User manual

STM8AF Series safety manual

Introduction

This document must be read along with the technical documentation such as reference manual(s) and datasheets for the STM8AF Series microcontroller devices, available on www.st.com.

It describes how to use the devices in the context of a safety-related system (STM8A-SafeASIL functional safety package), specifying the user's responsibilities for installation and operation in order to reach the targeted safety integrity level.

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 ISO26262:2018.

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 STM8AF Series microcontroller unit (MCU) devices (further also referred to as Device(s)) in the context of a safety-related system, specifying the user's responsibilities for installation and operation, in order to reach the desired safety integrity level.

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

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

1.2 Normative references

This document is written in compliance with the ISO26262:2018 2nd Edition - Road vehicles functional safety.

1.3 Reference documents

[1] UM2138 FMEDA analysis for STM8AF Series MCUs.

[2] UM2139 FMEDA handling for STM8AF Series MCUs.

[3] AN5494 STM8AF Evaluation Report for hardware component qualification according ISO26262:2018

2 Device development process

STM8AF 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

1 Conception

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

2 Design & validation

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

3 Qualification

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

3 Reference safety architecture

This section reports details of the STM8AF Series safety architecture.

3.1 Strategy for ISO26262 compliance for STM8AF microcontrollers

STM8AF microcontrollers can be considered, in the spirit of ISO26262:8, 5.4.1.1 requirement, as off-the-shelf elements not built-to-order to fulfil specific safety requirements. Accordingly, they can be qualified according to well-established procedures based on quality standards, and they can be evaluated according to ISO26262:8 Clause 13. Clause 13 activity aims to ensure that device functional behaviour is adequate to meet their assumed allocated safety requirements and, therefore, to ensure that the risk of a violation of a safety goal/safety requirement, due to a systematic fault of the hardware element, is sufficiently low.

STM8AF microcontrollers can be classified as Class III as per ISO26262:8, 13.4.1.1 requirement. Accordingly, the overall Clause13 procedure is followed to the extent required for Class III hardware components. In the specific:

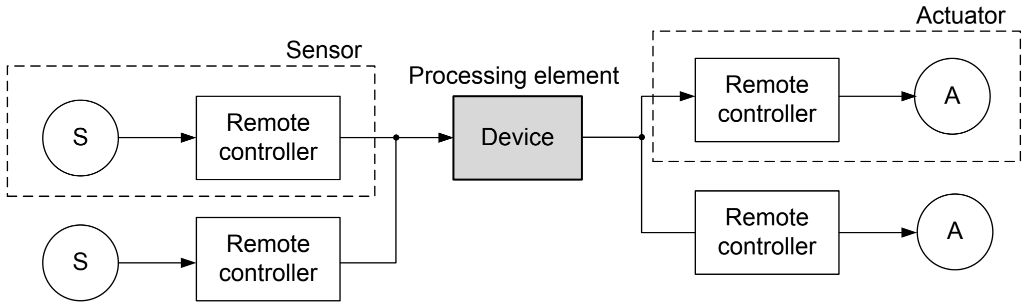
- The device intended use in the framework of safety related application is identified in Section 3.2
- Resulting assumptions (as per requirement ISO26262:8 13.4.3.3.3) are listed in Section 3.3
- The outcomes of the overall safety analysis in terms of resulting safety requirements to be implemented by the End User are eventually listed in Section 3.5 and Section 3.6
- Analysis details and conclusions, including the confirmation of the device suitability for the intended use (according to the original assumptions) are provided in Section 4

All evidences, rationales and explanations related to the overall Clause 13 evaluation procedure are collected in reference [3] (Evaluation Report as per ISO26262:8 13.4.3.6.1 requirement) which is available under NDA. It is worth to note that regardless the pattern selected to allow the use of the Device in a ISO26262 compliant application, the structure of this document is adherent to the SEooC approach as per ISO26262:10, with the SEooC assumption reported in Section 3.2 and the SEooC requirements reported in Section 3.5 and Section 3.6 .

3.2 Device intended use

In the framework of its reference safety architecture, the Device is assumed to be a standard hardware element/ component considered to be safety-related within the context of the ISO 26262 compliant item or element into which they are to be integrated. Its intended use is represented at high level in Figure 2, where the Device basically acts as processing unit between input and output entities.

Figure 2. Device intended use: high level representation



Other components might be related to the device, 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).

In essence, Device 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
- the computation processing elements can be involved (to the extent depending to the target safety integrity) in the implementation of local software-based diagnostic functions; this is represented by the block PEd
- processes external to the Device ensuring safety integrity, such as watchdog (WDTe) and voltage monitors (VMONe)

The role of WDTe and VMONe external processes is clarified in the sections where the safety mechanism are detailed:

- WDTe: refer to *External watchdog* – CPU_SM_4 and *Control flow monitoring* in *Application software* – CPU_SM_1,
- VMONe: refer to *System-level power supply management* - VSUP_SM_1.

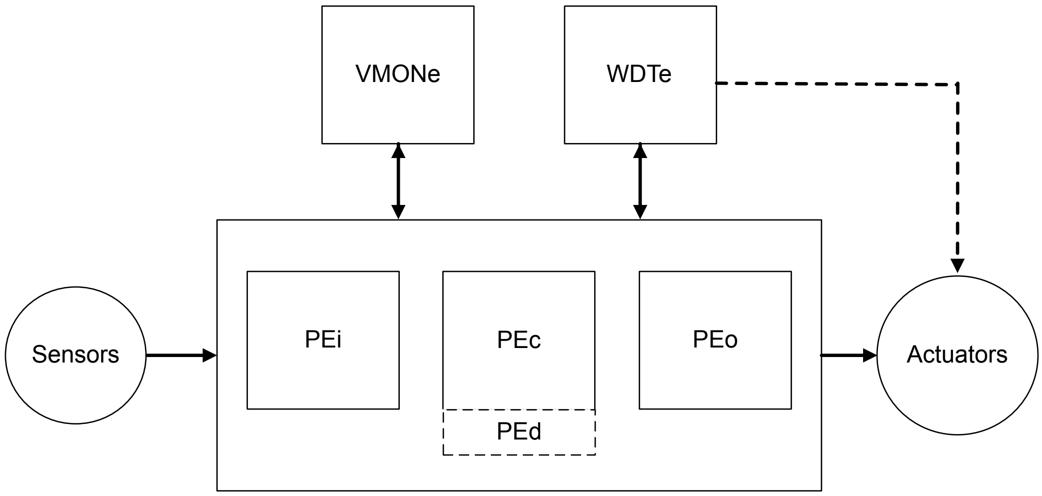
In summary, the devices support the implementation of *End user* safety functions consisting of three operations:

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

Claims on the Device and computation of safety metrics are done with respect to these three basic operations.

Reference safety architecture (Figure 3) ensures safety integrity of Device through combining device internal processes (implemented safety mechanisms) with external processes WDTe and VMONe.

Figure 3. Reference safety architecture



3.3 Evaluation of the safety analysis assumptions

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

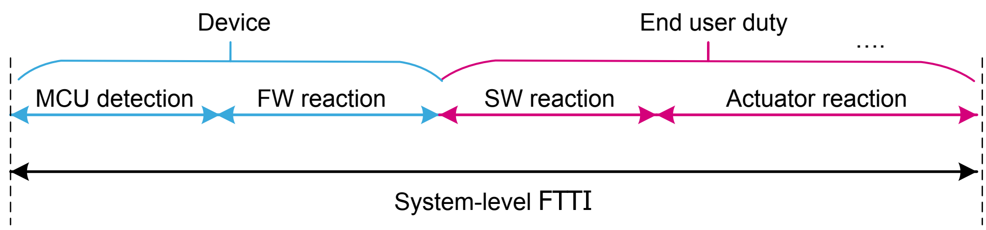
Caution: It is the *End user*'s responsibility to check the compliance of the final application with these assumptions.

ASR1: Device is assumed to be used in final system, implementing safety goal(s) up to ASIL B.

ASR2: Device is used to implement safety function(s) allowing a specific worst-case time budget (see note below) for the Device to detect and react to a failure. That time corresponds to the portion of the fault tolerant time interval (FTTI) allocated to the device (*Device* in Figure 4) in error reaction chain at system level.

Note: The computation for time budget mainly depends on the execution speed for periodic tests implemented by software. Such duration might depends on the actual amount of hardware resources (RAM memory, Flash memory, peripherals) actually declared as safety-related.

Figure 4. Allocation and target for STM8 FTTI



ASR3: Device is assumed to be integrated in a product with a lifetime up to 10 years.

ASR4: It is assumed that debug/test functions, BEEP and AWU module are not used for the implementation of safety functions.

ASR5: It is assumed that there are no *non-safety-related* functions implemented in application software, coexisting with safety functions.

ASR6: It is assumed that the implemented safety function(s) does (do) not depend on transition of the device to and from a low-power state.

ASR7: The local safe state of *Device* is the one in which either:

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

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

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

Table 1. SS1 and SS2 safe state details

Safe state	Condition	*Device* action	System transition to safe state
SS1	The application software is informed by the presence of a fault and a reaction by the application software itself is possible.	Fault reporting to application software	Application software drives the overall system in its safe state
SS2	The application software 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”)

ASR8: 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 Device.

3.4 Electrical specifications and environment limits

To ensure safety integrity, the user must operate the *Device* within its specified:

- absolute maximum rating
- capacity
- operating conditions

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

3.5 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 the 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 must disregard any recommendations not applicable to their *Device* part number of interest.

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

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

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

SM CODE	Unique safety mechanism code/identifier used also in *FMEA* document. Identifiers use the scheme *mmm_SM_x* where *mmm* is a 3- or 4-letter module (function, peripheral) short name, and *x* is a number. It is possible that the numbering is not sequential (although usually incremental) and/or that the module short name is different from that used in other documents.
Description	Short mnemonic description
Ownership	ST : means that method is available on silicon. *End user*: method must be implemented by *End user* through *Application software* modification, hardware solutions, or both.
Detailed implementation	Detailed implementation sometimes including notes about the safety concept behind the introduction of the safety mechanism.
Error reporting	Describes how the fault detection is reported to application software.
Fault detection time	Time that the safety mechanism needs to detect the hardware failure.
Addressed fault model	Reports fault model(s) addressed by the diagnostic (permanent, transient, or both), and other information: • If ranked for *Fault avoidance*: method contributes to lower the probability of occurrence of a failure • If ranked for *Systematic*: method is conceived to mitigate systematic errors (bugs) in application software design
Dependency on *Device* configuration	Reports if safety mechanism implementation or characteristics change among different *Device* part numbers.
Initialization	Specific operation to be executed to activate the contribution of the safety mechanism
Periodicity	Continuous : safety mechanism is active in continuous mode. Periodic: safety mechanism is executed periodically

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

Startup: safety mechanism is supposed to be executed only at power-up or during off-line maintenance periods.

Test for the diagnostic Reports specific procedure (if any and recommended) to allow on-line tests of safety mechanism efficiency.

Latent faults protection Reports the safety mechanism(s) associated in order to correctly manage a Latent faults scenario (refer to Section 3.7 Notes on latent faults mitigation).

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

3.5.1 STM8A Central processing unit (CPU)

Table 2. **CPU_SM_0**

SM CODE	CPU_SM_0
Description	Periodical core self-test software for STM8A CPU
Ownership	End user
Detailed implementation	The software test is built around well-known techniques already addressed by ISO26262-5 D.2.3.1 Self-test by software. To reach the required values of coverage, the self-test software is specified by means of a detailed analysis of all the CPU failure modes and related failure modes distribution
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen. The adoption of checksum protection on results variables and defensive programming are recommended.
Latent faults protection	CPU_SM_5: external watchdog
Recommendations and known limitations	This method is the main asset in STM8AF Series safety concept. CPU integrity is a key factor because the defined diagnostics for MCU peripherals are to major part software-based. Startup execution of this safety mechanism is recommended for latent fault mitigations - refer to Section 3.7 Notes on latent faults mitigation for details.

Table 3. CPU_SM_1

SM CODE	CPU_SM_1
Description	Control flow monitoring in Application software
Ownership	End user
Detailed implementation	A significant part of the failure distribution of CPU core for permanent faults is related to failure modes directly related to program counter loss of control or hang-up. Due to their intrinsic nature, such failure modes are not addressed by a standard software test method like SM_CPU_0. Therefore it is necessary to implement a run-time control of the Application software flow, in order to monitor and detect deviation from the expected behavior due to such faults. Linking this mechanism to watchdog firing assures that severe loss of control (or, in the worst case, a program counter hang-up) is detected. The guidelines for the implementation of the method are the following: • Different internal states of the Application software are well documented and described (the use of a dynamic state transition graph is encouraged). • Monitoring of the correctness of each transition between different states of the Application software is implemented. • Transition through all expected states during the normal Application software program loop is checked. • A function in charge of triggering the system watchdog is implemented in order to constrain the triggering (preventing the issue of CPU reset by watchdog) also to the correct execution of the above-described method for program flow monitoring. The use of window feature available on internal window watchdog (WWDG) is recommended. • The use of the external watchdog, helps to implement a more robust control flow mechanism fed by a different clock source. In any case, safety metrics do not depend on the kind of watchdog in use (the adoption of independent or external watchdog contributes to the mitigation of dependent failures, see Section 3.5.16 Reset (RST) and Clock control (CLK))
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	-

Table 4. **CPU_SM_2**

SM CODE	CPU_SM_2
Description	Double computation in Application software
Ownership	End user
Detailed implementation	A timing redundancy for safety-related computation is considered to detect transient faults affecting the STM8A CPU subparts devoted to mathematical computations and data access. The guidelines for the implementation of the method are the following: • The requirement needs be applied only to safety-relevant computation, which in case of wrong result could interfere with the system safety functions. Such computation must be therefore carefully identified in the original Application software source code • Both mathematical operation and comparison are intended as computation. • The redundant computation for mathematical computation is implemented by using copies of the original data for second computation, and by using an equivalent formula if possible
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	End user is responsible to carefully avoid that the intervention of optimization features of the used compiler removes timing redundancies introduced according to this condition of use.

Table 5. **CPU_SM_3**

SM CODE	CPU_SM_3
Description	STM8A CPU illegal op-code protection
Ownership	ST
Detailed implementation	Any illegal op-code read from the program space triggers a MCU reset. Because according ASR7 (refer to Section 3.3 Evaluation of the safety analysis assumptions) the issue of MCU reset is equivalent to safe state SS2, this protection feature can be considered a valid safety mechanisms
Error reporting	CPU reset
Fault detection time	Depends on implementation. Refer to functional documentation.
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	None
Periodicity	Continuous
Test for the diagnostic	Not directly available but it can be potentially implemented by software. Anyway this safety mechanism is never used as standalone mitigation for specific failure modes (refer to [1] for further information)
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 6. CPU_SM_4

SM CODE	CPU_SM_4
Description	Stack hardening for Application software
Ownership	End user
Detailed implementation	The stack hardening method is required to address faults (mainly transient) affecting CPU register bank. This method is based on source code modification, introducing information redundancy in register-passed information to called functions. The guidelines for the implementation of the method are the following: • To pass also a redundant copy of the passed parameters values (possibly inverted) and to execute a coherence check in the function. • To pass also a redundant copy of the passed pointers and to execute a coherence check in the function. • For parameters that are not protected by redundancy, to implement defensive programming techniques (plausibility check of passed values). For example enumerated fields are to be checked for consistency.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	This method partially overlaps with defensive programming techniques required by ISO26262-6 for software development. Therefore in presence of Application software qualified for safety integrity greater or equal to ASIL C, optimizations are possible.

Table 7. **CPU_SM_5**

SM CODE	CPU_SM_5
Description	External watchdog
Ownership	End user
Detailed implementation	Using an external watchdog linked to control flow monitoring method (refer to CPU_SM_1) addresses failure mode of program counter or control structures of CPU. External watchdog can be designed to be able to generate the combination of signals needed on the final system to achieve the safe state. It is recommended to carefully check the assumed requirements about system safe state reported in Safety requirement assumptions. It also contributes to reduce potential common cause failures, because the external watchdog is clocked and supplied independently of Device.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	To be defined at system level (outside the scope of Device analysis)
Latent faults protection	CPU_SM_1: control flow monitoring in Application software
Recommendations and known limitations	In case the external watchdog requires a clock input source , it must be different from the one feeding the Device. In case the bootloader feature is used, the external watchdog must be able to keep the system in safe state during the boot process (that is, until the main application software will start).

Table 8. **CPU_SM_6**

SM CODE	CPU_SM_6
Description	Window watchdog (WWDG)
Ownership	ST
Detailed implementation	Using the WWDG watchdog linked to control flow monitoring method (refer to CPU_SM_1) addresses failure mode of program counter or control structures of CPU. Its window feature can help to implement a more accurate control flow for the application software (refer to CPU_SM_1)
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	WWDG activation. It is recommended to use hardware watchdog in Option byte settings (WWDG is automatically enabled after reset)
Periodicity	Continuous
Test for the diagnostic	WDG_SM_1: Software test for watchdog at startup
Latent faults protection	CPU_SM_1: control flow monitoring in Application software WDG_SM_0: periodical read-back of configuration registers
Recommendations and known limitations	The WWDG intervention is able to achieve a potentially “incomplete” local safe state because it can only guarantee that CPU is reset. No guarantee that Application software can be still executed to generate combinations of output signals that might be needed by the external system to achieve the final safe state.

Table 9. CPU_SM_7

SM CODE	CPU_SM_7
Description	Independent watchdog (IWDG)
Ownership	ST
Detailed implementation	Using the IWDG watchdog linked to control flow monitoring method (refer to CPU_SM_1) addresses failure mode of program counter or control structures of CPU. Its window feature can help to implement a more accurate control flow for the application software (refer to CPU_SM_1)
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	IWDG activation. It is recommended to use hardware watchdog in Option byte settings (IWDG is automatically enabled after reset)
Periodicity	Continuous
Test for the diagnostic	WDG_SM_1: Software test for watchdog at startup
Latent faults protection	CPU_SM_1: control flow monitoring in Application software WDG_SM_0: periodical read-back of configuration registers
Recommendations and known limitations	The IWDG intervention is able to achieve a potentially “incomplete” local safe state because it can only guarantee that CPU is reset. No guarantee that Application software can be still executed to generate combinations of output signals that might be needed by the external system to achieve the final safe state.

3.5.2 Embedded Flash memory and BOOT ROM

Table 10. FLASH_SM_0

SM CODE	FLASH_SM_0
Description	Periodical software test for Flash memory
Ownership	*End user*
Detailed implementation	Permanent faults affecting the system Flash memory, memory cells and address decoder, are addressed through a dedicated software test that checks the memory cell contents versus the expected value, using signature-based techniques. According to ISO26262-11 5.1.13.2 , the effective diagnostic coverage of such techniques depends on the width of the signature in relation to the block length of the information to be protected - therefore the signature computation method is to be carefully selected. Note that the simple signature method (ISO26262-11 5.1.13.2 Modified checksum) is inadequate as it only achieves a low value of coverage. The information block does not need to be addressed with this test as it is not used during normal operation (no data nor program fetch).
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	Flash memory size changes according part number
Initialization	Memory signatures must be stored in Flash memory as well
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen
Latent faults protection	CPU_SM_1: control flow monitoring in application software

SM CODE	FLASH_SM_0
	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	This test is expected to have a relevant time duration – test integration must therefore consider the impact on application software execution. Unused Flash memory sections can be excluded from testing. Startup execution of this safety mechanism is recommended for latent fault mitigations - refer to Section 3.7 Notes on latent faults mitigation for details.

Table 11. FLASH_SM_1

SM CODE	FLASH_SM_1
Description	Control flow monitoring in application software
Ownership	*End user*
Detailed implementation	Permanent and transient faults affecting the system Flash memory, memory cells and address decoder, can interfere with the access operation by the *CPU*, leading to wrong data or instruction fetches. Such failures can be detected by control flow monitoring techniques implemented in the application software loaded from Flash memory. For more details on the implementation, refer to description CPU_SM_1.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	CPU_SM_1 correct implementation supersedes this requirement.

Table 12. FLASH_SM_2

SM CODE	FLASH_SM_3
Description	Flash (including option bytes) write protection
Ownership	ST
Detailed implementation	This safety mechanism prevents unintended writes on the Flash memory. After reset, the main program and DATA areas are protected against unintentional write operations. They must be unlocked before attempting to modify their content.
Error reporting	None
Fault detection time	Not applicable
Addressed fault model	None (Systematic only)
Dependency on *Device* configuration	None
Initialization	Not needed (enabled by default)
Periodicity	Continuous
Test for the diagnostic	Not needed
Latent faults protection	FLASH_SM_0: periodical software test for Flash memory

SM CODE	FLASH_SM_3
Recommendations and known limitations	This method addresses systematic faults in software application and it have zero efficiency in addressing hardware random faults affecting the option byte value during running time. No DC value is therefore associated.

Table 13. **FLASH_SM_3**

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

Table 14. **FLASH_SM_4**

SM CODE	FLASH_SM_6
Description	Flash memory unused area filling code
Ownership	*End user*
Detailed implementation	Used Flash memory area must be filled with deterministic data. This way in case that the program counter jumps outside the application program area due to a transient fault affecting *CPU*, the system evolves in a deterministic way.
Error reporting	NA
Fault detection time	NA
Addressed fault model	None (Fault avoidance)
Dependency on *Device* configuration	None
Initialization	NA
Periodicity	NA
Test for the diagnostic	NA
Latent faults protection	NA
Recommendations and known limitations	Filling code can be made of NOP instructions, or an illegal code that leads to a HardFault exception raise.

Table 15. **BOOT_SM_0**

SM CODE	BOOT_SM_0
Description	Application software integrity test

SM CODE	BOOT_SM_0
Ownership	End user
Detailed implementation	If used, the boot loader starts executing after reset. Permanent and transient faults affecting the boot ROM can lead to the load of wrong or missing software image and/or data into Flash/ EEPROM. Testing the integrity of the loaded application software and data before starting the application software implementing the safety function mitigates in correct way those failures.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Startup only
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_5: External watchdog
Recommendations and known limitations	This method is fully equivalent to the execution at startup of FLASH_SM_0 and the implementation of EEP_SM_0

3.5.3 Embedded SRAM

Table 16. **RAM_SM_0**

SM CODE	RAM_SM_0
Description	Periodical software test for static random access memory (SRAM or RAM)
Ownership	End user
Detailed implementation	To enhance the coverage on SRAM data cells and to ensure adequate coverage for permanent faults affecting the address decoder it is required to execute a periodical software test on the system RAM memory. The selection of the algorithm must ensure the target SFF coverage for both the RAM cells and the address decoder. Evidences of the effectiveness of the coverage of the selected method must be also collected
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	RAM size can change according to the part number
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Usage of a March test C- is recommended. Because the nature of this test can be destructive, RAM contents restore must be implemented. Possible interferences with interrupt-serving routines fired during test execution must be also considered (such routines can access to RAM invalid contents). Note: unused RAM section can be excluded by the testing, under end user responsibility on actual RAM usage by final application software Startup execution of this safety mechanism is recommended for multiple fault mitigations - refer to Section 3.7 Notes on latent faults mitigation for details.

Table 17. **RAM_SM_1**

SM CODE	RAM_SM_1
Description	Stack hardening for application software
Ownership	End user
Detailed implementation	The stack hardening method is used to enhance the application software robustness to SRAM faults that affect the address decoder. The method is based on source code modification, introducing information redundancy in the stack-passed information to the called functions. Method contribution is relevant in case the combination between the final application software structure and the compiler settings requires a significant use of the stack for passing function parameters. Implementation is the same as method CPU_SM_4
Error reporting	Refer to CPU_SM_4
Fault detection time	Refer to CPU_SM_4
Addressed fault model	Refer to CPU_SM_4
Dependency on *Device* configuration	Refer to CPU_SM_4
Initialization	Refer to CPU_SM_4
Periodicity	Refer to CPU_SM_4
Test for the diagnostic	Refer to CPU_SM_4
Latent faults protection	Refer to CPU_SM_4
Recommendations and known limitations	Refer to CPU_SM_4

Table 18. **RAM_SM_2**

SM CODE	RAM_SM_2
Description	Information redundancy for safety-related variables in application software
Ownership	End user
Detailed implementation	To address transient faults affecting SRAM controller, it is required to implement information redundancy on the safety-related system variables stored in the RAM. The guidelines for the implementation of this method are the following: • The system variables that are safety-related (in the sense that a wrong value due to a failure in reading on the RAM affects the safety functions) are well-identified and documented. • The arithmetic computation or decision based on such variables are executed twice and the two final results are compared. • Safety-related variables are stored and updated in two redundant locations, and comparison is checked before consuming data. • Enumerated fields must use non-trivial values, checked for coherence at least one time per *FTTI* • Data vectors stored in SRAM must be protected by a encoding checksum.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_0: periodical core self-test software

SM CODE	RAM_SM_2
Recommendations and known limitations	Implementation of this safety method shows a partial overlap with an already foreseen method for STM8A (CPU_SM_1); optimizations in implementing both methods are therefore possible

Table 19. RAM_SM_3

SM CODE	RAM_SM_3
Description	Control flow monitoring in application software
Ownership	End user
Detailed implementation	In case the end user application software is executed from SRAM, permanent and transient faults affecting the memory (cells and address decoder) can interfere with the program execution. To address such failures it is needed to implement this method. For more details on the implementation, refer to description CPU_SM_1
Error reporting	Depends on implementation
Fault detection time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Needed just in case of application software execution from SRAM. CPU_SM_1 correct implementation supersedes this requirement

Table 20. RAM_SM_4

SM CODE	RAM_SM_4
Description	Periodical integrity test for application software in RAM
Ownership	End user
Detailed implementation	In case application software or diagnostic libraries are executed in RAM, it is needed to protect the integrity of the code itself against soft-error corruptions and related code mutations. This method must check the integrity of the stored code by checksum computation techniques, on a periodic basis (at least once per *FTTI*). For implementation details refer to similar method FLASH_SM_0
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen.
Latent faults protection	CPU_SM_0: periodical core self test software CPU_SM_1: control flow monitoring in application software
Recommendations and known limitations	This method must be implemented only in case of application software or diagnostic libraries are executed from RAM

3.5.4 Address and Data bus

Table 21. BUS_SM_0

SM CODE	BUS_SM_0
Description	Periodical software test for interconnections
Ownership	*End user*
Detailed implementation	The intra-chip connection resources needs to be periodically tested for permanent faults detection. The test executes a connectivity test of these shared resources, including the testing of the arbitration mechanisms between peripherals.
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
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Implementation can be considered in large part as overlapping with the widely used *Periodical read-back of configuration registers* required for several peripherals

Table 22. BUS_SM_1

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

Table 23. LOCK_SM_0

SM CODE	LOCK_SM_0
Description	Lock mechanism for configuration options

SM CODE	LOCK_SM_0
Ownership	ST
Detailed implementation	The STM8AF Series devices feature spread protection to prevent unintended configuration changes for some peripherals and system registers (for example timers), the protection is implemented by specific lock functions, or by requiring a specific correct order for the register access. The spread protection mitigates the effects of systematic faults in software application. The use of this method is encouraged to enhance the end application robustness to systematic faults.
Error reporting	Not generated (when locked, register overwrites are just ignored)
Fault detection time	NA
Addressed fault model	None (Systematic only)
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Latent faults protection	Not needed
Recommendations and known limitations	No DC associated because this test addresses systematic faults

3.5.5 Data EEPROM memory

Table 24. **EEP_SM_0**

SM CODE	EEP_SM_0
Description	Information redundancy
Ownership	End User
Detailed implementation	To address permanent faults affecting the internal EEPROM bank it is required to implement information redundancy techniques. Possible techniques are: • use redundant copies of safety relevant data and perform coherence check before use, • organize data in arrays and compute checksum field to be checked before use. Due to its nature, the data stored in EEPROM is typically managed directly by the end user application software, therefore it is reasonable to rely to methods to be plugged into the final software solution.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on MCU 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	None

Table 25. **EEP_SM_1**

SM CODE	EEP_SM_1
Description	Software read-back after write operation
Ownership	End User

SM CODE	EEP_SM_1
Detailed implementation	To address missing writes on EEPROM cells due to hardware random faults related to EEPROM technology, it is required that the application software executes a read-back of written data after an update of the EEPROM values. Missing writes are handled as a hardware fault.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on MCU 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	None

3.5.6 Interrupt controller (ITC)

Table 26. ITC_SM_0

SM CODE	ITC_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This test is implemented by executing a periodical check of the configuration registers for a system peripheral against its expected value. Expected values are previously stored in RAM and adequately updated after each configuration change. The method mainly addresses transient faults affecting the configuration registers, by detecting bit flips in the registers contents. It addresses also permanent faults on registers because it is executed at least one time within *FTTI* after a peripheral update. Method must be implemented to any configuration register whose contents are able to interfere with ITC behavior in case of incorrect settings. Check includes ITC vector table. This method can achieve high levels of Diagnostic Coverage (refer to ISO26262-5, D.2.3.7 Configuration register test) An alternative valid implementation requiring less space in SRAM can be realized on the basis of signature concept: • Peripheral registers to be checked are read in a row, computing a CRC-like checksum • Obtained signature is compared with the golden value (computed in the same way after each register update, and stored in SRAM) • Coherence between signatures is checked by the application software – signature mismatch is considered as failure detection
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on MCU configuration	None
Initialization	Values of configuration registers must be read after the boot before executing the first check
Periodicity	Periodic
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	This method addresses only failures affecting configuration registers, and not peripheral core logic or external interface. Attention must be paid to registers containing mixed combination of configuration and status bits. Mask must be used before saving register contents affecting signature, and related checks, to avoid false positive detections

Table 27. ITC_SM_1

SM CODE	ITC_SM_1
Description	Expected and unexpected interrupt check
Ownership	End user
Detailed implementation	The method of expected and unexpected interrupt check is implemented at application software level. The guidelines for the implementation of the method are the following: • The list of the implemented interrupt for the MCU are well documented, reporting also the expected frequency of each request when possible (for example the interrupts related to ADC conversion completion, therefore coming on a deterministic way). • Individual counters are maintained for each interrupt request served, in order to detect in a given time frame the cases of a) no interrupt at all b) too many interrupt requests ("babbling idiot" interrupt source). The control of the time frame duration must be regulated according to the individual interrupt expected frequency.

SM CODE	ITC_SM_1
	• Interrupt vectors related to unused interrupt source point to a default handler that reports, in case of triggering, a faulty condition (unexpected interrupt). • In case an interrupt service routine is shared between different sources, a plausibility check on the caller identity is implemented. • Interrupt requests related to non-safety-related peripherals are handled with the same method here described, despite their originator safety classification
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	In order to decrease the complexity of method implementation, it is suggested to use polling technique (when possible) instead of interrupt for end system implementation

3.5.7 Controller area network (beCAN)

Table 28. **CAN_SM_0**

SM CODE	CAN_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to CAN configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple faults protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 29. **CAN_SM_1**

SM CODE	CAN_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	CAN communication module embeds protocol error checks (like error counters) conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a marginal percentage of hardware random failures affecting the module itself. Error signals connected to these checkers are normally handled in a standard communication software, so the overhead is reduced
Error reporting	Several error condition are reported by flag bits in related CAN registers.
Fault detection time	Depends on peripheral configuration (for example baud rate), refer to functional documentation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults 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 30. **CAN_SM_2**

SM CODE	CAN_SM_2
Description	Information redundancy techniques on messages, including end-to-end protection.
Ownership	*End user*
Detailed implementation	This method aims to protect the communication between a peripheral and his external counterpart establishing a kind of "protected" channel. The aim is to specifically address communication failure modes as reported in ISO 26262-6:2018, D.2.4. Implementation guidelines are the following: • Data packet must be protected (encapsulated) by an information redundancy check, like for instance a CRC checksum computed over the packet and added to payload. Checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. • Additional field added in payload reporting an unique identification of sender or receiver and an unique increasing sequence packet number • Timing monitoring of the message exchange (for example check the message arrival within the expected time window), detecting therefore missed message arrival conditions • Application software must verify before consuming data packet its consistency (CRC check), its legitimacy (sender or receiver) and the sequence correctness (sequence number check, no packets lost)
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* 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	Important note: it is assumed that the remote CAN counterpart has an equivalent capability of performing the checks described. A major overlap between the requirements of this method and the implementation of complex communication software protocols can exists. Due to large adoption of these protocols in industrial applications, optimizations can be possible

3.5.8 Universal asynchronous receiver/transmitter (UART)

Note: The indicated set of safety mechanisms applies also when LIN option is used on UART.

Table 31. UART_SM_0

SM CODE	UART_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to UART configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Latent faults protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 32. UART_SM_1

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

Table 33. UART_SM_2

SM CODE	UART_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by UART a redundancy check with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	It is assumed that the remote UART counterpart has an equivalent capability of performing the check described. Transmission full redundancy (message repetition) should not be used because its detection capability is limited to a subset of communication unit failure modes. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated.

Table 34. UART_SM_3

SM CODE	UART_SM_3
Description	Information redundancy techniques on messages, including end-to-end protection.
Ownership	*End user*
Detailed implementation	This method aims to protect the communication between a peripheral and his external counterpart establishing a kind of “protected” channel. The aim is to specifically address communication failure modes as reported in ISO 26262-6:2018, D.2.4. Implementation guidelines are the following: • Data packet must be protected (encapsulated) by an information redundancy check, like for instance a CRC checksum computed over the packet and added to payload. Checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. • Additional field added in payload reporting an unique identification of sender or receiver and an unique increasing sequence packet number • Timing monitoring of the message exchange (for example check the message arrival within the expected time window), detecting therefore missed message arrival conditions • Application software must verify before consuming data packet its consistency (CRC check), its legitimacy (sender or receiver) and the sequence correctness (sequence number check, no packets lost)
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None

SM CODE	UART_SM_3
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Important note: it is assumed that the remote UART counterpart has an equivalent capability of performing the checks described. A major overlap between the requirements of this method and the implementation of complex communication software protocols can exists. Due to large adoption of these protocols in industrial applications, optimizations can be possible

3.5.9 Inter-integrated circuit (I2C)

Table 35. IIC_SM_0

SM CODE	IIC_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to I2C configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Latent faults protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 36. IIC_SM_1

SM CODE	IIC_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	I2C communication module embeds protocol error checks (like overrun, underrun, packet error etc.) conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a marginal percentage of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection time	Depends on peripheral configuration (for example baud rate), refer to functional documentation.
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous

SM CODE	IIC_SM_1
Test for the diagnostic	Not needed
Latent faults protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Adoption of SMBus option (when available on selected partnumber) 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 37. **IIC_SM_2**

SM CODE	IIC_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by I2C a redundancy check with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	It is assumed that the remote I2C counterpart has an equivalent capability of performing the check described. Transmission full redundancy (message repetition) should not be used because its detection capability is limited to a subset of communication unit failure modes. To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. This method is overlapped with IIC_SM_3 if hardware handled *CRC* insertion is possible.

Table 38. **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 peripheral configuration (for example baud rate), refer to functional documentation.
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation

SM CODE	IIC_SM_3
Periodicity	Continuous
Test for the diagnostic	Not needed
Latent faults protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	This method can be part of the implementation for IIC_SM_2 Enabling related interrupt generation on the detection of errors is highly recommended.

Table 39. **IIC_SM_4**

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

3.5.10 Serial peripheral interface (SPI)

Table 40. **SPI_SM_0**

SM CODE	SPI_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to SPI configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Latent faults protection	Refer to ITC_SM_0

SM CODE	SPI_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 41. **SPI_SM_1**

SM CODE	SPI_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	SPI communication module embeds protocol error checks (like overrun, underrun, timeout and so on) conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a marginal percentage of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional interrupt event generation
Fault detection time	Depends on peripheral configuration (for example baud rate), refer to functional documentation.
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Latent faults protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Enabling related interrupt generation on the detection of errors is highly recommended.

Table 42. **SPI_SM_2**

SM CODE	SPI_SM_2
Description	Information redundancy techniques on messages
Ownership	*End user*
Detailed implementation	This method is implemented adding to data packets transferred by SPI a redundancy check (such as a *CRC* check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Latent faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	It is assumed that the remote SPI counterpart has an equivalent capability of performing the check described. Transmission full redundancy (message repetition) should not be used because its detection capability is limited to a subset of communication unit failure modes.

SM CODE	SPI_SM_2
	To give an example on checksum encoding capability, using just a bit-by-bit addition is unappropriated. This method is overlapped with SSP_SM_3 if hardware handled *CRC* insertion is possible.

Table 43. **SPI_SM_3**

SM CODE	SPI_SM_3
Description	*CRC* packet-level
Ownership	ST
Detailed implementation	SPI communication module allows to activate automatic insertion (and check) of CRC-8 or CRC-18 checksums to packet data.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection time	Depends on peripheral configuration (for example baud rate), refer to functional documentation.
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Latent faults protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	This method can be part of the implementation for SPI_SM_2 Enabling related interrupt generation on the detection of errors is highly recommended.

Table 44. **SPI_SM_4**

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

3.5.11 Analog-to-digital converters (ADC)

Table 45. **ADC_SM_0**

SM CODE	ADC_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to ADC configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC)
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple faults protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 46. **ADC_SM_1**

SM CODE	ADC_SM_1
Description	Multiple acquisition by application software
Ownership	*End user*
Detailed implementation	This method implements a timing information redundancy by executing multiple acquisitions on the same input signal. Multiple acquisition data are then combined by a filter algorithm to determine the signal correct value
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	It is highly probable that this recommendation is satisfied by design by the end user application software. 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 47. **ADC_SM_2**

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

Table 48. **ADC_SM_3**

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

Table 49. **ADC_SM_4**

SM CODE	ADC_SM_4
Description	1oo2 scheme for ADC inputs
Ownership	*End user*
Detailed implementation	This safety mechanism is implemented using two different ADC channels belonging to separate ADC modules to acquire the same input signal. The application software checks the coherence between the two readings.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	This method is applicable only on partnumbers with two separate ADC modules
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple faults protection	ADC_SM_0: Periodical read-back of ADC configuration registers
Recommendations and known limitations	This method can be used in conjunction with ADC_SM_0/ ADC_SM_2/ ADC_SM_3 to achieve highest level of ADC module diagnostic coverage

3.5.12 Basic timers TIM 4/6

Table 50. **BTIM_SM_0**

SM CODE	BTIM_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to basic counter timer TIM6 or TIM7 configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC)
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple faults protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 51. **BTIM_SM_1**

SM CODE	BTIM_SM_1
Description	1oo2 for counting timers
Ownership	*End user*
Detailed implementation	This method implements via software a 1oo2 scheme between two counting resources. The guidelines for the implementation of the method are the following: • Two timers are programmed with same time base or frequency. • In case of timer use as a time base: use in the application software one of the timer as time base source, and the other one just for check. Coherence check for the 1oo2 is done at application level, comparing two counters values each time the timer value is used to affect safety function. • In case of interrupt generation usage: use the first timer as main interrupt source for the service routines, and use the second timer as a “reference” to be checked at the initial of interrupt routine
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic

3.5.13 Advanced, general purpose timers TIM1/2/3/5

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

Table 52. **TIM_SM_0**

SM CODE	TIM_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to advanced, general and low-power timers TIM1/2/3/5 configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple faults protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 53. **ATIM_SM_1**

SM CODE	TIM_SM_1
Description	1oo2 for counting timers
Ownership	*End user*
Detailed implementation	This method implements via software a 1oo2 scheme between two counting resources. The guidelines for the implementation of the method are the following: • Two timers are programmed with same time base or frequency. • In case of timer use as a time base: use in the application software one of the timer as time base source, and the other one just for check. Coherence check for the 1oo2 is done at application level, comparing two counters values each time the timer value is used to affect safety function. • In case of interrupt generation usage: use the first timer as main interrupt source for the service routines, and use the second timer as a “reference” to be checked at the initial of interrupt routine
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic. This method apply to timer channels merely used as elapsed time counters

Table 54. **TIM_SM_2**

SM CODE	TIM_SM_2
Description	1oo2 for input capture timers
Ownership	*End user*
Detailed implementation	This method is conceived to protect timers used for external signal acquisition and measurement, like “input capture” and “encoder reading”. Implementation requires to connect the external signals also to a redundant timer, and to perform a coherence check on the measured data at application level. Coherence check between timers is executed each time the reading is used by the application software
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	To reduce the potential effect of common cause failures, it is suggested to use for redundant check a channel belonging to a different timer module and mapped to non-adjacent pin on the device package

Table 55. **TIM_SM_3**

SM CODE	TIM_SM_3
Description	Loop-back scheme for PWM outputs
Ownership	*End user*
Detailed implementation	This method is implemented by connecting the PWM to a separate timer channel to acquire the generated waveform characteristics. The guidelines are the following: • Both PWM frequency and duty cycle are measured and checked versus the expected value. • To reduce the potential effect of common cause failure, it is suggested to use for the loopback check a channel belonging to a different timer module and mapped to non-adjacent pins on the device package. This measure can be replaced under the end-user responsibility by different loopback schemes already in place in the final application and rated as equivalent. For example if the PWM is used to drive an external power load, the reading of the on-line current value can be used instead of the PWM duty cycle measurement.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and Transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of PWM wrong signal permanence (wrong frequency, wrong duty, or both) required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm

Table 56. **TIM_SM_4**

SM CODE	TIM_SM_4
Description	Lock bit protection for timers
Ownership	ST
Detailed implementation	This safety mechanism allows the end user to lock down specific configuration options, (like lock break, dead time and output idle state registers setting) avoiding unintended modifications by application software. It addresses therefore software development systematic faults
Error reporting	NA
Fault detection time	NA
Addressed fault model	None (Systematic)
Dependency on *Device* configuration	None
Initialization	Lock protection must be enabled using LOCK bits in the related register
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults protection	NA
Recommendations and known limitations	This method does not addresses timer configuration changes due to soft-errors

3.5.14 General-purpose input/output (GPIO)

Table 57. GPIO_SM_0

SM CODE	GPIO_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to GPIO configuration registers. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	GPIO availability can differ according to part number
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple-fault protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 58. GPIO_SM_1

SM CODE	GPIO_SM_1
Description	1oo2 for input GPIO lines
Ownership	*End user*
Detailed implementation	This method addresses GPIO lines used as inputs. Implementation is done by connecting the external safety-related signal to two independent GPIO lines. Comparison between the two GPIO values is executed by application software each time the signal is used to affect application software behavior.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed
Multiple-fault protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	To reduce the potential impact of common cause failure, it is recommended to use GPIO lines: • belonging to different I/O ports (for instance port B and C) • with different bit port number (for instance PB5 and PC1) • mapped to non-adjacent pins on the device package

Table 59. GPIO_SM_2

SM CODE	GPIO_SM_2
Description	Loopback scheme for output GPIO lines

SM CODE	GPIO_SM_2
Ownership	*End user*
Detailed implementation	This method addresses GPIO lines used as outputs. Implementation is done by a loopback scheme, connecting the output to a different GPIO line programmed as input and by using the input line to check the expected value on output port. Comparison is executed by application software periodically and each time output is updated.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	To reduce the potential impact of common cause failure, it is recommended to use GPIO lines: • belonging to different I/O ports (for instance port B and C) • with different bit port number (for instance PB5 and PC1) • mapped to non-adjacent pins on the device package Efficiency versus transient failures is linked to final application characteristics. We define as Tm the minimum duration of GPIO output wrong signal permanence required to violate the related safety function(s). Efficiency is maximized when execution test frequency is higher than 1/Tm.

3.5.15 Power controller (PWR)

Table 60. **VSUP_SM_0**

SM CODE	VSUP_SM_0
Description	External watchdog
Ownership	End user
Detailed implementation	Failures in the power supplies for digital logic (core or peripherals) may lead to alteration of the application software timing, which can be detected by the external watchdog as safety mechanism introduced to monitor the application software control flow. Refer to CPU_SM_1 and CPU_SM_4 for further information.
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval)
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_1: control flow monitoring in application software
Recommendations and known limitations	-

Table 61. VSUP_SM_1

SM CODE	VSUP_SM_1
Description	System-level power supply management
Ownership	*End user*
Detailed implementation	This method is implemented at system level in order to guarantee the stability of power supply value over time. It can include a combination of different overlapped solutions, some listed here below (but not limited to): • Additional voltage monitoring by external components • Passive electronics devices able to mitigate overvoltage • Specific design of power regulator in order to avoid power supply perturbation in presence of a single failure
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Fault avoidance
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	N/A
Multiple faults protection	N/A
Recommendations and known limitations	Usually this method is already required/implemented to guarantee the stability of each component of the final electronic board

3.5.16 Reset (RST) and Clock control (CLK)

Table 62. CLK_SM_0

SM CODE	CLK_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to configuration registers for clock and reset system (refer to related register map). Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple-fault protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 63. **CLK_SM_1**

SM CODE	CLK_SM_1
Description	Clock security system (CSS)
Ownership	ST
Detailed implementation	The clock security system (CSS) detects the loss of high-speed external (HSE) oscillator clock activity and executes the corresponding recovery action, such as: • Switch-off HSE • Commutation on the HSI • Generation of related NMI
Error reporting	NMI
Fault detection time	Depends on implementation (clock frequency value).
Addressed fault model	Permanent and transient
Dependency on *Device* configuration	None
Initialization	CSS protection must be enabled through Clock interrupt register (RCC_CIR) after boot stabilization.
Periodicity	Continuous
Test for the diagnostic	CLK_SM_0: periodical read-back of configuration registers
Multiple-fault protection	CPU_SM_5: external watchdog
Recommendations and known limitations	It is recommended to carefully read reference manual instruction on NMI generation, in order to correctly managing the faulty situation by *Application software*.

Table 64. **CLK_SM_2**

SM CODE	CLK_SM_2
Description	External watchdog
Ownership	End user
Detailed implementation	The external watchdog is able to detect failures in internal main *MCU* clock (lower frequency).
Error reporting	Reset signal generation
Fault detection time	Depends on implementation (watchdog timeout interval).
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPU_SM_1: control flow monitoring in application software
Recommendations and known limitations	If watchdog window function is used, *End user* must consider possible tolerance in application software execution, to avoid false error reports (affecting system availability).

3.5.17 Independent and system window watchdogs (IWDG and WWDG)

Table 65. **WDG_SM_0**

SM CODE	WDG_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*

SM CODE	WDG_SM_0
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 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple-fault protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

Table 66. **WDG_SM_1**

SM CODE	WDG_SM_1
Description	Software test for watchdog at startup
Ownership	*End user*
Detailed implementation	This safety mechanism ensures the right functionality of the internal watchdogs in use. At startup, the software test programs the watchdog with the required expiration timeout, stores a specific non-trivial code in SRAM and waits for the reset signal. After the watchdog reset, the software understands that the watchdog has correctly triggered, and does not execute the procedure again.
Error reporting	Depends on implementation
Fault detection time	Depends on implementation
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Startup (see below)
Test for the diagnostic	Not needed
Multiple-fault protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	In a typical *End user* application, this test can be executed only at startup and so it cannot be accounted for a diagnostic coverage contribution during operating time.

3.5.18 Single wire interface module (SWIM) and debug module (DM)

Table 67. **DBG_SM_0**

SM CODE	DBG_SM_0
Description	Watchdog protection
Ownership	ST
Detailed implementation	The debug unintentional activation due to hardware random fault results in the massive disturbance of *CPU* operations, leading to intervention of the independent watchdog or alternately, the other system watchdog WWGDG or an external one.
Error reporting	Reset signal generation

SM CODE	DBG_SM_0
Fault detection time	Depends on implementation (watchdog timeout interval).
Addressed fault model	Permanent
Dependency on *Device* configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple-fault protection	CPU_SM_1: control flow monitoring in application software
Recommendations and known limitations	None

3.5.19 System configuration controller (SYSCFG)

Table 68. **SYSCFG_SM_0**

SM CODE	SYSCFG_SM_0
Description	Periodical read-back of configuration registers
Ownership	*End user*
Detailed implementation	This method must be applied to System Configuration controller configuration registers. This method is strongly recommended to protect registers related to hardware diagnostics activation and error reporting chain related features. Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple-fault protection	Refer to ITC_SM_0
Recommendations and known limitations	This method is mainly overlapped by several other “configuration register read-backs” required for other *MCU* peripherals. It is reported here for the sake of completeness.

Table 69. **DIAG_SM_0**

SM CODE	DIAG_SM_0
Description	Periodical read-back of hardware diagnostics configuration registers
Ownership	*End user*
Detailed implementation	In STM8AF Series a few hardware-based safety mechanisms are available (they are reported in this manual with the wording Ownership=ST). This method must be applied to any configuration register related to diagnostic measure operations, including error reporting. *End user* must therefore individuate configuration registers related to: • Hardware diagnostic enable • Interrupt/NMI enable (if used for diagnostic error management)
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0

SM CODE	DIAG_SM_0
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple-fault protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

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

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

Table 70. **FFI_SM_0**

SM CODE	FFI_SM_0
Description	Unused peripherals disable
Ownership	*End user*
Detailed implementation	This method contributes to the reduction of the probability of cross-interferences caused by peripherals not used by the software application, in case a hardware failure causes an unintentional activation. After the system boot, the application software must disable all unused peripherals, also by using Peripheral clock-gating (PCG) to avoid clock propagation into non used logic.
Error reporting	NA
Fault detection time	NA
Addressed fault model	NA
Dependency on *Device* configuration	None
Initialization	NA
Periodicity	Startup
Test for the diagnostic	Not needed
Multiple faults protection	FFI_SM_1: Periodical read-back of interference avoidance registers
Recommendations and known limitations	None

Table 71. **FFI_SM_1**

SM CODE	FFI_SM_1
Description	Periodical read-back of interference avoidance registers
Ownership	*End user*
Detailed implementation	This method contributes to the reduction of the probability of cross-interferences between peripherals that can potentially conflict on the same input/output pins, including for instance unused peripherals. This diagnostic measure must be applied to following registers: • clock enable and disable registers • alternate function programming registers Detailed information on the implementation of this method can be found in Section 3.5.6 Interrupt controller (ITC).
Error reporting	Refer to ITC_SM_0
Fault detection time	Refer to ITC_SM_0

SM CODE	FFI_SM_1
Addressed fault model	Refer to ITC_SM_0
Dependency on *Device* configuration	Refer to ITC_SM_0
Initialization	Refer to ITC_SM_0
Periodicity	Refer to ITC_SM_0
Test for the diagnostic	Refer to ITC_SM_0
Multiple faults protection	Refer to ITC_SM_0
Recommendations and known limitations	Refer to ITC_SM_0

3.6 List of required safety mechanisms

The table below provides a summary of the safety concept recommendations reported in Section 3.5 Hardware and software diagnostics.

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

- M = this safety mechanism is always operating during normal operations – no end user activity can deactivate it.
- ++ = Highly recommended being a common practice and considered in this Safety Manual for the computation of the safety metrics and as an integral part of the safety concept. Missing implementation may lead to invalidate any safety feature claimed in this manual.
- + = Recommended as additional safety measure, but not considered in this Safety Manual for the computation of safety metrics. STM8AF Series users can skip the implementation in case it is in contradiction with functional requirements or overlapped by another mechanism marked as "++".
- o = optional, not needed or related to specific MCU configuration

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

Table 72. **List of safety mechanisms**

Device function	Diagnostic	Description	Rank	Perm	Trans
STM8A CPU	CPU_SM_0	Periodical software test addressing permanent faults in STM8A CPU core	++	X	-
	CPU_SM_1	Control flow monitoring in application software	++	X	X
	CPU_SM_2	Double computation in application software	++	-	X
	CPU_SM_3	STM8A CPU illegal op-code protection	M	X	X
	CPU_SM_4	Stack hardening for application software	+	X	X
	CPU_SM_5	External watchdog	+	X	X
	CPU_SM_6	Window watchdog (WWDG)	+	X	X
	CPU_SM_7	Independent watchdog (IWDG)	+	X	X
Embedded Flash memory	FLASH_SM_0	Periodical software test for Flash memory	++	X	-
	FLASH_SM_1	Control flow monitoring in application software	++	X	X
	FLASH_SM_2	Flash (including option bytes) write protection	M	-	-
	FLASH_SM_3	Static data encapsulation	++	X	X
	FLASH_SM_4	Flash unused area filling code	+	-	-
	BOOT_SM_0	Loaded application software integrity test	++	X	X
Embedded SRAM	RAM_SM_0	Periodical software test for SRAM memory	++	X	-
	RAM_SM_1	Stack hardening for application software	+	X	X

Device function	Diagnostic	Description	Rank	Perm	Trans
Embedded SRAM	RAM_SM_2	Information redundancy for system variables in application software	++	X	X
	RAM_SM_3	Control flow monitoring in application software	o[1]	X	X
	RAM_SM_4	Periodical integrity test for application software in RAM	o[1]	X	X
Address and Data bus	BUS_SM_0	Periodical software test for interconnections	++	X	-
	BUS_SM_1	Information redundancy in intra-chip data exchanges	++	X	X
Data EEPROM memory	EEP_SM_0	Information redundancy	++	X	X
	EEP_SM_1	Software read-back after write operation	+	X	X
Interrupt controller (ITC)	ITC_SM_0	Periodical read-back of configuration registers	++	X	X
	ITC_SM_1	Expected and unexpected interrupt check by application software	++	X	X
Control area network (beCAN)	CAN_SM_0	Periodical 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
UART	UART_SM_0	Periodical read-back of configuration registers	++	X	X
	UART_SM_1	Protocol error signals	++	X	X
	UART_SM_2	Information redundancy techniques on messages	++	X	X
	UART_SM_3	Information redundancy techniques on messages, including end-to-end protection	++	X	X
I2C	IIC_SM_0	Periodical read-back of configuration registers	++	X	X
	IIC_SM_1	Protocol error signals	++	X	X
	IIC_SM_2	Information redundancy techniques on messages	++	X	X
	IIC_SM_3	CRC packet-level	+	X	X
	IIC_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X
SPI	SPI_SM_0	Periodical read-back of configuration registers	++	X	X
	SPI_SM_1	Protocol error signals	++	X	X
	SPI_SM_2	Information redundancy techniques on messages	++	X	X
	SPI_SM_3	CRC packet-level	+	X	X
	SPI_SM_4	Information redundancy techniques on messages, including end-to-end protection	+	X	X
ADC	ADC_SM_0	Periodical read-back of configuration registers	++	X	X
	ADC_SM_1	Multiple acquisition by application software	++	-	X
	ADC_SM_2	Range and plausibility check by application software	++	X	X
	ADC_SM_3	Periodical software test for ADC	++	X	-
	ADC_SM_4	1oo2 scheme for ADC inputs	+	X	X
Basic timers TIM4/6	BTIM_SM_0	Periodical read-back of configuration registers	++	X	X
	BTIM_SM_1	1oo2 for counting timers	++	X	X
Advanced, general and low-power timers TIM1/2/3/5	TIM_SM_0	Periodical read-back of configuration registers	++	X	X
	TIM_SM_1	1oo2 for counting timers	++	X	X
	TIM_SM_2	1oo2 for input capture timers	++	X	X
	TIM_SM_3	Loopback scheme for PWM outputs	++	X	X

Device function	Diagnostic	Description	Rank	Perm	Trans
Advanced, general and low-power timers TIM1/2/3/5	TIM_SM_4	Lock bit protection for timers	+(2)	-	-
GPIO	GPIO_SM_0	Periodical read-back of configuration registers	++	X	X
	GPIO_SM_1	1oo2 for input GPIO lines	++	X	X
	GPIO_SM_2	Loopback scheme for output GPIO lines	++	X	X
	GPIO_SM_3	GPIO port configuration lock register	+	-	-
RTC	RTC_SM_0	Periodical read-back of configuration registers	++	X	X
	RTC_SM_1	Application check of running RTC	++	X	X
	RTC_SM_2	Application-level measures to detect failures in timestamps or event capture	o	X	X
PWR	VSUP_SM_0	External Watchdog	++	X	-
	VSUP_SM_1	System-level power supply management	++	X	-
Clock and Reset	CLK_SM_0	Periodical read-back of configuration registers	++	X	X
	CLK_SM_1	Clock security system (CSS)	++	X	-
	CLK_SM_2	Independent Watchdog	++	X	-
IWDG/WWDG	WDG_SM_0	Periodical read-back of configuration registers	++	X	X
	WDG_SM_1	Software test for watchdog at startup	o	X	-
SWIM/DM	DBG_SM_0	Independent watchdog	++	X	X
System configuration controller	LOCK_SM_0	Lock mechanism for configuration options	+	-	-
	SYSCFG_SM_0	Periodical read-back of configuration registers	++	X	X
	DIAG_SM_0	Periodical read-back of hardware diagnostics configuration registers	++	X	X
Device	FFI_SM_0	Unused peripherals disable	++	-	-
	FFI_SM_1	Periodical read-back of interference avoidance registers	++	-	-
	AoU_1	End user must implement the required combination of safety mechanism for each Device peripheral used in the implementation of safety function(s)	++	X	X
Flash memory	AoU_2	During Flash memory bank mass erase and reprogramming there must not be safety functions(s) executed by Device.	++	-	-

1. *Must be considered ranked as “++” if the application software is executed on RAM.*
2. *Must be considered ++ for motor control applications*

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

The safety analysis highlights two major partitions inside the MCU:

- System-critical MCU modules. Every end-user application is affected from a safety point of view by a failure on these modules. Because these modules are used by every end user application, related methods or safety mechanism are mainly conceived to be application-independent. MCU system critical modules are: CPU, CLK, RST, Power, Address and Data bus and Flash and RAM memories (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.

3.7 Notes on latent faults mitigation

ISO 26262 defines also a metric for “latent” faults. The latent fault is a multiple-point fault which presence is not detected by a safety mechanism nor perceived by the driver within the multiple-point fault detection interval. In practical words, the latent fault is a combination of a fault in a safety mechanism - that by itself does NOT cause the violation of the safety goal (function) - and a fault in the mission logic supervised by that safety mechanism.

Because of the nature of Device, its safety concept includes a very limited amount of safety mechanisms implemented in the hardware structure, while the major part of them are actually implemented by software. Because of that, and due to the specific computation formula indicated in ISO26262:5, safety metrics related to latent faults are easily well above the limit fixed for the assumed target safety integrity level. Anyway, the section 3.6 tables includes a dedicated field where the measure to mitigate latent faults for the given safety mechanism is indicated. It is worth to note that for each software-based safety mechanisms (e.g. peripheral configuration read back), the CPU has been considered as “acting as safety mechanisms” because executing the software implementing the test. Accordingly, the periodical test for CPU integrity is indicated as mitigation for latent faults.

4 Safety results

This section reports the results of the safety analysis of the STM8AF Series devices, according to ISO26262:2018 standard 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 STM8AF Series devices reported in this safety manual is executed according to STMicroelectronics methodology flow for safety analysis of semiconductor devices in compliance with ISO26262:2018. The accuracy of results obtained are guaranteed by two factors:

- STMicroelectronics methodology flow strict adherence to ISO26262:2018 requirements and prescriptions
- the use, during the analysis, of detailed and reliable information on microcontroller design

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 related *FMEA* document (reference [1]), provided on demand by local STMicroelectronics sales office.

The resulting relative safety metrics and absolute safety metrics are not reported in this section but in the failure mode effect diagnostic analysis (FMEDA) snapshot, due to:

- a large number of different STM8AF 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 (reference [1]) is a static document reporting the safety metrics computed at different detail levels (at microcontroller level and for microcontroller basic functions) for a given combination of safety mechanisms and for a given part number. If *FMEDA* computation sheet is needed, early contact the local STMicroelectronics sales representative, in order to receive information on expected delivery dates for specific device target part number.

Note: Safety metrics computations are restricted to STM8AF Series boundary, hence they do not include the prescripted external watchdog nor any other external component.).

In summary, with the adoption of the safety mechanisms and conditions of use reported in List of required safety mechanisms, Device is suitable to be used in systems implementing safety goals (functions) up to ASIL B

4.1.1 Safety analysis result customization

The safety analysis executed for STM8AF Series devices documented in this safety manual considers all microcontrollermicroprocessor modules used during operational time 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 microcontrollermicroprocessor, in order to be agnostic versus the final application. This means that no MCU module has been declared *safe* as per ISO2626:1, except some logic belonging to debug/AWU/BEEP functions as per ASR4 (refer to Section 3.3 Evaluation of the safety analysis assumptions).

Therefore, all microcontroller functional modules are included in SPFM computations.

In actual *End user* applications, not all the STM8AF 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 could be considered in principle as non safety-related (for example, a GPIO line driving a *power-on* signaling light on an electronic board).

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 STM8AF 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.
- Fulfilling the general condition for the mitigation of intra-*MCU* interferences (see Table 72. List of safety mechanisms).

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 72. List of safety mechanisms.

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 Safety analysis result customization.

End user must implement the safety mechanisms listed in Table 73 (implementation details in Hardware and software diagnostics) regardless any evaluation of their contribution to safety metrics.

Table 73. **List of general requirements for FFI**

Diagnostic	Description
FFI_SM_0	Unused peripheral disable
FFI_SM_1	Periodical read-back of interference avoidance registers
BUS_SM_0	Periodical software test for interconnections
ITC_SM_0	Periodical read-back of configuration registers
ITC_SM_1	Expected and unexpected interrupt check by application software
GPIO_SM_0	Periodical read-back of configuration registers

4.2 Analysis of dependent failures

The analysis of dependent failures is important for microcontroller and microprocessor devices. The analysis executed on Device is compliant with requirements and guidelines reported in ISO26262:11, section 4.7 Semiconductor dependent failure analysis.

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

4.2.1 Power supply

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

- VSUP_SM_1: the system-level external measures mitigate the possibility of abnormal power supply values;
- VSUP_SM_0: the external watchdog is different from the digital core of the *MCU*, and this diversity helps to mitigate dependent failures related to the main supply alterations, because affected in a different way by power supply anomalies.

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

4.2.2 Clock

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

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

The adoption of such safety mechanism is therefore highly recommended despite their minor contribution to the safety metrics to reach the required safety integrity level. Refer to Section 3.5.16 Reset (RST) and Clock control (CLK) for detailed safety mechanisms description.

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

As these materials contain STMicroelectronics' confidential information, they are only available for the purpose of audit and inspection by authorized bodies, without being published.

Revision history

Table 74. Document revision history

Date	Version	Changes
07-Jul-2015	1	Initial version
18-May-2018	2	Updated Introduction, Section 1.3: Reference normative, Section 1.4: Annexes, Section 3.4: Electrical specifications and environment limits, Section 3.6: Safety mechanisms/measures, Section 4: Safety analysis results, Section 4.1: Hardware random failure analysis, Section 4.1.1: Safety analysis result customization, Section 4.1.2: General requirements for FFI (freedom from interferences),Section 4.2: Dependent failures analysis, Section 5: List of evidences, Appendix A: Change impact analysis for other safety standards and its subsections. Updated Table 2: List of STM8AF assumed requirements, Table 4: List of safety mechanisms, Table 6: Some reference architectures for IEC 61508 and Table 8: IEC 61508 work product grid. Updated Figure 1: Definition of the STM8AF as a SEooC, Figure 2: Relationship between assumptions and SEooC development, Figure 3: STM8AF FTTI allocation and cycle time and Figure 4: Correlation matrix between SIL and ASIL. Removed former Section 2.2: YOGITECH fRMethodology process, Appendix A: Overview of fRMethodology, B.1: IEC 60730-1:2010, B.2: Architectural categories, B.3: Safety metrics re-computation and B.4: Work products. Minor text edits across the whole document.
01-Oct-2019	3	Updated functional safety documentation framework. Updated Section 1.3: Reference normative. Minor text edits across the whole document.
30-Apr-2020	4	Update of the document structure and content.

Glossary

AoU Assumption of use

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

ASIL Automotive safety integrity level

CCF common cause failure

COTS commercial off-the-shelf

CPU central processing unit

CRC cyclic redundancy check

DC diagnostic coverage

Device depending on context, any single or all of the STM8AF Series silicon products

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

FIT failure in time

FMEA failure mode effect analysis

FMEDA failure mode effect diagnostic analysis

FTTI Fault tolerant time interval

HFT hardware fault tolerance

HW hardware

ITRS international technology roadmap for semiconductors

MCU microcontroller unit

MPFDI multiple point fault detection interval

MTBF mean time between failures

NA not applicable/available

PEc computation processing element

PEd diagnostic processing element

PEi input processing elements

PEo output processing elements

PMHF probabilistic metric for random hardware failures

SPFM single point fault metric

Contents

List of tables

List of figures