> • Check for RNG error conditions. • Check the difference between generated random number and the previous one.
Error reporting	CEIS, SEIS error bits of the RNG status register (RNG_SR) <i>Application software</i> error for FIPS PUB 140-2 test fail
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	RNG module available only on specific part numbers
Initialization	Permanent/transient
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	None

3.6.22 Cryptographic processor (CRYP)

Table 75. CRYP_SM_0

SM CODE	CRYP_SM_0
Description	Periodic read-back of CRYP configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to CRYP configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> configuration	CRYP module available only on specific part numbers
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 76. CRYP_SM_1

SM CODE	CRYP_SM_1
Description	Encryption/decryption collateral detection
Ownership	ST
Detailed implementation	Encryption and decryption operations performed by CRYP module are composed by several data manipulations and checks, with different level of complexity according to the selected chaining algorithm. A major part of the hardware random failures affecting CRYP module leads to algorithm violations/errors. Leading to decoding errors on the receiver side.
Error reporting	Several error conditions can happen, check functional documentation.
Fault detection time	Dependency on <i>Device</i> configuration
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	CRYP module available only on specific part numbers
Initialization	Dependency on <i>Device</i> configuration
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRYP efficiency is not available. CRYP run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field <i>Multiple-fault protection</i> .
Multiple-fault protection	CRYP_SM_2: Information redundancy techniques on messages, including end-to-end protection
Recommendations and known limitations	None

Table 77. CRYP_SM_2

SM CODE	CRYP_SM_2
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	<i>End user</i>
Detailed implementation	This method aim to protect the communication between a peripheral and his external counterpart. It is used in CRYP local safety concept to address failures not detected by the encryption/decryption features. Refer to UART_SM_3 description for detailed information.
Error reporting	Refer to UART_SM_3
Fault detection time	Refer to UART_SM_3
Addressed fault model	Refer to UART_SM_3
Dependency on <i>Device</i> configuration	CRYP module available only on specific part numbers
Initialization	Refer to UART_SM_3
Periodicity	Refer to UART_SM_3
Test for the diagnostic	Refer to UART_SM_3
Multiple-fault protection	Refer to UART_SM_3
Recommendations and known limitations	Important note: it is assumed that the remote counterpart has an equivalent capability of performing the checks described. Refer to UART_SM_3 for further notice.

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

3.6.23 Advanced-control/General-purpose and low-power timers

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

Table 78. ATIM_SM_0

SM CODE	ATIM_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
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 3.6.10 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 <i>Device</i> 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 79. ATIM_SM_1

SM CODE	ATIM_SM_1
Description	1oo2 for counting timers
Ownership	<i>End user</i>
Detailed implementation	<p>This method implements via software a 1oo2 scheme between two counting resources.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> Two timers are programmed with same time base or frequency. In case of timer use as a time base: use in <i>Application software</i> 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 <i>Device</i> 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	<p>Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic.</p> <p>This method applies to timer channels merely used as elapsed time counters.</p> <p>Events related to timers protected by the safety mechanisms must be monitored inside the routine managing the external watchdog (CPU_SM_5) reset.</p> <p><i>Note:</i> <i>One timer may act as a reference for multiple other timers.</i></p>

Table 80. ATIM_SM_2

SM CODE	ATIM_SM_2
Description	1002 for input capture timers
Ownership	<i>End user</i>
Detailed implementation	<p>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.</p> <p>Coherence check between timers is executed each time the reading is used by <i>Application software</i>.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 81. ATIM_SM_3

SM CODE	ATIM_SM_3
Description	Loopback scheme for pulse width modulation (PWM) outputs
Ownership	<i>End user</i>
Detailed implementation	<p>This method is implemented by connecting the PWM to a separate timer channel to acquire the generated waveform characteristics.</p> <p>The guidelines are the following:</p> <ul style="list-style-type: none"> 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. <p>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.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Depends on implementation
Dependency on <i>Device</i> 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 T_m 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 $1/T_m$.

Table 82. ATIM_SM_4

SM CODE	ATIM_SM_4
Description	Lock bit protection for timers
Ownership	ST
Detailed implementation	This safety mechanism allows <i>End user</i> to lock down specified configuration options, thus avoiding unintended modifications by <i>Application software</i> . Therefore, it addresses software development systematic faults.
Error reporting	Not applicable
Fault detection time	Not applicable
Addressed fault model	None (Fault avoidance)
Dependency on <i>Device</i> 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.*

3.6.24 Basic timers

Table 83. GTIM_SM_0

SM CODE	GTIM_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
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 3.6.10 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 <i>Device</i> 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 84. GTIM_SM_1

SM CODE	GTIM_SM_1
Description	1oo2 for counting timers
Ownership	<i>End user</i>
Detailed implementation	<p>This method implements via software a 1oo2 scheme between two counting resources. The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> Two timers are programmed with same time base or frequency. In case of timer use as a time base: use in <i>Application software</i> 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 <i>“reference”</i> 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 <i>Device</i> 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	<p>Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic.</p> <p>Events related to timers protected by the safety mechanisms must be monitored inside the routine managing the external watchdog reset.</p> <p><i>Note:</i> <i>One timer may act as a reference for multiple other timers.</i></p>

3.6.25 Independent and system window watchdogs (IWDG and WWDG)

Table 85. WDG_SM_0

SM CODE	WDG_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to IWDG/WWDG configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 86. WDG_SM_1

SM CODE	WDG_SM_1
Description	Software test for watchdog at startup
Ownership	<i>End user</i>
Detailed implementation	This safety mechanism ensures the right functionality of the internal watchdogs in use. The test implementation allows the application software to induce a watchdog reset for a specific purpose such as at startup, and to determine that the cause of the reset was the test procedure itself, and not a software/hardware malfunction. This is confirmed by reading the associated hardware flag in the RCC status register before and after the test and applying specific SW flag, which stores nontrivial pattern at SRAM, just during the test execution. Both the <i>HW</i> and <i>SW</i> flags must be cleared once the test is done. This is essential to avoid repeating the test in a loop, and to correctly manage watchdog resets related to failures.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Startup
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	In a typical <i>End user</i> application, this test can be executed only at startup and during maintenance or offline periods. It could be associated to IEC 61508 concept of “proof test” and so it cannot be accounted for a diagnostic coverage contribution during operating time.

3.6.26 Real-time clock module (RTC)

Table 87. RTC_SM_0

SM CODE	RTC_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to RTC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 88. RTC_SM_1

SM CODE	RTC_SM_1
Description	Application check of running RTC
Ownership	<i>End user</i>
Detailed implementation	The <i>Application software</i> 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: <ul style="list-style-type: none"> RTC backup registers are used to store coded information in order to detect the absence of VBAT during power-off period. RTC backup registers are used to periodically store compressed information on current date or time The <i>Application software</i> executes minimal consistence checks for date reading after power-on (detecting "past" date or time retrieve). The <i>Application software</i> 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 <i>Device</i> 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	This method provides a limited diagnostic coverage for RTC failure modes. In case of <i>End user</i> 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 89. RTC_SM_2

SM CODE	RTC_SM_2
Description	Information redundancy on backup registers
Ownership	<i>End user</i>
Detailed implementation	Data stored in RTC backup registers must be protected by a checksum with encoding capability (for instance, CRC). Checksum must be checked by application software before consuming stored data. This method guarantees data versus erases due to backup battery failures.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	None

Table 90. RTC_SM_3

SM CODE	RTC_SM_3
Description	Application-level measures to detect failures in timestamps/event capture
Ownership	<i>End user</i>
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 <i>Device</i> 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 <i>MPU</i> in Sleep or Stop mode is prevented by the assumed requirement ASR7 (refer to Section 3.3.1 Safety requirement assumptions).

3.6.27 Tamper and backup registers (TAMP)

Table 91. TAMP_SM_0

SM CODE	TAMP_SM_0
Description	Information redundancy on tamper backup registers
Ownership	<i>End user</i>
Detailed implementation	Data stored in tamper backup registers must be protected by a checksum with encoding capability (for instance, <i>CRC</i>). Checksum must be checked by <i>Application software</i> before consuming stored data. This method guarantees data versus erases due to backup battery failures.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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

3.6.28 Inter-integrated circuit (I2C)

Table 92. IIC_SM_0

SM CODE	IIC_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to I2C configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 93. IIC_SM_1

SM CODE	IIC_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	I2C communication module embeds protocol error checks (like overrun, underrun, packet error etc.) conceived to detect network-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 94. IIC_SM_2

SM CODE	IIC_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i>
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 <i>Application software</i> before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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. <i>Important:</i> This method must be considered as a subset of IIC_SM_4. Therefore, the implementation of IIC_SM_4 completely overlap this method. Refer to [4] for additional details.

Table 95. 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 <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field <i>Multiple-fault protection</i> .
Multiple-fault protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	<p>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 <i>Test for the diagnostic</i> field, this mechanism can not be the only one to guarantee message integrity.</p> <p>Enabling related interrupt generation on the detection of errors is highly recommended.</p>

Table 96. IIC_SM_4

SM CODE	IIC_SM_4
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	<i>End user</i>
Detailed implementation	<p>This method aims to protect the communication between a I2C peripheral and his external counterpart.</p> <p>Refer to UART_SM_3 description for detailed information.</p>
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 <i>Device</i> 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	<p>It is assumed that the remote I2C counterpart has an equivalent capability of performing the checks described.</p> <p>Refer to UART_SM_3 for further notice.</p>

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

Table 97. 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 and LPUART configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 98. UART_SM_1

SM CODE	UART_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	USART and LPUART communication module embeds protocol error checks (like additional parity bit check, overrun, frame error) conceived to detect network-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself. Error signals connected to these checkers are normally handled in a standard communication software, so the overhead is reduced.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 and LPUART communication module allows several different configurations. The actual composition of communication error checks depends on the selected configuration. Enabling related interrupt generation on the detection of errors is highly recommended.

Table 99. UART_SM_2

SM CODE	UART_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i>
Detailed implementation	<p>This method is implemented by adding to data packets transferred by this peripheral a redundancy check (such as a <i>CRC</i> 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.</p> <p>Consistency of data packet must be checked by <i>Application software</i> before consuming data.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	<p>It is assumed that the remote counterpart has an equivalent capability of performing the check described.</p> <p>To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated.</p> <p><i>Important:</i> <i>This method must be considered as a subset of UART_SM_3. Therefore, the implementation of UART_SM_3 completely overlap this method. Refer to [4] for additional details.</i></p>

Table 100. UART_SM_3

SM CODE	UART_SM_3
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	<i>End user</i>
Detailed implementation	<p>This method aims to protect the communication between a peripheral and his external counterpart establishing a kind of “protected” channel. The aim is to specifically address communication failure modes as reported in IEC 61508-2, 7.4.11.1.</p> <p>Implementation guidelines are as follows:</p> <ul style="list-style-type: none"> • 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. • <i>Application software</i> 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 <i>Device</i> 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	<p>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.</p> <p>It is assumed that the remote counterpart has an equivalent capability of performing the checks described.</p>

3.6.30 Serial peripheral interface (SPI)

Table 101. SPI_SM_0

SM CODE	SPI_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to SPI configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 102. SPI_SM_1

SM CODE	SPI_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	SPI communication module embeds protocol error checks (like overrun, underrun, timeout and so on) conceived to detect network-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 103. SPI_SM_2

SM CODE	SPI_SM_2
Description	Information redundancy techniques on messages
Ownership	End user
Detailed implementation	<p>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.</p> <p>Consistency of data packet must be checked by <i>Application software</i> before consuming data.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	<p>It is assumed that the remote counterpart has an equivalent capability of performing the check described.</p> <p>To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated.</p> <p><i>Important:</i> This method must be considered as a subset of SPI_SM_4. Therefore, the implementation of SPI_SM_4 completely overlap this method. Refer to [4] for additional details.</p>

Table 104. 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 <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field <i>Multiple-fault protection</i> .
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 <i>Test for the diagnostic</i> field, this mechanism can not be the only one to guarantee message integrity.

Table 105. SPI_SM_4

SM CODE	SPI_SM_4
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	<i>End user</i>
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 <i>Device</i> 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.

3.6.31 Serial audio interface (SAI)

Table 106. SAI_SM_0

SM CODE	SAI_SM_0
Description	Periodic read-back of SAI configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to SAI configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 107. SAI_SM_1

SM CODE	SAI_SM_1
Description	SAI output loopback scheme
Ownership	<i>End user</i>
Detailed implementation	This method uses a loopback scheme to detect permanent and transient faults on the output channel used for serial audio frame generation. It is implemented by connecting the second serial audio interface as input for primary output generation. <i>Application software</i> is able therefore to identify wrong or missing audio frame generation.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous/ On demand
Test for the diagnostic	Not applicable
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	<p>Efficiency versus transient failures is linked to final application characteristics. We define as T_m the minimum duration of serial audio wrong signal permanence required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than $1/T_m$.</p> <p>Method to be used when SAI interface safety-related use is <i>audio stream generation</i>.</p>

Table 108. SAI_SM_2

SM CODE	SAI_SM_2
Description	1002 scheme for SAI module
Ownership	<i>End user</i>
Detailed implementation	This safety mechanism is implemented using the two SAI interfaces to decode/receive the same input stream audio. <i>Application software</i> checks the coherence between the received data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	<p>The MCU performance overload and the implementation complexity associated to this method can be relevant.</p> <p>Method to be used when SAI interface safety-related use is <i>audio stream receive</i>.</p>

3.6.32 SPDIF receiver interface (SPDIFRX)

Table 109. SPDF_SM_0

SM CODE	SPDF_SM_0
Description	Periodic read-back of SPDIF configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to SPDIF configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 110. SPDF_SM_1

SM CODE	SPDF_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	IEC60598 S/PDIF data frame specification used in SPDIF interface embeds protocol error checks (like overrun, underrun, bit timing violations, parity, etc.) conceived to detect transmission-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 implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	SPDF_SM_0: Periodic read-back of SPDIF configuration registers
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 111. SPDF_SM_2

SM CODE	SPDF_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i>
Detailed implementation	This method is implemented adding to data S/PDIF data stream some form of information redundancy, possibly including information repetition, to address failure modes affecting the decoding section of the module.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 could be replaced by application-level alternative measures checking the correctness of the audio stream received. One given example could be represented by a set of plausibility checks executed after post-elaboration by voice recognition algorithms.

3.6.33 Management data input/output (MDIOS)

Table 112. MDIO_SM_0

SM CODE	MDIO_SM_0
Description	Periodic read-back of MDIO slave configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to MDIO slave configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple-fault protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 113. MDIO_SM_1

SM CODE	MDIO_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	MDIO communication protocol is based on a packet handling concept, including preamble/ start/stop correct conditions checks. This mechanism, mainly implemented to manage on field communication disturbance, is able to achieve a relevant diagnostic coverage on several MDIO module failure modes.

SM CODE	MDIO_SM_1
Error reporting	Error conditions are reported by flag bits in related registers, and interrupt generation.
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Permanent/transient
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	DSI_SM_0: Periodic read-back of DSI configuration registers
Recommendations and known limitations	Not applicable

Table 114. MDIO_SM_2

SM CODE	MDIO_SM_2
Description	Information redundancy techniques on MDIO registers contents, including register update awareness
Ownership	<i>End user</i>
Detailed implementation	<p>Information provided by external parties by MDIO communication must be protected by redundancy schemes (encoded data values and possibly the definition of a <i>checksum</i> register).</p> <p><i>Application software</i> must be aware of any register value update executed by external parties, so it is needed the implementation of a <i>validate/invalidate</i> mechanism to:</p> <ul style="list-style-type: none"> report to external party that updated data have been consumed mark as <i>invalidated</i> any data already consumed allow external party to inform <i>Application software</i> that new data are available
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not required
Multiple-fault protection	CPU_SM_0: Periodic core self-test software
Recommendations and known limitations	It is assumed that the external entity responsible to update/send data to <i>Application software</i> by the MDIO communication link is able to contribute to the MDIO failure mitigation, by detecting missing or incomplete data consumption.

3.6.34 Secure digital input/output MultiMediaCard interface (SDMMC)

Table 115. SDIO_SM_0

SM CODE	SDIO_SM_0
Description	Periodic read-back of SDIO/SMMC configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to SDIO/SMMC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 116. SDIO_SM_1

SM CODE	SDIO_SM_1
Description	Protocol error signals including hardware CRC
Ownership	ST
Detailed implementation	SDIO/SMMC communication module embeds protocol error checks (like overrun, underrun, timeout etc.) and CRC-packet checks as well, conceived to detect network-related abnormal conditions. These mechanisms are only able to detect a small fraction of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation.
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not applicable
Multiple-fault protection	SDIO_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 117. SDIO_SM_2

SM CODE	SDIO_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i>
Detailed implementation	<p>This method is implemented adding to data packets transferred by SDIO/SMMC a redundancy check (like a CRC check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet.</p> <p>Consistency of data packet must be checked by <i>Application software</i> before consuming data.</p>
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	<p>To give an example on checksum encoding capability, using just a bit-by-bit addition is inappropriate.</p> <p>This safety mechanism can overlap with information redundancy techniques implemented at system level to address failure of physical device connected to SDIO/SMMC port.</p>

Note: *The safety mechanisms mentioned above are addressing the SDIO/SMMC interface included in STM32 MCUs. No claims are done in this Safety Manual about the mitigation of hardware random faults affecting the external memory connected to SDIO/SMMC port.*

3.6.35 Secure digital input/output MultiMediaCard interface (SDMMC3)

Table 118. SDIO_SM_0

SM CODE	SDIO_SM_0
Description	Periodical read-back of SDMMC3 configuration registers
Ownership	<i>End user</i>
Detailed implementation	<p>This method must be applied to SDMMC3 configuration registers.</p> <p>Detailed information on the implementation of this method can be found in EXTI controller.</p>
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 <i>Device</i> 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 faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 119. SDIO_SM_1

SM CODE	SDIO_SM_1
Description	Protocol error signals including hardware CRC
Ownership	ST
Detailed implementation	SDMMC3 communication module embeds protocol error checks (such as overrun, underrun, timeout) and CRC-packet checks as well, conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a percentage of hardware random failures affecting the module itself
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection time	Depends on peripheral configuration (for example baud rate), refer to functional documentation
Addressed fault model	Permanent and transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults protection	SDIO_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 120. SDIO_SM_2

SM CODE	SDIO_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i>
Detailed implementation	This method is implemented adding to data packets transferred by SDMMC3 a redundancy check (like a CRC check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: Periodical core self test software
Recommendations and known limitations	To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. This safety mechanism can overlap with information redundancy techniques implemented at system level to address failure of physical device connected to SDIO/SMMC port

Note: *The safety mechanisms mentioned above are addressing the SDMMC3 interface included in STM32MP1. No claims are done in this Safety Manual about the mitigation of hardware random faults affecting the external memory connected to SDMMC3 port.*

3.6.36 Controller area network (FDCAN)

Table 121. CAN_SM_0

SM CODE	CAN_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to FDCAN configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 122. 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 only able to detect a small fraction of hardware random failures affecting the module itself. Error signals connected to these checkers are normally handled in a standard communication software, so the overhead is reduced.
Error reporting	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 <i>Device</i> 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 123. CAN_SM_2

SM CODE	CAN_SM_2
Description	Information redundancy techniques on messages, including end-to-end protection.
Ownership	<i>End user</i>
Detailed implementation	<p>This method aims to protect the communication between a peripheral and his external counterpart establishing a kind of “protected” channel. The aim is to specifically address communication failure modes as reported in IEC 61508-2, 7.4.11.1.</p> <p>Implementation guidelines are as follows:</p> <ul style="list-style-type: none"> • 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. • <i>Application software</i> 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 <i>Device</i> 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	<p>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.</p> <p>It is assumed that the remote counterpart has an equivalent capability of performing the checks described.</p>

3.6.37 USB on-the-go (OTG)

Table 124. USB_SM_0

SM CODE	USB_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to USB configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 125. 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 only able to detect a small fraction of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate). Refer to functional documentation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 126. USB_SM_2

SM CODE	USB_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i> 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 <i>Device</i> 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	<i>Important: This method must be considered as a subset of USB_SM_3. Therefore, the implementation of USB_SM_3 completely overlap this method. Refer to [4] for additional details.</i>

Table 127. USB_SM_3

SM CODE	USB_SM_3
Description	Information redundancy techniques on messages, including end-to-end protection.
Ownership	<i>End user</i>
Detailed implementation	This method aims to protect the communication between the OTG 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 <i>Device</i> 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.

3.6.38 Ethernet (ETH): media access control (MAC) with DMA controller

Table 128. ETH_SM_0

SM CODE	ETH_SM_0
Description	Periodic read-back of Ethernet configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to Ethernet configuration registers (including those relate to unused module features). Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 129. ETH_SM_1

SM CODE	ETH_SM_1
Description	Protocol error signals including hardware CRC
Ownership	ST
Detailed implementation	Ethernet communication module embeds protocol error checks (like overrun, underrun, timeout, packet composition violation etc.) and CRC-packet checks as well, conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a 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 <i>Device</i> configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Direct test procedure for CRC efficiency is not available. CRC run-time hardware failures leading to disabling such protection fall into multiple-fault scenario, from IEC 61508 perspective. Related failures are adequately mitigated by the combination of safety mechanisms reported in this table, field <i>Multiple-fault protection</i> .
Multiple-fault protection	ETH_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 130. ETH_SM_2

SM CODE	ETH_SM_2
Description	Information redundancy techniques on messages, including end-to-end protection
Ownership	<i>End user</i>
Detailed implementation	This method aim to protect the communication between a peripheral and its external counterpart. It is used in Ethernet local safety concept to address failures not detected by ETH_SM_1 and to increase its associated diagnostic coverage. Refer to UART_SM_3 description for detailed information.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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	The implementation on <i>Application software</i> of complex Ethernet-based communication stacks (like TCP/IP) is able to satisfy the requirements of this method.

Note: The use of the DMA feature inside Ethernet module requires the adoption of the set of safety mechanisms defined for DMA (refer to [Section 3.6.9 Direct memory access controller \(DMA/ DMA2D/ DMAMUX\)](#))).

3.6.39 HDMI-CEC (CEC)

Table 131. HDMI_SM_0

SM CODE	HDMI_SM_0
Description	Periodic read-back of configuration registers
Ownership	<i>End user</i>
Detailed implementation	This method must be applied to CEC configuration registers. Detailed information on the implementation of this method can be found in Section 3.6.10 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 <i>Device</i> 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 132. HDMI_SM_1

SM CODE	HDMI_SM_1
Description	Protocol error signals
Ownership	ST
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 <i>Device</i> 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 133. HDMI_SM_2

SM CODE	HDMI_SM_2
Description	Information redundancy techniques on messages
Ownership	<i>End user</i>
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 <i>Application software</i> before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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.

3.6.40 Boot and security and OTP control (BSEC)

Table 134. BSEC_SM_0

SM CODE	BSEC_SM_0
Description	Periodical read-back of BSEC configuration registers
Ownership	End User
Detailed implementation	This method must be applied to BSEC configuration registers, to check the correct expected configuration for the module (for example debug disabled, lock protection enable). Detailed information on the implementation of this method can be found in EXTI controller.
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 faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 135. BSEC_SM_1

SM CODE	BSEC_SM_1
Description	Hardware protection for OTP words
Ownership	ST
Detailed implementation	OTP lower region is protected by a 2:1 redundancy OTP upper region is protected by ECC (error correction code) redundancy, implementing a protection feature at double-word (64-bit) level.
Error reporting	Errors in OTP words, or any reading anomaly in values load, are reported in appropriated OTP word status.
Fault detection Time	ECC bits are checked during their load into shadow registers.
Addressed Fault Model	Permanent/transient
Dependency on device configuration	None
Initialization	None
Periodicity	On demand
Test for the diagnostic	Direct test procedure for ECC efficiency or 2:1 redundancy check is not available. Diagnostics run-time hardware failures leading to disabling such protection, or to wrong corrections, 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 faults protection	BSEC_SM_2: application software measures for end user configuration check. DIAG_SM_0: periodic read-back of hardware diagnostics configuration registers. CPU_SM_5: external watchdog.
Recommendations and known limitations	Further mitigation for latent failures affecting the 2:1 redundancy check comes from the application of CoU_10.

Table 136. BSEC_SM_2

SM CODE	BSEC_SM_2
Description	OTP read/write protection
Ownership	ST
Detailed implementation	BSEC module features several methods to protect OTP words from unintended writes (permanent write lock), disturbed read operation, unintended shadow register reload (sticky registers).
Error reporting	Refer to functional documentation.
Fault detection Time	Refer to functional documentation.
Addressed Fault Model	Systematic
Dependency on device configuration	None
Initialization	Not required
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults protection	BSEC_SM_0 Periodical read-back of BSEC configuration registers.
Recommendations and known limitations	Correct enable of these protection must be checked by BSEC_SM_0 method.

Table 137. BSEC_SM_3

SM CODE	BSEC_SM_3
Description	Application software measures for end user configuration check.
Ownership	End user
Detailed implementation	OTP upper words usually contain end user configuration data for connectivity and security peripherals. Because of marginal risk of failures not addressed by hardware-implemented protections (BSEC_SM_1), a further check on the correctness of those loaded data is required. Significant overlap may exist with already implemented safety recommendations - for instance, end-to-end protection methods on communication peripherals may detect in efficient way wrong peripherals configuration due to OPT load failure.
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	NA
Multiple faults protection	CPU_SM_0: periodic core self-test software.
Recommendations and known limitations	None

3.6.41 Extended TrustZone® protection controller (ETZPC)

Table 138. ETZPC_SM_0

SM CODE	ETZPC_SM_0
Description	Periodical read-back of eTZPC system configuration registers
Ownership	End User
Detailed implementation	This method must be applied to eTZPC configuration registers, including the programming for the implementation of the separation for isolated peripherals (exclusively allocated to Arm® Cortex®-M4 CPU). Refer to Section 3.2.5 The separation concept for details on the isolation concept.

SM CODE	ETZPC_SM_0
	Detailed information on the implementation of this method can be found in EXTI controller.
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 faults protection	CPU_SM_0: Periodical core self test software
Recommendations and known limitations	Startup execution of this safety mechanism is highly recommended - refer to Section 4.2.7 STM32MP1 boot process for details. Note that eTZPC programming is executed during boot phase by A7 software.

Table 139. ETZPC_SM_1

SM CODE	ETZPC_SM_1
Description	Periodical software test for ETZPC
Ownership	End User
Detailed implementation	The method can be implemented by accessing in reading to all eTZPC configuration/status registers and checking them against the expected values. Method implementation is overlapped with ETPC_SM_0.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on device configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: Periodical core self test software
Recommendations and known limitations	None

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

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

Table 140. FFI_SM_0

SM CODE	FFI_SM_0
Description	Disable of unused peripherals
Ownership	<i>End user</i>
Detailed implementation	<p>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.</p> <p>After the system boot, <i>Application software</i> must disable all unused peripherals with this procedure:</p> <ul style="list-style-type: none"> • 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 <i>Device</i> 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 141. FFI_SM_1

SM CODE	FFI_SM_1
Description	Periodic read-back of interference avoidance registers
Ownership	<i>End user</i>
Detailed implementation	<p>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:</p> <ul style="list-style-type: none"> • clock enable and disable registers • alternate function programming registers <p>Detailed information on the implementation of this method can be found in Section 3.6.10 Extended interrupt and events controller (EXTI).</p>
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 <i>Device</i> 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 142. FFI_SM_2

SM CODE	FFI_SM_2
Description	eTZPC protection on MCU isolated peripherals/resources
Ownership	ST and End User
Detailed implementation	In the framework of the isolation concept between the STM32MP1 Safe Partition and Non Safe Partition (refer to section 3.2.5), eTZPC protect MCU allocated peripherals and resources from unintended write access by AXI masters like A7 CPU
Error reporting	Error reporting is not available (M4 CPU is not informed that a forbidden master access has been stopped)
Fault detection Time	NA
Addressed Fault Model	Permanent, Transient and Systematic for software running on A7 CPU
Dependency on device configuration	None
Initialization	eTZPC must be correctly configured to protect MCU isolated peripherals. eTZPC programming is done by A7 CPU during boot, therefore the prerequisite is successful execution of FFI_SM_3
Periodicity	Continuous
Test for the diagnostic	ETZPC_SM_0: Periodical read-back of eTZPC system configuration registers
Multiple faults protection	Additional layers of safety mechanism implemented on each safety relate peripherals as required by CoU_4
Recommendations and known limitations	-

Table 143. FFI_SM_3

SM CODE	FFI_SM_3
Description	Startup check for system configuration
Ownership	End User
Detailed implementation	Because several configuration registers used for the STM32MP1 safety concept (and isolation concept) are programmed by A7 CPU during boot phase, this method checks that the programmed value into such registers are compliant with the expected ones. Also application software integrity is checked, because depending on correct boot sequence execution. Basically, this method is implemented by the startup execution of following methods (elsewhere already described): CLK_SM_0 VSUP_SM_0 SYSCFG_SM_0 ETZPC_SM_0 RAM_SM_5
Error reporting	Depends on implementation - this method is executed when the end user application software is not yet started, therefore the only entity to be informed could be the external watchdog.
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent, Transient Addresses also systematic errors of the boot software running on A7 CPU.
Dependency on device configuration	None
Initialization	NA
Periodicity	Startup only
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: Periodical core self test software.

SM CODE	FFI_SM_3
Recommendations and known limitations	<p>Because some checked configuration registers are not accessible in writing by M4 CPU (for example eTZPC ones), if this method fails the application software must not start because a restore of the expected values could be not possible. There are two main possibility then, either:</p> <ul style="list-style-type: none"> to stay indefinitely in power-up safe state with the combination of external watchdog intervention (see CoU_10), or to rely in external watchdog to keep system safe state (see CoU_10), and to apply for an overall STM32MP1 reboot.

3.6.43 System

Table 144. DUAL_SM_0

SM CODE	DUAL_SM_0
Description	Cross-check between two STM32 devices
Ownership	<i>End user</i>
Detailed implementation	<p>This method is implemented in the spirit of technique described in IEC 61508-7, A.3.5 "Reciprocal comparison by software", which is rated in IEC 61508-2 Table A.4 as capable to achieve high level of diagnostic coverage.</p> <p>The two processing units exchange data reciprocally, and a fail in the comparison is considered as a detection of a failure in one of the two unit. The guidelines for the implementation are the following:</p> <ul style="list-style-type: none"> Data exchanged include output results, intermediate results⁽¹⁾ and the results (pass/fail) of each software-implemented safety mechanisms executed on periodical basis on both MCUs (for example CPU_SM_0) Software routines devoted to data exchange/comparison must be logically separated from the software implementing the safety function(s). Systematic capability of software implementing this method must be equal or above the one of the software implementing the safety function(s). Independence and lack of interference between the software implementing the data exchange/comparison and the one implementing the safety function(s) must be proven. Frequency of data exchange/comparison is imposed by the system PST (refer to related timing constraints for periodic safety mechanisms), except for output results which needs to be exchanged/compared at the same rate they are potentially updated.
Error reporting	-
Fault detection time	Depends on implementation
Addressed fault model	Permanent/transient
Dependency on <i>Device</i> 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 (individually executed on both processing units)
Recommendations and known limitations	<p>This method is usually rated as optional because it is not strictly needed in the framework of 1oo2 architecture described in Section 3.2.4 Reference safety architectures - 1oo2. Anyway, it is included here only for its use in such an architecture.</p> <p>This method can provide additional safety margin for systems that need further protection against fault accumulation.</p> <p>Because this method could be a potential source of common cause failure between the two 1oo2 channels (in case of incorrect implementation), <i>End user</i> is recommended to closely follow the Detailed implementation guidelines in this table.</p>

1. the value of each variable able to directly influence the final individual channel output, such as:

- variables included in computation of the final result; for example, of a PWM rate
- variables involved in a decision determining the final result; for example, two variables used in a comparison which determines if a GPIO output is set high or low.

3.7 Conditions of use

The table below provides a summary of the safety concept recommendations reported in [Section 3.6 Hardware and software diagnostics](#). The conditions of use to be applied to STM32MP1 series devices are reported in form of safety mechanism requirements. Exception is represented by some conditions of use introduced by FMEA, FTA, and FFI analysis in order to correctly address specific failure modes and/or contribute to the implementation of the separation concept (refer to [Section 3.2.5 The separation concept](#) for details). These conditions of use are reported at the end of the table presented in this section.

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

- ++ The safety mechanism is highly recommended as common practice. It is considered in this document for the computation of safety metrics to allow the use of *Device* in systems implementing safety functions up to *SIL2* with a single *MPU* or up to *SIL3* with two *MPUs* in 1oo2 scheme. Missing implementation may lead to invalidate any safety feature claimed in this manual and must be supported by adequate arguments under end user responsibility (refer to [Section 4.1.1 Safety analysis result customization](#) for guidance).
- + The safety mechanism is recommended as additional safety measure, but not considered in this document for the computation of safety metrics. The *End user* can skip the implementation in case it is in contradiction with functional requirements or overlapped by another mechanism ranked ++.
- o The safety mechanism is optional. It is not strictly required for the implementation of safety functions up to *SIL2*, or it is related to a specific *MPU* configuration.

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

Attention: The full list of *Device* functions belonging to Safe Partition is included in the first column of [Table 145](#). Any *Device* function/module not listed there must be considered belonging to Non Safe Partition and therefore not suitable for the implementation of safety functions.

Table 145. List of safety recommendations

Diagnostic	Description	Rank	Perm	Trans	Separation
Arm® Cortex®-M4					
CPU_SM_0	Periodic core self-test software for Arm® Cortex®-M4 <i>CPU</i>	++	X	-	-
CPU_SM_1	Control flow monitoring in <i>Application</i> software	++	X	X	-
CPU_SM_2	Double computation in <i>Application</i> software	++	-	X	-
CPU_SM_3	Arm®Cortex®-M4 HardFault exceptions	++	X	X	-
CPU_SM_4	Stack hardening for <i>Application</i> software	++	X	X	-
CPU_SM_5	External watchdog	++	X	X	-
CPU_SM_6	MCU window watchdog	++	X	X	-
CPU_SM_7	Memory protection unit (<i>MPU</i>)	++	X	X	-
MPU_SM_0	Periodic read-back of <i>MPU</i> configuration registers	++	X	X	-
MPU_SM_1	<i>MPU</i> software test	+(1)	X	-	-
System bus architecture/BusMatrix					
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_2	Stack hardening for <i>Application</i> software	++	X	X	X
RAM_SM_3	Information redundancy for safety-related variables in the <i>Application</i> software	++	X	X	X

Diagnostic	Description	Rank	Perm	Trans	Separation
RAM_SM_4	Control flow monitoring in <i>Application software</i>	++	X	X	X
RAM_SM_5	Periodic integrity test for <i>Application software</i> in RAM	++	X	X	X
RAM_SM_9	SRAM static data encapsulation	++	X	X	-
Power controller (PWR)					
VSUP_SM_0	Periodic read-back of configuration registers	++	X	X	X
VSUP_SM_1	Supply voltage internal monitoring (PVD)	++	X	-	-
VSUP_SM_2	External watchdog	++	X	-	-
VSUP_SM_3	Internal temperature sensor check	++	-	-	-
VSUP_SM_5	System-level power supply management	++	-	-	-
VSUP_SM_6	PWR TrustZone® security	++	-	-	X
Reset and clock controller (RCC)					
CLK_SM_0	Periodic read-back of configuration registers	++	X	X	X
CLK_SM_1	Clock security system (CSS)	+	X	-	-
CLK_SM_2	External watchdog	++	X	-	-
CLK_SM_3	Internal clock cross-measurement	+	X	-	-
CLK_SM_4	RCC TrustZone® function	++	-	-	X
General-purpose input/output (GPIO)					
GPIO_SM_0	Periodic read-back of configuration registers	++	X	X	X
GPIO_SM_1	1002 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	++	-	-	X
Debug system or peripheral control					
DBG_SM_0	Watchdog protection	++	X	X	-
LOCK_SM_0	Lock mechanism for configuration options	++	-	-	X
System configuration controller (SYSCFG)					
SYSCFG_SM_0	Periodic read-back of configuration registers	++	X	X	-
DIAG_SM_0	Periodic read-back of hardware diagnostics configuration registers	++	X	X	-
Direct memory access controller (DMA/ DMA2D/ DMAMUX)					
DMA_SM_0	Periodic read-back of configuration registers	++	X	X	X
DMA_SM_1	Information redundancy on data packet transferred via <i>DMA</i>	++	X	X	-
DMA_SM_2	Information redundancy by including sender or receiver identifier on data packet transferred via <i>DMA</i>	++	X	X	-
DMA_SM_3	Periodic software test for <i>DMA</i>	++	X	-	-
DMA_SM_4	<i>DMA</i> transaction awareness	++	X	X	-
Extended interrupt and events controller (EXTI)					
NVIC_SM_0	Periodic read-back of configuration registers	++	X	X	X
NVIC_SM_1	Expected and unexpected interrupt check	++	X	X	-
Cyclic redundancy-check calculation unit (CRC2)					
CRC_SM_0	CRC2 self-coverage	++	X	X	-
Flexible static memory controller (FSMC)					
FSMC_SM_0	Control flow monitoring in <i>Application software</i>	++ ⁽²⁾	X	X	-

Diagnostic	Description	Rank	Perm	Trans	Separation
FSMC_SM_1	Information redundancy on external memory connected to FSMC	++ ⁽²⁾	X	X	-
FSMC_SM_2	Periodic read-back of FSMC configuration registers	++	X	X	-
FSMC_SM_3	ECC engine on NAND interface in FSMC module	o	X	X	-
Quad-SPI interface and Octo-SPI interface (QUADSPI/OCTOSPI)					
QSPI_SM_0	Periodic read-back of QUADSPI/OCTOSPI configuration registers	++	X	X	-
QSPI_SM_1	Protocol error signals including hardware CRC	++	X	X	-
QSPI_SM_2	Information redundancy techniques on messages	++	X	X	-
Delay block (DLYB)					
DLB_SM_0	Periodic read-back of DLYB configuration registers	++	X	X	-
Analog-to-digital converter (ADC)					
ADC_SM_0	Periodic read-back of configuration registers	++	X	X	-
ADC_SM_1	Multiple acquisition by <i>Application software</i>	++	-	X	-
ADC_SM_2	Range check by <i>Application software</i>	++	X	X	-
ADC_SM_3	Periodic software test for ADC	++	X	-	-
ADC_SM_4	1002 scheme for ADC inputs	+	X	X	-
Digital-to-analog converter (DAC)					
DAC_SM_0	Periodic read-back of configuration registers	++	X	X	-
DAC_SM_1	DAC output loopback on ADC channel	++	X	X	-
Voltage reference buffer (VREFBUF)					
VREF_SM_0	Periodic read-back of VREFBUF system configuration registers	++	X	X	-
VREF_SM_1	VREF cross-check by ADC reading	+	X	-	-
Digital filter for sigma delta modulators (DFSDM)					
DFS_SM_0	Periodic read-back of DFSDM configuration registers	++	X	X	-
DFS_SM_1	Multiple acquisition by <i>Application software</i>	++	-	X	-
DFS_SM_2	Range check by <i>Application software</i>	++	X	X	-
DFS_SM_3	1002 scheme for DFSM inputs	+	X	X	-
Digital camera interface (DCMI)					
DCMI_SM_0	Periodic read-back of DCMI configuration registers	++	X	X	-
DCMI_SM_1	DCMI video input data synchronization	++	X	X	-
True random number generator (RNG)					
RNG_SM_0	Periodic read-back of RNG configuration register	++	X	X	-
RNG_SM_1	RNG module entropy on-line tests	++	X	-	-
Cryptographic processor (CRYP)					
CRYP_SM_0	Periodic read-back of CRYP configuration registers	++	X	X	-
CRYP_SM_1	Encryption/decryption collateral detection	++	X	X	-
CRYP_SM_2	Information redundancy techniques on messages, including end-to-end protection	++	X	X	-
HASH processor (HASH)					
HASH_SM_0	Periodic read-back of HASH configuration registers	++	X	X	-
HASH_SM_1	HASH processing collateral detection	++	X	X	-
Advanced-control/General-purpose and low-power timers					
ATIM_SM_0	Periodic read-back of configuration registers	++	X	X	-

Diagnostic	Description	Rank	Perm	Trans	Separation
ATIM_SM_1	1002 for counting timers	++	X	X	-
ATIM_SM_2	1002 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					
GTIM_SM_0	Periodic read-back of configuration registers	++	X	X	-
GTIM_SM_1	1002 for counting timers	++	X	X	-
Independent and system window watchdogs (IWDG and WWDG)					
WDG_SM_0	Periodic read-back of configuration registers	++	X	X	-
WDG_SM_1	Software test for watchdog at startup	o	X	-	-
Real-time clock module (RTC)					
RTC_SM_0	Periodic read-back of configuration registers	++	X	X	-
RTC_SM_1	Application check of running RTC	++	X	X	-
RTC_SM_2	Information redundancy on backup registers	+	X	X	-
RTC_SM_3	Application-level measures to detect failures in timestamps/event capture	o	X	X	-
Tamper and backup registers (TAMP)					
TAMP_SM_0	Information redundancy on tamper backup registers	++	X	X	-
Inter-integrated circuit (I2C)					
IIC_SM_0	Periodic read-back of configuration registers	++	X	X	-
IIC_SM_1	Protocol error signals	++	X	X	-
IIC_SM_2	Information redundancy techniques on messages	++	X	X	-
IIC_SM_3	CRC packet-level	+	X	X	-
IIC_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X	-
Universal synchronous/asynchronous receiver/transmitter and low-power universal asynchronous receiver/transmitter (USART and LPUART)					
UART_SM_0	Periodic read-back of configuration registers	++	X	X	-
UART_SM_1	Protocol error signals	++	X	X	-
UART_SM_2	Information redundancy techniques on messages	++	X	X	-
UART_SM_3	Information redundancy techniques on messages, including end-to-end protection	++	X	X	-
Serial peripheral interface (SPI)					
SPI_SM_0	Periodic read-back of configuration registers	++	X	X	-
SPI_SM_1	Protocol error signals	++	X	X	-
SPI_SM_2	Information redundancy techniques on messages	++	X	X	-
SPI_SM_3	CRC packet-level	+	X	X	-
SPI_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X	-
Serial audio interface (SAI)					
SAI_SM_0	Periodic read-back of SAI configuration registers	++	X	X	-
SAI_SM_1	SAI output loopback scheme	++	X	X	-
SAI_SM_2	1002 scheme for SAI module	++	X	X	-

Diagnostic	Description	Rank	Perm	Trans	Separation
SPDIF receiver interface (SPDIFRX)					
SPDF_SM_0	Periodic read-back of SPDIF configuration registers	++	X	X	-
SPDF_SM_1	Protocol error signals	++	X	X	-
SPDF_SM_2	Information redundancy techniques on messages	++	X	X	-
Management data input/output (MDIOS)					
MDIO_SM_0	Periodic read-back of MDIO slave configuration registers	++	X	X	-
MDIO_SM_1	Protocol error signals	++	X	X	-
MDIO_SM_2	Information redundancy techniques on MDIO registers contents, including register update awareness	++	X	X	-
Secure digital input/output MultiMediaCard interface (SDMMC)					
SDIO_SM_0	Periodic read-back of SDIO/SMMC configuration registers	++	X	X	-
SDIO_SM_1	Protocol error signals including hardware CRC	++	X	X	-
SDIO_SM_2	Information redundancy techniques on messages	++	X	X	-
Controller area network (FDCAN)					
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	-
Secure digital input/output MultiMediaCard interface (SDMMC3)					
SDIO_SM_0	Periodical read-back of SDMMC3 configuration registers	++	X	X	-
SDIO_SM_1	Protocol error signals including hardware CRC	++	X	X	-
SDIO_SM_2	Information redundancy techniques on messages	++	X	X	-
USB on-the-go (OTG)					
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	-
Ethernet (ETH): media access control (MAC) with DMA controller					
ETH_SM_0	Periodic read-back of Ethernet configuration registers	++	X	X	-
ETH_SM_1	Protocol error signals including hardware CRC	++	X	X	-
ETH_SM_2	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	-
Boot and security and OTP control (BSEC)					
BSEC_SM_0	Periodical read-back of BSEC configuration registers	++	X	X	-
BSEC_SM_1	Hardware protection for OTP words	++	X	X	-
BSEC_SM_2	OTP read/write protection	++	X	X	-
BSEC_SM_3	Application software measures for end user configuration check.	++	X	X	-

Diagnostic	Description	Rank	Perm	Trans	Separation
Extended TrustZone® protection controller (ETZPC)					
ETZPC_SM_0	Periodical read-back of eTZPC system configuration registers	++	X	X	X
ETZPC_SM_1	Periodical software test for ETZPC	++	X	X	X
Disable and periodic cross-check of unintentional activation of unused peripherals					
FFI_SM_0	Disable of unused peripherals	++	-	-	X
FFI_SM_1	Periodic read-back of interference avoidance registers	++	-	-	X
FFI_SM_2	eTZPC protection on MCU isolated peripherals/resources	++	-	-	X
FFI_SM_3	Startup check for system configuration	++	-	-	X
Arm®Cortex®-M4 CPU					
CoU_1	The reset condition of Arm® Cortex®- M4 CPUs must be compatible as valid safe state at system level	++	-	-	-
Debug					
CoU_2	Device debug features must not be used in safety function(s) implementation.	++	-	-	-
Arm®Cortex®-M4 Supply system					
CoU_3	Low-power or standby mode states for Arm® Cortex®- M4 CPU or overall STM32MP1 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	-
CPU subsystem					
CoU_7	In case of multiple safety functions implementations, methods to guarantee their mutual independence must include MPU use.	++	-	-	-
CoU_9.1	Any security violation event must be considered equivalent to the detection of a non-controllable hardware random failure; accordingly, it must lead to the transition to a safe state (de-energize) at system level.	++	X	X	-
CoU_9.2	For each implemented safety mechanisms, related error/fault detection signal(s)/message(s) must be processed by the application software for the correct management of SS1 safe state. Related software routines must be considered as safety related in the framework of the overall policy for software systematic capability in the final system. ⁽³⁾	++	X	X	X
Part separation					
CoU_10	A combination of the external watchdog and system-level measures must guarantee the overall safe state in case of missed correct application software start, including fail of startup tests FFI_SM_3. Exit from power-up safe state must be linked to the sending to the external watchdog of a non-trivial message reporting successful execution of FFI_SM_3. ⁽⁴⁾	++	-	-	X
CoU_11	If implemented, data exchanges between the Safe Partition and the Non- Safe Partition must follow these rules: <ul style="list-style-type: none"> The exchange of safety related data is forbidden Exchange uses statically allocated locations in SYSRAM bank Exchange is based on the use of HSEM module It is End User responsibility to guarantee and verify with adequate accuracy that anomalies affecting the communication (including, but not limited to, data corruption, delays, wrong interrupt generation by HSEM) are not able to interfere with the safety function(s) 	++	-	-	X

Diagnostic	Description	Rank	Perm	Trans	Separation
CoU_12	The use of the peripheral direct connection function to connect safety related peripherals to non-safety related ones, or to modules belonging to Non Safe Partition is forbidden.	++	-	-	X
CoU_13	Correct configuration for power, clocks and SYSCFG must be checked at Arm® Cortex®-M4 boot before application software start.	++	-	-	X
CoU_14	Correct expected configuration and programming for eTZPC module must be checked at Arm® Cortex®-M4 boot before application software start.	++	-	-	X
CoU_15	An application software running on M4 CPU and another running on A7 CPU cannot share the same GPIO port.	++	-	-	X
CoU_16	Safety related functions (implemented in Safe Partition) and non-safety related functions (implemented in Non-Safe Partition) cannot share the use of the same physical package pin.	++	-	-	X
CoU_17	<p>It must be proven by the end user that the software running on A7 CPU has been developed and verified according some generic quality standards.</p> <p>Furthermore, for following specific functions:</p> <ul style="list-style-type: none"> • STM32MP1 clock/power configuration settings • bootloader software • access drivers to peripherals (HAL) • GPIO management • external memories access <p>Dedicated verification activities must guarantee that the related software functions have been correctly designed and functionally verified.</p> <p>(5)</p>	++	-	-	X
CoU_18	<p>The end user must activate and manage the security features available for the integrity check of software images load during boot process of A7 and M4 CPUs.(</p> <p>(6)</p>	++	-	-	X
CoU_19	<p>In 1oo2 architecture the PEv must keep the final system in safe state after the power-up until the message specified in CoU_10 is successfully received from both Arm® Cortex®-M4 CPUs.</p> <p>(7)</p>	++ ⁽⁷⁾	-	-	X
System					
DUAL_SM_0	Cross-check between two STM32 devices	o	X	X	-

1. Must be considered ranked as “++” in case MPU features are used for the separation of different safety functions which need to be prevented to interfere.
2. Can be considered ranked as “o” depending on the intended use of external memory connected to FMC.
3. To provide an example, safety recommendation “CPU_SM_3 Arm® Cortex®- M4 HardFault exceptions” cannot be considered correctly satisfied if related exception handler is not implemented in software.
4. The information of successful execution of startup test must be realized by a numeric message or by a combination of signals. A simple signal line transition is too simple to be valid.
5. This requirement is essential to mitigate the presence of systematic errors in the A7 software potentially leading to an excessive number of intervention by the protection mechanisms of the Safe Partition, and so an excessive switching rate to safe state. Indeed, the necessary performances of the final safety system may be no longer guaranteed if the A7 software interferes too often with the M4 CPU software or the Safe Partition resources.
6. The availability of integrity check features may change according to the software distribution. Some examples of such features are the TF-A authentication in FSBL and the SHA authentication available among SSBL U-Boot FIT options.
7. This CoU is recommended only if 1oo2 safety architecture is used. Refer to [Section 3.2.4](#).

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 *MPU*:

- System-critical MPU 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, ETZPC, bus matrix and interconnect 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 STM32MP1 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 STM32MP1 series devices reported in this safety manual is executed according to STMicroelectronics methodology flow for safety analysis of semiconductor devices in compliance with IEC 61508 (refer to [4] for more details). The accuracy of results obtained are guaranteed by three factors:

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

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

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

Table 146. Overall achievable safety integrity levels

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

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

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

- a large number of different STM32MP1 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* document is needed, contact the local STMicroelectronics sales representative as early as possible, in order to receive information on expected delivery dates for specific *Device* target part numbers.

Note: Safety metrics computations are restricted to STM32MP1 series boundary, hence they do not include the WDT_E, PE_V, and VMOne processes described in [Section 3.3.1 Safety requirement assumptions](#).

4.1.1 Safety analysis result customization

The safety analysis executed for STM32MP1 series devices documented in this safety manual considers all microprocessor 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 microprocessor, in order to be agnostic versus the final application. This means that no microprocessor module has been declared safe as per IEC 61508-4, 3.6.8. Therefore, all microcontroller modules are included in *SFF* computations.

In actual *End user* applications, not all the STM32MP1 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 STM32MP1 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-MPU 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 1.

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

4.1.2 General requirements for freedom from interferences (FFI)

A dedicated analysis has highlighted a list of general requirements to be followed in order to mitigate potential interferences between *Device* internal 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 147](#) (implementation details in [Section 3.6 Hardware and software diagnostics](#)) regardless any evaluation of their contribution to safety metrics.

Table 147. 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 <i>DMA</i> ⁽¹⁾
DMA_SM_4	<i>DMA</i> transaction awareness ⁽¹⁾
NVIC_SM_0	Periodic read-back of configuration registers
NVIC_SM_1	Expected and unexpected interrupt check
FFI_SM_0	Disable of unused peripherals
FFI_SM_1	Periodic read-back of interference avoidance registers

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

4.1.3 Notes on multiple-fault scenario

According to the requirements of IEC 61508, the safety analysis for STM32MP1 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 STM32MP1 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, , 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

To guarantee freedom from interference in a safety related microcontroller/microprocessor device means to analyze any potential source of interference with the correct execution for the safety function. Interferences can be of very different nature:

- Non safety related software interfering with safety related software
- Safety related software interfering with other safety related software, or even with itself
- Non safety related hardware interfering with safety related hardware or software due to random hardware failures
- Safety related hardware interfering with itself due to random hardware failures

In IEC61508 the analysis of hardware interferences (more precisely defined as dependent/common cause failure) is ruled by the IEC 61508:2 annex E, while software interferences analysis is ruled by IEC61508-3, Annex F. Because of the complexity of STM32MP1 device, and its safety concept based on the separation between two diverse CPUs, methods and analysis concepts from the more advanced ISO26262 2nd edition have been used as well.

Requirements derived by the above analysis are reported in this section. Their adoption is therefore highly recommended despite their minor contribution to the safety metrics to reach the required safety integrity level.

It is worth to note that as there is no on-chip redundancy on STM32MP1 devices, the CCF quantification through the β IC 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 β D parameters (used in PFH computation) that reflect the common cause factors between the two channels.

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 , and this diversity helps to mitigate dependent failures related to the main supply alterations. As reported in VSUP_SM_2 description, separate power supply for IWDG or/and the adoption of an external watchdog (CPU_SM_5) increase such diversity.
- VSUP_SM_5: power supply stability (guaranteed by system level measures) is an important mitigation factor

The adoption of such safety mechanisms is therefore highly recommended despite their minor contribution to the safety metrics to reach the required safety integrity level. Refer to [Section 3.6.4 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 *MPU* reset by watchdog. The adoption of external watchdog (CPU_SM_5) provides additional diversity and so further mitigation of clock-related common cause failures.

The adoption of such safety mechanism is therefore highly recommended despite their minor contribution to the safety metrics to reach the required safety integrity level. Refer to [Section 3.6.5 Reset and clock controller \(RCC\)](#) for detailed safety mechanisms description.

4.2.3 DMA

The *DMA* function can be involved in data transfers operated by most of the peripherals. Failures of *DMA* can interfere with the behavior of the system peripherals or *Application software*, leading to dependent failures. The adoption of the following safety mechanisms is therefore highly recommended (refer to [Section 3.6.9 Direct memory access controller \(DMA/ DMA2D/ DMAMUX\)](#) for description):

- DMA_SM_0
- DMA_SM_1
- DMA_SM_2

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

4.2.4 Internal temperature

The abnormal increase of the internal temperature is a potential source of dependent failures, as it can affect many *MPU* parts. The following safety mechanism mitigates this potential effect (refer to [Section 3.6.4 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.

4.2.5 Non Safe Partition hardware

Hardware failures in Non Safe Partition hardware could lead to unintended accesses to peripheral and resources belonging to Safe Partitions, and therefore to violation of safety function execution. The separation concept, built by the eTZPC protection and reinforced by a second layer of redundant safety mechanisms at application level, mitigates the effects of these dependent failures.

Related requirements are: FFI_SM_2: eTZPC protection on MCU isolated peripherals/resources, CoU_4, CoU_12 and CoU_15. Refer to [Section 3.2.5 The separation concept](#) for more information on separation concept.

4.2.6 Non safety related software running on A7 CPU

Systematic errors affecting the non-safety related software running on A7 CPU could lead to unintended accesses to peripheral and resources belonging to Safe Partitions, and therefore to violation of safety function execution.

The separation concept, built by the eTZPC protection and reinforced by a second layer of redundant safety mechanisms at application level, mitigates the effects of these dependent failures. An additional requirement about minimum quality level for some specific aspects of that non-safety related software is placed with CoU_17.

Related requirements are: FFI_SM_2: eTZPC protection on MCU isolated peripherals/resources, CoU_4, CoU_15 and CoU_17. Refer to [Section 3.2.5 The separation concept](#) for more information on separation concept.

Caution: The STM32MP1 safety concept presented in this safety manual is robust enough to manage and mitigate interferences due to systematic failures in the A7 software, because of the separation concept. End User must be aware, anyway, that excessive interferences by the A7 non safety related software may lead the final system to switch in safe state too often. In such a case, the required performances of the safety application may be compromised. Accordingly, End User is recommended to carefully read and strictly follow CoU_17 recommendations.

4.2.7

STM32MP1 boot process

The STM32MP1 boot process is based on software execution on A7 CPU which is in charge, among other tasks, to load the firmware image on M4 CPU and to start its application software execution. Random hardware failures on Non Safe Partition and/or systematic errors in non-safety related software running on A7 CPU may lead to incorrect boot process for M4 CPU and so to wrong safety function execution. These failures are mitigated by the combination of different measures: FFI_SM_3 Startup check for system configuration, CoU_10, CoU_13, CoU_14, CoU_16, CoU_17 and CoU_18. The combination of measures guarantees that either the M4 boot process has been correctly executed or the system is held in safe state.

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 STM32MP1 series devices and the compliance to IEC 61508 applicable clauses
- STMicroelectronics' internal fault injection campaign database including tool configuration and settings, fault injection logs and results, related to the *MPU* modules for which fault injection is adopted as verification method.

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

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

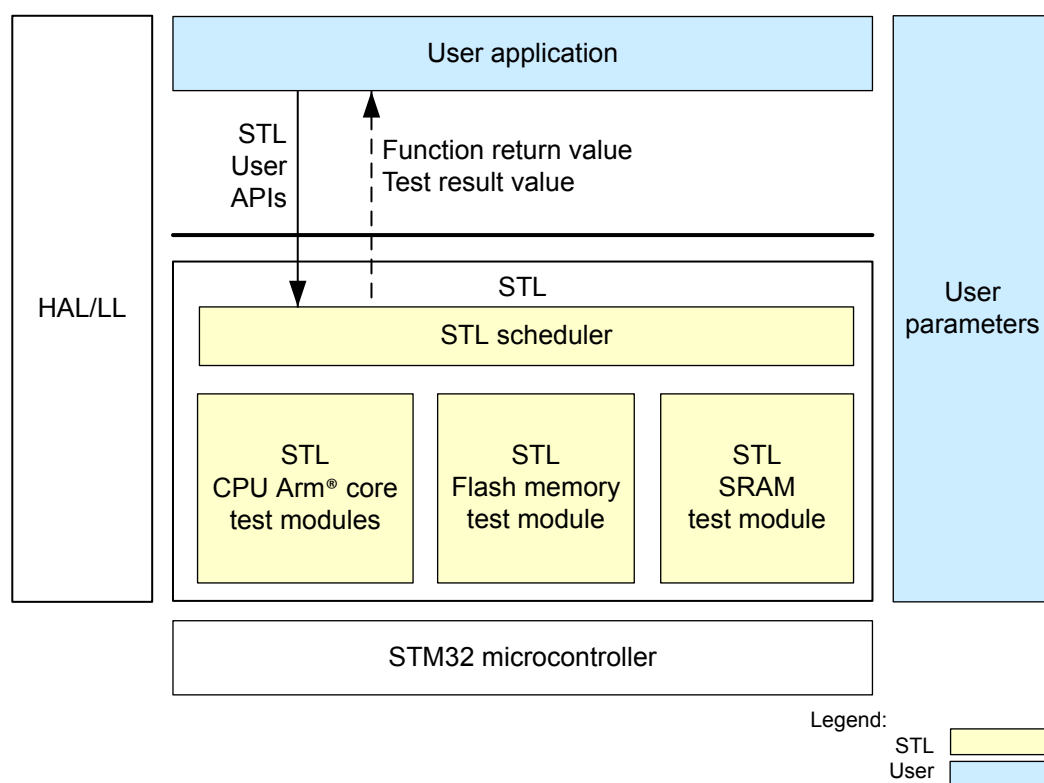
Appendix A X-CUBE-STL self-test software library

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

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

- **CPU_SM_0** Periodic core self-test software for *CPU*.
- **RAM_SM_5** Periodical integrity test for application software in *RAM*
- **RAM_SM_0** Periodic software test for static random access memory (*SRAM*)

Figure 8. STL architecture



X-CUBE-STL characteristics:

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

- Diagnostic coverage verified by state-of-the-art ST proprietary fault injection methodology
 - Development flow compliant to SC3 systematic capability requirements from IEC 61508
 - Certified by TÜV Rheinland (certification covers claims related to achieved DC and SC3 development flow)
- X-CUBE-STL is available on demand under *NDA* agreement (contact your local ST representative).

Revision history

Table 148. Document revision history

Date	Version	Changes
08-Sep-2020	1	Initial release.
28-Oct-2020	2	<p>Updated:</p> <p>Functional safety documentation framework.</p> <ul style="list-style-type: none"> Section Introduction Section 1.1 Purpose and scope Section 3.2.1 Definition of Compliant item Section 3.3.1 Safety requirement assumptions Table 23. NVIC_SM_1 Section 3.6.14 General-purpose input/output (GPIO) - port A/B/C/D/E/F/G/H/I/J/K Table 61. GPIO_SM_0 Table 63. GPIO_SM_2 Table 64. GPIO_SM_3 Table 140. FFI_SM_3 Section 3.6.41 Interferences mitigation Table 142. List of safety recommendations Section 4.2.6 Non safety related software running on A7 CPU
27-Jul-2021	3	<p>Updated:</p> <ul style="list-style-type: none"> Section Introduction Section 1.2 Normative references Section 1.3 Reference documents Section 3.2.2 Safety functions performed by Compliant item Section 3.2.4 Reference safety architectures - 1oo2 Section 3.3.1 Safety requirement assumptions Section 3.7 Conditions of use Table 142. List of safety recommendations
06-Sep-2021	4	Updated CRC2 and BSEC in Table 142. List of safety recommendations.
25-Oct-2023	5	<p>Updated:</p> <ul style="list-style-type: none"> Section 1.3 Reference documents Section 3.2.2 Safety functions performed by Compliant item Section 3.2.4 Reference safety architectures - 1oo2 Safety requirement assumptions Section 3.5 Systematic safety integrity Minor updates of tables and table titles in Section 3.6 Hardware and software diagnostics Section 4.1 Random hardware failure safety results Table 146. Overall achievable safety integrity levels Section 4.1.1 Safety analysis result customization Section 4.2.1 Power supply Section 5 List of evidences <p>Added:</p> <ul style="list-style-type: none"> Appendix A X-CUBE-STL self-test software library <p>Replacement of Flash by flash in the document.</p>

Glossary

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

ASR assumed safety requirement

CCF common cause failure

CM continuous mode

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

COTS commercial off-the-shelf

CoU conditions of use

CPU central processing unit

CRC cyclic redundancy check

DC diagnostic coverage

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

DMA direct memory access

DTI diagnostic test interval

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

EUC equipment under control

FIT failure in time

FMEA failure mode effect analysis

FMEDA failure mode effect diagnostic analysis

HD high-demand

HFT hardware fault tolerance

HW hardware

ITRS international technology roadmap for semiconductors

LD low-demand

MPU microprocessor unit

MPU memory protection unit

MTBF mean time between failures

MTTFd mean time to dangerous failure

NDA non disclosure agreement

PEc computation processing elements

PEi input processing elements

PEo output processing elements

PEv voting processing element

PFD probability of dangerous failure on demand

PFH probability of failure per hour

PL performance level

PST process safety time

SFF safe failure fraction

SIL safety integrity level

SoC system on chip

VMONe voltage monitors

WDTe watchdog

Contents

1	About this document	2
1.1	Purpose and scope	2
1.2	Normative references	2
1.3	Reference documents	2
2	Device development process	3
3	Reference safety architecture	4
3.1	Safety architecture introduction	4
3.2	Compliant item	4
3.2.1	Definition of Compliant item	4
3.2.2	Safety functions performed by Compliant item	5
3.2.3	Reference safety architectures - 1oo1	6
3.2.4	Reference safety architectures - 1oo2	7
3.2.5	The separation concept	7
3.3	Safety analysis assumptions	10
3.3.1	Safety requirement assumptions	10
3.4	Electrical specifications and environment limits	11
3.5	Systematic safety integrity	12
3.6	Hardware and software diagnostics	12
3.6.1	Arm® Cortex®-M4 CPU	13
3.6.2	System bus architecture/BusMatrix	19
3.6.3	Embedded SRAM	21
3.6.4	Power controller (PWR)	25
3.6.5	Reset and clock controller (RCC)	28
3.6.6	General-purpose input/output (GPIO)	31
3.6.7	Debug system or peripheral control	33
3.6.8	System configuration controller (SYSCFG)	34
3.6.9	Direct memory access controller (DMA/ DMA2D/ DMAMUX)	35
3.6.10	Extended interrupt and events controller (EXTI)	38
3.6.11	Cyclic redundancy-check calculation unit (CRC2)	40
3.6.12	Flexible static memory controller (FSMC)	40
3.6.13	Quad-SPI interface and Octo-SPI interface (QUADSPI/OCTOSPI)	42
3.6.14	Delay block (DLYB)	44
3.6.15	Analog-to-digital converter (ADC)	44
3.6.16	Digital-to-analog converter (DAC)	47
3.6.17	Voltage reference buffer (VREFBUF)	48

3.6.18	Digital filter for sigma delta modulators (DFSDM)	49
3.6.19	Digital camera interface (DCMI)	51
3.6.20	HASH processor (HASH)	52
3.6.21	True random number generator (RNG)	53
3.6.22	Cryptographic processor (CRYP)	54
3.6.23	Advanced-control/General-purpose and low-power timers	55
3.6.24	Basic timers	59
3.6.25	Independent and system window watchdogs (IWDG and WWDG)	60
3.6.26	Real-time clock module (RTC)	61
3.6.27	Tamper and backup registers (TAMP)	63
3.6.28	Inter-integrated circuit (I2C)	63
3.6.29	Universal synchronous/asynchronous receiver/transmitter and low-power universal asynchronous receiver/transmitter (USART and LPUART)	66
3.6.30	Serial peripheral interface (SPI)	69
3.6.31	Serial audio interface (SAI)	71
3.6.32	SPDIF receiver interface (SPDIFRX)	73
3.6.33	Management data input/output (MDIOS)	74
3.6.34	Secure digital input/output MultiMediaCard interface (SDMMC)	76
3.6.35	Secure digital input/output MultiMediaCard interface (SDMMC3)	77
3.6.36	Controller area network (FDCAN)	79
3.6.37	USB on-the-go (OTG)	81
3.6.38	Ethernet (ETH): media access control (MAC) with DMA controller	83
3.6.39	HDMI-CEC (CEC)	84
3.6.40	Boot and security and OTP control (BSEC)	86
3.6.41	Extended TrustZone® protection controller (ETZPC)	87
3.6.42	Disable and periodic cross-check of unintentional activation of unused peripherals	89
3.6.43	System	91
3.7	Conditions of use	92
4	Safety results	100
4.1	Random hardware failure safety results	100
4.1.1	Safety analysis result customization	100
4.1.2	General requirements for freedom from interferences (FFI)	101
4.1.3	Notes on multiple-fault scenario	102
4.2	Analysis of dependent failures	102
4.2.1	Power supply	102
4.2.2	Clock	103
4.2.3	DMA	103
4.2.4	Internal temperature	103

4.2.5	Non Safe Partition hardware	103
4.2.6	Non safety related software running on A7 CPU	103
4.2.7	STM32MP1 boot process	104
5	List of evidences	105
Appendix A	X-CUBE-STL self-test software library	106
	Revision history	108
	Glossary	109

List of tables

Table 1.	Document sections versus IEC 61508-2 Annex D safety requirements	2
Table 2.	SS1 and SS2 safe state details	11
Table 3.	CPU_SM_0	13
Table 4.	CPU_SM_1	14
Table 5.	CPU_SM_2	15
Table 6.	CPU_SM_3	15
Table 7.	CPU_SM_4	16
Table 8.	CPU_SM_5	17
Table 9.	CPU_SM_6	17
Table 10.	CPU_SM_7	18
Table 11.	MPU_SM_0	18
Table 12.	MPU_SM_1	19
Table 13.	BUS_SM_0	19
Table 14.	BUS_SM_1	20
Table 15.	RAM_SM_0	21
Table 16.	RAM_SM_2	22
Table 17.	RAM_SM_3	23
Table 18.	RAM_SM_4	24
Table 19.	RAM_SM_5	24
Table 20.	RAM_SM_9	25
Table 21.	VSUP_SM_0	25
Table 22.	VSUP_SM_1	26
Table 23.	VSUP_SM_2	26
Table 24.	VSUP_SM_3	27
Table 25.	VSUP_SM_5	27
Table 26.	VSUP_SM_6	28
Table 27.	CLK_SM_0	28
Table 28.	CLK_SM_1	29
Table 29.	CLK_SM_2	29
Table 30.	CLK_SM_3	30
Table 31.	CLK_SM_4	30
Table 32.	GPIO_SM_0	31
Table 33.	GPIO_SM_1	31
Table 34.	GPIO_SM_2	32
Table 35.	GPIO_SM_3	32
Table 36.	DBG_SM_0	33
Table 37.	LOCK_SM_0	33
Table 38.	SYSCFG_SM_0	34
Table 39.	DIAG_SM_0	34
Table 40.	DMA_SM_0	35
Table 41.	DMA_SM_1	35
Table 42.	DMA_SM_2	36
Table 43.	DMA_SM_3	36
Table 44.	DMA_SM_4	37
Table 45.	NVIC_SM_0	38
Table 46.	NVIC_SM_1	39
Table 47.	CRC_SM_0	40
Table 48.	FSMC_SM_0	40
Table 49.	FSMC_SM_1	41
Table 50.	FSMC_SM_2	41
Table 51.	FSMC_SM_3	42
Table 52.	QSPI_SM_0	42
Table 53.	QSPI_SM_1	43

Table 54.	QSPI_SM_2	43
Table 55.	DLB_SM_0	44
Table 56.	ADC_SM_0	44
Table 57.	ADC_SM_1	45
Table 58.	ADC_SM_2	45
Table 59.	ADC_SM_3	46
Table 60.	ADC_SM_4	46
Table 61.	DAC_SM_0	47
Table 62.	DAC_SM_1	47
Table 63.	VREF_SM_0	48
Table 64.	VREF_SM_1	48
Table 65.	DFS_SM_0	49
Table 66.	DFS_SM_1	49
Table 67.	DFS_SM_2	50
Table 68.	DFS_SM_3	50
Table 69.	DCMI_SM_0	51
Table 70.	DCMI_SM_1	51
Table 71.	HASH_SM_0	52
Table 72.	HASH_SM_1	52
Table 73.	RNG_SM_0	53
Table 74.	RNG_SM_1	53
Table 75.	CRYP_SM_0	54
Table 76.	CRYP_SM_1	54
Table 77.	CRYP_SM_2	55
Table 78.	ATIM_SM_0	55
Table 79.	ATIM_SM_1	56
Table 80.	ATIM_SM_2	57
Table 81.	ATIM_SM_3	58
Table 82.	ATIM_SM_4	58
Table 83.	GTIM_SM_0	59
Table 84.	GTIM_SM_1	59
Table 85.	WDG_SM_0	60
Table 86.	WDG_SM_1	60
Table 87.	RTC_SM_0	61
Table 88.	RTC_SM_1	61
Table 89.	RTC_SM_2	62
Table 90.	RTC_SM_3	62
Table 91.	TAMP_SM_0	63
Table 92.	IIC_SM_0	63
Table 93.	IIC_SM_1	64
Table 94.	IIC_SM_2	64
Table 95.	IIC_SM_3	65
Table 96.	IIC_SM_4	65
Table 97.	UART_SM_0	66
Table 98.	UART_SM_1	66
Table 99.	UART_SM_2	67
Table 100.	UART_SM_3	68
Table 101.	SPI_SM_0	69
Table 102.	SPI_SM_1	69
Table 103.	SPI_SM_2	70
Table 104.	SPI_SM_3	70
Table 105.	SPI_SM_4	71
Table 106.	SAI_SM_0	71
Table 107.	SAI_SM_1	72
Table 108.	SAI_SM_2	72

Table 109.	SPDF_SM_0	73
Table 110.	SPDF_SM_1	73
Table 111.	SPDF_SM_2	74
Table 112.	MDIO_SM_0	74
Table 113.	MDIO_SM_1	74
Table 114.	MDIO_SM_2	75
Table 115.	SDIO_SM_0	76
Table 116.	SDIO_SM_1	76
Table 117.	SDIO_SM_2	77
Table 118.	SDIO_SM_0	77
Table 119.	SDIO_SM_1	78
Table 120.	SDIO_SM_2	78
Table 121.	CAN_SM_0	79
Table 122.	CAN_SM_1	79
Table 123.	CAN_SM_2	80
Table 124.	USB_SM_0	81
Table 125.	USB_SM_1	81
Table 126.	USB_SM_2	82
Table 127.	USB_SM_3	82
Table 128.	ETH_SM_0	83
Table 129.	ETH_SM_1	83
Table 130.	ETH_SM_2	84
Table 131.	HDMI_SM_0	84
Table 132.	HDMI_SM_1	85
Table 133.	HDMI_SM_2	85
Table 134.	BSEC_SM_0	86
Table 135.	BSEC_SM_1	86
Table 136.	BSEC_SM_2	87
Table 137.	BSEC_SM_3	87
Table 138.	ETZPC_SM_0	87
Table 139.	ETZPC_SM_1	88
Table 140.	FFI_SM_0	89
Table 141.	FFI_SM_1	89
Table 142.	FFI_SM_2	90
Table 143.	FFI_SM_3	90
Table 144.	DUAL_SM_0	91
Table 145.	List of safety recommendations	92
Table 146.	Overall achievable safety integrity levels	100
Table 147.	List of general requirements for FFI	101
Table 148.	Document revision history	108

List of figures

Figure 1.	STMicroelectronics product development process	3
Figure 2.	STM32 as <i>Compliant item</i>	4
Figure 3.	1oo1 reference architecture	6
Figure 4.	1oo2 reference architecture	7
Figure 5.	Spatial separation concept	8
Figure 6.	Temporal separation concept	9
Figure 7.	Allocation and target for STM32 <i>PST</i>	10
Figure 8.	STL architecture	106

IMPORTANT NOTICE – READ CAREFULLY

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

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

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

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

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

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

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

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

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

© 2023 STMicroelectronics – All rights reserved