

## Introduction

This document describes how to use the microcontrollers of STM32F0 Series in the context of a safety-related system, specifying the user's responsibilities for installation and operation in order to reach the targeted safety integrity level.

This manual applies to the microcontrollers of the STM32F0 Series and to STM32-SafeSIL part number.

If the STM32F0 Series microcontrollers are used in adherence to this manual, system designers avoid going into the details of the application of functional safety standards to STM32F0 microcontrollers.

This manual is written in compliance with IEC 61508. It indicates how to use the STM32F0 Series microcontrollers in the context of other functional safety standards such as safety machine directives ISO 13849.

The safety analysis summarized in this manual takes into account the variation in terms of memory size, internal peripheral number and package among the different part numbers of the Arm<sup>®</sup> Cortex<sup>®</sup>-M0 based STM32F0 Series microcontrollers.

This manual has to be read along with the technical documentation on related part numbers (such as reference manuals and datasheets) available on [www.st.com](http://www.st.com).

# Contents

<b>1</b>	<b>About this document</b>	<b>9</b>
1.1	Purpose and scope	9
1.2	Terms and abbreviations	9
1.3	Reference normative	10
<b>2</b>	<b>STM32F0 Series microcontroller development process</b>	<b>12</b>
2.1	STMicroelectronics standard development process	12
<b>3</b>	<b>Reference safety architecture</b>	<b>14</b>
3.1	Introduction	14
3.2	Compliant item	14
3.2.1	Definition of the compliant item	14
3.2.2	Safety functions performed by the compliant item	15
3.2.3	Reference Safety Architectures - 1001	15
3.2.4	Reference Safety Architectures - 1002	16
3.3	Assumed requirements	17
3.3.1	Assumed safety requirements	17
3.4	Electrical specifications and environment limits	19
3.5	Systematic safety integrity	20
3.6	Description of hardware and software diagnostics	20
3.6.1	Cortex®-M0 CPU	21
3.6.2	System FLASH memory	26
3.6.3	System SRAM memory	30
3.6.4	System bus interconnect	33
3.6.5	NVIC and EXTI controller	35
3.6.6	DMA	37
3.6.7	CAN	40
3.6.8	USART 1/2/3/4/5/6/7/8	42
3.6.9	I2C 1/2	44
3.6.10	SPI 1/2	47
3.6.11	USB - 2.0 Universal Serial Bus interface FS module	49
3.6.12	HDMI CEC module	51
3.6.13	Touch Sensing Controller (TSC)	53

3.6.14	Analog to Digital Converters (ADC)	55
3.6.15	DAC	57
3.6.16	Comparator	58
3.6.17	TIM 6/7	60
3.6.18	TIM1/2/3/14/15/16/17	61
3.6.19	GPIO - PORT A/B/C/D/E/F	65
3.6.20	Real Time Clock module (RTC)	67
3.6.21	Supply voltage system	69
3.6.22	Reset and clock control subsystem	72
3.6.23	Watchdogs (IWDG, WWDG)	74
3.6.24	Debug	75
3.6.25	Cyclic Redundancy Check module (CRC)	75
3.6.26	System configuration controller (SYSCFG)	76
3.6.27	Disable and periodic cross-check of unintentional activation of unused peripherals	76
3.6.28	Notes on multiple faults scenario	78
3.7	Conditions of use	78
<b>4</b>	<b>Safety results</b>	<b>83</b>
4.1	Random hardware failure safety results	83
4.1.1	Safety analysis results customization	84
4.1.2	General requirements for freedom from interferences (FFI)	84
4.2	Dependent failures analysis	85
4.2.1	Power supply	85
4.2.2	Clock	86
4.2.3	DMA	86
4.2.4	Internal temperature	86
<b>5</b>	<b>List of evidences</b>	<b>87</b>
<b>Appendix A</b>	<b>Change impact analysis for other safety standards.</b>	<b>88</b>
A.1	ISO 13849-1 / ISO 13849-2.	88
A.1.1	Architectural categories	88
A.1.2	Safety metrics computation	91
A.1.3	Work products.	92
A.2	IEC 62061:2012-11	94
A.2.1	Architectural categories	95

---

A.2.2	Safety metrics computation .....	97
A.2.3	Work products .....	98
A.3	IEC 61800-5-2:2007 .....	99
A.3.1	Architectural categories .....	99
A.3.2	Safety metrics computation .....	100
A.3.3	Work products .....	100
A.4	IEC 60730-1:2010 .....	101
A.4.1	Architectural categories .....	101
A.4.2	Safety metrics computation .....	102
A.4.3	Work products .....	107
A.5	ISO 26262:2010 .....	109
A.5.1	Architectural categories .....	110
A.5.2	Safety metrics computation .....	110
A.5.3	Work products .....	111
<b>Revision history</b>	.....	<b>112</b>

# List of tables

Table 1.	Terms and abbreviations .....	9
Table 2.	Mapping between this document content and IEC 61508-2 Annex D requirements .....	11
Table 3.	SS1 and SS2 safe state details .....	19
Table 4.	Safety mechanism field explanation .....	20
Table 5.	CPU_SM_0 .....	21
Table 6.	CPU_SM_1 .....	22
Table 7.	CPU_SM_2 .....	23
Table 8.	CPU_SM_3 .....	23
Table 9.	CPU_SM_4 .....	24
Table 10.	CPU_SM_5 .....	24
Table 11.	CPU_SM_6 .....	25
Table 12.	FLASH_SM_0 .....	26
Table 13.	FLASH_SM_1 .....	27
Table 14.	FLASH_SM_2 .....	27
Table 15.	FLASH_SM_3 .....	28
Table 16.	FLASH_SM_4 .....	28
Table 17.	FLASH_SM_5 .....	29
Table 18.	FLASH_SM_6 .....	29
Table 19.	RAM_SM_0 .....	30
Table 20.	RAM_SM_1 .....	30
Table 21.	RAM_SM_2 .....	31
Table 22.	RAM_SM_3 .....	31
Table 23.	RAM_SM_4 .....	32
Table 24.	RAM_SM_5 .....	33
Table 25.	BUS_SM_0 .....	33
Table 26.	BUS_SM_1 .....	34
Table 27.	LOCK_SM_0 .....	34
Table 28.	NVIC_SM_0 .....	35
Table 29.	NVIC_SM_1 .....	36
Table 30.	DMA_SM_0 .....	37
Table 31.	DMA_SM_1 .....	37
Table 32.	DMA_SM_2 .....	38
Table 33.	DMA_SM_3 .....	38
Table 34.	DMA_SM_4 .....	39
Table 35.	CAN_SM_0 .....	40
Table 36.	CAN_SM_1 .....	40
Table 37.	CAN_SM_2 .....	41
Table 38.	UART_SM_0 .....	42
Table 39.	UART_SM_1 .....	42
Table 40.	UART_SM_2 .....	43
Table 41.	UART_SM_3 .....	43
Table 42.	IIC_SM_0 .....	44
Table 43.	IIC_SM_1 .....	44
Table 44.	IIC_SM_2 .....	45
Table 45.	IIC_SM_3 .....	45
Table 46.	IIC_SM_4 .....	46
Table 47.	SPI_SM_0 .....	47

Table 48.	SPI_SM_1	47
Table 49.	SPI_SM_2	48
Table 50.	SPI_SM_3	48
Table 51.	SPI_SM_4	49
Table 52.	USB_SM_0	49
Table 53.	USB_SM_1	50
Table 54.	USB_SM_2	50
Table 55.	USB_SM_3	51
Table 56.	HDMI_SM_0	51
Table 57.	HDMI_SM_1	52
Table 58.	HDMI_SM_2	52
Table 59.	TSC_SM_0	53
Table 60.	TSC_SM_1	53
Table 61.	TSC_SM_2	54
Table 62.	ADC_SM_0	55
Table 63.	ADC_SM_1	55
Table 64.	ADC_SM_2	56
Table 65.	ADC_SM_3	56
Table 66.	DAC_SM_0	57
Table 67.	DAC_SM_1	57
Table 68.	COMP_SM_0	58
Table 69.	COMP_SM_1	58
Table 70.	COMP_SM_2	59
Table 71.	COMP_SM_3	59
Table 72.	COMP_SM_4	60
Table 73.	GTIM_SM_0	60
Table 74.	GTIM_SM_1	61
Table 75.	ATIM_SM_0	62
Table 76.	ATIM_SM_1	62
Table 77.	ATIM_SM_2	63
Table 78.	ATIM_SM_3	63
Table 79.	ATIM_SM_4	64
Table 80.	GPIO_SM_0	65
Table 81.	GPIO_SM_1	65
Table 82.	GPIO_SM_2	66
Table 83.	GPIO_SM_3	66
Table 84.	RTC_SM_0	67
Table 85.	RTC_SM_1	68
Table 86.	RTC_SM_2	68
Table 87.	RTC_SM_3	69
Table 88.	VSUP_SM_0	69
Table 89.	VSUP_SM_1	70
Table 90.	VSUP_SM_2	70
Table 91.	VSUP_SM_3	71
Table 92.	CLK_SM_0	72
Table 93.	CLK_SM_1	72
Table 94.	CLK_SM_2	73
Table 95.	CLK_SM_3	73
Table 96.	WDG_SM_0	74
Table 97.	WDG_SM_1	74
Table 98.	DBG_SM_0	75
Table 99.	CRC_SM_0	75

---

Table 100.	SYSCFG_SM_0	76
Table 101.	FFI_SM_0	77
Table 102.	FFI_SM_1	77
Table 103.	List of safety mechanisms	79
Table 104.	Overall achievable safety integrity levels	83
Table 105.	List of general requirements for FFI	85
Table 106.	IEC 13849 architectural categories	89
Table 107.	IEC 13849 work product grid	92
Table 108.	SIL classification versus HFT	95
Table 109.	IEC 62061 architectural categories	96
Table 110.	IEC 62061 work product grid	98
Table 111.	IEC 61800 work product grid	100
Table 112.	IEC 60730 required safety mechanism for Class B/C compliance	103
Table 113.	IEC 60730 work product grid	107
Table 114.	IEC 26262 work product grid	111
Table 115.	Document revision history	112

## List of figures

Figure 1.	STMicroelectronics product development process . . . . .	13
Figure 2.	Definition of the compliant item. . . . .	14
Figure 3.	1oo1 reference architecture . . . . .	16
Figure 4.	1oo2 reference architecture . . . . .	17
Figure 5.	Allocation and target for STM32 PST . . . . .	18
Figure 6.	Block diagram for IEC 13849 Cat. B and Cat. 1 . . . . .	90
Figure 7.	Block diagram for IEC 13849 Cat. 2 . . . . .	90
Figure 8.	Block diagram for IEC 13849 Cat. 3 and Cat. 4 . . . . .	91
Figure 9.	SRECS high-level diagram . . . . .	97
Figure 10.	Correlation matrix between SIL and ASIL. . . . .	110



# 1 About this document

## 1.1 Purpose and scope

This document describes how to use the STM32F0 Series microcontrollers 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.

This document is useful to system designers willing evaluate the safety of their solution embedding one or more STM32F0 microcontroller(s).

## 1.2 Terms and abbreviations

**Table 1. Terms and abbreviations**

Acronym	Definition
CCF	Common cause failure
CM	Continuous mode
COTS	Commercial off-the-shelf
CoU	Conditions of use
CPU	Central processing unit
CRC	Cyclic redundancy check
DC	Diagnostic coverage
DMA	Direct memory access
DTI	Diagnostic test interval
ECM	Engine control module
ECU	Electronic control unit
EUC	Equipment under control
FIT	Failure in time
FMEA	Failure mode effect analysis
FMEDA	Failure mode effect diagnostic analysis
HD	High demand
HFT	Hardware fault tolerance
HW	Hardware
ITRS	International technology roadmap for semiconductors
LD	Low demand
MCU	Microcontroller unit
MTBF	Mean time between failure
MTTFd	Mean time to failure
NA	Not Available

**Table 1. Terms and abbreviations (continued)**

Acronym	Definition
PDS(SR)	Power drive system (safety related)
PEc	Programmable electronics - core
PEd	Programmable electronics - diagnostic
PFH	Probability of failure per hour
PL	Performance level
PST	Process safety time
SFF	Safe failure fraction
SIL	Safety integrity level
SRCF	Safety-related control function
SRECS	Safety-related electrical control systems
SRP/CS	Safety-related parts of control systems
SW	Software

Abbreviations related to STM32F0 hardware modules (e.g. DMA, GPIO etc.) are the same used in STM32F0 technical documentation.

Read also following definitions used within this manual:

- end user: the STM32F0 final user of that is in charge of integrating the MCU in a real application (for example an electronic control board).
- application software: the actual software running on the STM32F0 MCU and implementing the safety function

### 1.3 Reference normative

This document is written in compliance with the IEC 61508 international norm for functional safety of electrical, electronic and programmable electronic safety-related systems.

The version used as reference is IEC 61508:1-7 © IEC:2010.

The other functional safety standards considered in this manual are the following:

- ISO 26262-1, 2, 3, 4, 5, 6, 7, 8, 9: 2011(E), ISO 26262-10: 2012(E),
- ISO 13849-1:2006, ISO 13849-2:2010,
- IEC 62061:2012-11, ed. 1.1,
- IEC 61800-5-2:2007, ed.1.0,
- IEC 60730-1:2010, ed. 4.0.

[Table 2](#) reports the mapping of this document content with respect to the requirements listed in the IEC 61508-2 Annex D.

**Table 2. Mapping between this document content and IEC 61508-2 Annex D requirements**

IEC 61508 requirement (part 2 annex D)	Reference
D2.1 a) a functional specification of the functions capable of being performed	<a href="#">Section 3</a>
D2.1 b) identification of the hardware and/or software configuration of the compliant item	<a href="#">Section 3.2</a>
D2.1 c) constraints on the use of the compliant item or assumptions on which analysis of the behavior or failure rates of the item are based	<a href="#">Section 3.2</a>
D2.2 a) the failure modes of the compliant item due to random hardware failures, that result in a failure of the function and that are not detected by diagnostics internal to the compliant item;	<a href="#">Section 3.7</a>
D2.2 b) for every failure mode in a), an estimated failure rate;	
D2.2 c) the failure modes of the compliant item due to random hardware failures, that result in a failure of the function and that are detected by diagnostics internal to the compliant item;	
D2.2 d) the failure modes of the diagnostics, internal to the compliant item due to random hardware failures, that result in a failure of the diagnostics to detect failures of the function;	
D2.2 e) for every failure mode in c) and d), the estimated failure rate;	
D2.2 f) for every failure mode in c) that is detected by diagnostics internal to the compliant item, the diagnostic test interval;	<a href="#">Section 3.2.2</a>
D2.2 g) for every failure mode in c) the outputs of the compliant item initiated by the internal diagnostics;	<a href="#">Section 3.6</a>
D2.2 h) any periodic proof test and/or maintenance requirements;	<a href="#">Section 3.7</a>
D2.2 i) for those failure modes, in respect of a specified function, that are capable of being detected by external diagnostics, sufficient information must be provided to facilitate the development of an external diagnostics capability.	
D2.2 j) the hardware fault tolerance;	<a href="#">Section 3</a>
D2.2 k) the classification as type A or type B of that part of the compliant item that provides the function (see 7.4.4.1.2 and 7.4.4.1.3);	

The safe failure fraction reported in this manual has been computed under the assumptions described in this document and especially according to the conditions of use described in [Section 3.7: Conditions of use](#).

## 2 STM32F0 Series microcontroller development process

The development process of a microelectronic device that is used in safety critical application takes into account the adequate management to reduce the probability of systematic faults introduced during the design phase.

IEC 61508:2 in Annex F (Techniques and measures for ASICs - avoidance of systematic failures) act as a guidance in tailoring the microcontroller standard design and manufacturer process to the compliance of the IEC 61508 requirements. The checklist reported in the named Annex F helps to collect all related evidences of a given real process.

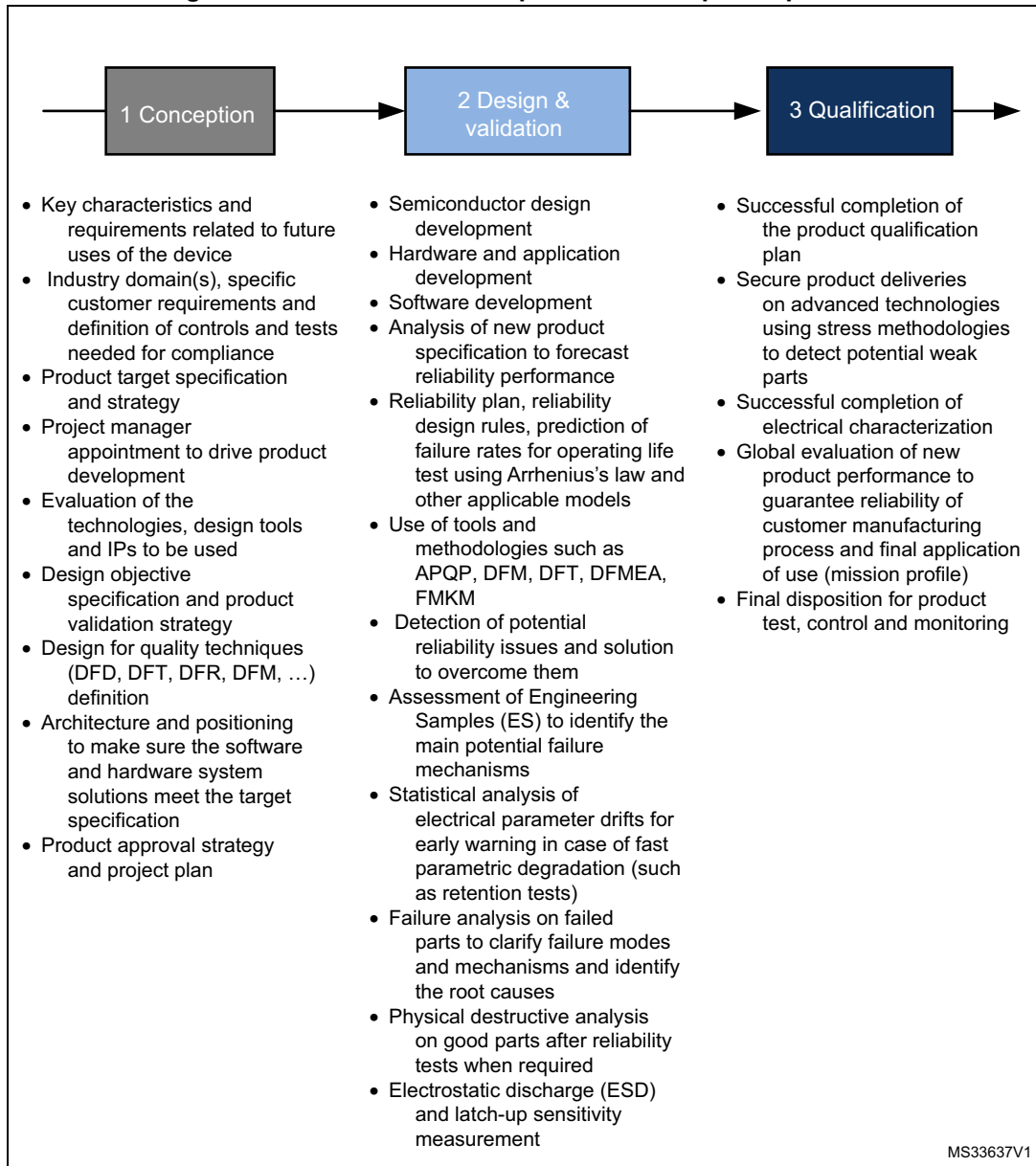
### 2.1 STMicroelectronics standard development process

STMicroelectronics (ST) serves four industry domains:

- Standard products,
- Automotive products: ST automotive products are AEC-Q100 compliant. They are subject to specific stress testing and processing instructions in order to achieve the required quality levels and product stability.
- Automotive safety: a subset of the automotive domain. ST uses as a reference the ISO 26262 Road vehicles Functional safety standard. ST supports customer inquiries regarding product failure rates and FMEDA to support hardware system compliance to established safety goals. ST provides products that are safe in their intended use, working in cooperation with customers to understand the mission profile, adopt common methods and define countermeasures for residual risks.
- Medical products: ST complies with applicable regulations for medical products and applies due diligence in the development and validation of these products.

STMicroelectronics product development process, compliant with the ISO/TS 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, application, hardware, software and documentation), qualified respecting ST internal procedures and able to be manufactured using ST internal or subcontracted technologies.

Figure 1. STMicroelectronics product development process



## 3 Reference safety architecture

### 3.1 Introduction

The STM32F0 Series microcontroller analyzed in this document can be used as a compliant item within different safety applications.

The aim of this section is to identify such compliant item and therefore to define the context of the analysis in terms of assumptions with respect to a reference concept definition. This concept definition will include therefore reference safety requirements as also assumptions on the design external to the defined compliant item.

As a consequence of compliant item approach, the goal is not to provide an exhaustive hazard and risk analysis of the system around the microcontroller, but rather to list the system-related information considered during the analysis. Such information include - among others - application related assumptions for dangerousness factors, frequency of failures and diagnostic coverage already guaranteed by the application.

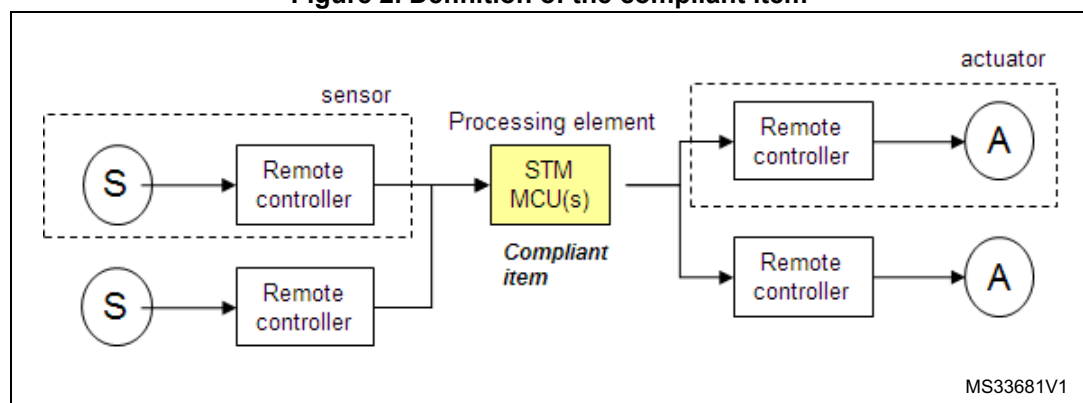
### 3.2 Compliant item

#### 3.2.1 Definition of the compliant item

According to IEC 61508:1 clause 8.2.12, a compliant item is any item (for example an element) on which a claim is being made with respect to the clauses of IEC 61508 series. With respect to its user, at the end of its development the compliant item must be described by a safety manual.

In this document, the compliant item is defined as a system including one or two STM32 microcontrollers (MCU) (see [Figure 2](#)). The communication bus is directly or indirectly connected to sensors and actuators.

**Figure 2. Definition of the compliant item**



Other components might be related to the compliant item, like the external HW components needed to guarantee either the functionality of the STM32F0 (external memory, clock quartz etc) or its safety (for example the external watchdog, voltage supervisors).

Defined compliant item can be classified as “element” according IEC61508-4, 3.4.5.

### 3.2.2 Safety functions performed by the compliant item

In essence, the compliant item architecture can be represented as composed by the following processes performing the safety function or part of it:

- Input processing elements (PEi) reading safety related data from the remote controller connected to the sensor(s) and transferring them to the following computation elements;
- Computation processing elements (PEc) performing the algorithm required by the safety function and transferring the results to the following output elements;
- Output processing elements (PEo) transferring safety related data to the remote controller connected to the actuator;
- in the case of the 1oo2 architecture, a further voting processing element (PEv) can be present;
- processes external to the compliant item are considered to guarantee safety integrity, such as a watchdog (WDTe) and voltage monitors (VMONE).

The role of the PEv and of the external processes WDTe and VMONE is clarified in the sections where the CoU (definition of safety mechanism) are detailed:

- WDTe: refer to Independent watchdog – VSUP\_SM\_2, Control flow monitoring in application software – CPU\_SM\_1,
- VMONE: refer to Supply Voltage Monitoring – VSUP\_SM\_1.

In summary, STM32F0 microcontrollers support the implementation of end user safety functions composed by 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 compliant item and computation of safety metrics are done with respect to these three basic operations.

According to above reported definition for implemented safety functions, the compliant item i.e. the element can be regarded as type B (as per IEC61508-2, 7.4.4.1.2 definition). Despite accurate, exhaustive and detailed failure analysis has been done for STM32F0, this device has to be considered intrinsically complex and therefore type B classification is appropriate.

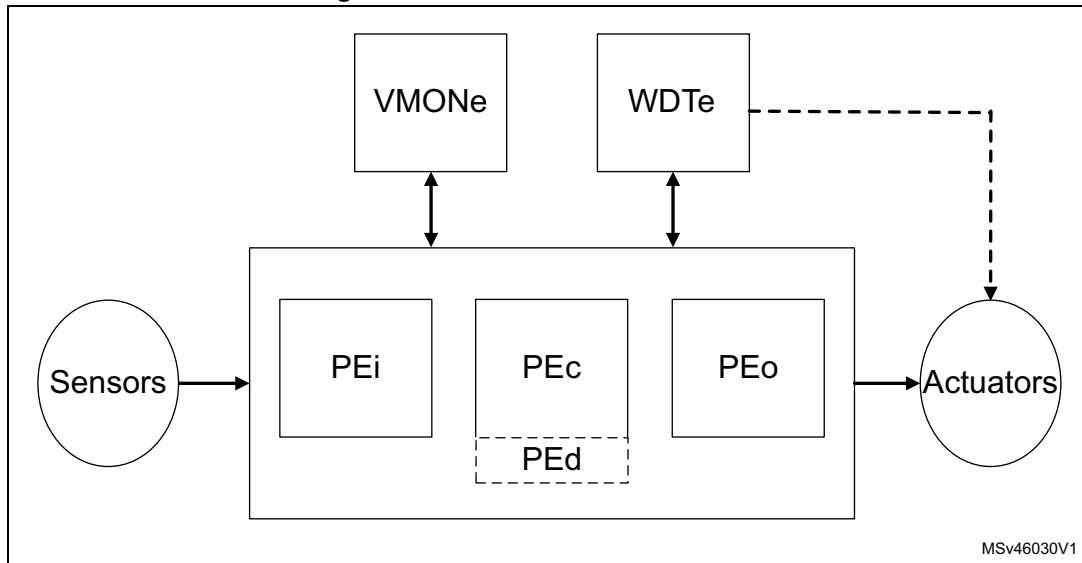
Two main safety architecture are therefore identified: 1oo1 (using one MCU) and 1oo2 (using two MCUs).

### 3.2.3 Reference Safety Architectures - 1oo1

In 1oo1 reference architecture (shown in below [Figure 3](#)) the safety integrity of the compliant item is guaranteed by the combination of STM32F0 internal processes (implemented safety mechanisms) and external processes WDTe and VMONE.

Target for 1oo1 reference architecture is SIL2.

Figure 3. 1oo1 reference architecture



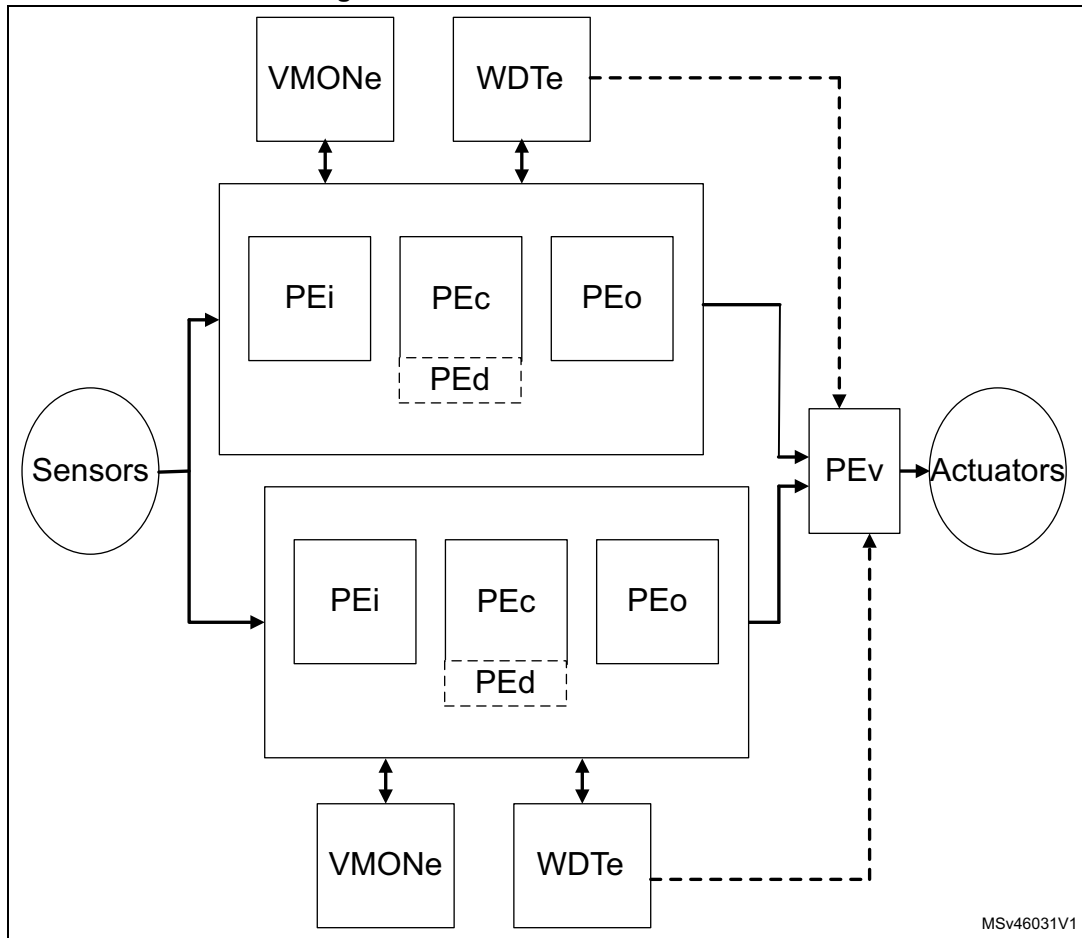
**3.2.4 Reference Safety Architectures - 1oo2**

1oo2 reference architecture (shown in below [Figure 4](#)) is composed by two separate channels, each of them implemented in the same way of 1oo1 reference architecture. Safety integrity of each channel is guaranteed by the combination of STM32F0 internal processes (implemented safety mechanisms) and external processes WDTe and VMONe. Safety integrity of overall compliant item is guaranteed by the external voter PEv allowing to claim HFT=1. Achievement of higher safety integrity levels as per IEC61508-2 Table 3 is therefore possible. Appropriate separation between the two channels (including power supply separation) should be implemented in order to avoid huge impact of common-cause failures (refer to [Section 4.2](#)).  $\beta$ D computation is anyway required.

Target for 1oo2 reference architecture is SIL3.



Figure 4. 1oo2 reference architecture



### 3.3 Assumed requirements

This section collects all assumptions done during the safety analysis of STM32F0 Series microcontrollers

#### 3.3.1 Assumed safety requirements

The concept specification, the hazard and risk analysis, the overall safety requirement specification and the consequent allocation has determined the requirements for the compliant item (ASR, Assumed Safety Requirements) listed here below.

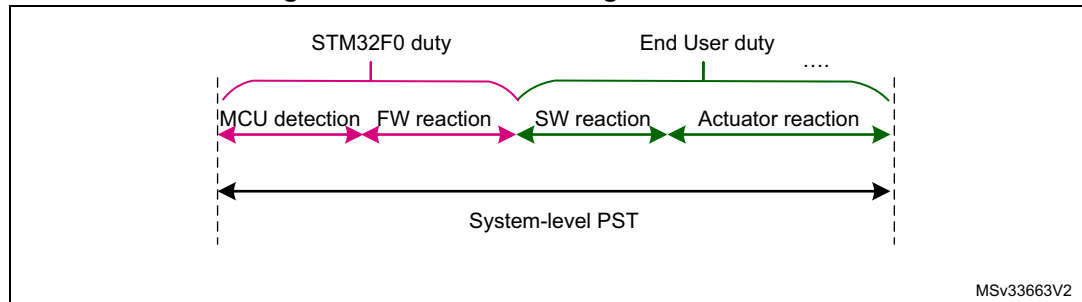
**Caution:** It is end user’s responsibility to check the compliance of the final application with these assumptions.

**ASR1:** The compliant item can be used for four kinds of safety functions mode of operations according part 4, 3.5.16:

- A continuous mode or high-demand SIL3 safety function (CM3), or
- A low-demand SIL3 safety function (LD3), or
- A continuous mode or high-demand SIL2 safety function (CM2), or
- A low-demand SIL2 safety function (LD2).

**ASR2:** The compliant item is used to implement a safety function allowing a time budget of 10 ms (worst case) for the STM32 MCU to detect and react to a failure. That time corresponds to the portion of the Process Safety Time allocated to STM32F0 MCU (“STM32F0 duty” in *Figure 5*) in error reaction chain at system level.

**Figure 5. Allocation and target for STM32 PST**



**ASR3:** The compliant item is used in a safety function that can be continuously powered-on for a time higher than 8 hours. It is assumed to not require any proof test and the lifetime of the product is considered to be not less than 10 years.

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

**ASR5:** In case of multiple safety functions implementations, it is assumed that end user is responsible to guarantee their needed mutual independence.

**ASR6:** It is assumed that there are no “non-safety related” functions implemented in application software and coexisting with the safety functions.

**ASR7:** It is assumed that the implemented safety function(s) is not depending on STM32F0 transition to and from a low-power state.

**ASR8:** The local safe state of the compliant item 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<sup>(a)</sup>

Details on safe states SS1 and SS2 are provided in following table:

a. The end user must take into account that random hardware failures affecting the STM32 can compromise the MCU capability of operating properly (for example failure modes affecting the program counter prevent the correct execution of software).

**Table 3. SS1 and SS2 safe state details**

Safe state	Condition	Compliant item action	System Transition to Safe state – 1oo1 architecture	System Transition to Safe state – 1oo2 architecture
SS1	The application software 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 will drive the overall system in his safe state	Application software in one of the two channels will drive the overall system in his 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 will drive the overall system in his safe state (“safe shut-down”) <sup>(1)</sup>	PEv will drive the overall system in his safe state

1. safe state achievement intended here is compliant to Note on IEC61508-2, 7.4.8.1 a)

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

**ASR10:** The compliant item is assumed to be analyzed according to routes 1H and 1S of IEC 61508-2.<sup>(a)</sup>

**ASR11:** The compliant item is assumed to be regarded as type B as per IEC61508:2, 7.4.4.1.2

### 3.4 Electrical specifications and environment limits

The user must not exceed the electrical specification and the environmental limits defined in the below list as reported in the STM32F0 user manual in order to guarantee the STM32F0 safety integrity:

- Absolute maximum rating,
- Capacity,
- Operating conditions.

Due to the large number of STM32F0 part numbers, the related user manuals and datasheets are not listed in this document; users are responsible to carefully check the above reported limits in the technical documentation on the related part number available on [www.st.com](http://www.st.com).

a. Please refer to [Section 3.5: Systematic safety integrity](#) and [Section 3.6: Description of hardware and software diagnostics](#).

### 3.5 Systematic safety integrity

According to the requirements of IEC 61508 -2, 7.4.2.2, the Route 1S has been considered in the STM32F0 development. As clearly authorized by IEC61508-2, 7.4.6.1, STM32 MCU series can be considered a standard, mass-produced electronic integrated device – for which stringent development procedures, rigorous testing and extensive experience of use minimizes the likelihood of design faults. Anyway, an internal assessment against the compliance of STM32 MCU development flow with the techniques and measures suggested in IEC 61508-2 Annex F has been executed. The Safety Case Database ([Section 5: List of evidences](#)) maintains the evidences of the compliance to the norm.

### 3.6 Description of hardware and software diagnostics

This section lists all the safety mechanisms (hardware, software and application level) considered in the safety analysis of the microcontrollers of the STM32F0 series. It is expected that users are familiar with the STM32F0 architecture, and that this document is used in conjunction with the related device datasheet, user manual and reference information. Therefore, to avoid the eventuality of mistakes and reduce the amount of information to be shown, minimum functional details are included in this document. In following descriptions the words “safety mechanism”, “method” or “requirement” are used as synonym.

Note that each part number of the STM32F0 series owns different combinations of peripherals (for instance, some of them are not equipped with USB peripheral). To reduce the number of documents and avoid information-less repetitions, the current Safety Manual (and therefore this section) addresses the overall possible peripherals available in the targeted part numbers. Users have to select which peripherals are really available on their devices, and discard the meaningless recommendations accordingly.

The implementation guidelines reported in the following section are for reference only. The safety verification executed by ST during STM32F0 safety analysis and related diagnostic coverage figures reported in this manual (or its Annexes) are based on such guidelines. For the sake of clarity, safety mechanism are reported grouped for MCU basic functions.

Information are organized in form of tables (one for each safety mechanism). Here below is reported the explanation of each field:

**Table 4. Safety mechanism field explanation**

SM CODE	Unique safety mechanism code or identifier used also in FMEA document.
Description	Short mnemonic description
Ownership	ST : means that method is available on silicon End user: method must be implemented by the end user by 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

**Table 4. Safety mechanism field explanation (continued)**

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 MCU configuration	Reports if safety mechanism implementation or characteristics will change among different part numbers belonging to STM32F0 series.
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. Note that safety mechanism can be accounted for diagnostic coverage contribution only if it is executed at least one per PST On Demand: safety mechanism is activated in correspondence of 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
Multiple faults protection	Reports the safety mechanism(s) associated in order to correct manage a multi-fault scenario (refer to <a href="#">Section 3.6.28</a> )
Recommendations and known limitations	Additional recommendations or limitations (if any) not reported in other fields

### 3.6.1 Cortex®-M0 CPU

**Table 5. CPU\_SM\_0**

SM CODE	CPU_SM_0
Description	Periodical core self-test software for Arm Cortex CPU
Ownership	End user or ST
Detailed implementation	The software test is built around well-known techniques already addressed by IEC 61508:7, A.3.2 (Self-test by software: walking bit one-channel). To reach the required values of coverage, the self-test software is specified by means of a detailed analysis of all the CPU failure modes and related failure modes distribution.
Error reporting	Depending on implementation
Fault detection Time	Depending on implementation
Addressed Fault Model	Permanent
Dependency on MCU 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.

**Table 5. CPU\_SM\_0 (continued)**

Multiple faults protection	CPU_SM_5 : external watchdog
Recommendations and known limitations	This method is the main asset in STM32F0 safety concept. CPU integrity is a key factor, given that the major part of defined diagnostics for MCU peripherals are software-based.

**Table 6. CPU\_SM\_1**

SM CODE	CPU_SM_1
Description	Control flow monitoring in application software
Ownership	End user
Detailed implementation	<p>A significant part of the failure distribution of 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) will be detected.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> <li>– The different internal states of the application software is well documented and described (the use of a dynamic state transition graph is encouraged).</li> <li>– The monitoring of the correctness of each transition between different states of the application software is implemented.</li> <li>– The transition through all expected states during the normal application software program loop is checked.</li> <li>– The 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.</li> <li>– The use of the window feature of the independent watchdog (IWDG) (or an external one) helps to implement a more robust control flow mechanism fed by a different clock source.</li> </ul> <p><i>Note: 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 <a href="#">Section 4.2.2: Clock</a>)</i></p>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	

Table 7. CPU\_SM\_2

SM CODE	CPU_SM_2
Description	Double computation in application software
Ownership	End user
Detailed implementation	<p>A timing redundancy for safety-related computation is considered to detect transient faults affecting the Arm Cortex-M0 CPU subparts devoted to mathematical computations and data access.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> <li>– 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</li> <li>– Both mathematical operation and comparison are intended as computation.</li> <li>– 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.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	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	End user is responsible to carefully avoid that the intervention of optimization features of the used compiler will remove timing redundancies introduced according to this condition of use

Table 8. CPU\_SM\_3

SM CODE	CPU_SM_3
Description	Arm Cortex-M0 HardFault exceptions
Ownership	ST
Detailed implementation	<p>HardFault exception raise is an intrinsic safety mechanism implemented in Arm Cortex-M0 core, mainly devoted to intercept systematic faults due to software limitations or error in software design (causing for example execution of undefined operations, unaligned address access). This safety mechanism is also able to detect hardware random faults inside the CPU bringing to such described abnormal operations.</p>
Error reporting	High-priority Interrupt Event
Fault detection Time	Depending on implementation, refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	None

Table 8. CPU\_SM\_3 (continued)

Periodicity	Continuous
Test for the diagnostic	It is possible to write a test procedure to verify the generation of the HardFault exception; anyway, given the expected minor contribution in terms of hardware random failure detection, such implementation is not recommended.
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 9. CPU\_SM\_4

SM CODE	CPU_SM_4
Description	Stack hardening for application software
Ownership	End user
Detailed implementation	<p>The stack hardening method is required to address faults (mainly transient) affecting CPU register bank. This method is based on source code modification, introducing information redundancy in register-passed information to called functions.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> <li>– To pass also a redundant copy of the passed parameters values (possibly inverted) and to execute a coherence check in the function.</li> <li>– To pass also a redundant copy of the passed pointers and to execute a coherence check in the function.</li> <li>– 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.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	This method partially overlaps with defensive programming techniques required by IEC61508 for software development. Therefore in presence of application software qualified for safety integrity greater or equal to SC2, optimizations are possible.

Table 10. CPU\_SM\_5

SM CODE	CPU_SM_5
Description	External watchdog
Ownership	End user



**Table 10. CPU\_SM\_5 (continued)**

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 <a href="#">Section 3.3.1</a> . It also contributes to dramatically reduce potential common cause failures, because the external watchdog will be clocked and supplied independently from the STM32F0.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation (watchdog timeout interval)
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	To be defined at system level (outside the scope of compliant item analysis)
Multiple faults protection	CPU_SM_1: control flow monitoring in application software
Recommendations and known limitations	In case of usage of windowed watchdog, end user must consider possible tolerance in application software execution, to avoid false error reports (affecting system availability).

**Table 11. CPU\_SM\_6**

SM CODE	CPU_SM_6
Description	Independent watchdog
Ownership	ST
Detailed implementation	Using the IDWG watchdog linked to control flow monitoring method (refer to CPU_SM_1) addresses failure mode of program counter or control structures of CPU.
Error reporting	Reset signal generation
Fault detection Time	Depends on implementation (watchdog timeout interval)
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	IWDG activation. It is recommended to use the "Hardware watchdog" in Option byte settings (IWDG will be automatically enabled after reset)
Periodicity	Continuous
Test for the diagnostic	WDG_SM_1: Software test for watchdog at startup

**Table 11. CPU\_SM\_6 (continued)**

Multiple 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. If this limitation turn out in a blocking point, end user must adopt CPU_SM_5

### 3.6.2 System FLASH memory

**Table 12. FLASH\_SM\_0**

SM CODE	FLASH_SM_0
Description	Periodical software test for Flash memory
Ownership	End user or ST
Detailed implementation	Permanent faults affecting the system Flash memory, memory cells and address decoder, are addressed through a dedicated software test that checks the memory cell contents versus the expected value, using signature-based techniques. According to IEC 61508:2 Table A.5, the effective diagnostic coverage of such techniques depends on the width of the signature in relation to the block length of the information to be protected - therefore the signature computation method is to be carefully selected. Note that the simple signature method (IEC 61508:7 - A.4.2 Modified checksum) is inadequate as it only achieves a low value of coverage. The information block does not need to be addressed with this test as it is not used during normal operation (no data nor program fetch).
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU configuration	Flash size changes according part number
Initialization	Memory signatures must be stored in Flash as well.
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen.
Multiple faults protection	CPU_SM_1: control flow monitoring in application software 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. The use of internal CRC module is recommended. In principle DMA feature for data transfer can be used. Note: unused Flash sections can be excluded from testing.

**Table 13. 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 & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	CPU_SM_1 correct implementation supersedes this requirement.

**Table 14. FLASH\_SM\_2**

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

Table 15. FLASH\_SM\_3

SM CODE	FLASH_SM_3
Description	Option byte write protection
Ownership	ST
Detailed implementation	This safety mechanism prevents unintended writes on the option byte. The use of this method is encouraged to enhance end application robustness for systematic faults.
Error reporting	Write protection exception
Fault detection Time	Not applicable
Addressed Fault Model	None (Systematic only)
Dependency on MCU configuration	None
Initialization	Not needed (enabled by default)
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	This method addresses systematic faults in software application and it have zero efficiency in addressing hardware random faults affecting the option byte value during running time. No DC value is therefore associated.

Table 16. FLASH\_SM\_4

SM CODE	FLASH_SM_4
Description	Static data encapsulation
Ownership	End user
Detailed implementation	If static data are stored in Flash memory, encapsulation by a checksum field with encoding capability (like CRC) must be implemented. Checksum validity is checked by application software before static data consuming.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on 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 17. FLASH\_SM\_5

SM CODE	FLASH_SM_5
Description	Option byte redundancy with load verification
Ownership	ST
Detailed implementation	During option byte loading after each power-on reset, the bit-wise complementarity of the option byte and its corresponding complemented option byte is verified. Mismatches are reported as error.
Error reporting	Option byte error (OPTERR) generation
Fault detection Time	Not applicable
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	None (always enabled)
Periodicity	Startup
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 18. FLASH\_SM\_6

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

### 3.6.3 System SRAM memory

**Table 19. RAM\_SM\_0**

SM CODE	RAM_SM_0
Description	Periodical software test for SRAM memory
Ownership	End user or ST
Detailed implementation	To enhance the coverage on SRAM data cells and to ensure adequate coverage for permanent faults affecting the address decoder it is required to execute a periodical software test on the system RAM memory. The selection of the algorithm must ensure the target SFF coverage for both the RAM cells and the address decoder. Evidences of the effectiveness of the coverage of the selected method must be also collected.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU configuration	RAM size can change according to the part number
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen.
Multiple 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.

**Table 20. RAM\_SM\_1**

SM CODE	RAM_SM_1
Description	Parity bit check
Ownership	ST
Detailed implementation	Internal SRAM is protected by additional parity bits (1 bit per byte). The parity bits are computed and stored when writing into the SRAM.
Error reporting	NMI raise
Fault detection Time	Parity bits are checked during a reading.
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	The end user must enable the parity check using the option bit RAM_PARITY_CHECK in the user option byte, after the boot.
Periodicity	Continuous

**Table 20. RAM\_SM\_1 (continued)**

Test for the diagnostic	NA
Multiple faults protection	RAM_SM_2: Information redundancy for safety-related variables in application software RAM_SM_0: periodical software test for SRAM memory
Recommendations and known limitations	It is advised to initialize by software the whole RAM memory at MCU boot, to avoid getting parity errors when reading non-initialized locations.

**Table 21. RAM\_SM\_2**

SM CODE	RAM_SM_2
Description	Stack hardening for application software
Ownership	End user
Detailed implementation	The stack hardening method is used to enhance the application software robustness to SRAM faults that affect the address decoder. The method is based on source code modification, introducing information redundancy in the stack-passed information to the called functions. Method contribution is relevant in case the combination between the final application software structure and the compiler settings requires a significant use of the stack for passing function parameters. Implementation is the same as method 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 MCU configuration	Refer to CPU_SM_4
Initialization	Refer to CPU_SM_4
Periodicity	Refer to CPU_SM_4
Test for the diagnostic	Refer to CPU_SM_4
Multiple faults protection	Refer to CPU_SM_4
Recommendations and known limitations	Refer to CPU_SM_4

**Table 22. RAM\_SM\_3**

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

**Table 22. RAM\_SM\_3 (continued)**

Detailed implementation	<p>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:</p> <ul style="list-style-type: none"> <li>– 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.</li> <li>– The arithmetic computation or decision based on such variables are executed twice and the two final results are compared.</li> <li>– Safety-related variables are stored and updated in two redundant locations, and comparison is checked before consuming data.</li> <li>– Enumerated fields must use non-trivial values, checked for coherence at least one time per PST</li> <li>– Data vectors stored in SRAM must be protected by a encoding checksum (like CRC)</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	Implementation of this safety method shows a partial overlap with an already foreseen method for Cortex®-M0 (CPU_SM_1); optimizations in implementing both methods are therefore possible.

**Table 23. RAM\_SM\_4**

SM CODE	RAM_SM_4
Description	Control flow monitoring in application software
Ownership	End user
Detailed implementation	<p>In case the end user application software will be executed from SRAM, permanent and transient faults affecting the memory (cells and address decoder) can interfere with the program execution.</p> <p>To address such failures it is needed to implement this method.</p> <p>For more details on the implementation, refer to description CPU_SM_1</p>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation. Higher value is fixed by watchdog timeout interval.
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous



**Table 23. RAM\_SM\_4 (continued)**

Test for the diagnostic	NA
Multiple 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 24. RAM\_SM\_5**

SM CODE	RAM_SM_5
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 PST). For implementation details refer to similar method FLASH_SM_0.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Self-diagnostic capabilities can be embedded in the software, according the test implementation design strategy chosen.
Multiple 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.

### 3.6.4 System bus interconnect

**Table 25. BUS\_SM\_0**

SM CODE	BUS_SM_0
Description	Periodical software test for interconnections
Ownership	End user
Detailed implementation	The intra-chip connection resources (Bus Matrix, AHB or APB bridges) needs to be periodically tested for permanent faults detection. Note that STM32F0 series MCUs have no hardware safety mechanism to protect these structures. The test executes a connectivity test of these shared resources, including the testing of the arbitration mechanisms between peripherals. According to IEC 61508:2 Table A.8, A.7.4 the method is considered able to achieve high levels of coverage.
Error reporting	Depends on implementation

**Table 25. BUS\_SM\_0 (continued)**

Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU 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	Implementation can be considered in large part overlapping with the widely used "Periodical read-back of configuration registers" required for several peripherals.

**Table 26. 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 (e.g. a CRC checksum at packet level) to each data message exchanged inside the MCU. Message integrity is verified using the checksum by the application software, before consuming data.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on 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	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 27. LOCK\_SM\_0**

SM CODE	LOCK_SM_0
Description	Lock mechanism for configuration options
Ownership	ST

Table 27. LOCK\_SM\_0 (continued)

Detailed implementation	The STM32F0 series MCUs feature spread protection to prevent unintended configuration changes for some peripherals and system registers (for example PVD_LOCK, timers); the spread protection detects systematic faults in software application. The use of this method is encouraged to enhance the end application robustness to systematic faults.
Error reporting	Not generated (when locked, register overwrites are just ignored)
Fault detection Time	NA
Addressed Fault Model	None (Systematic only)
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	Not needed
Recommendations and known limitations	No DC associated because this test addresses systematic faults.

### 3.6.5 NVIC and EXTI controller

Table 28. NVIC\_SM\_0

SM CODE	NVIC_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	<p>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 will be executed at least one time within PST after a peripheral update.</p> <p>Method must be implemented to any configuration register whose contents are able to interfere with NVIC or EXTI behavior in case of incorrect settings. Check includes NVIC vector table.</p> <p>According the state of the art automotive safety standard ISO26262, this method can achieve high levels of Diagnostic Coverage (refer to ISO26262:5, Table D.4)</p> <p>An alternative valid implementation requiring less space in SRAM can be realized on the basis of signature concept:</p> <ul style="list-style-type: none"> <li>– Peripheral registers to be checked are read in a row, computing a CRC checksum (use of hardware CRC is encouraged)</li> <li>– Obtained signature is compared with the golden value (computed in the same way after each register update, and stored in SRAM)</li> <li>– Coherence between signatures is checked by the application software – signature mismatch is considered as failure detection.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation

Table 28. NVIC\_SM\_0 (continued)

Addressed Fault Model	Permanent & 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 29. NVIC\_SM\_1

SM CODE	NVIC_SM_1
Description	Expected and unexpected interrupt check
Ownership	End user
Detailed implementation	<p>According to IEC 61508:2 Table A.1 recommendations, a diagnostic measure for continuous, absence or cross-over of interrupt must be implemented. The method of expected and unexpected interrupt check is implemented at application software level. The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> <li>– 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).</li> <li>– 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.</li> <li>– Interrupt vectors related to unused interrupt source point to a default handler that will report, in case of triggering, a faulty condition (unexpected interrupt).</li> <li>– In case an interrupt service routine is shared between different sources, a plausibility check on the caller identity is implemented.</li> <li>– Interrupt requests related to non-safety-related peripherals are handled with the same method here described, despite their originator safety classification.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed

**Table 29. NVIC\_SM\_1 (continued)**

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.

**3.6.6 DMA**

**Table 30. DMA\_SM\_0**

SM CODE	DMA_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to DMA configuration register and channel addresses register as well. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 31. DMA\_SM\_1**

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

**Table 31. DMA\_SM\_1 (continued)**

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	To give an example about checksum encoding capability, using just a bit-by-bit addition will be unappropriated.

**Table 32. DMA\_SM\_2**

SM CODE	DMA_SM_2
Description	Information redundancy including sender or receiver identifier on data packet transferred via DMA
Ownership	End user
Detailed implementation	<p>This method helps to identify inside the MCU the source and the originator of the message exchanged by DMA.</p> <p>Implementation is realized by adding an additional field to protected message, with a coding convention for message type identification fixed at MCU level. Guidelines for the identification fields are:</p> <ul style="list-style-type: none"> <li>– Identification field value must be different for each possible couple of sender or receiver on DMA transactions</li> <li>– Values chosen must be enumerated and non-trivial</li> <li>– Coherence between the identification field value and the message type is checked by application software before consuming data.</li> </ul> <p>This method, when implemented in combination with DMA_SM_4, makes available a kind of “virtual channel” between source and destinations entities.</p>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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 33. DMA\_SM\_3**

SM CODE	DMA_SM_3
Description	Periodical software test for DMA

Table 33. DMA\_SM\_3 (continued)

Ownership	End user
Detailed implementation	<p>This method requires the periodical testing of the DMA basic functionality, implemented through a deterministic transfer of a data packet from one source to another (for example from memory to memory) and the checking of the correct transfer of the message on the target. Data packets are composed by non-trivial patterns (avoid the use of 0x0000, 0xFFFF values) and organized in order to allow the detection during the check of the following failures:</p> <ul style="list-style-type: none"> <li>– Incomplete packed transfer</li> <li>– Errors in single transferred word</li> <li>– Wrong order in packed transmitted data</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	None

Table 34. DMA\_SM\_4

SM CODE	DMA_SM_4
Description	DMA transaction awareness
Ownership	End user
Detailed implementation	<p>DMA transactions are non-deterministic by nature, because typically driven by external events like communication messages reception. Anyway, well-designed safety systems should keep much control as possible of events – refer for instance to IEC61508:3 Table 2 item 13 requirements for software architecture.</p> <p>This method is based on system knowledge of frequency and type of expected DMA transaction. For instance, an externally connected sensor supposed to send periodically some messages to a STM32 peripheral. Monitoring DMA transaction by a dedicated state machine will allow to detect missing or unexpected DMA activities.</p>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed

**Table 34. DMA\_SM\_4 (continued)**

Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	Because DMA transaction termination is often linked to an interrupt generation, implementation of this method can be merged with the safety mechanism NVIC_SM_1: expected and unexpected interrupt check

**3.6.7 CAN**

**Table 35. 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 <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 36. 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 (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None



**Table 36. CAN\_SM\_1 (continued)**

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 safing
Recommendations and known limitations	None

**Table 37. CAN\_SM\_2**

SM CODE	CAN_SM_2
Description	Information redundancy techniques on messages, including end-to-end safing
Ownership	End user
Detailed implementation	<p>This method aims to protect the communication between a peripheral and his external counterpart establishing a kind of “protected” channel. The aim is to specifically address communication failure modes as reported in IEC61508:2, 7.4.11.1.</p> <p>Implementation guidelines are the following:</p> <ul style="list-style-type: none"> <li>– 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.</li> <li>– Additional field added in payload reporting an unique identification of sender or receiver and an unique increasing sequence packet number</li> <li>– Timing monitoring of the message exchange (for example check the message arrival within the expected time window), detecting therefore missed message arrival conditions</li> <li>– 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)</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	<p>Important note: it is assumed that the remote CAN counterpart will have an equivalent capability of performing the checks described.</p> <p>A major overlap between the requirements of this method and the implementation of complex communication software protocols can exists. Due to large adoption of these protocols in industrial applications, optimizations can be possible.</p>

## 3.6.8 USART 1/2/3/4/5/6/7/8

Table 38. 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 <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 39. UART\_SM\_1

SM CODE	UART_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	USART communication module embeds protocol error checks (like additional parity bit check, overrun, frame error) conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a marginal percentage of hardware random failures affecting the module itself. Error signals connected to these checkers are normally handled in a standard communication software, so the overhead is reduced.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection Time	Depends on peripheral configuration (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not required

**Table 39. UART\_SM\_1 (continued)**

Multiple faults protection	UART_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	USART communication module is fitted by several different configurations – the actual composition of communication error checks depends on selected configuration.

**Table 40. 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 (like a CRC check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	It is assumed that the remote UART counterpart will have 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 will be unappropriated.

**Table 41. UART\_SM\_3**

SM CODE	UART_SM_3
Description	Information redundancy techniques on messages, including end-to-end safing
Ownership	End user
Detailed implementation	This method aims to protect the communication between a peripheral and his external counterpart. Refer to CAN_SM_2 description for detailed information.
Error reporting	Refer to CAN_SM_2
Fault detection Time	Refer to CAN_SM_2
Addressed Fault Model	Refer to CAN_SM_2

Table 41. UART\_SM\_3 (continued)

Dependency on MCU configuration	Refer to CAN_SM_2
Initialization	Refer to CAN_SM_2
Periodicity	Refer to CAN_SM_2
Test for the diagnostic	Refer to CAN_SM_2
Multiple faults protection	Refer to CAN_SM_2
Recommendations and known limitations	Important note: it is assumed that the remote UART counterpart will have an equivalent capability of performing the checks described. Refer to CAN_SM_2 for further notice.

### 3.6.9 I2C 1/2

Table 42. 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 <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 43. 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

**Table 43. IIC\_SM\_1 (continued)**

Fault detection Time	Depends on peripheral configuration (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	Adoption of SMBus option grants the activation of more efficient protocol-level hardware checks like CRC-8 packet protection

**Table 44. 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 (like a CRC check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	It is assumed that the remote I2C counterpart will have 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 will be unappropriated. This method is superseded by IIC_SM_3 if hardware handled CRC insertion is possible

**Table 45. IIC\_SM\_3**

SM CODE	IIC_SM_3
Description	CRC packet-level

**Table 45. IIC\_SM\_3 (continued)**

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 (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	IIC_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

**Table 46. IIC\_SM\_4**

SM CODE	IIC_SM_4
Description	Information redundancy techniques on messages, including end-to-end safing
Ownership	End user
Detailed implementation	This method aims to protect the communication between a I2C peripheral and his external counterpart. Refer to CAN_SM_2 description for detailed information.
Error reporting	Refer to CAN_SM_2
Fault detection Time	Refer to CAN_SM_2
Addressed Fault Model	Refer to CAN_SM_2
Dependency on MCU configuration	Refer to CAN_SM_2
Initialization	Refer to CAN_SM_2
Periodicity	Refer to CAN_SM_2
Test for the diagnostic	Refer to CAN_SM_2
Multiple faults protection	Refer to CAN_SM_2
Recommendations and known limitations	Important note: it is assumed that the remote I2C counterpart will have an equivalent capability of performing the checks described. Refer to CAN_SM_2 for further notice.

## 3.6.10 SPI 1/2

Table 47. 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 <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 48. 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 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 (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

Table 49. 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 (like a CRC check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	It is assumed that the remote SPI counterpart will have 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 will be unappropriated. This method is superseded by SSP_SM_3 if hardware handled CRC insertion is possible

Table 50. 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 (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed



**Table 50. SPI\_SM\_3 (continued)**

Multiple faults protection	SPI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

**Table 51. SPI\_SM\_4**

SM CODE	SPI_SM_4
Description	Information redundancy techniques on messages, including end-to-end safing
Ownership	End user
Detailed implementation	This method aims to protect the communication between SPI peripheral and his external counterpart. Refer to CAN_SM_2 description for detailed information.
Error reporting	Refer to CAN_SM_2
Fault detection Time	Refer to CAN_SM_2
Addressed Fault Model	Refer to CAN_SM_2
Dependency on MCU configuration	Refer to CAN_SM_2
Initialization	Refer to CAN_SM_2
Periodicity	Refer to CAN_SM_2
Test for the diagnostic	Refer to CAN_SM_2
Multiple faults protection	Refer to CAN_SM_2
Recommendations and known limitations	Important note: it is assumed that the remote SPI counterpart will have an equivalent capability of performing the checks described. Refer to CAN_SM_2 for further notice.

**3.6.11 USB - 2.0 Universal Serial Bus interface FS module**

**Table 52. USB\_SM\_0**

SM CODE	USB_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to USB configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0

**Table 52. USB\_SM\_0 (continued)**

Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 53. USB\_SM\_1**

SM CODE	USB_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	USB communication module embeds protocol error checks (like overrun, underrun, NRZI, bit stuffing etc.) conceived to detect network-related abnormal conditions. These mechanisms are able anyway to detect a marginal percentage of hardware random failures affecting the module itself.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection Time	Depends on peripheral configuration (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	USB_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

**Table 54. USB\_SM\_2**

SM CODE	USB_SM_2
Description	Information redundancy techniques on messages
Ownership	ST or end user
Detailed implementation	The implementation of required information redundancy on messages, USB communication module is fitted by hardware capability. It basically allows to activate the automatic insertion (and check) of CRC checksums to packet data.
Error reporting	Error flag raise and optional Interrupt Event generation
Fault detection Time	Depends on peripheral configuration (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Error reporting configuration, if interrupt events are planned

**Table 54. USB\_SM\_2 (continued)**

Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	USB_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

**Table 55. USB\_SM\_3**

SM CODE	USB_SM_3
Description	Information redundancy techniques on messages, including end-to-end safing
Ownership	End user
Detailed implementation	This method aims to protect the communication between an USB peripheral and his external counterpart. Refer to CAN_SM_2 description for detailed information.
Error reporting	Refer to CAN_SM_2
Fault detection Time	Refer to CAN_SM_2
Addressed Fault Model	Refer to CAN_SM_2
Dependency on MCU configuration	Refer to CAN_SM_2
Initialization	Refer to CAN_SM_2
Periodicity	Refer to CAN_SM_2
Test for the diagnostic	Refer to CAN_SM_2
Multiple faults protection	Refer to CAN_SM_2
Recommendations and known limitations	This method apply in case USB bulk or isochronous transfers are used. For other transfers modes the USB hardware protocol already implements several features of this requirement. Refer to CAN_SM_2 for further notice.

### 3.6.12 HDMI CEC module

**Table 56. HDMI\_SM\_0**

SM CODE	HDMI_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to HDMI CEC configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0

**Table 56. HDMI\_SM\_0 (continued)**

Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 57. HDMI\_SM\_1**

SM CODE	HDMI_SM_1
Description	Protocol error signals
Ownership	ST
Detailed implementation	HDMI communication module embeds protocol error checks (like overrun, underrun, bit timing violations, 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 (e.g. baud rate), refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	HDMI_SM_2: Information redundancy techniques on messages
Recommendations and known limitations	None

**Table 58. HDMI\_SM\_2**

SM CODE	HDMI_SM_2
Description	Information redundancy techniques on messages
Ownership	End user
Detailed implementation	This method is implemented adding to data packets transferred by HDMI or CEC a redundancy check (like a CRC check, or similar one) with encoding capability. The checksum encoding capability must be robust enough to guarantee at least 90% probability of detection for a single bit flip in the data packet. Consistency of data packet must be checked by the application software before consuming data.
Error reporting	Depends on implementation

**Table 58. HDMI\_SM\_2 (continued)**

Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	It is assumed that the remote HDMI or CEC counterpart will have 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 will be unappropriated.

**3.6.13 Touch Sensing Controller (TSC)**

**Table 59. TSC\_SM\_0**

SM CODE	TSC_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to TSC configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 60. TSC\_SM\_1**

SM CODE	TSC_SM_1
Description	Multiple acquisition by application software
Ownership	End user

**Table 60. TSC\_SM\_1 (continued)**

Detailed implementation	This method implements a timing information redundancy by executing multiple acquisitions on TSC input data. Multiple acquisition data are then used to determine the acquisition correct state. This method overlaps on the native features of the TSC module of counting events in order to ensure a stable acquisition against external noise.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Transient
Dependency on 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	None

**Table 61. TSC\_SM\_2**

SM CODE	TSC_SM_2
Description	Application-level detection of permanent failures of TSC acquisition
Ownership	End user
Detailed implementation	This method must detect TSC module permanent failure leading to wrong or missing acquisition of touch sensing events.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU 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	Due to the nature strictly application-dependent of this solution, no detailed guidelines for its implementation are given here. Being not possible a solution fully based on microcontroller resources, leveraging on the contribution from other components of the final system is necessary.

### 3.6.14 Analog to Digital Converters (ADC)

**Table 62. 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 <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 63. 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 & 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	This recommendation will most probably be 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 64. ADC\_SM\_2

SM CODE	ADC_SM_2
Description	Range check by application software
Ownership	End user
Detailed implementation	<p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> <li>– 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).</li> <li>– 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.</li> <li>– 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.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	The implementation (and the related diagnostic efficiency) of this safety mechanism are strongly application-dependent.

Table 65. ADC\_SM\_3

SM CODE	ADC_SM_3
Description	Periodical software test for ADC
Ownership	End user
Detailed implementation	<p>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:</p> <ul style="list-style-type: none"> <li>– Basic complexity: acquisition and check of upper or lower rails (VDD or VSS) and internal reference voltage</li> <li>– High complexity: in addition to basic complexity tests, acquisition of a DAC output connected to ADC input and checking all voltage excursion and linearity</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent



**Table 65. ADC\_SM\_3 (continued)**

Dependency on MCU 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.

**3.6.15 DAC**

**Table 66. DAC\_SM\_0**

SM CODE	DAC_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to DAC configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 67. DAC\_SM\_1**

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

Table 67. DAC\_SM\_1 (continued)

Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Continuous or on demand
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 $T_m$ the minimum duration of DAC wrong signal permanence required to violate the safety function. Efficiency is maximized when execution test frequency is higher than $1/T_m$

### 3.6.16 Comparator

Table 68. COMP\_SM\_0

SM CODE	COMP_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to COMP configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 69. COMP\_SM\_1

SM CODE	COMP_SM_1
Description	1oo2 scheme for comparator
Ownership	End user
Detailed implementation	This safety mechanism is implemented using the two internal comparators to take the same decision. It requires that the comparator voting is handled accordingly.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation

**Table 69. COMP\_SM\_1 (continued)**

Addressed Fault Model	Permanent & 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	This method is not compatible with “window” comparator feature

**Table 70. COMP\_SM\_2**

SM CODE	COMP_SM_2
Description	Plausibility check on inputs
Ownership	End user
Detailed implementation	This method is used to redundantly acquire on dedicated ADC channels the analog inputs that are subjected to comparator function, and to periodically check the coherence of the comparator output on the measured values.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	None

**Table 71. COMP\_SM\_3**

SM CODE	COMP_SM_3
Description	Multiple acquisition by application software
Ownership	End user
Detailed implementation	This method requires that application software will take a decision not on the basis of a comparator single-shot transition, but after multiple events or after the permanence of comparator trigger conditions for a certain amount of time.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Transient

**Table 71. COMP\_SM\_3 (continued)**

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	This recommendation will most probably be satisfied by design on end user application - multiple acquisition is a common technique in industrial applications where it is needed to survive with spurious EMI disturbs on sensor lines

**Table 72. COMP\_SM\_4**

SM CODE	COMP_SM_4
Description	Comparator Lock mechanism
Ownership	ST
Detailed implementation	This safety mechanism prevents configuration changes for comparator control and status registers; it addresses therefore systematic faults in the software application.
Error reporting	NA
Fault detection Time	NA
Addressed Fault Model	None (Fault avoidance)
Dependency on MCU configuration	None
Initialization	Lock protection must be enabled using COMPxLOCKbits in COMP_CSR register.
Periodicity	Continuous
Test for the diagnostic	NA
Multiple faults protection	NA
Recommendations and known limitations	This method doesn't addresses comparator configuration changes due to soft-errors.

**3.6.17 TIM 6/7**

**Table 73. GTIM\_SM\_0**

SM CODE	GTIM_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to general purpose counter timer TIM6 or TIM7 configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0

**Table 73. GTIM\_SM\_0 (continued)**

Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 74. GTIM\_SM\_1**

SM CODE	GTIM_SM_1
Description	1oo2 for counting timers
Ownership	End user
Detailed implementation	<p>This method implements via software a 1oo2 scheme between two counting resources. The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> <li>– Two timers are programmed with same time base or frequency.</li> <li>– 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 will be done at application level, comparing two counters values each time the timer value is used to affect safety function.</li> <li>– 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.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	Tolerance implementation in timer checks is recommended to avoid false positive outcomes of the diagnostic.

### 3.6.18 TIM1/2/3/14/15/16/17

*Note: As the advanced 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 75. ATIM\_SM\_0

SM CODE	ATIM_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to advanced timers TIM1,2,3, 14, 15,16 or 17 configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

Table 76. ATIM\_SM\_1

SM CODE	ATIM_SM_1
Description	1oo2 for counting timers
Ownership	End user
Detailed implementation	This method implements via software a 1oo2 scheme between two counting resources. The guidelines for the implementation of the method are the following: <ul style="list-style-type: none"> <li>– Two timers are programmed with same time base or frequency.</li> <li>– 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 will be done at application level, comparing two counters values each time the timer value is used to affect safety function.</li> <li>– 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.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	On demand
Test for the diagnostic	Not needed

**Table 76. ATIM\_SM\_1 (continued)**

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 77. ATIM\_SM\_2**

SM CODE	ATIM_SM_2
Description	1oo2 for input capture timers
Ownership	End user
Detailed implementation	This method is conceived to protect timers used for external signal acquisition and measurement, like "input capture" and "encoder reading". Implementation requires to connect the external signals also to a redundant timer, and to perform a coherence check on the measured data at application level. Coherence check between timers is executed each time the reading is used by the application software.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	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 78. ATIM\_SM\_3**

SM CODE	ATIM_SM_3
Description	Loopback scheme for PWM outputs
Ownership	End user

**Table 78. ATIM\_SM\_3 (continued)**

Detailed implementation	<p>This method is implemented by connecting the PWM to a separate timer channel to acquire the generated waveform characteristics.</p> <p>The guidelines are the following:</p> <ul style="list-style-type: none"> <li>– Both PWM frequency and duty cycle are measured and checked versus the expected value.</li> <li>– 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.</li> </ul> <p>This measure can be replaced under the end-user responsibility by different loopback schemes already in place in the final application and rated as equivalent. For example if the PWM is used to drive an external power load, the reading of the on-line current value can be used instead of the PWM duty cycle measurement.</p>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	Efficiency versus transient failures is linked to final application characteristics. We define as $T_m$ the minimum duration of PWM wrong signal permanence (wrong frequency, wrong duty, or both) required to violate the safety function. Efficiency is maximized when execution test frequency is higher than $1/T_m$

**Table 79. ATIM\_SM\_4**

SM CODE	ATIM_SM_4
Description	Lock bit protection for timers
Ownership	ST
Detailed implementation	This safety mechanism allows the end user to lock down specified configuration options, avoiding unintended modifications by application software. It addresses therefore software development systematic faults.
Error reporting	NA
Fault detection Time	NA
Addressed Fault Model	None (Fault avoidance)
Dependency on MCU configuration	None
Initialization	Lock protection must be enabled using LOCK bits in the TIMx_BDTR register.
Periodicity	Continuous
Test for the diagnostic	NA



**Table 79. ATIM\_SM\_4 (continued)**

Multiple faults protection	NA
Recommendations and known limitations	This method doesn't addresses timer configuration changes due to soft-errors.

**3.6.19 GPIO - PORT A/B/C/D/E/F**

**Table 80. 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 <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 81. 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 & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	On demand

**Table 81. GPIO\_SM\_1 (continued)**

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 impact of common cause failure, it is recommended to use GPIO lines: <ul style="list-style-type: none"> <li>– belonging to different i/o ports (for instance PORT A and B)</li> <li>– with different bit port number (for instance PORTA.1 and PORTB.5)</li> <li>– mapped to non-adjacent pins on the device package.</li> </ul>

**Table 82. GPIO\_SM\_2**

SM CODE	GPIO_SM_2
Description	Loopback scheme for output GPIO lines
Ownership	End user
Detailed implementation	This method addresses GPIO lines used as outputs. Implementation is done by a loopback scheme, connecting the output to a different GPIO line programmed as input and by using the input line to check the expected value on output port. Comparison is executed by application software periodically and each time output is updated.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on 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	To reduce the potential impact of common cause failure, it is recommended to use GPIO lines: <ul style="list-style-type: none"> <li>– belonging to different i/o ports (for instance PORT A and B)</li> <li>– with different bit number (for instance PORTA.1 and PORTB.5)</li> <li>– mapped to non-adjacent pins on the device package</li> </ul> Efficiency versus transient failures is linked to final application characteristics. We define as $T_m$ the minimum duration of GPIO output wrong signal permanence required to violate the safety function. Efficiency is maximized when execution test frequency is higher than $1/T_m$

**Table 83. GPIO\_SM\_3**

SM CODE	GPIO_SM_3
Description	GPIO port configuration lock register
Ownership	ST

**Table 83. GPIO\_SM\_3 (continued)**

Detailed implementation	This safety mechanism prevents configuration changes for GPIO registers; it addresses therefore systematic faults in software application. The use of this method is encouraged to enhance the end-application robustness for systematic faults
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	None (Systematic only)
Dependency on MCU configuration	None
Initialization	The correct write sequence must be applied to bit 16 (LCKK) of GPIOx_LCKR after the final GPIO configuration has been written by the application software.
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	Not needed
Recommendations and known limitations	This method does not address transient faults (soft errors) that can possibly cause bit-flips on GPIO registers at running time.

**3.6.20 Real Time Clock module (RTC)**

**Table 84. RTC\_SM\_0**

SM CODE	RTC_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to RTC configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 85. RTC\_SM\_1**

SM CODE	RTC_SM_1
Description	Application check of running RTC
Ownership	End user
Detailed implementation	<p>The application software implements some plausibility check on RTC calendar or timing data, mainly after a power-up and further date reading by RTC.</p> <p>The guidelines for the implementation of the method are the following:</p> <ul style="list-style-type: none"> <li>– RTC backup registers are used to store coded information in order to detect the absence of VBAT during power-off period.</li> <li>– RTC backup registers are used to periodically store compressed information on current date or time</li> <li>– The application software will execute minimal consistence checks for date reading after power-on (detecting “past” date or time retrieve).</li> <li>– Application software will periodically check that RTC is actually running, by reading RTC timestamp progress and comparing with an elapsed time measurement based on STM32 internal clock or timers.</li> </ul>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Periodical
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: periodical core self-test software
Recommendations and known limitations	This method provides a limited diagnostic coverage for RTC failure modes. In case of end user application where RTC timestamps accuracy can affect in severe way the safety function (e.g. medical data storage devices), it is strongly recommended to adopt more efficient system-level measures.

**Table 86. RTC\_SM\_2**

SM CODE	RTC_SM_2
Description	Information redundancy on backup registers
Ownership	End user
Detailed implementation	<p>Data stored in RTC backup registers must be protected by a checksum with encoding capability (for instance, CRC). Checksum must be checked by application software before consuming stored data.</p> <p>This method will guarantee data versus erases due to backup battery failures.</p>
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None

**Table 86. RTC\_SM\_2 (continued)**

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	Implementation of this safety method shows a partial overlap with an already foreseen method for Cortex®-M0 (CPU_SM_1); optimizations in implementing both methods are therefore possible.

**Table 87. RTC\_SM\_3**

SM CODE	RTC_SM_3
Description	Application-level measures to detect failures in timestamps/event capture
Ownership	End user
Detailed implementation	This method must detect failures affecting the RTC capability to correct execute the timestamps/event capture functions. Due to the nature strictly application-dependent of this solution, no detailed guidelines for its implementation are given here.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Periodic/On demand
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_0: Periodical core self-test software
Recommendations and known limitations	This method must be used only if the timestamps/event capture function is used in the safety function implementation. It is worth to note that the use of timestamp/event capture in safety related applications where the MCU is in sleep or stop mode is prevented by the assumed requirement ASR7 (refer to <a href="#">Section 3.3.1</a> )

### 3.6.21 Supply voltage system

**Table 88. VSUP\_SM\_0**

SM CODE	VSUP_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0

**Table 88. VSUP\_SM\_0 (continued)**

Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 89. VSUP\_SM\_1**

SM CODE	VSUP_SM_1
Description	Supply voltage internal monitoring (PVD)
Ownership	ST
Detailed implementation	The device features an embedded programmable voltage detector (PVD) that monitors the VDD power supply and compares it to the VPVD threshold. An interrupt can be generated when VDD drops below the VPVD threshold or when VDD is higher than the VPVD threshold.
Error reporting	Interrupt Event generation
Fault detection Time	Depends on threshold programming, refer to functional documentation
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	Protection enable by PVDE bit and threshold programming in Power control register (PWR_CR)
Periodicity	Continuous
Test for the diagnostic	VSUP_SM_0: periodical read-back of configuration registers
Multiple faults protection	CPU_SM_5: external watchdog
Recommendations and known limitations	After a PDR violation is detected, the power switch between VBAT and VDD remains connected to VBAT – refer to MCU functional documentation for more details. Internal monitoring PVD has limited capability to address failures affecting STM32F0 internal voltage regulator. Refer to device FMEA for details. To further mitigate related failure mode, it is possible to disable the internal regulator and directly feed V18 section by an external regulator that can be checked with dedicated method by end user application.

**Table 90. VSUP\_SM\_2**

SM CODE	VSUP_SM_2
Description	Independent watchdog
Ownership	ST

**Table 90. VSUP\_SM\_2 (continued)**

Detailed implementation	The independent watchdog is fed directly by VDD; therefore, major failures in the 1.8 V supply for digital logic (core or peripherals) will not affect its behavior but may lead to a violation of the IDWG timeout.
Error reporting	Reset signal generation
Fault detection Time	Depends on implementation (watchdog timeout interval)
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	IWDG activation. It is recommended to use the “Hardware watchdog” in Option byte settings (IWDG will be automatically enabled after reset)
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_1: control flow monitoring in application software
Recommendations and known limitations	None

**Table 91. VSUP\_SM\_3**

SM CODE	VSUP_SM_3
Description	Internal temperature sensor check
Ownership	End user
Detailed implementation	The internal temperature sensor must be periodically tested in order to detect abnormal increase of the die temperature – hardware faults in supply voltage system may cause excessive power consumption and consequent temperature rise.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	None
Periodicity	Periodic
Test for the diagnostic	Not needed
Multiple faults protection	VSUP_SM_3: Supply voltage internal monitoring (PVD)
Recommendations and known limitations	This method also mitigates the eventuality of common-cause affecting the MCU and due to too high temperature. Refer to the device datasheet to set the threshold temperature.

### 3.6.22 Reset and clock control subsystem

**Table 92. CLK\_SM\_0**

SM CODE	CLK_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to configuration registers for clock and reset system (refer to RCC register map). Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 93. 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 HSE clock activity and executes the corresponding recovery action, such as: <ul style="list-style-type: none"> <li>– switch-off HSE</li> <li>– commutation on the HIS</li> <li>– generation of related NMI</li> </ul>
Error reporting	NMI
Fault detection Time	Depends on implementation (clock frequency value)
Addressed Fault Model	Permanent & Transient
Dependency on MCU configuration	None
Initialization	CSS protection must be enabled on Clock interrupt register (RCC_CIR) after boot stabilization
Periodicity	Continuous
Test for the diagnostic	CLK_SM_0: periodical read-back of configuration registers



**Table 93. CLK\_SM\_1 (continued)**

Multiple faults 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 correct managing the faulty situation by application software features.

**Table 94. CLK\_SM\_2**

SM CODE	CLK_SM_2
Description	Independent watchdog
Ownership	ST
Detailed implementation	The independent watchdog IWDG is able to detect failures in internal main MCU clock (lower frequency).
Error reporting	Reset signal generation
Fault detection Time	Depends on implementation (watchdog timeout interval)
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	IWDG activation. It is recommended to use the “Hardware watchdog” in Option byte settings (IWDG will be automatically enabled after reset)
Periodicity	Continuous
Test for the diagnostic	Not needed
Multiple faults protection	CPU_SM_1: control flow monitoring in application software
Recommendations and known limitations	In case of usage of IWDG window option, end user must consider possible tolerance in application software execution, to avoid false error reports (affecting system availability).

**Table 95. CLK\_SM\_3**

SM CODE	CLK_SM_3
Description	Internal clock cross-measure
Ownership	End user
Detailed implementation	This method is implemented using TIM14 capabilities to be fed by the 32 KHz RTC clock or an external clock source (if available). TIM14 counter progresses are compared with another counter (fed by internal clock). Abnormal values of oscillator frequency can be therefore detected.
Error reporting	Depends on implementation
Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent
Dependency on MCU configuration	None
Initialization	Depends on implementation
Periodicity	Periodic
Test for the diagnostic	Not needed

**Table 95. CLK\_SM\_3 (continued)**

Multiple faults protection	CPU_SM_1: control flow monitoring in application software CPU_SM_5: external watchdog
Recommendations and known limitations	Efficiency versus transient faults is negligible. It provides only medium efficiency in permanent clock-related failure mode coverage.

### 3.6.23 Watchdogs (IWDG, WWDG)

**Table 96. WDG\_SM\_0**

SM CODE	WDG_SM_0
Description	Periodical read-back of configuration registers
Ownership	End user
Detailed implementation	This method must be applied to WDG or WDG configuration registers. Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

**Table 97. WDG\_SM\_1**

SM CODE	WDG_SM_1
Description	Software test for watchdog at startup
Ownership	End user
Detailed implementation	This safety mechanism ensures the right functionality of the internal watchdogs in use. 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 MCU configuration	None

**Table 97. WDG\_SM\_1 (continued)**

Initialization	Depends on implementation
Periodicity	Startup (see below)
Test for the diagnostic	Not needed
Multiple faults 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 during maintenance or offline periods. It could be associated to IEC61508 concept of “proof test” and so it cannot be accounted for a diagnostic coverage contribution during operating time.

**3.6.24 Debug**

**Table 98. DBG\_SM\_0**

SM CODE	DBG_SM_0
Description	Independent watchdog
Ownership	ST
Detailed implementation	The debug unintentional activation due to hardware random fault will result in the massive disturbance of CPU operations, leading to intervention of the independent watchdog or alternately, the other system watchdog WWGDG or an external one.
Error reporting	Reset signal generation
Fault detection Time	Depends on implementation (watchdog timeout interval)
Addressed Fault Model	Permanent
Dependency on MCU 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	None

**3.6.25 Cyclic Redundancy Check module (CRC)**

**Table 99. CRC\_SM\_0**

SM CODE	CRC_SM_0
Description	CRC self-coverage
Ownership	ST
Detailed implementation	The CRC algorithm implemented in this module (CRC-32 Ethernet polynomial: 0x4C11DB7) offers excellent features in terms of error detection in the message. Therefore permanent and transient faults affecting CRC computations are easily detected by any operations using the module to recompute an expected signature.
Error reporting	Depends on implementation

Table 99. CRC\_SM\_0 (continued)

Fault detection Time	Depends on implementation
Addressed Fault Model	Permanent & 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	None

### 3.6.26 System configuration controller (SYSCFG)

Table 100. 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 <a href="#">Section 3.6.5</a> .
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0
Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	This method is mainly overlapped by several other “configuration register read-back” required for other MCU peripherals. It is reported here for the sake of completeness.

### 3.6.27 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 101. FFI\_SM\_0

SM CODE	FFI_SM_0
Description	Unused peripherals disable
Ownership	End user
Detailed implementation	<p>This method contributes to the reduction of the probability of cross-interferences caused by peripherals not used by the software application, in case a hardware failure will cause an unintentional activation.</p> <p>After the system boot, the application software must disable all unused peripherals with this procedure:</p> <ul style="list-style-type: none"> <li>– Enable reset flag on AHB and APB peripheral reset register</li> <li>– Disable clock distribution on AHB and APB peripheral clock enable register</li> </ul>
Error reporting	NA
Fault detection Time	NA
Addressed Fault Model	NA
Dependency on MCU 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 102. FFI\_SM\_1

SM CODE	FFI_SM_1
Description	Periodical read-back of interference avoidance registers
Ownership	End user
Detailed implementation	<p>This method contributes to the reduction of the probability of cross-interferences between peripherals that can potentially conflict on the same input/output pins, including for instance unused peripherals. This diagnostic measure must be applied to following registers:</p> <ul style="list-style-type: none"> <li>– Clock enable and disable registers</li> <li>– Alternate functions programming registers</li> </ul> <p>Detailed information on the implementation of this method can be found in <a href="#">Section 3.6.5</a>.</p>
Error reporting	Refer to NVIC_SM_0
Fault detection Time	Refer to NVIC_SM_0
Addressed Fault Model	Refer to NVIC_SM_0
Dependency on MCU configuration	Refer to NVIC_SM_0
Initialization	Refer to NVIC_SM_0
Periodicity	Refer to NVIC_SM_0

Table 102. FFI\_SM\_1 (continued)

Test for the diagnostic	Refer to NVIC_SM_0
Multiple faults protection	Refer to NVIC_SM_0
Recommendations and known limitations	Refer to NVIC_SM_0

### 3.6.28 Notes on multiple faults scenario

In principle, IEC61508 requires to analyze multiple faults scenarios, therefore restrictions to one fault at time conditions could be not acceptable. Accordingly, the safety analysis for STM32F0 took into consideration also multiple faults scenarios. Furthermore, following the spirit of ISO26262 (the reference and state-of-art standard norm for integrated circuit safety analysis) the analysis investigated faults able to “disable” prescript safety mechanisms, in order to individuate mitigation measure for such condition. Then the descriptions for each given safety mechanisms in above sections reports in field “Multiple faults protection” which is the mitigation measure against disable. The robustness of STM32F0 safety concept is therefore confirmed.

## 3.7 Conditions of use

[Table 103](#) provides a summary of the safety concept recommendations reported in [Section 3.6: Description of hardware and software diagnostics](#). The conditions of use to be applied to STM32F0 Series MCUs are reported in form of safety mechanism requirements. Exception is represented by some conditions of use introduced by FMEA analysis in order to correctly address specific failure modes. These conditions of use are reported at the end of [Table 103](#).

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 to achieve SIL2 on a single MCU.
- + = Recommended as additional safety measure, but not considered in this Safety Manual for the computation of safety metrics. STM32F0 series users can skip the implementation in case it is in contradiction with functional requirements or overlapped by another mechanism marked as “++”.
- o = optional, not needed or related to specific MCU configuration

The “X” marker in the “Perm” and “Trans” columns in [Table 103](#), indicates that the related safety mechanism is effective for such fault model.

Table 103. List of safety mechanisms

STM32F0 function	Diagnostic	Description	Rank	Perm	Trans
Arm Cortex-M0 CPU	CPU_SM_0	Periodical software test addressing permanent faults in Arm Cortex-M0 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	Arm Cortex-M0 HardFault exceptions	M	X	X
	CPU_SM_4	Stack hardening for application software	+	X	X
	CPU_SM_5	External watchdog	++	X	X
	CPU_SM_6	Independent watchdog	++	X	X
System Flash	FLASH_SM_0	Periodical software test for Flash memory	++	X	-
	FLASH_SM_1	Control flow monitoring in application software	+	X	X
	FLASH_SM_2	Arm Cortex-M0 HardFault exceptions	M	X	X
	FLASH_SM_3	Option byte write protection	M	-	-
	FLASH_SM_4	Static data encapsulation	++	X	X
	FLASH_SM_5	Option byte redundancy with load verification	M	X	X
	FLASH_SM_6	Flash unused area filling code	+	-	-
System SRAM	RAM_SM_0	Periodical software test for SRAM memory	++	X	-
	RAM_SM_1	Parity bit check	++	X	X
	RAM_SM_2	Stack hardening for application software	+	X	X
	RAM_SM_3	Information redundancy for system variables in application software	++	X	X
	RAM_SM_4	Control flow monitoring in application software	o	X	X
	RAM_SM_5	Periodical integrity test for application software in RAM	o	X	X
System interconnect	BUS_SM_0	Periodical software test for interconnections	++	X	-
	BUS_SM_1	Information redundancy in intra-chip data exchanges	++	X	X
NVIC	NVIC_SM_0	Periodical read-back of configuration registers	++	X	X
	NVIC_SM_1	Expected and unexpected interrupt check by application software	++	X	X
DMA	DMA_SM_0	Periodical read-back of configuration registers	++	X	X
	DMA_SM_1	Information redundancy on data packet transferred via DMA	++	X	X
	DMA_SM_2	Information redundancy including sender or receiver identifier on data packet transferred via DMA	++	X	X
	DMA_SM_3	Periodical software test for DMA	++	X	-
	DMA_SM_4	DMA transaction awareness	++	X	X

Table 103. List of safety mechanisms (continued)

STM32F0 function	Diagnostic	Description	Rank	Perm	Trans
CAN	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 safing	++	X	X
USART	USART_SM_0	Periodical read-back of configuration registers	++	X	X
	USART_SM_1	Protocol error signals	++	X	X
	USART_SM_2	Information redundancy techniques on messages	++	X	X
	USART_SM_3	Information redundancy techniques on messages, including end-to-end safing	++	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 safing	+	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 Safing	+	X	X
USB	USB_SM_0	Periodical read-back of configuration registers	++	X	X
	USB_SM_1	Protocol error signals	++	X	X
	USB_SM_2	Information redundancy techniques on messages	++	X	X
	USB_SM_3	Information redundancy techniques on messages, including end-to-end safing	+	X	X
HDMI-CEC	HDMI_SM_0	Periodical read-back of configuration registers	++	X	X
	HDMI_SM_1	Protocol error signals	+	X	X
	HDMI_SM_2	Information redundancy techniques on messages	++	X	X
TSC	TSC_SM_0	Periodical read-back of configuration registers	++	X	X
	TSC_SM_1	Multiple acquisition by application software	++	-	X
	TSC_SM_2	Application-level detection of permanent failures of TSC acquisition	+	X	-



Table 103. List of safety mechanisms (continued)

STM32F0 function	Diagnostic	Description	Rank	Perm	Trans
ADC	ADC_SM_0	Periodical read-back of configuration registers	++	X	X
	ADC_SM_1	Multiple acquisition by application software	++	-	X
	ADC_SM_2	Range check by application software	++	X	X
	ADC_SM_3	Periodical software test for ADC	++	X	-
DAC	DAC_SM_0	Periodical read-back of configuration registers	++	X	X
	DAC_SM_1	DAC output loopback on ADC channel	++	X	X
COMP	COMP_SM_0	Periodical read-back of configuration registers	++	X	X
	COMP_SM_1	1oo2 scheme for comparator	++	X	X
	COMP_SM_2	Plausibility check on inputs	+	X	-
	COMP_SM_3	Multiple acquisition by application software	+	-	X
	COMP_SM_4	Comparator LOCK mechanism	+	-	-
TIM6/7	GTIM_SM_0	Periodical read-back of configuration registers	++	X	X
	GTIM_SM_1	1oo2 for counting timers	++	X	X
TIM1/2/3/14/15/16/17	ATIM_SM_0	Periodical read-back of configuration registers	++	X	X
	ATIM_SM_1	1oo2 for counting timers	++	X	X
	ATIM_SM_2	1oo2 for input capture timers	++	X	X
	ATIM_SM_3	Loopback scheme for PWM outputs	++	X	X
	ATIM_SM_4	Lock bit protection for timers	+	-	-
CRC	CRC_SM_0	CRC self-coverage	++	X	X
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	Information redundancy on backup registers	+	X	X
	RTC_SM_3	Application-level measures to detect failures in timestamps or event capture	o	X	X
Supply system	VSUP_SM_0	Periodical read-back of configuration registers	++	X	X
	VSUP_SM_1	Supply voltage monitoring	++	X	-
	VSUP_SM_2	Independent Watchdog	++	X	-
	VSUP_SM_3	Internal temperature sensor check	o	-	-

Table 103. List of safety mechanisms (continued)

STM32F0 function	Diagnostic	Description	Rank	Perm	Trans
Clock and Reset	CLK_SM_0	Periodical read-back of configuration registers	++	X	X
	CLK_SM_1	CSS Clock Security System	++	X	-
	CLK_SM_2	Independent Watchdog	++	X	-
	CLK_SM_3	Internal clock cross-measure	+	X	-
Watchdogs	WDG_SM_0	Periodical read-back of configuration registers	++	X	X
	WDG_SM_1	Software test for watchdog at startup	o	X	-
Debug	DBG_SM_0	Independent watchdog	++	X	X
System or peripheral control	LOCK_SM_0	Lock mechanism for configuration options	+	-	-
	SYSCFG_SM_0	Periodical read-back of configuration registers	++	X	X
Part separation (no interference)	FFI_SM_0	Unused peripherals disable	++	-	-
	FFI_SM_1	Periodical read-back of interference avoidance registers	++	-	-
Arm Cortex-M0 CPU	CoU_1	The reset condition of Arm Cortex-M0 CPU must be compatible as valid safe state at system level	++	-	-
Debug	CoU_2	STM32F0 debug features must not be used in safety function(s) implementation	++	-	-
Arm Cortex-M0 / Supply system	CoU_3	Low power mode state must not be used in safety function(s) implementation	++	-	-
STM32F0 peripherals	CoU_4	End user must implement the required combination of safety mechanism/CoUs for each STM32 peripherals used in safety function(s) implementation	++	X	X

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 will be affected from a safety point of view by a failure on these modules. Because these modules are used by every end user application, related methods or safety mechanism are mainly conceived to be application-independent. For STM32F0 series microcontroller system critical modules are: CPU, Reset, Power, Clocks, Busmatrix and Interconnect, 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.

## 4 Safety results

This section reports the results of the safety analysis of the STM32F0 Series MCU, according to IEC 61508 and to ST methodology flow, related to the hardware random and dependent failures.

### 4.1 Random hardware failure safety results

The analysis for random hardware failures of STM32F0 Series devices reported in this Safety Manual is executed according to ST methodology flow for safety analysis of semiconductor devices according IEC61508. The accuracy of results obtained are guaranteed by three factors:

- ST methodology flow strict adherence to IEC61508 requirements and prescriptions
- the use during the analysis of detailed and reliable information on microcontroller design
- the use of state-of-the-art fault injections methods and tools for safety metrics verification

The STM32F0 safety analysis has been therefore able to explore the overall and exhaustive list of MCU failure modes, and to individuate for each of them an adequate mitigation measure (safety mechanism). The overall list of STM32F0 failure modes is maintained in related FMEA document. STM32F0 FMEA document can be provided on demand, refer to your local ST sales contact.

In summary, with the adoptions of the safety mechanisms and conditions of use reported in [Section 3.7: Conditions of use](#), it is possible to achieve the integrity levels summarized in [Table 104](#).

**Table 104. Overall achievable safety integrity levels**

MCUs used	Safety architecture	Target	Safety analysis result
1	1oo1/1oo1D	SIL2 LD	Achievable
		SIL2 HD/CM	Achievable with potential performance impact <sup>(1)</sup>
2	1oo2	SIL3 LD	Achievable
		SIL3 HD/CM	Achievable with potential performance impact

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

The resulting relative safety metrics (DC and SFF) and absolute safety metrics (PFH, PFD) are not reported in this section but in the FMEDA snapshot, due to:

- the large number of STM32F0 series part numbers,
- the possibility to declare non-safety-relevant unused peripherals, and
- the possibility to enable or not the different available safety mechanisms.

The FMEDA snapshot is a static document reporting the safety metrics computed at different detail levels (at microcontroller level and for microcontroller basic functions) for a given combination of safety mechanisms and for a given part number. If FMEDA

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

*Note:* Safety metrics computations are restricted to STM32F0 boundary, therefore not including the WDTe, PEv and VMONE (they are described in [Section 3.2](#))

#### 4.1.1 Safety analysis results customization

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

In actual end-user applications, not all the STM32F0 Series parts or modules are used for the implementation of the safety function. That can happen under two possible alternative conditions:

- The part is not used at all (disabled)
- The part is used for the implementation of a function that is non-safety related (for example a GPIO line driving a “power-on” signaling led on an electronic board)

Requiring the implementation of the respective safety mechanism for those unused parts could result in an overkill. The safety analysis results can be therefore customized.

The end user can define the selected STM32F0 Series parts as “non-safety related” under the following conditions (end user responsibility):

- collect rationales and evidences that the parts play no role in safety function implementation
- collect rationales and evidences that the parts do not interfere with the safety function during normal operation due to final system design decisions.
- fulfill the below-reported general condition for the mitigation of the intra-MCU interferences ([Table 105](#)).

The end user is therefore allowed for “non-safety related” parts to do the following:

- discard the part contribution from metrics computations in FMEDA;
- do not implement the related safety mechanisms listed in [Table 103: List of safety mechanisms](#).

With regard to SFF computation, this procedure is equivalent to declare “no part/no effect” as per IEC61508-4, 3.6.13 or 14 definition any failure of discarded modules.

#### 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 STM32F0 internal modules in case of internal failures (freedom from interferences, FFI). These precautions are integral part of the STM32F0 safety concept and they can play a relevant role when multiple microcontroller modules are declared as “non-safety related” by the end user as per [Section 4.1.1](#).

The requirement for the end user is to implement the safety mechanism listed in [Table 105](#) (implementation details can be found in Description of hardware and software diagnostics ) despite any evaluation about their contribution for the safety metrics computations.

**Table 105. List of general requirements for FFI**

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

1. To be implemented only if DMA is actually used

## 4.2 Dependent failures analysis

The analysis of dependent failures is important for microcontrollers. The main sub-classes of dependent failures are the Common Cause Failures (CCF). Their analysis is ruled by the IEC 61508:2 annex E that lists the design requirements to be verified to allow the use of on-chip redundancy for ICs with one common semiconductor substrate. However, annexes E.1 and E.2 apply for HFT=1 while the Annex E.3 must be applied to every on-chip redundancy, intended also in terms of diagnostic implemented on the same silicon.

As there are no on-chip redundancy on STM32F0 Series devices, the CCF quantification through the BetaIC computation method is not required. Note that in the case of 1oo2 safety architecture implementation, the end user is required to evaluate the parameter  $\beta D$ , which is the measure of the common-cause between the two channels) used in PFH computation.

The STM32F0 Series 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 detail in [Section 3.6: Description of hardware and software diagnostics](#).

### 4.2.1 Power supply

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

- VSUP\_SM\_1: detection of abnormal value of supply voltage;
- VSUP\_SM\_2: the independent watchdog has a different supply source from the digital core of the MCU, and this diversity helps to mitigate dependent failures related to the main supply alterations.

The adoption of such safety mechanisms is therefore highly recommended despite their minor contribution to the safety metrics to reach the required safety integrity level. Refer to [Section 3.6.21: Supply voltage system](#) 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 those dependent failures:

- CLK\_SM\_1: the clock security system is able to detect hard alterations (stop) of system clock and activate the adequate recovery actions.
- CLK\_SM\_2: the independent watchdog has a dedicated clock source. The frequency alteration of the system clock leads to the watchdog window violations by the triggering routine on 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 3.6.22: Reset and clock control subsystem](#) for detailed safety mechanisms description.

#### 4.2.3 DMA

DMA is a widely shared resource involved in data transfers operated mainly by all peripherals. Failures of DMA can interfere with the behavior of the system peripherals or application software, leading to non independent failures. The safety mechanisms addressing such dependent failures are the following:

- DMA\_SM\_0,
- DMA\_SM\_1,
- DMA\_SM\_2.

The adoption of such safety mechanisms is therefore highly recommended. It is worth to note that if DMA will not be used for data transfer, then only DMA\_SM\_0 must be implemented. Refer to [Section 3.6.6: DMA](#) for detailed safety mechanisms description.

#### 4.2.4 Internal temperature

The abnormal increase of the internal temperature is a potential source of dependent failures, because it can affect many MCU parts and therefore lead to not-independent failures. The safety mechanism to be used to mitigate this potential effect is the following:

- VSUP\_SM\_3: the internal temperature read and check allow the user to quickly detect potential risky conditions before they lead to a series of internal failures. Refer to [Section 3.6.21: Supply voltage system](#) for the detailed safety mechanism descriptions.

## 5 List of evidences

The Safety Case Database stores all the informations related to the safety analysis performed to derive the results and conclusions reported in this Safety Manual.

In detail, the Safety Case Database is composed of the following:

- Safety Case with the full list of all safety analysis related documents
- ST internal FMEDA tool database for the computation of the safety metrics, including the estimated and measured values
- Safety Report, the document that describes in detail the safety analysis executed on STM32F0 Series devices and the clause-by-clause compliance to IEC 61508
- ST internal fault injection campaign database including tools configuration and settings, fault injection logs and results

Due to presence of ST confidential information, above-described contents are not publicly available and are available for possible competent bodies audit and inspections. This is in line with the clarification of Note 2 on IEC61508:2, 7.4.9.7 requirement.

## Appendix A Change impact analysis for other safety standards

The safety analysis reported in this Safety Manual is executed according to IEC 61508 safety norm. This appendix reports the outcomes of a change impact analysis with respect to different safety standards. The following topics are considered for each addressed new safety standard:

- Differences in the suggested hardware architecture (architectural categories), and how to map to safety architectures of IEC 61508.
- Differences in the safety integrity level definitions and metrics computation methods, and how to recompute and judge the safety performances of STM32F0 Series devices according to the new standard.
- Work products required by the new safety norms, and how to remap or rework (if needed) the existing work products resulting as output of the IEC 61508 compliance activity.

The safety standards examined within this change impact analysis are the followings:

- ISO 13849-1:2006, ISO 13849-2:2010 – Safety of machinery and Safety-related parts of control systems,
- IEC 62061:2012-11, ed. 1.1 –Safety of machinery and Functional safety of safety-related electrical, electronic and programmable electronic control systems,
- IEC 61800-5-2:2007, ed.1.0 –Adjustable speed electrical power drive systems – Part 5-2: Safety requirements – Functional,
- IEC 60730-1:2010, ed. 4.0 –Automatic electrical controls for household and similar use – Part 1: General requirements,
- ISO 26262:2010 – Road vehicles - Electrical or electronic (EE) systems.

### A.1 ISO 13849-1 / ISO 13849-2

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

#### A.1.1 Architectural categories

The section §6.2 of ISO 13849 identifies five categories for the basic parameters, DC, MTTFd and CCF, reflecting the expected resistance to faults of SRP or CS under design and needed for achieving the required PLr. For each category, the standard suggests a typical architecture that meets the related requirements.

Considering ISO 13849 architectural categories defined in §6.2 and focusing on microcontrollers, [Table 106](#) presents a summary for end users willing to develop Logic Solver units suitable for safety critical channels and performing a defined safety function.

The assumptions are listed hereafter:



1. The safety function is realized by combining in series the elements (SRP or CS) input system, signal processing unit, output system.
2. The SRP or CSs elements may be assigned to one or different categories and different PLs.
3. The safety function is completely in the scope of the end user application.
4. The STM32F0 Series MCU with the adoption of safety mechanism described in this Safety Manual as single compliant item is by itself suitable for CM application up to PLd (equivalent to SIL2).

The ISO 13849 architectural categories for Logic Solver are shown in [Table 106](#).

**Table 106. IEC 13849 architectural categories**

Cat.	Ref. §	Summary	Designated architecture of Logic	Block diagram
B	6.2.3	The main category; occurrence of one fault can lead to the loss of the safety function. No need of DC and CCF (usually single channel), MTTFd is low or medium. Highest achievable is PL = b	Single channel architecture, one MCU in 1oo1 Refer to <a href="#">Section 3</a> Compliant item's MTTFd = high	<a href="#">Figure 6</a>
1	6.2.4	Enforcing category B requirements by adopting solutions based on "well tried components" for safety critical application and "well tried" safety principles. A microprocessor is not classified as a "well tried" component. No need of DC and CCF (usually single channel), MTTFd is high. Highest achievable is PL = c.	Single channel architecture, one MCU in 1oo1 Refer to <a href="#">Section 3</a> Compliant item's MTTFd = high	<a href="#">Figure 6</a>
2	6.2.5	With respect to category 1, it is expected to include in the architecture a test equipment performing checks on the safety function and reporting its loss. Overall DC is low, CCF must be evaluated, MTTFd can range from low to high allowing up to PL = d.	Single channel architecture, one MCU in 1oo1d Refer to <a href="#">Section 3</a> Compliant item's MTTFd = high TE is in charge of end user, PL = d	<a href="#">Figure 7</a>
3	6.2.6	With respect to category 1, it is expected to have fault detection mechanisms and any single fault occurrence does not lead the loss of the safety function. Overall DC is low, CCF must be evaluated for the channels, MTTFd can range from low to high allowing up to PL = d	Double channel architecture, two identical MCUs in 1oo2 Refer to <a href="#">Section 3</a> Continuous testing or monitoring Compliant item's MTTFd = high	<a href="#">Figure 8</a>
4	6.2.7	With respect to category 1, it is expected to have fault detection mechanisms and any single fault occurrence does not lead the loss of the safety function. Overall DC is high, CCF must be evaluated for the channels, MTTFd is high allowing PL = e	Double channel architecture, two identical MCUs in 1oo2 Refer to <a href="#">Section 3</a> Continuous testing or monitoring Compliant item's MTTFd = high PLe achievable	<a href="#">Figure 8</a>

Figure 6. Block diagram for IEC 13849 Cat. B and Cat. 1

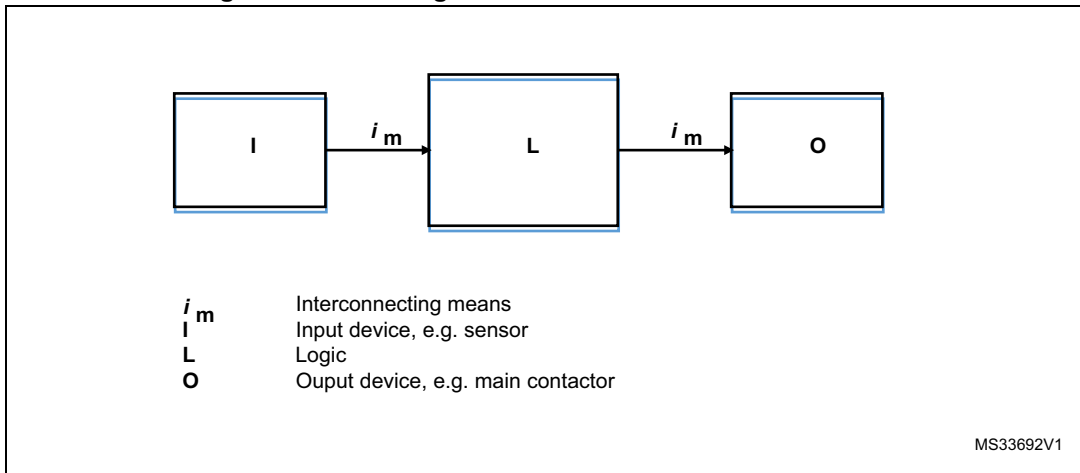


Figure 7. Block diagram for IEC 13849 Cat. 2

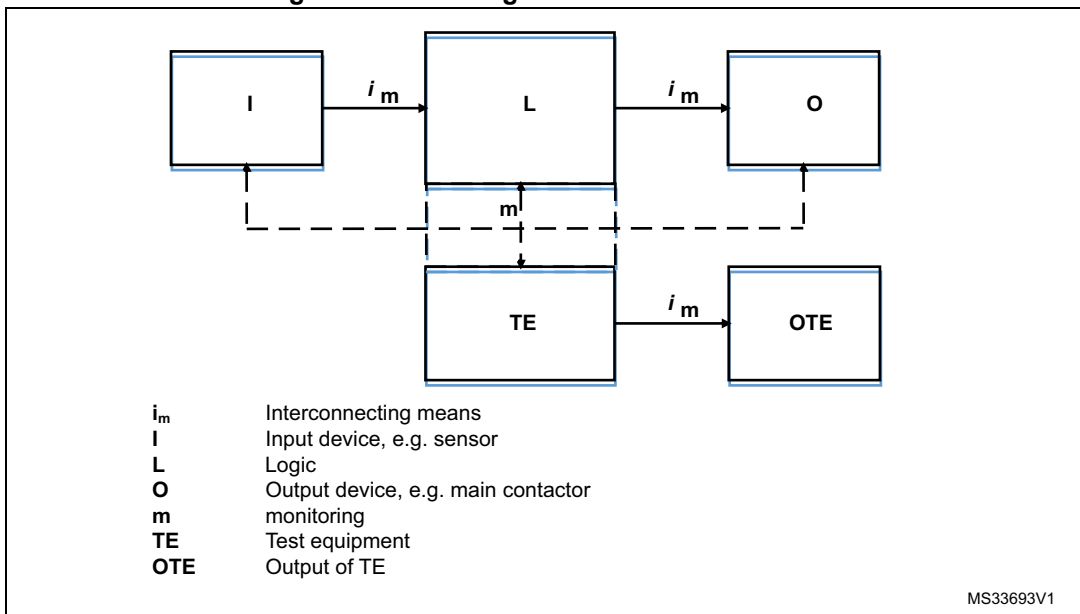
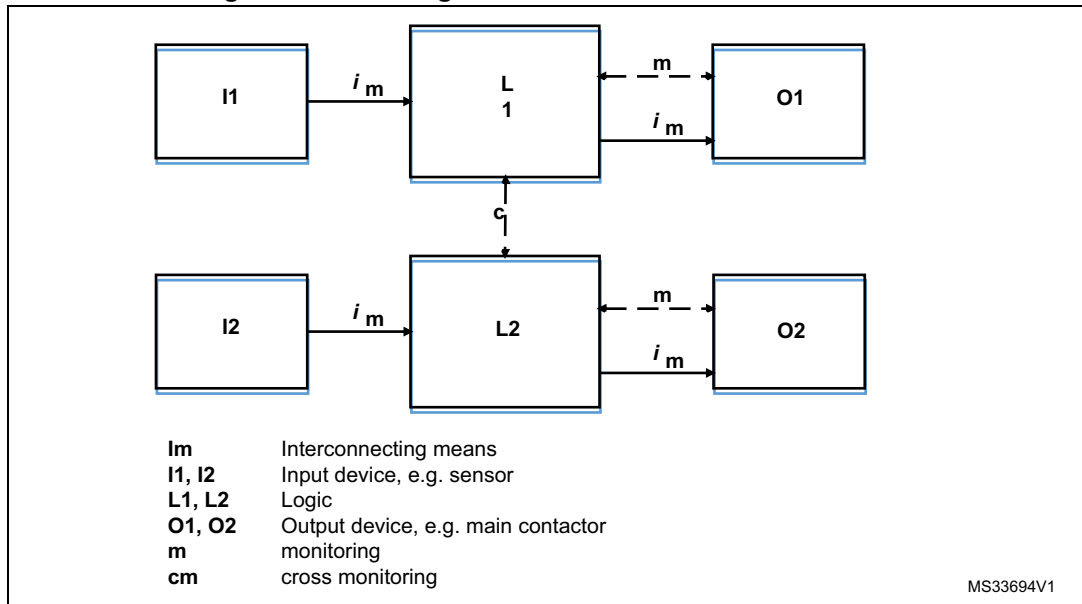


Figure 8. Block diagram for IEC 13849 Cat. 3 and Cat. 4



### A.1.2 Safety metrics computation

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

When for a certain component  $PFH \ll 1$  we can assume that  $MTTFd = 1 / PFH$  [years].

From the reliability theory, MTTF (the inverse of  $\lambda$  and PFH) is a metric applicable only to not repairable systems. Nowadays it is a common practice to use MTBF also for not repairable systems where MTBF has to be understood as the average time for the first (and only) failure of the equipment; in this case MTBF is equal to MTTF.

In ISO 13849-1 the DC for each single component has the same meaning of the IEC 61508 metric; results of this Safety Manual can therefore be reused. However, this standard defines the concept of  $DC_{avg}$  applicable to the whole SRP or CS in the form of the equation defined in Annex E, formula E.1, where the contribution of each part of the control system is weighted with respect to MTTF of the various subsystems of the channel. The standard denies any possibility of fault exclusion while calculating  $DC_{avg}$  (ISO13849-2 Tab.D.21 no exclusion allowed) and this is the same assumption done in STM32F0 analysis in this Safety Manual.

It is necessary to calculate the  $DC_{avg}$  only for subsystem made of a 2 MCUs architecture by applying the formula:

$$DC_{avg} = \frac{\frac{DC_{MCU1}}{MTTF_{MCU1}} + \frac{DC_{MCU2}}{MTTF_{MCU2}}}{\frac{1}{MTTF_{MCU1}} + \frac{1}{MTTF_{MCU2}}}$$

For two identical MCUs having the same DC and MTTF,  $DC_{avg} = DC$ .

*Note:* An evaluation of the possible common failure modes is required for any architectural solution implemented with two channels. ISO 13849 defines a simplified approach with respect to IEC 61508 approach.

Table 7 of the IEC 13849 standard provides a simplified procedure for PL evaluation of SRP or CS based on category,  $DC_{avg}$  and MTTFd. It is worth to note that each architectural solution analyzed in this Safety Manual results in PFH-values producing high values of MTTF.

### A.1.3 Work products

Table 107 lists the work products required by the IEC 13849, and how to map these into available work products from IEC 61508 compliance activity:

**Table 107. IEC 13849 work product grid**

ISO 13849-1		STM32F0 Series IEC 61508 document
Information to be provided	ISO 13849-1 Part-Clause	
Safety functions provided by the SRP or CS	10 Technical documentation	End user responsibility
Characteristics of each safety function		
Exact points at which the safety-related part(s) start and end		
Environmental conditions		
Performance level (PL)		
Category or categories selected		
Parameters relevant to the reliability (MTTFd, DC, CCF and mission time)	10 Technical documentation	STM32F0 Safety Manual and FMEA
Measures against systematic failure		
Technology or technologies used;		
All safety-relevant faults considered		
Justification for fault exclusions (see ISO 13849-2)	10 Technical documentation	End user responsibility
Design rationale (e.g. faults considered, faults excluded)	10 Technical documentation	STM32F0 Safety Manual
Measures against reasonably foreseeable misuse		
Dated reference to this part of ISO 13849 (that is "ISO 13849-1:2006");	11 Information for use	
Category (B, 1, 2, 3, or 4)		
Performance level (a, b, c, d, or e)		
Use of de-energization (see ISO 13849-2)	G.2 Measures for the control of systematic failures	
Measures for controlling the effects of voltage breakdown, voltage variations, overvoltage, under voltage		

Table 107. IEC 13849 work product grid (continued)

ISO 13849-1		STM32F0 Series IEC 61508 document
Information to be provided	ISO 13849-1 Part-Clause	
Measures for controlling or avoiding the effects of the physical environment (for example, temperature, humidity, water, vibration, dust, corrosive substances, electromagnetic interference and its effects)	G.2 Measures for the control of systematic failures	End user responsibility
Program sequence monitoring must be used with SRP or CS containing software to detect defective program sequences		
Measures for controlling the effects of errors and other effects arising from any data communication process (see IEC 61508-2:2000, 7.4.8)		
Failure detection by automatic tests	G.2 Measures for the control of systematic failures	STM32F0 Safety Manual
Computer-aided design tools capable of simulation or analysis	G.3 Measures for avoidance of systematic failures	End user responsibility
Simulation	-	
Safety-related specification for machine control	App. J, tab.J.1 (SW)	End user responsibility
Definition of the control architecture		
Software descriptions	App. J, tab.J.1 (SW)	Software User Guide (End user responsibility because in charge of implementing software-based diagnostics)
Function block modeling	App. J, tab.J.1 (SW)	SW requirements specification (End user responsibility because in charge of implementing software-based diagnostics)
Encoding comments in the code	App. J, tab.J.1 (SW)	Code inspection results (End user responsibility because in charge of implementing software-based diagnostics)
Encoding re-reading sheets		

Table 107. IEC 13849 work product grid (continued)

ISO 13849-1		STM32F0 Series IEC 61508 document
Information to be provided	ISO 13849-1 Part-Clause	
Correspondence matrix	App. J, tab.J.1 (SW)	Software module test specification Software system integration test specification Programmable electronic hardware and software integration tests specification (End user responsibility because in charge of implementing software-based diagnostics)
Test sheets	App. J, tab.J.1 (SW)	Software module test report Software system integration test report Programmable electronic hardware and software integration tests report SW verification report (End user responsibility because in charge of implementing software-based diagnostics)

## A.2 IEC 62061:2012-11

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

With respect of the safety lifecycle, the scope of this standard is limited from safety requirements allocation to safety validation.

IEC 62061 is the special standard for the machine domain within the framework of the more generic IEC 61508:2010. Since it is just an application standard, IEC 62061 is not strict with respect to the technical solutions. Moreover it is focused on electrical, electronic and programmable electronic parts of safety-related control systems.

Note that §3.2.26 and §3.2.27 in IEC 62061 apply only to SRECS in HD or CM, suitable for the machines domain. LD equipment are still ruled by IEC 61508 requirements.

The close relationship with IEC 61508:2010 is synthesized by the main assumption that the design of complex electronic components as subsystems or elements of subsystems has to be compliant with requirements of IEC 61508:2010 part 2, Route 1H, ref. to §7.4.4.2. Coming from the IEC 62061 definition §3.2.8, natively a microprocessor has to be considered as a complex component.

For this reason, the results reported in this Safety Manual for the STM32F0 Series item (refer to [Section 4: Safety results](#)), in the scope of IEC 61508 are still applicable also in the machines context ruled by IEC 62061.

End-users can effectively adopt the STM32F0 Series compliant item to design SRECS suitable for the achievement of SIL2 or SIL3 (by adopting two STM32F0 Series MCUs) machines control loops.

The standard defines as “subsystem” (refer to §3.2.5) the level of parts for a system architecture where a dangerous failure could lead to the loss of the safety function.

Concerning the integrity levels achievable for subsystems, the standard suggests a classification based on HFT and SFF as shown in [Table 108](#).

**Table 108. SIL classification versus HFT**

SFF	HFT		
	0	1	2
<60%	Not allowed	SIL1	SIL2
60% - <90%	SIL1	SIL2	SIL3
60% - <99%	SIL2	SIL3	SIL3
≥90%	SIL3	SIL3	SIL3

SIL 3 is the highest requirement for SRCF in this context. SIL 4 is out of scope since the final outcome of the development is a control system for one machine only.

For the designer, the SIL values listed in the table has to be seen as the SILCL for the subsystem where SILCL is the maximum SIL claimable for a SRECS subsystem, as defined in IEC 62061, §3.2.24.

## A.2.1 Architectural categories

The standard in §6.7.8.2 defines a set of basic system architectures to be used for the design of SRECS implementing their SRCFs. A key point is the definition of “subsystem”, refer to §3.2.5, as the level of parts for a system architecture where a dangerous failure could lead to the loss of the safety function.

Focusing on the microcontrollers, IEC 62061 proposed architectures are here quickly summarized for supporting end users in the development of their Logic Solver units usable as subsystems for the implementation of a SRCF.

The assumptions for the correct understanding of the architectures are listed hereafter:

1. The SRCF is completely in the scope of the end user.
2. The STM32F0 Series device with the adoption of safety mechanism described in this Safety Manual as single compliant item is by itself suitable for applications up to SILCL 2.
3. Two identical STM32F0 Series devices with the adoption of safety mechanism described in this Manual must be used for achieving HFT ≠ 0, when required by basic architectures.
4. For a microcontroller, the parameter T1, mentioned in the standard as the minimum between service life or proof test, is intended as the lifetime (mission time) assumed equal to 10 years, as per [Section 3.3.1: Assumed safety requirements](#) of this Manual.

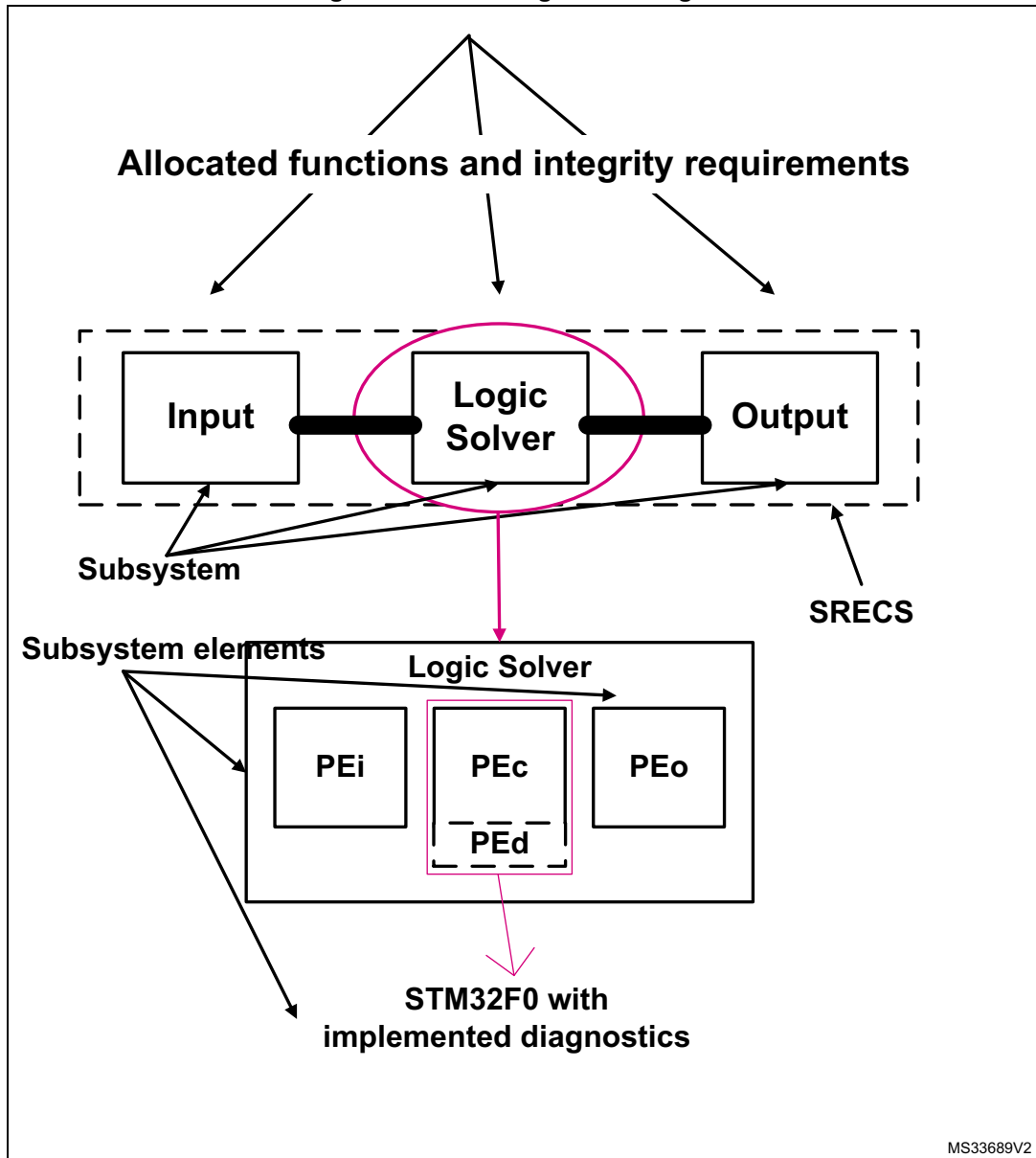
Table 109. IEC 62061 architectural categories

Cat.	Ref. §	Summary	Basic architecture of Logic
A	6.7.8.2.2	Equivalent of 1oo1, with HFT = 0, no diagnostic function(s). Overall PFH <sub>DSSA</sub> is the probability of dangerous failure of MCU	Single channel architecture, one MCU in 1oo1, n=1  $PFH_{DSSA} = \lambda_{De1} \left[ \frac{1}{\text{Hours}} \right]$ <ul style="list-style-type: none"> <li>- SILCL = 1 if SFF &lt; 90%</li> <li>- SILCL = 2 if 90 ≤ SFF &lt; 99%</li> <li>- SILCL = 3 if SFF ≥ 99%</li> </ul>
B	6.7.8.2.3	Equivalent to 1oo2 with HFT = 1, a single failure does not lead to the loss of SRCF. No diagnostic function(s).	Dual channel architecture with two identical MCUs <ul style="list-style-type: none"> <li>- SILCL = 1 if SFF &lt; 60%</li> <li>- SILCL = 2 if 60% ≤ SFF &lt; 90%</li> <li>- SILCL = 3 if SFF ≥ 90%</li> </ul> In this case: $\lambda_{De1} = \lambda_{De2} = \lambda_{De}$ $\lambda_{DSSB} = (1 - \beta)^2 \times \lambda_{De}^2 \times T_1 + \beta \times \lambda_{De}$ For β factor see <a href="#">Section 4.2</a>
C	6.7.8.2.4	It is the equivalent of 1oo1d with a diagnostic function that initiates a reaction function as a dangerous failure happens on SRCF. NOTE: diagnostic function provides the Logic Solver with a diagnosis of an external subsystem, e.g. the actuator	Single channel architecture, one MCU in 1oo1, n=1 Diagnostic function is in charge of end user <ul style="list-style-type: none"> <li>- SILCL = 1 if SFF &lt; 90%</li> <li>- SILCL = 2 if 90 &lt; SFF &lt; 99%</li> <li>- SILCL = 3 if SFF ≥ 99%</li> </ul> $\lambda_{DSSC} = \lambda_{De1} (1 - DC_1)$ DC (Diagnostic Coverage) as resulting from FMEDA $PFH_{DSSC} = \lambda_{DSSC} \left[ \frac{1}{\text{Hours}} \right]$
D	6.7.8.2.5	Any single failure does not lead to a loss of the SRCF; it is equivalent to 1oo2d with HFT = 1, with diagnostic function(s). NOTE: diagnostic function provides the Logic Solver with a diagnosis of an external subsystem, e.g. the actuator	Dual channel architecture with two identical MCUs Diagnostic function is in charge of end user <ul style="list-style-type: none"> <li>- SILCL = 1 if SFF &lt; 60%</li> <li>- SILCL = 2 if 60% ≤ SFF &lt; 90%</li> <li>- SILCL = 3 if SFF ≥ 90%</li> </ul> For β factor see <a href="#">Section 4.2</a> DC (Diagnostic Coverage) as resulting from FMEDA In this case: <ul style="list-style-type: none"> <li>- <math>\lambda_{De1} = \lambda_{De2} = \lambda_{De}</math></li> <li>- T2 has to be defined at Logic Solver level by end user</li> </ul>

Based on IEC 62061 §6, [Figure 9](#) shows how to proceed with the development of SRECS implementing the generic control architecture depicted in figure B.1 of the standard.



Figure 9. SRECS high-level diagram



MS33689V2

Where the microprocessor here presented is an STM32F0 Series device with the adoption of the safety mechanisms as defined in [Section 3.7: Conditions of use](#).

### A.2.2 Safety metrics computation

Again, as already seen in ISO13849, is still considered valid the approximation §6.7.8.2.1 NOTE2:

$$\lambda = \frac{1}{MTTF}$$

Where is assumed to be:

$$1 \gg \lambda \times T$$

The failure rate ( $\lambda$ ) in T is the smaller proof test interval or the life time of the subsystem. But from being:

$$PFH_D = \lambda_D \times 1h$$

again:

$$PFH_D = \lambda_D \times 1h$$

Safety analysis executed for STM32F0 according IEC61508 is more and more accurate for the definition of dangerous failure identifications that can be re-mapped in IEC 62061 domain. Thus, values of  $\lambda$  and PFH that are reported in the FMEDA (refer to [Section 4: Safety results](#)), are still valid and can be used into formulas of the previous paragraph.

There is no need for re-computation for the SFF of a microcontroller. The end-user uses the same value resulting from this Safety Manual.

As previously discussed in [Section 4.2: Dependent failures analysis](#), in evaluating CCF for those basic architectures with an HFT = 1, the end-user uses the same result, if available, as achieved by the IEC 61508 approach (refer to IEC 61508:2010-6 Annex D). Alternatively, the end-user can apply the simplified approach from the standard (refer to Annex F) to calculate the  $\beta$  factor value to be used in formulas for PFHD.

### A.2.3 Work products

[Table 110](#) lists the work products required by the IEC 62061 standard and their mapping with the work products from IEC 61508 compliance activity:

**Table 110. IEC 62061 work product grid**

IEC 62061 1.1 Tab.8		STM32F0 Series IEC 61508 document
Information to be provided	IEC 62061-1.1 Clause	
Functional safety plan	4.2.1	End user responsibility
Specification of requirements for SRCFs	5.2	
Functional safety requirements specification for SRCFs	5.2.3	
Safety integrity requirements specification for SRCFs	5.2.4	
SRECS design	6.2.5	STM32F0 Series Safety Manual
Structured design process	6.6.1.2	End user responsibility
SRECS design documentation	6.6.1.8	
Structure of function blocks	6.6.2.1.1	
SRECS architecture	6.6.2.1.5	STM32F0 Series Safety Manual
Subsystem safety requirements specification	6.6.2.1.7	End user responsibility
Subsystem realization	6.7.2.2	

Table 110. IEC 62061 work product grid (continued)

IEC 62061 1.1 Tab.8		STM32F0 Series IEC 61508 document
Information to be provided	IEC 62061-1.1 Clause	
Subsystem architecture (elements & their interrelationships)	6.7.4.3.1.2	STM32F0 Series Safety Manual
Fault exclusions claimed when estimating fault tolerance or SFF	6.7.6.1c / 6.7.7.3	End user responsibility
Software safety requirements specification	6.10.1	
Software based parameterization	6.11.2.4	
Software configuration management items	6.11.3.2.2	
Suitability of software development tools	6.11.3.4.1	
Documentation of the application program	6.11.3.4.5	
Results of application software module testing	6.11.3.7.4	
Results of application software integration testing	6.11.3.8.2	
Documentation of SRECS integration testing	6.12.1.3	
Documentation of SRECS installation	6.13.2.2	
Documentation for installation, use and maintenance	7.2	
Documentation of SRECS validation testing	8.2.4	
Documentation for SRECS configuration management	9.3.1	

### A.3 IEC 61800-5-2:2007

The scope of this standard is the functional safety of adjustable speed electric drive systems. Part 5.2 of the IEC 61800 defines the requirements for the design, development, integration and validation of the safety related parts for power drive speed applications, PDS(SR), within the framework of IEC 61508 first edition. More precisely, this part of IEC 61800 just limits its application to those PSD(RS) operating in HD or CM, referring to §3.10 NOTE1, implementing safety functions with a target integrity up to SIL 3.

From the architectural point of view, this limitation is reflected in two tables, §6.2.2.3 Tab. 3 and Tab. 4, for the two different types of classified devices. The CPU or the whole microcontroller, since these are complex electronics parts, is classified as Type B. Also the concept of HFT is derived from IEC 61508 as it is.

#### A.3.1 Architectural categories

From the architectural point of view, IEC 61800 application is reflected in two tables, §6.2.2.3 Tab. 3 and Tab. 4, for the two different types of classified devices. The CPU or the whole microcontroller, considered as complex electronics parts, are classified as Type B. Also the concept of HFT is derived from IEC 61508 as it is. Architectural remapping on IEC61508 is therefore straightforward.

### A.3.2 Safety metrics computation

The PFH of a safety function performed by PDS(SR) is evaluated by the application of IEC 61508-2. The strong link with the norm IEC 61508 is reflected also by the adoption in IEC 61800-5-2 of the same relevant metrics PFH, ref. to §6.2.1, and SFF, ref. to §6.2.3. So, results of this Safety Manual (and related FMEA or FMEDA) can be re-mapped in IEC61800 domain.

### A.3.3 Work products

Table 111 lists the work products required by the IEC 61800-5-2 standard and their mapping with the work products from IEC 61508 compliance activity.

**Table 111. IEC 61800 work product grid**

IEC 618000 5.2		STM32F0 Series IEC 61508 document
Information to be provided	IEC 61800-5.2 Part-Clause	
Safety requirements specification (SRS) for PDS(SR) including safety function requirements and safety integrity requirements	5.4	End user responsibility
Verification of PDS(SR) safety requirements specification	8.2	
Hardware design on an architectural level	6	
Software design on an architectural level	IEC 61508-3	
Estimation of the probability of failure of safety functions due to random hardware failures on a level of functional block diagrams	IEC 61508-2	STM32F0 Series Safety Manual and FMEDA
Reviews of system design	8.2	End user responsibility
Detailed planning of the validation of safety related PDS(SR).	8.3	
Hardware design	6	
Software design		
Reliability Prediction	6	STM32F0 Series Safety Manual and FMEDA

Table 111. IEC 61800 work product grid (continued)

IEC 618000 5.2		STM32F0 Series IEC 61508 document
Information to be provided	IEC 61800-5.2 Part-Clause	
Reviews of the system design	8.2	End user responsibility
Functional tests on module level		
Integration and test of the safety related PDS(SR).	6.5	
Review of HW or SW integration test results and documentation	8.2	
Develop user documentation describing PDS(SR) installation, commissioning, operation and maintenance.	7	
Complete software and appropriate documentation	8.3	
Documentation of the results of the validation tests		
Validation tests and procedures according to the validation plan		
Documentation of the results of the validation tests		
Subsystem testing plan	6.2.4.1.4	
Integration testing plan		
Validation testing plan		
Configuration testing plan		
Detailed results of each test	9.2.g)	
Any discrepancy between expected and actual results	9.2.h)	
Conclusion of the test: either it has been passed or the reasons for failure	9.2.i)	

## A.4 IEC 60730-1:2010

This IEC 60730-1:2010 version.4.0 standard is applicable to the electrical or electronics control systems guaranteeing safety and reliability for household equipment, building and automation appliances that have a rated power supply voltage ≤ 690 V and a current ≤ 63 A

Annex H of the standard defines specific requirements for electronics controls (HW and SW) which are mandatory when designing safe compliant electronics parts of control systems.

### A.4.1 Architectural categories

In this standard the proposed architectures (refer to § H.11.12.1) are mainly classified on software basis, where specific suggestions are given at software level to avoid systematic faults.

IEC 60730-1:2010 §H.2.22 defines three classes of compliance for the performed control functions:

- Class A (§H.2.22.1): control functions which are not intended to be relied upon for the safety of the application
- Class B (§H.2.22.2): control functions which are intended to prevent an unsafe state of the controlled equipment. Failure of the control function will not lead directly to a hazardous situation
- Class C (§H.2.22.3): control functions which are intended to prevent special hazards such as explosion or which failure could directly cause a hazard in the appliance

For those systems implementing their control function by software, the standard in §H.2.16 defines the set of applicable architectures. The following list summarizes typical solutions for the design of Class B control functions.

- Single channel with functional test (§H.2.16.5). A single CPU executes the software control functions as required. A functional test is performed as the software starts. It guarantees that all critical features are properly working. This solution can be mapped to 1oo1 LD case in IEC61508
- Single channel with periodic self-test (§H.2.16.6). A single CPU executes the software control functions, but embedded periodical tests check the various critical functions of the system without impacting the performance of the fully planned control tasks. This solution can be mapped to 1oo1 HD or CM case in IEC61508
- Dual channel (homogeneous) with comparison (§H.2.16.3): The software is designed to execute identical control functions on two independent CPUs. Both CPUs compare internal signals for fault detection before starting any safety critical task. This solution can be mapped to 1oo2 case in IEC61508

For Class C control functions, the adequate architectural solutions are listed hereafter. The comparison can be achieved by a comparator or by software.

- Dual-channel (homogeneous) with comparison (§H.2.16.3),
- Dual-channel (diverse) with comparison (§H.2.16.2).

The standard states the need for control measures (see §H.11.12.2), and explains how to avoid faults or errors for Class B or Class C software control functions (see §H.11.12.3).

## A.4.2 Safety metrics computation

This safety standard does not mention safety metrics, therefore there it is no need to recompute anything.

The possibility for a system to achieve Class B or Class C ranking is related to the adoption of a certain set of methods; the qualitative table (see Table H.1) in the standard lists the respective types of safety mechanism that need to be present in a Class B or C system.

[Table 112](#) lists the requirements of the standard versus the target ranking (B or C) for the various parts or functions of the STM32F0 Series device, that are detailed in [Section 3.6: Description of hardware and software diagnostics](#). In case the IEC 60730 requires a safety method not yet foreseen in the framework of the IEC 61508 safety analysis, the gap is reported in the related field. For sake of clarity the original text of the standard requirement is omitted in the table (refer to standard).

Table 112. IEC 60730 required safety mechanism for Class B/C compliance

Component <sup>(1)</sup>	Fault/error	Software class		Definitions	SM for Class B <sup>(2)</sup>	SM for Class C <sup>(3)</sup>	Gaps/Notes
		B	C				
1. CPU 1.1 Registers	Stuck at	X	-	H.2.16.5 H.2.16.6 H.2.19.6 H.2.19.8.2	CPU_SM_0	-	None
	DC fault	-	X	H.2.18.15 H.2.18.3 H.2.18.9 H.2.19.5 H.2.19.7 H.2.19.1 H.2.19.2.1 H.2.19.8.1 H.2.19.6 H.2.20.8.2	-	CPU_SM_1 CPU_SM_5 or 1002 architecture	In case of CPU_SM_5 adoption, external watchdog is assumed to be able to drive the safety relevant outputs in safe state.
1.2 Instruction decoding and execution	Wrong decoding and execution	-	X	H.2.18.15 H.2.18.3 H.2.18.9 H.2.18.5	-	1002 architecture	In case of single-MCU architecture the safety mechanism CPU_SM_0 described in the Manual could be used but it must explicitly implement also the "equivalent class test" as described in H.2.18.5 (this test is not needed for IEC 61508).
1.3 Program counter	Stuck at	X	-	H.2.16.5 H.2.16.6 H.2.18.10.4 H.2.18.10.2	CPU_SM_0	-	None
	DC fault	-	X	H.2.16.7 H.2.18.10.3 H.2.18.9 H.2.18.15 H.2.18.3	-	CPU_SM_1 CPU_SM_5 or 1002 architecture	None
1.4 Addressing	DC fault	-	X	H.2.18.15 H.2.18.3 H.2.18.9 H.2.16.7 H.2.18.22 H.2.18.1.1 H.2.18.1.2	-	1002 architecture or BUS_SM_0	None
1.5 Data paths instruction decoding	DC fault and execution	-	X	H.2.18.15 H.2.18.3 H.2.18.9 H.2.16.7 H.2.18.22 H.2.18.1.2	-	1002 architecture or CPU_SM_0	None

Table 112. IEC 60730 required safety mechanism for Class B/C compliance (continued)

Component <sup>(1)</sup>	Fault/error	Software class		Definitions	SM for Class B <sup>(2)</sup>	SM for Class C <sup>(3)</sup>	Gaps/Notes
2. Interrupt handling and execution	No interrupt or too frequent interrupts	X	-	H.2.16.5 H.2.18.10.4	NVIC_SM_1	-	None
	No interrupt or too frequent interrupts related to different sources	-	X	H.2.18.15 H.2.18.3 H.2.18.10.3	-	NVIC_SM_1 or 1oo2 architecture	None
3. Clock	Wrong frequency (for quartz synchronized clock: harmonics or subharmonics only)	X	-	H.2.18.10.1 H.2.18.10.4	CPU_SM_1 CLK_SM_3	-	None
		-	X	H.2.18.10.1 H.2.18.10.4 H.2.18.15 H.2.18.3	-	CPU_SM_1 CLK_SM_3 or 1oo2 architecture	None
4. Memory	All single bit faults	X	-	H.2.19.3.1 H.2.19.3.2 H.2.19.8.2	FLASH_SM_0	-	None
	4.1 Invariable memory 99,6% coverage of all information errors	-	X	H.2.18.15 H.2.18.3 H.2.19.5 H.2.19.4.1 H.2.19.4.2 H.2.19.8.1	-	1oo2 architecture or FLASH_SM_0	In case of adoption of FLASH_SM_0 evidences must be given about the 99,6% target of information errors
4.2 Variable memory	DC fault	X	-	H.2.19.6 H.2.19.8.2	RAM_SM_0	-	None
	DC fault and dynamic cross links	-	X	H.2.18.15 H.2.18.3 H.2.19.5 H.2.19.7 H.2.19.1 H.2.19.2.1 H.2.19.8.1	-	1oo2 architecture	RAM_SM_0 proposed for IEC 61508 needs to be enriched to add features like Abraham or GALPAT
4.3 Addressing (relevant for variable and invariable memory)	Stuck at	X	-	H.2.19.18.2	FLASH_SM_0 RAM_SM_0 or DUAL_SM_0	-	None
	DC fault	-	X	H.2.18.15 H.2.18.3 H.2.18.1.1 H.2.18.22 H.2.19.4.1 H.2.19.4.2 H.2.19.8.1	-	FLASH_SM_0 RAM_SM_0 or 1oo2 architecture	None



Table 112. IEC 60730 required safety mechanism for Class B/C compliance (continued)

Component <sup>(1)</sup>	Fault/error	Software class		Definitions	SM for Class B <sup>(2)</sup>	SM for Class C <sup>(3)</sup>	Gaps/Notes
5. Internal data path 5.1 Data	Stuck at	X	-	H.2.19.8.2	BUS_SM_1	-	None
	DC fault	-	X	H.2.18.15 H.2.18.3 H.2.19.8.1 H.2.18.2.1 H.2.18.22 H.2.18.14	-	1002 architecture or DMA_SM_1 DMA_SM_3 BUS_SM_0 BUS_SM_1	None
5.2 Addressing	Wrong address	X	-	H.2.19.8.2	BUS_SM_1	-	None
	Wrong address and multiple addressing	-	X	H.2.18.15 H.2.18.3 H.2.19.8.1 H.2.18.1.1 H.2.18.22	-	1002 architecture or BUS_SM_0 BUS_SM_1	None
6. External communication 6.1 Data	Hamming distance 3	X	-	H.2.19.8.1 H.2.19.4.1 H.2.18.2.2 H.2.18.14	CAN_SM_2 UART_SM_2 IIC_SM_2 SPI_SM_2 USB_SM_2 HDMI_SM_2	-	None
	Hamming distance 4	-	X	H.2.19.4.2 H.2.18.2.1 H.2.18.15 H.2.18.3	-	1002 architecture or CAN_SM_2 UART_SM_2 IIC_SM_2 SPI_SM_2 USB_SM_2 HDMI_SM_2	In case of adoption of 1002 architecture the communications are managed by both cores In case of adoption of individual safety mechanism for peripherals, the use of CRC32 is mandatory
6.2 Addressing	Wrong address	X	-	H.2.19.8.1 H.2.19.4.1 H.2.18.2.2 H.2.18.14	CAN_SM_2 UART_SM_2 IIC_SM_2 SPI_SM_2 USB_SM_2 HDMI_SM_2 DMA_SM_2	-	None
	Wrong and multiple addressing	-	X	H.2.19.4.2 H.2.18.1.1 H.2.18.15 H.2.18.3	-	1002 architecture	Communications are managed by both cores

Table 112. IEC 60730 required safety mechanism for Class B/C compliance (continued)

Component <sup>(1)</sup>	Fault/error	Software class		Definitions	SM for Class B <sup>(2)</sup>	SM for Class C <sup>(3)</sup>	Gaps/Notes
6.3 Timing	Wrong point in time	X	-	H.2.18.10.4 H.2.18.18	CPU_SM_1	-	None
	Wrong point in time	-	X	H.2.18.10.3 H.2.18.15 H.2.18.3	-	CPU_SM_1 or 1oo2 architecture	None
	Wrong sequence	X	-	H.2.18.10.2 H.2.18.10.4 H.2.18.18	CPU_SM_1	-	None
	Wrong sequence	-	X	H.2.18.10.3 H.2.18.15 H.2.18.3	-	CPU_SM_1 or 1oo2 architecture	None
7. Input/output periphery 7.1 Digital I/O	Fault conditions specified in H.27	X	-	H.2.18.13	GPIO_SM_1 GPIO_SM_2	-	None
		-	X	H.2.18.15 H.2.18.3 H.2.18.8 H.2.18.11 H.2.18.12 H.2.18.22 H.2.18.2	-	GPIO_SM_1 GPIO_SM_2	None
7.2 Analog I/O 7.2.1 AD-and DA- converter	Fault conditions specified in H.27	X	-	H.2.18.13	ADC_SM_2	-	None
		-	X	H.2.18.15 H.2.18.3 H.2.18.8 H.2.18.11 H.2.18.12 H.2.18.22	-	ADC_SM_1 ADC_SM_3 DAC_SM_1 or 1oo2 architecture	In case of adoption of 1oo2 architecture the analog values are acquired by both MCUs
7.2.2 Analog multiplexer	Wrong addressing	X	-	H.2.18.13	ADC_SM_2	-	None
		-	X	H.2.18.15 H.2.18.3 H.2.18.8 H.2.18.22	-	ADC_SM_1 ADC_SM_3 or 1oo2 architecture	In case of adoption of DUAL_SM_01oo2 architecture the analog values needs to be acquired by both MCUs
8. Monitoring devices and comparators	Any output outside the static and dynamic functional specification	-	X	H.2.18.21 H.2.18.17 H.2.18.6	-	WDG_SM_1 CPU_SM_5	None

**Table 112. IEC 60730 required safety mechanism for Class B/C compliance (continued)**

Component <sup>(1)</sup>	Fault/error	Software class		Definitions	SM for Class B <sup>(2)</sup>	SM for Class C <sup>(3)</sup>	Gaps/Notes
		X	-				
9. Custom chips 5) for example. ASIC, GAL, Gate array	Any output outside the static and dynamic functional specification	X	-	H.2.16.6	NA	NA	Not applicable
		-	X	H.2.16.7 H.2.16.2 H.2.18.6	NA	NA	Not applicable

1. For fault or error assessment, some components are divided into their sub functions.
2. It is recognized that some of the addressed measures provide a higher level of assurance than required by this standard for Class B.
3. For each sub function in the table, the software class C measure will cover the software class B fault/error.

*Note: Safety mechanisms separated by “or” word are alternative; safety mechanism listed together are intended to be applied all together.*

### A.4.3 Work products

*Table 113* provides the list of work products that are required by the IEC 60730 standard and their mapping with the work products from the IEC 61508 compliance activity:

**Table 113. IEC 60730 work product grid**

IEC 60730 5.2		STM32F0 Series IEC 61508 document
Information to be provided	IEC 60730 Part-Clause	
Purpose of control	Tab.1 - 6	End user responsibility
Construction of control and whether the control is electronic	Tab.1 - 6a	
Which of the terminals for external conductors are for a wider range of conductor sizes than those indicated in the table of 10.1.4.	Tab.1 - 18	
For screw-less terminals the method of connection and disconnection	Tab.1 - 19	
Details of any special conductors which are intended to be connected to the terminals for internal conductors	Tab.1 - 20	
Method of mounting control	Tab.1 - 31	
Method of providing earthing of control	Tab.1 - 31a	
Method of attachment for non-detachable cords	Tab.1 - 32	STM32F0 Series Safety Manual
Details of any limitation of operating time	Tab.1 - 34	

Table 113. IEC 60730 work product grid (continued)

IEC 60730 5.2		STM32F0 Series IEC 61508 document
Information to be provided	IEC 60730 Part-Clause	
Type 1 or Type 2 action	Tab.1 - 39	End user responsibility
Additional features of Type 1 or Type 2 actions	Tab.1 - 40	
Reset characteristics for cut-out action	Tab.1 - 43	
Any limitation to the number or distribution of flat push-on receptacles which can be fitted	Tab.1 - 45	
Any Type 2 action must be designed so that manufacturing deviation and drift of its operating value, operating time or operating sequence is within the limit declared in requirements 41, 42 and 46 of Table 1 (7.2 of the previous edition)	Tab.1 - 46	
Extent of any sensing element	Tab.1 - 47	
Operating value (or values) or operating time	Tab.1 - 48	STM32F0 Series Safety Manual
Control pollution degree	Tab.1 - 49	End user responsibility
Rated impulse voltage	Tab.1 - 75	
Temperature for the ball pressure test	Tab.1 - 76	
Maximum declared torque on single bush mounting using thermoplastic material	Tab.1 - 78	
Pollution degree in the micro-environment of the creep age or clearance if cleaner than that of the control, and how this is designed	Tab.1 - 79	End user responsibility
Rated impulse voltage for the creep age or clearance if different from that of the control, and how this is ensured	Tab.1 - 80	
The values designed for tolerances of distances for which the exclusion from fault mode "short" is claimed	Tab.1 - 81	
For SELV or PELV circuits, the ELV limits realized	Tab.1 - 86	
Value of accessible voltage of SELV or PELV circuit, if different from 8.1.1, product standard referred to for the application of the control, in which standard(s) the accessible SELV or PELV level(s) is (are) given	Tab.1 - 87	
The minimum parameters of any heat dissipater (for example heat sink) not provided with an electronic control but essential to its correct operation	H.7 - 52	
Software sequence documentation	H.7 - 66	
Program documentation	H.7 - 67	
Software fault analysis	H.7 - 68	
Software class(es) and structure	H.7 - 69	
Analytical measures and fault or error control techniques employed	H.7 - 70	

Table 113. IEC 60730 work product grid (continued)

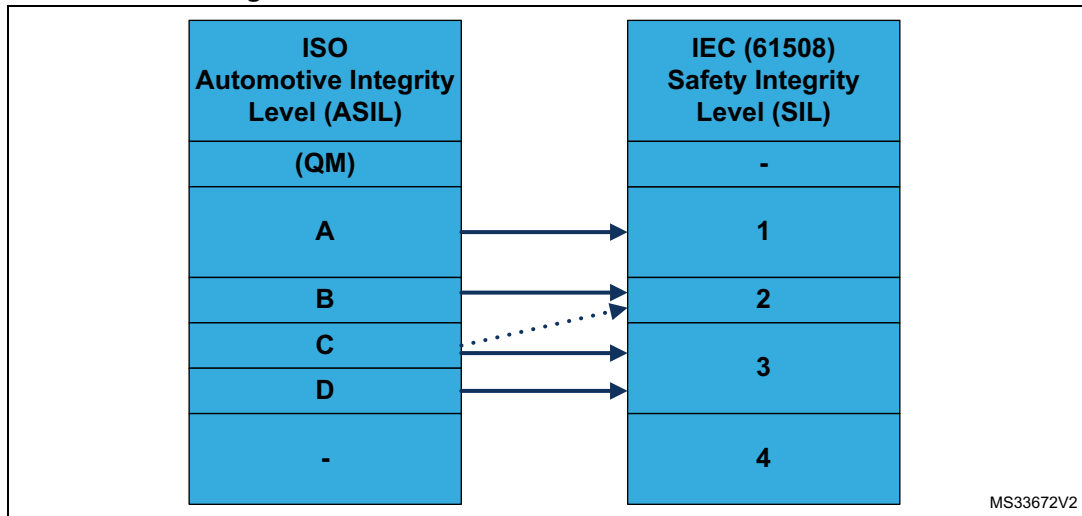
IEC 60730 5.2		STM32F0 Series IEC 61508 document
Information to be provided	IEC 60730 Part-Clause	
Software fault or error detection time(s) for controls with software classes B or C	H.7 - 71	STM32F0 Safety Manual and End user responsibility
Control response(s) in case of detected fault or error	H.7 - 72	End user responsibility
Software safety requirements	H.11.12.3.2.1	
Software architecture	H.11.12.3.2.2	
Module design and coding	H.11.12.3.2.3	
Design and coding standards	H.11.12.3.2.4	
Testing	H.11.12.3.3	
Inspection	H.2.17.5	
Walk-through	H.2.17.9	
Static analysis	H.2.17.7.1	
Dynamic analysis	H.2.17.1	
Hardware analysis	H.2.17.3	
Hardware simulation	H.2.17.4	
Failure rate calculation	H.2.17.2	STM32F0 Series safety manual and FMEDA
FMEA	H.2.20.2	STM32F0 FMEA
Operational test	H.2.17.6	End user responsibility

## A.5 ISO 26262:2010

This international standard is the reference for the functional safety for the automotive domain. It derives from IEC 61508 standard, and includes relevant modifications.

ISO 26262 redefines the safety integrity levels in term of Automotive SIL (ASIL) with a scale from A, the lowest level, to D, the highest level. A correlation matrix between SIL and ASIL values has been empirically identified by TÜV SÜD and is illustrated in [Figure 10](#).

Figure 10. Correlation matrix between SIL and ASIL



**A.5.1 Architectural categories**

Not Applicable - since ISO 26262 does not define any category.

**A.5.2 Safety metrics computation**

Hardware metrics in ISO 26262 standard have been defined with a slightly different perspective from IEC61508:

- Single Point Fault Metric (SPFm): defined with the same formula of SFF in IEC61508, can differ according to different definition of safe faults (see below)
- Diagnostic Coverage (DC) is defined in the same way of IEC61508;
- Latent Faults Metric (LFm): dedicated ISO26262 safety metrics to evaluate the robustness of the design against faults affecting diagnostic parts. We have no equivalent in IEC61508.

It is worth noting that these failures that are classified in IEC 61508 standard as no-parts/no-effect, in ISO26262 are classified as “safe failures”. As a result, IEC61508 computations for SFF are “conservative” and so using as SPF values taken from STM32F0 FMEDA is possible.

For such kind of Commercial Off-the-Shelf (COTS) microcontroller as STM32F0, the natural target in ISO scenario is ASIL B (90% SPF target for permanent and transient, and 60% for latent). As these are the same targets as for 1oo1 SIL2 case, one can assume that the same set of conditions of use or safety mechanisms apply. Metrics computations are detailed into the FMEDA for microcontrollers of the STM32F0 Series; note that the resulting PMHF values comply with the expectations for an ASIL B MCU.

We can conclude that the ASIL B target is achievable with some constraints for the final application. Note that safety diagnostic measures based on periodical execution of software are executed at least once each FTTI.

For the STM32F0 Series device, the fulfillment of ASIL B latent faults metrics (60%) is achievable with the adoption of the same safety mechanism combination that guarantees the microcontroller to be suitable for SIL2 applications.

*Note: Due to differences between IEC61508 and ISO26262 interpretation on local targets for microcontroller modules or functions, safety performances achieved by STM32F0 in a SIL2 scenario could be not compatible with an ISO26262 application based on ISO26262-5, 9.4.3 section (the so-called 'cut-set' approach). If your ISO26262 safety analysis uses such approach, check carefully STM32F0 FMEDA failure rates at function level.*

### A.5.3 Work products

Table 114 lists the work products required by the ISO 26262 standard and their mapping with the work products from IEC 61508 compliance activity:

**Table 114. IEC 26262 work product grid**

IEC 26262		STM32F0 Series IEC 61508 document
Information to be provided	IEC 26262 Part-Clause	
Technical safety requirements specification	4-6.5.1	STM32F0 Series Safety Manual
Technical safety concept	4-7.5.1	
Safety analysis reports resulting from requirement	4-7.5.6	
Hardware safety requirements verification report	5-6.5.3	
Hardware safety analysis report	5-7.5.2	
Analysis of the effectiveness of the architecture of the item to cope with the random hardware failures	5-8.5.1	
Review report of evaluation of the effectiveness of the architecture of the item to cope with the random hardware failures	5-8.5.2	
Analysis of safety goal violations due to random hardware failures	5-9.5.1	
Review report of evaluation of safety goal violations due to random hardware failures	5-9.5.3	STM32F0 FMEDA
Software safety requirements specification	6-6.5.1	End user Responsibility
Software architectural design specification	6-7.5.1	
Software verification report (refined)	6-11.5.3	
Results of safety analyses	9-8.5.1	STM32F0 Series Safety Manual, FMEA and FMEDA

*Note: STM32F0 FMEA should be reworked in order to map IEC61508 reference failure modes into ISO26262 ones*

## Revision history

**Table 115. Document revision history**

Date	Revision	Changes
19-Jun-2014	1	Initial release.
30-Jan-2015	2	Extended the user manual applicability to STM32F0 series and to STM32-SafeSIL part number. Updated: <a href="#">Figure 1: STMicroelectronics product development process</a> <a href="#">Figure 16: Block diagram for IEC 62061 Cat. B</a> <a href="#">Figure 18: Block diagram for IEC 62061 Cat. D</a>
3-Mar-2015	3	Replaced all NVC occurrences with NVIC in <a href="#">Table 3: List of safety mechanisms</a> and in <a href="#">Table 17: List of STM32F0 Series safety mechanism overlapped by fRSTL_STM32F0_SIL2(3)</a> .
5-Oct-2017	4	Removed: <a href="#">former fRMethodology</a> , <a href="#">Dual MCU architecture</a> , <a href="#">Latent Fault detection</a> , <a href="#">examples of safety architecture</a> , <a href="#">fRSTL_STM32F0_SIL2(3)</a> Added: <a href="#">Figure 3: 1oo1 reference architecture</a> , <a href="#">Figure 4: 1oo2 reference architecture</a> , <a href="#">Section 3.6.26: System configuration controller (SYSCFG)</a> , <a href="#">Section 3.6.28: Notes on multiple faults scenario</a> , <a href="#">Table 5 to Table 102</a> for description of hardware and software diagnostics Updated: <a href="#">Section 3.3.1: Assumed safety requirements</a> , <a href="#">Section A.4.1: Architectural categories</a> , <a href="#">Table 104: Overall achievable safety integrity levels</a> , <a href="#">Table 112: IEC 60730 required safety mechanism for Class B/C compliance</a>



**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

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

No license, express or implied, to any intellectual property right is granted by ST herein.

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

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

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

© 2017 STMicroelectronics – All rights reserved