Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

UM1741
User manual

STM32F0 series safety manual

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 can avoid going into the details of the functional safety design and validation to give an estimation about the impact to the overall safety function.

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. This manual and FMEDA data were developed in cooperation with the safety expertise company YOGITECH using their faultRobust Methodology (fRMethodology).

The safety analysis summarized in this manual takes into account the variation in terms of memory size, internal peripheral number and presence and package between the different part numbers of the ARM® Cortex®-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*.

Contents

List of tables

List of figures

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.

1.2 Terms and abbreviations

Table 1. Terms and abbreviations

Acronym	Definition
ADAS	Advanced Driver Assistance System
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
EHSR	Essential Health and Safety Requirement
EUC	Equipment Under Control
FE	Final Element (that is generalized actuator)
FIT	Failure In Time
FMEA	Failure Mode Effect Analysis
FMEDA	Failure Mode Effect Diagnostic Analysis
FPU	Floating Processing Unit
HD	High Demand
HFT	Hardware Fault Tolerance
HW	Hardware
INTC	Interrupt Controller
ITRS	International Technology Roadmap for Semiconductors
LD	Low Demand

Table 1. Terms and abbreviations (continued)

Acronym	Definition
MCU	Microcontroller Unit
MTBF	Mean Time Between Failure
MTTFd	Mean Time to Failure
OC	Output Circuit
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
SE	Sensor Element
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

1.3 Reference normative

This document is written in compliance with the IEC 61508 international norm for functional safety of electrical/electronic/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	*Section 3*
D2.1 b) identification of the hardware and/or software configuration of the compliant item	*Section 3.2*
D2.1 c) constraints on the use of the compliant item and/or assumptions on which analysis of the behavior or failure rates of the item are based	*Section 3.2*
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;	*Section 3.7*
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;	*Section 3.2.2*
D2.2 g) for every failure mode in c) the outputs of the compliant item initiated by the internal diagnostics;	*Appendix B*
D2.2 h) any periodic proof test and/or maintenance requirements;	*Section 3.7*
D2.2 i) for those failure modes, in respect of a specified function, that are capable of being detected by external diagnostics, sufficient information shall be provided to facilitate the development of an external diagnostics capability.	
D2.2 j) the hardware fault tolerance;	*Section 3*
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.

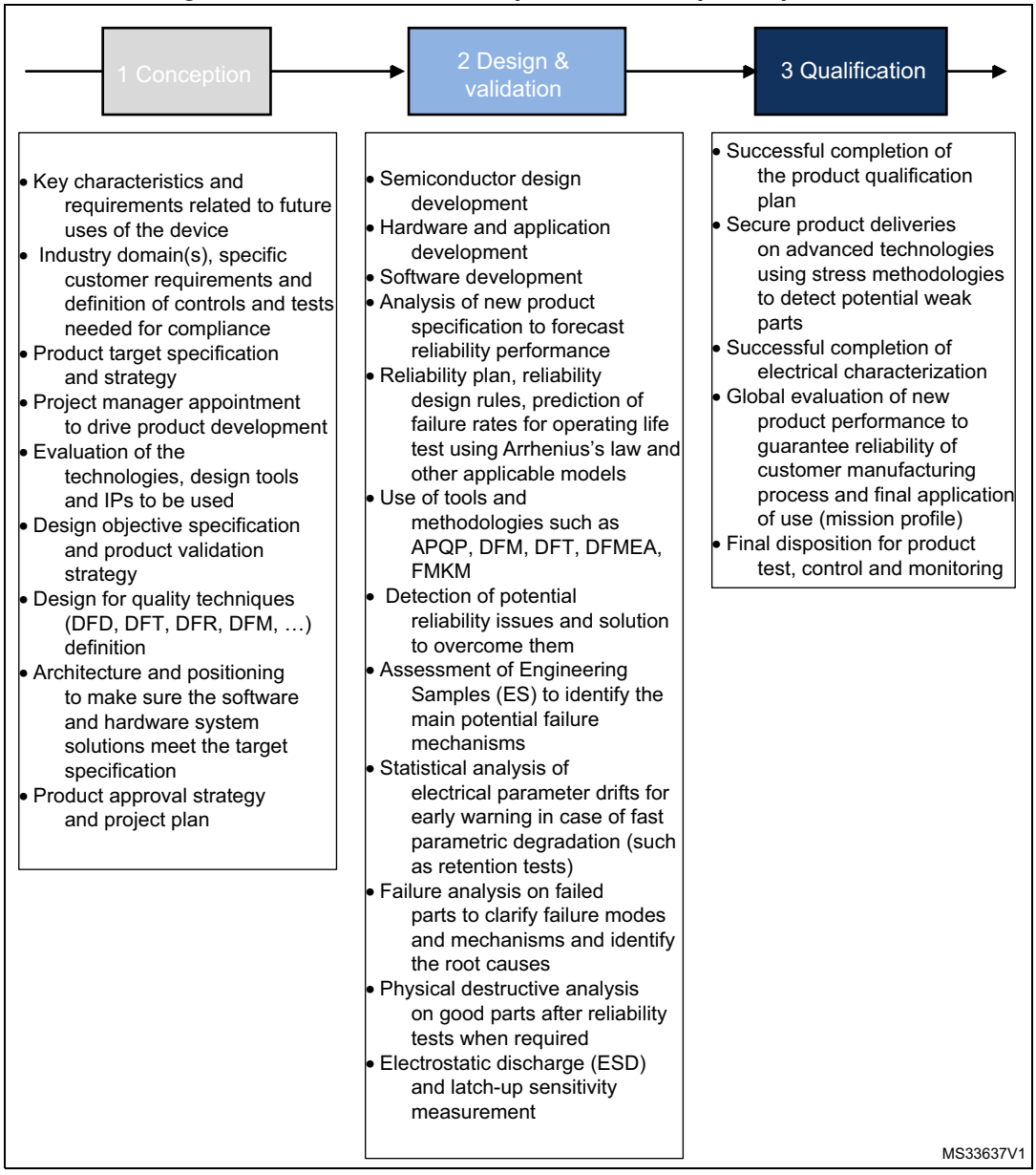
2.1 STMicroelectronics standard development process

STMicroelectronics (ST) serves four industry domains:

1. Standard products,
2. 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.
3. 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.
4. 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



2.2 Yogitech fRMethodology process

Yogitech fRMethodology is the “white-box” approach for safety design exploration proprietary of Yogitech, including tools and methodology to FMEA/FTA analysis and fault injection of integrated circuits. *Appendix A: Overview of fRMethodology* reports additional informations.

Yogitech contribution to IEC 61508 compliance of STMicroelectronics development process can be summarized in these key elements:

- Failure rate estimation based on multiple industry standards as well as STMicroelectronics manufacturing data,
- Application of Yogitech fault injection techniques/tools to validate the safety metrics claimed for STMicroelectronics devices belonging to STM32 program.

3 Reference safety architecture

3.1 Introduction

The STM32F0 series microcontrollers described in this document is a Safety Element out of Context (SEooC), that is, the intent is to describe a compliant item that can be used 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, that is with respect to reference safety requirements as also assumptions with respect to the design external to that compliant item.

As a consequence of the SEooC 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 - such as the application-related assumptions for dangerousness factors, frequency of failures and diagnostic coverage already guaranteed by the application - that have been considered during the following steps of the analysis.

Additional details on the reference safety architecture are given in *Appendix B: Examples of safety architectures – Informative*.

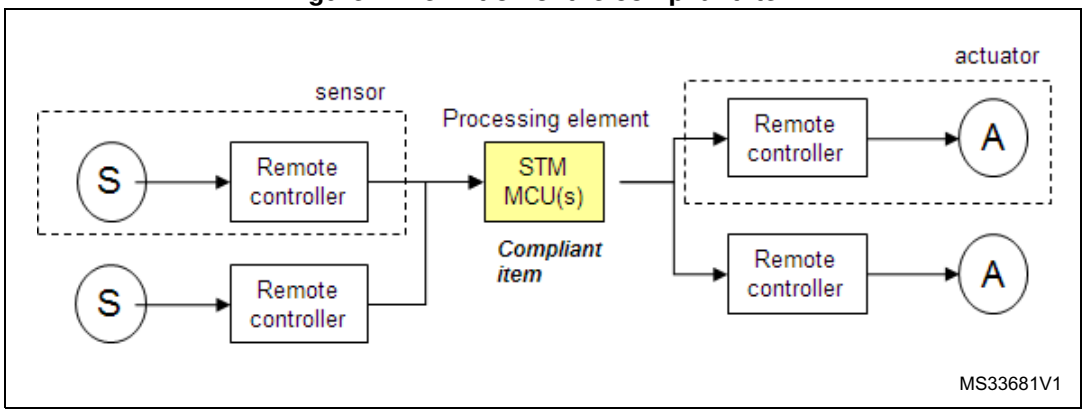
3.2 Compliant item

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 shall 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



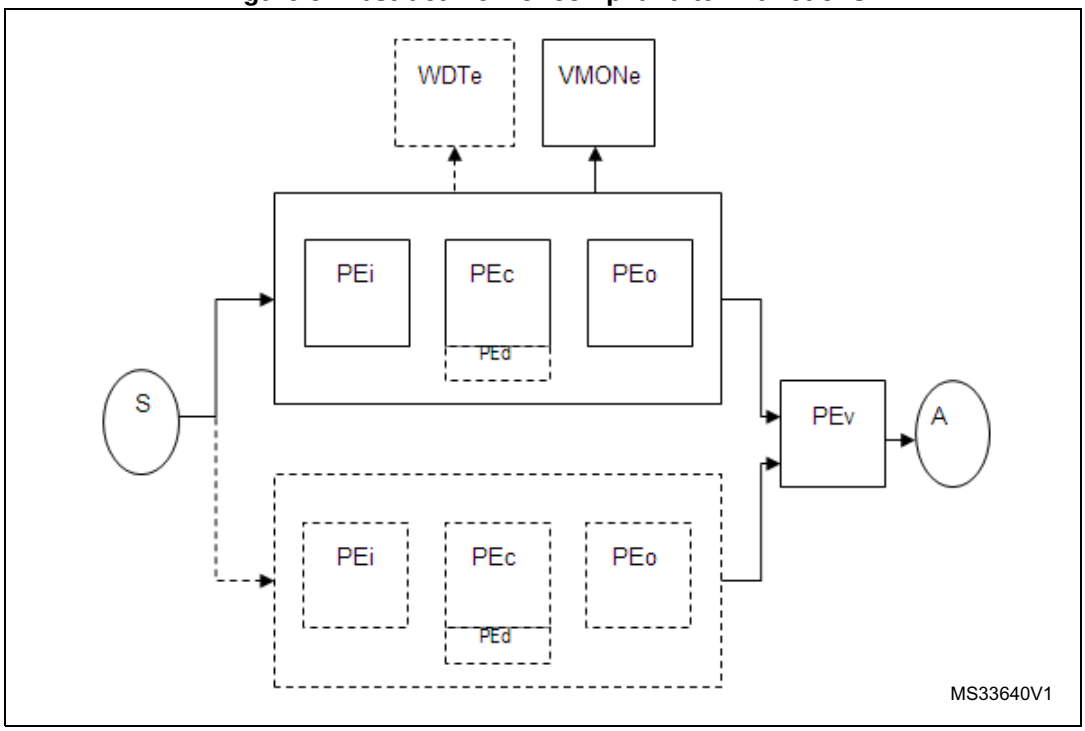
Other components might be related to the compliant item, like the external HW components needed to guarantee either the functionality of the STM32F0 (external memory, clock quartz etc) or its safety (for example the external watchdog, voltage supervisors).

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, 1oo2d or 2oo2 architecture (see *Appendix B: Examples of safety architectures – Informative*), a further voting processing element (PEv) can be present;
- in the 1oo2d case (see again *Appendix B: Examples of safety architectures – Informative*), the abstract view is the same as 1oo2 but with the addition of diagnostic processing elements (PEd), having the role of performing cross-diagnostic functions and contributing to the decision of the voter PEv;
- processes external to the compliant item are considered to guarantee functional safety, such as a watchdog (WDTe) and voltage monitors (VMONe).

Figure 3. Abstract view of compliant item functions



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*.

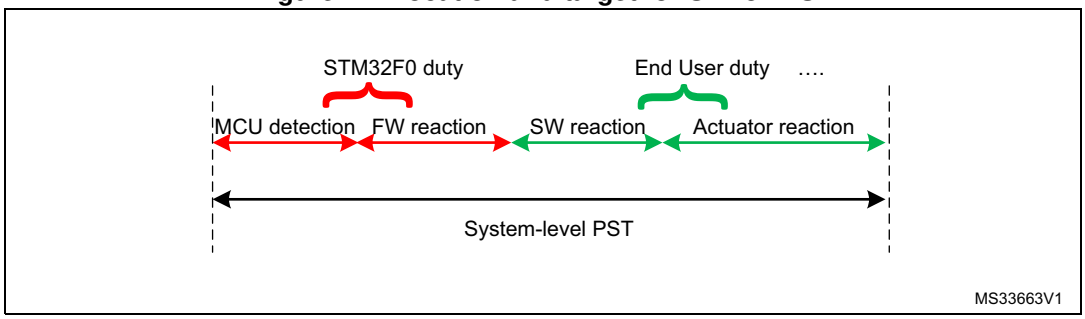
3.3 Assumed requirements

3.3.1 Assumed safety requirements

It is assumed that the concept specification, the hazard and risk analysis, the overall safety requirement specification and the consequent allocation has determined the following requirements for the compliant item (assumed safety requirements):

- The compliant item can be used for four kinds of safety functions:
  - A continuous mode / high-demand SIL3 safety function (CM3), or
  - A low-demand SIL3 safety function (LD3), or
  - A continuous mode / high-demand SIL2 safety function (CM2), or
  - A low-demand SIL2 safety function (LD2).
- The compliant item is used in a safety function with a worst case budget of 10 ms for the STM32 MCU to detect and react to a failure, which corresponds to the portion of the Process Safety Time[a] allocated to the STM32F0 MCU ("STM32F0 duty" in *Figure 4*)

Figure 4. Allocation and target for STM32 PST



- The compliant item is used in a safety function powered-on for a long time. It is assumed to not require any proof test and the lifetime of the product is considered to be not less than 10 years.
- The safe state of the compliant item is the one in which either:
  - the operating system (OS) is informed by the presence of a fault and a reaction is possible, or
  - if the OS cannot be informed or the OS is not able to execute a reaction[b]:

    in case of a 1oo2 or 1oo2D architecture, the PEv shall be directly informed so that the PEv itself is able to achieve or maintain the safe state of the system or,

    in case of 1oo1 and 1oo2/1oo2D high demand or continuous mode architectures, the safe state of the electronic system is "de-energize".
- The compliant item is assumed to be analyzed according to routes 1H and 1S of IEC 61508:2.

a. As explained in the following section, for the HFT=1 computations the value of the process safety time is not as stringent as it is for HFT=0 architectures (that is clarified further in the document).

b. The end user shall take into account that hardware random 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).

The base assumptions about the de-energize state and the repair conditions, in the computation of the PFD/PFH, are as follows;

- In the 1oo1 mode:
  - The system is de-energized as soon as a fault is identified by the HW or SW diagnostics.
  - If the fault has been identified as a transient fault, the compliant item can be reset and the safety function can continue after reset.
  - If the fault has been identified as a permanent fault or if it has not been identified, the compliant item is assumed to be kept de-energized.
- In the 1oo2 / 1oo2D low-demand modes:
  - The 1oo2 system is NOT de-energized if a fault is identified in one of the two channels.
  - The faulty compliant item can be repaired, that is the faulty STM32F0 MCU can be replaced or one of the external components might be replaced.
- In the 1oo2 / 1oo2D high-demand or continuous modes:
  - The system is de-energized as soon as a fault is identified in one of the two channels or in the shared logic.
  - If the fault has been identified as a transient fault, the compliant item can be reset and the safety function can continue after reset.
  - If the fault has been identified as a permanent fault or if it has not been identified, the compliant item is assumed to be kept de-energized, and the compliant item cannot be repaired, that is the faulty STM32F0 MCU cannot be replaced or one of the external components cannot be replaced.

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/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*.

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 and the techniques and measures given in IEC 61508-2 Annex F have been applied. 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 informations to be shown, no functional details are included in this document.

Note that the part numbers of the STM32F0 series represent 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 Yogitech and related coverage figures reported in this manual are based on such guidelines.

Please read the following definitions:

- **end user:** the final user of STM32F0 that is in charge of integrating the MCU in a real application (for example an electronic control board).
- **application software**: the real software that runs on the STM32F0 and that is used to implement the safety function.

3.6.1 Cortex®-M0 CPU

Periodical core self test software - CPU_SM_0

Permanent faults affecting the CPU Core ARM® Cortex®-M0 are addressed through a dedicated software test executing a sequence of instructions and data transfers.

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). A detailed safety analysis has shown that a self-test software based only on software testing operation is not able to reach the required values of coverage due to the complexity of the CPU. Therefore, in order to reach the required values of coverage, the self-test software has to be specified by means of a detailed analysis of all the CPU failure modes and related failure modes distribution. Moreover, it has to be verified by means of fault injection (according to ISO 26262:10, Annex A - the state of the art in terms of safety analysis applied to integrated circuits - fault injection is the recommended method for the verification of failure modes coverage in modern and complex microprocessor like Cortex®-M0).

The overall test software suite is assumed to be periodically executed with a time period compatible with the IEC 61508 requirements for the relationship between PST and the diagnostic test interval.

Control flow monitoring in application software – CPU_SM_1

A significant part of the failure distribution of ARM® Cortex®-M0 core for permanent faults is related to failure modes directly related to program counter loss of control or hang-up. Due to their intrinsic nature, such failure modes are not addressed by a standard software test method based on the execution of sequences of instruction/data access and consequent checks. 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 within DTI.

This diagnostic measure also contributes to the transient fault detection affecting the program counter and branch execution subpart in ARM® Cortex®-M0.

The guidelines for the implementation of the method are the following:

- The different internal states of the application software is well documented and described (the use of a dynamic state transition graph is encouraged).
- The monitoring of the correctness of each transition between different states of the application software is implemented.
- The transition through all expected states during the normal application software program loop is checked.
- The function in charge of triggering the system watchdog is implemented in order to constrain the triggering (preventing the watchdog reset) also to the correct execution of the above-described method for program flow monitoring.

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. In any case the safety metrics do not depend on the watchdog in use (the adoption of independent or external watchdog contributes to the mitigation of dependent failures, see *Section 4.2.2: Clock*).

Double computation in application software – CPU_SM_2

A timing redundancy for safety-related computation is considered to detect transient faults affecting the ARM® Cortex®-M0 CPU subparts devoted to mathematical computations and data access.

The guidelines for the implementation of the method are the following:

- The requirement needs be applied only to safety-relevant computation, that is to the computations that in case of wrong result could interfere with the system safety

functions. Such computation shall be therefore carefully identified in the original application software source code.

- Both mathematical operation and comparison are intended as computation.
- The redundant computation for comparison could be implemented according to the following template:
  - Original code:

```
If (VarA > VarB) then { ( execute function) }
```

  - Modified code:

```
copyVarA:=VarA;
copyVarB:=VarB;
If (VarA > VarB) then {
If (copyVarA <= copyVarB) then { (signal_error);  break }
( execute function)
}
```

- 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.
- End users are responsible to carefully avoid that the intervention of optimization features of the used compiler removes the timing redundancy introduced according to this current condition of use.

ARM® Cortex®-M0 HardFault exceptions – CPU_SM_3

HardFault exception raise is an intrinsic safety mechanism implemented in ARM® Cortex®-M0 core, mainly devoted to intercept systematic faults due to software limitations and/or error in software design, leading for example to execution of undefined operations, unaligned address access. This safety mechanism is therefore able to detect hardware random faults inside the CPU bringing to such described abnormal operations.

Stack hardening for application software – CPU_SM_4

The stack hardening method is required to address faults affecting the CPU register bank. This method is based on source code modification, introducing information redundancy in register-passed information to the called functions.

The guidelines for the implementation of the method are the following:

- Pass also the redundant copy of the passed parameters values (possibly inverted) and execute a coherence check in the function.
- Pass also the redundant copy of the passed pointers and execute a coherence check in the function.
- For the parameters that are not protected by redundancy, implement defensive programming techniques (plausibility check of passed values). For example enumerated fields are to be checked for consistency.

External watchdog – CPU_SM_5

Using an external watchdog for the control flow monitoring method (CPU_SM_1) contributes to further reduce potential common cause failures, because the external watchdog will be clocked and supplied independently from the STM32F0.

3.6.2 System FLASH memory

Periodical software test for Flash memory – FLASH_SM_0

Permanent faults affecting the system Flash memory (that is the 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/program fetch).

Without information over the frequency of usage of different occupied Flash sections, in principle, all used Flash area are assumed to be tested with a time period compatible with the IEC 61508 requirements for the relationship between PST and the diagnostic test interval.

Control flow monitoring in application software – FLASH_SM_1

Permanent and transient faults affecting the system Flash memory (that is the memory cells and address decoder) can interfere with the access operation by the CPU, leading to wrong data or instruction fetches. Such wrong data and operation, if able to heavily interfere with the expected flow of the application software, are detected by strong control flow mechanism linked to a system watchdog. For more detailed implementation guidelines for such technique refer to safety mechanism CPU_SM_1 in *Control flow monitoring in application software – CPU_SM_1*.

ARM® Cortex®-M0 Hardfault exceptions – FLASH_SM_2

Hardfault exception raise is an intrinsic safety mechanism implemented in ARM® Cortex®-M0 core, mainly devoted to intercept systematic faults that are due to software limitations and/or error in software design, leading for example to the execution of undefined operations, unaligned address access. This safety mechanism is therefore able to detect hardware random faults (both permanent and transient) that affect the system Flash memory (cells and address decoder) bringing to such described abnormal operations.

Option byte write protection – FLASH_SM_3

This safety mechanism prevents unintended writes on the option byte; it addresses therefore systematic faults in software application and not hardware random faults affecting the option byte value during running time. The use of this method is encouraged to enhance end application robustness for systematic faults.

3.6.3 System SRAM memory

Periodical software test for SRAM memory – RAM_SM_0

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 ensures the target SFF coverage for both the RAM cells and the address decoder. The end user provides also evidences of the effectiveness of the coverage of the selected method.

The overall test software suite is assumed to be periodically executed with a time period compatible with the IEC 61508 requirements for the relationship between PST and the diagnostic test interval.

Parity bit check – RAM_SM_1

The Parity check on the system SRAM provides a relevant contribution to the detection of hardware random faults (both permanent and transient) that affect the RAM data cells (no expected contribution is expected for address decoder faults detection). This option is assumed to be enabled by the user after the boot.

Stack hardening for application software – RAM_SM_2

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. This method 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.

The guidelines for the implementation of the method are the following:

- Pass also the redundant copy of the passed parameters values (possibly inverted) and execute a coherence check in the function.
- Pass also the redundant copy of the passed pointers and execute a coherence check in the function.
- For parameters that are not protected by redundancy, implement defensive programming techniques such as the plausibility check of the passed values for example to check the consistency of enumerated fields.

Information redundancy for safety-related variables in application software – RAM_SM_3

To address transient faults affecting SRAM controller, it is required to implement information redundancy on the safety-related system variables stored in the RAM.

The guidelines for the implementation of this method are the following:

- The system variables that are safety-related (in the sense that a wrong value due to a failure in reading on the RAM affects the safety functions) are well-identified and documented.
- The arithmetic computation and/or decision based on such variables are/is executed twice and the two final results are compared.

Note that the 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 (see *Control flow monitoring in application software – CPU_SM_1*).

3.6.4 System bus interconnect

Periodical software test for interconnections – BUS_SM_0

The intra-chip connection resources (Bus Matrix, AHB/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. This method which is based on the periodical execution of software-based tests is executed at least once per DTI.

According to IEC 61508:2 Table A.8, A.7.4 the above-described method is considered able to achieve high levels of coverage (and it has to be verified by means of fault injection).

Information redundancy in intra-chip data exchanges – BUS_SM_1

Both permanent and transient fault affecting the intra-chip connection features (Bus Matrix, AHB/APB bridges) are addressed by information redundancy techniques implemented over the messages exchanged inside the MCU.

Lock mechanism for configuration options – LOCK_SM_0

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 and transient faults such as soft errors, that cause some bit-flip on registers during running time. The use of this method is encouraged to enhance the end application robustness to systematic faults.

The method described in this section provides a marginal protection against permanent and transient faults affecting system interconnect bus.

3.6.5 NVIC and EXTI controller

Periodical read-back of configuration registers – NVIC_SM_0

This diagnostic measure that is typically referred to as “Read Back Periodic by Software of Configuration Registers” is implemented by executing a periodical check of the configuration registers of each used system peripheral against its expected value that was previously stored in RAM and adequately updated after each configuration change. It mainly addresses the transient faults that affect the configuration registers, by detecting bit flips in the registers due to these transient faults. The register test is executed at least once per DTI in order to be able to claim the related diagnostic coverage.

Expected and unexpected interrupt check – NVIC_SM_1

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. It contributes to detecting both permanent and transient fault for all the above-reported failure modes affecting interrupt handling.

The guidelines for the implementation of the method are the following:

- The list of the implemented interrupt for the MCU are well documented, reporting also the expected frequency of each request when possible (for example the interrupts related to ADC conversion completion, therefore coming on a deterministic way).
- Individual counters are maintained for each interrupt request served, in order to detect in a given time frame the cases of a) no interrupt at all b) too many interrupt requests ("babbling idiot" interrupt source). The control of the time frame duration shall be regulated according to the individual interrupt expected frequency.
- Interrupt vectors related to unused interrupt source point to a default handler that will report, in case of triggering, a faulty condition (unexpected interrupt).
- In case an interrupt service routine is shared between different sources, a plausibility check on the caller identity is implemented.
- Interrupt requests related to not-safety-relevant peripherals are handled with the same method here described, despite their originator safety classification; in order to decrease the complexity of this method implementation, the use of polling instead of interrupt for not-safety-relevant peripherals is suggested.

3.6.6 DMA

Periodical read-back of configuration registers – DMA_SM_0

This diagnostic measure that is typically referred to as "Read Back Periodic by Software of Configuration Registers" is implemented by executing a periodical check of the configuration registers of the DMA peripheral against its expected value that was previously stored in RAM and adequately updated after each configuration change. It mainly addresses the transient faults that affect the configuration registers, by detecting bit flips in the registers due to these transient faults. The register test is executed at least once per DTI in order to be able to claim the related diagnostic coverage.

Information redundancy on data packet transferred via DMA – DMA_SM_1

The information redundancy required on the DMA data transfer cannot be implemented, in line with the DMA concept, with multiple transfers of the same data block. Therefore, this method is implemented by constraining the use of DMA to the transfer of data packed covered by a redundancy check (like CRC, or similar techniques). Note that other diagnostic measures on data communication peripherals (potential DMA sources or destinations) already foresee the implementation of information redundancy at message level – therefore the overlap with those measures could reduce the potential complexity in the referred-method implementation.

Information redundancy including sender/receiver identifier on data packet transferred via DMA – DMA_SM_2

This method requires that the information redundancy introduced at message level (therefore adding a checksum field to the message) helps to identify inside the MCU the source and the originator of the message exchange (that is which peripherals dialogs with the RAM or the CPU). This is implemented by adding an additional field to the protected message, with a coding convention for message type identification fixed at MCU level. That is, this method implements some "virtual channel" between the peripheral and the target.

Periodical software test for DMA – DMA_SM_3

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. The data packets are composed by not-trivial patterns (avoid the use of 0x0000, 0xFFFF values) and organized in order to allow the detection during the check of the following failures:

- Incomplete packed transfer,
- Errors in single transferred word,
- Wrong order in packed transmitted data.

The use of CRC packet protection is a way to simplify such checking operations.

3.6.7 CAN

Periodical read-back of configuration registers – CAN_SM_0

This diagnostic measure that is typically referred to as “Read Back Periodic by Software of Configuration Registers” executes a periodical check of the configuration registers of CAN peripheral against its expected value that is previously stored in the RAM and adequately updated after each configuration change. It mainly addresses the transient faults affecting the configuration registers, detecting bit flips in the registers due to transient faults. The register test is executed at least once per DTI in order to be able to claim the related diagnostic coverage.

Protocol error signals – CAN_SM_1

The CAN protocol error counters, which are entirely managed by the module at hardware level despite being conceived to detect network-related abnormal conditions, are able to contribute to the detection of the faults that lead to error messages generation.

The handling at application level of such error signals is a common technique in embedded applications. Their use is highly recommended.

Information redundancy techniques on messages, including End to End safing – CAN_SM_2

The CAN communications are protected by addressing both the permanent and transient faults with the redundant information technique that includes the End to End Safing.

For the implementation of redundant information, it is possible to adopt a different approach:

- Multiple sending of the same message, with comparison of the received results.
- Addition by the sender of a checksum field to the message to be verified by the receiver.

In case the checksum field approach is adopted, the selection of the algorithm for checksum computation shall ensure a similar protection against message corruption as the one ensured by a full redundancy.

For End to End Safing, additional measures are implemented:

- Additional field in payload allowing the unique identification of sender/receiver, and coherence check by receiver side.
- Timing monitoring of the message exchange (for example check the message arrival within the expected time window)
- Check of the message consistence using a message counter in the additional payload field and checking the right sequence of messages on the receiver side.

The use of a safe communication protocol such as ProfiSAFE is recommended for the correct implementation of this safety mechanism.

3.6.8 USART 1/2/3/4

Periodical read-back of configuration registers – UART_SM_0

This diagnostic measure that is typically referred to as “Read Back Periodic by Software of Configuration Registers” executes a periodical check of the configuration registers of USART against their expected value (previously stored in RAM and adequately updated after each configuration change). It mainly addresses transient faults affecting the configuration registers, detecting bit flips in the registers due to transient faults. The registers test is executed at least once per DTI in order to be able to claim the related diagnostic coverage.

Protocol error signals – UART_SM_1

The UART protocol errors signals (if used) despite being conceived to detect physical layer related abnormal conditions, are able to contribute to the detection to faults leading to error messages generation. For instance, option parity bit in data byte frame, overrun error.

The handling at application level of such error signals is a common technique in embedded applications. Their use is highly recommended.

Information redundancy techniques on messages – UART_SM_2

The redundant information technique is used to protect the USART communications by detecting both the permanent and transient faults. There are two different approaches to implement this technique:

- Multiple sending of the same message, with comparison of the received results.
- Addition by the sender of a checksum field to the message to be verified by the receiver.

In case the checksum field approach is adopted, the selection of the algorithm for checksum computation shall ensure a similar protection against message corruption as the one ensured by a full redundancy. Theoretic demonstrations on coverage capability are admitted – the use of CRC coding is anyway suggested.

The above-reported approaches are equivalent; an additional criteria for the selection of the approach is the availability of a quick hardware support on the MCU platform, and the evaluation of the computation capability of the external device with which the STM32F0 will exchange data.

Note that if the message is transferred by the DMA, the implementation of this safety mechanism takes into account also the overlap of DMA-related safety mechanism related to the message exchange inside the MCU and already declared as highly recommended.