AN4750
Application note

Handling of soft errors in STM32 applications

Introduction

Safety requirements on electronic devices increase permanently with massive expansion of electronic control into almost any human activity, this huge expansion requires processes compliant with specific standards. Primary target is to prevent human death, injury and environmental damage, but many other lower level factors must be considered besides safety during the design of an application, among them loss/ devaluation of a technological process, or of critical data or communications. Not less relevant is “simple” customer dissatisfaction in case of malfunction of a device under electronic control.

Developing processes satisfying national and international safety standards is a complex task. The main factors are field experience, market requirements, insurance coverage, globalization of trade and business. The standards are produced by specific legislative and executive bodies while specific recognized test houses take care about safety inspection and verification of all the required rules, processes and appliances.

STMicroelectronics supports two basic general level safety standards (ISO26262 functional safety standard is focused at specific automotive products):

- IEC 60730 & IEC 60335 – specific ones targeting household appliances under electric and electronic control, better known as “Class B” or “Class C” standard;
- IEC 61508 - more common industrial norm targeting safety integrity levels known as a ”SIL” or “ASIL” in the automotive industry.

For the first one, the main ST focus is Class B level support. The second one is a generic industrial “mother” norm which produces many derivative standards for different application fields like automotive, health, railway, avionic and many others. ST focuses applications targeting up to SIL3 level of this standard.

An application targeting safety can get a speed-up of software development when some pre-certified embedded self-test software modules (provided either by ST or by another third party) are implemented properly into the final firmware. Both built-in HW features of an implemented programmable electronic component and proper HW and SW methods applied during the application design phase decrease probability of hazardous events, thanks to efficient and early diagnostic of the component malfunction. Some applied HW methods can even increase the component reliability.

This application note provides an overview of errors and their possible impact on applications based on STM32 microcontrollers, focusing on so called “soft” errors handling, while highlighting proper software and hardware methods to eliminate them. For additional information and examples user can refer to following documents, available at *www.st.com*:

- AN3307 “Guidelines for obtaining IEC 60335 Class B certification for any STM32 application”
- AN4435 “Guidelines for obtaining UL/CSA/IEC 60335 Class B certification in any STM32 application”
- STM32 safety manuals.

Contents

List of tables

List of figures

1 Basic classification of failures

Safety standards require to take care of both systematic and random failures.

Systematic failures are considered as predictable and can be deal with by strict application of correct processes during the life time of both the electronic component and the implemented firmware or software.

In STMicroelectronics, the requirements are collected in specific internal quality documentation covering management of wide number activities like manufacturing, operational procedures, design (including verification and validation), materials handling, production testing, quality management, software development, documentation publishing, field feedback, tracking of issues etc. Compliance and application of these internal rules is subject to regular inspections and audits performed by worldwide recognized bodies.

On the other hand, limiting non predictable random failures requires the use of specific software and hardware design techniques and methods described in more details in this document.

From the product point of view we can recognize single point, latent or common cause type of failures. Single point failures have an immediate effect, while the latent ones are dormant and can aggregate with other faults. The common cause failures require special focus, as they can potentially make useless even quite complex safety structures, e.g when several components are affected contemporary. The structures most sensitive to common cause failures are usually shared systems like power rails, clock distribution tree or common control and synchronization signals. The heavier impact can be expected at extreme ambient temperatures, in low power modes or under specific emission, radiation or mechanical demands.

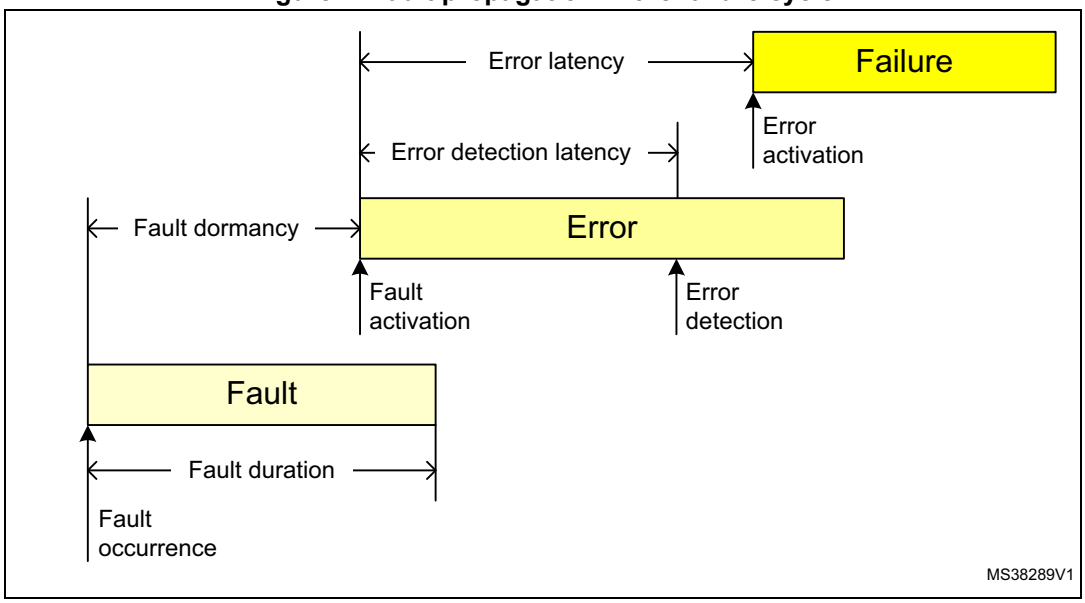
2 Random failures methodology

Not all the random failures result in hazardous events. In this case, they are considered as “safe” failures in relation to the given safety task if there is no potential latent correlation with some other error or fault. Basically, standards require monitoring dangerous failures related directly or indirectly to defined safety task with potential to escalate a dangerous situation at equipment under electronic control either directly or at whatever consequence.

Probability of a failure propagating into hazardous event is then decreasing when higher ratio of just the dangerous errors is discovered and prevented (detected) in time while minimizing number of dangerous errors staying dormant and undetected at the system. Propagation of a fault into a failure cycle is shown in *Figure 1*, where

- Fault status means occurrence of abnormal condition with potential to cause a system failure
- Error status is a discrepancy between the correct / true and the measured / observed / computed value (necessary condition for possible detection)
- Failure status happens when system is not able to perform required function (and then to prevent a hazardous event)

Figure 1. Fault propagation in the failure cycle



The detection and prevention action has to fit into overall Process Safety Time (PST) available in the application to prevent any hazardous event once a dangerous fault occurs. This interval must include all the possible delays and system reaction times (including sensors and actuators). When calculating PST all the delays need be considered, among them the so called Diagnostic test interval (DTI), i.e. the time necessary to perform a test with specific diagnostic coverage.

For quantification purpose, standards recognize a sure failure rate distribution scheme (sketched in *Figure 2*) and some quantification factors calculated upon this categorization, namely

- Safe Failure Fraction (SFF): the ratio between the entire covered failures rate including the dangerous detected failures and the total failure rate;
- Diagnostic Coverage (DC), the ratio between the dangerous detected failures rate and the entire dangerous failures rate.

SFF is computed from rates (approximate probabilities) of the different error classes and it is strongly dependent on DC, as can be seen from the equations below:

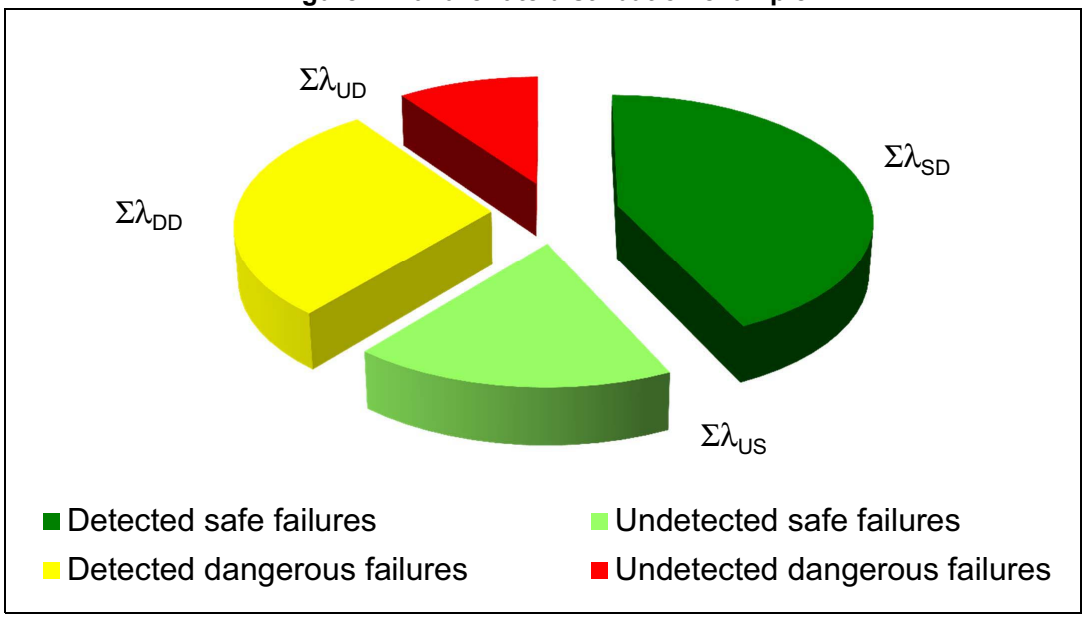
$SFF = (\sum\lambda_S + \sum\lambda_{DD}) / (\sum\lambda_S + \sum\lambda_D)$

$DC = \sum\lambda_{DD} / \sum\lambda_D$

where:

- $\sum\lambda_S$ is the total rate of safe failures (detected and undetected), equal to $\sum\lambda_{SD} + \sum\lambda_{SU}$
- $\sum\lambda_D$ is the total rate of dangerous failures (detected and undetected), or $\sum\lambda_{DD} + \sum\lambda_{DU}$
- $\sum\lambda_{DD}$ is the total rate of dangerous detected failures, equal to $\sum\lambda_D$ x DC
- $\sum\lambda_{DU}$ is the total rate of dangerous undetected failures, or $\sum\lambda_D$ x (1-DC)

Figure 2. Failure rate distribution example



The rates are calculated based on PHF (probability of Failure per Hour) when system operates at continuous or high demand mode (safety task is required permanently or routinely) or on PFD (probability of Failure on Demand) when system works at low demand mode (the safety task is not systematically required, e.g. an emergency button is pressed). The rates are strongly dependent on application and definition of the safety task, which implies that parts of hardware affect the safety.

Random failures can cause permanent (hard) or recoverable (soft) errors.

Random hard failures cause permanent physical destruction of the component. System is unable to continue in normal operation without any compensation of the raised damage. If no compensation is possible, the system has to be put into safe state until it's repaired (the

error is passivated). Typically, the signal is stuck-at or shorted with another one. Standard rate of hard failures can be expected in the range of 1-100 FIT/device.

Note: 1 FIT (Failure in Time) is the occurrence of a single failure in 10^9 hours of device operation.

Random soft failures are reversible, and some kind of recovery process is usually applicable. Typically flip flop gate, a register or SRAM cell can be recoverably affected by EMI or SER - cosmic or other flux (alpha or neutron particles or intermittent). This can cause latent weakness, noise, a cross-talk or spike of a signal. Exceptionally, high energy particles can cause permanent physical defects, too. Cosmic ray (neutron flux) depends on location (latitude) and elevation so it's partially predictable. The alpha flux is strongly dependent on several technology parameters (e.g. mold compounds, soldering material emitting isotopes) so it's difficult to estimate or calculate it (material contamination adds further variables).

Radiation level is expressed in counts per hour per cm^2 (cph/cm^2) and flux can range from 0.001 up to 100 cph/cm^2 in dependency of the contamination level of the used materials.

There is a strong dependence on technology used. Soft errors become more and more a problem with increasing component density and layout rules below 130 nm, especially on SRAM (usually other logic blocks of a microcontroller do not use so aggressive design rules. and not all the flips in these blocks result into an error).

Other the most negative factors are lower supply voltage, lower cell capacitance, higher clock speed, design complexity and frequency of use.

Even latitude and altitude influence the exposure level (the latter being more important), so the problem can be serious for electronic dedicated to avionic and cosmic systems.

On the other hand, short bit-lines, wafer thinning, radiation hardening techniques, and use of radiation protection materials are positively influencing factors. Thanks to these mitigation techniques especially the scaling trend stays nearly flat for standard latitude systems in the range of today's commonly used technologies (180-40 nm).

Soft failures happen much more frequent then the hard ones. Typically, they are measured in FIT per Mbit, and can be expected in the range of 200 to 2000 FIT/Mbit for a 90 nm technology, affecting mainly flip-flop gates (internal logic) and SRAM cells (memories, registers, buffers etc.). It could become a serious problem for devices featuring large memories. Customer can ask for FIT information to be obtained from ST for specific products.

Soft failures can be either latch-up or transient type. A detection-only method can be used to deal with them, it can be integrated with a compensation, i.e. a more complex procedure based on detection which allows the system to be resistant to errors and continue normal operation. Usually both these methods are based on a minimum level of redundancy present in the system.

Latch-up failures can be managed by both hardware and software, while transient failures need fast hardware methods just to be captured. Software tests cannot ensure full covering of these temporary and short life errors efficiently, as they are inherently slower and limited by the execution time (DTI). That is why the diagnostic coverage of software tests can never achieve the level of coverage provided by built-in dedicated hardware testing or compensation structures.

3 Random failures control techniques

In principle, random failures can be managed by detection, optionally followed by compensation.

3.1 Detection methods

When a redundancy is available in the system, the easiest method is a comparison of two independent calculation results, inputs, data etc. Valuable detection method can be based on acceptance test (e.g. valid interval, range or combination of values) too, when the functional result is compared with some consistency constraints. Another detection method is comparison of results from several components relative to each other or with specifically calculated expected result (e.g. orthogonal tests used e.g. for RAM testing).

When detection is the only applicable method, after a detection of a dangerous error, system should be stopped and placed into a fail-safe state or pass sure recovery process like reset, roll back or another specific check like replacement of the impacted component (to passivate the error). Exceptionally, if tolerable, system can continue to operate in a someway degraded mode, or provide wrong / not valid results, indicating to its environment that it has failed.

3.2 Compensation methods

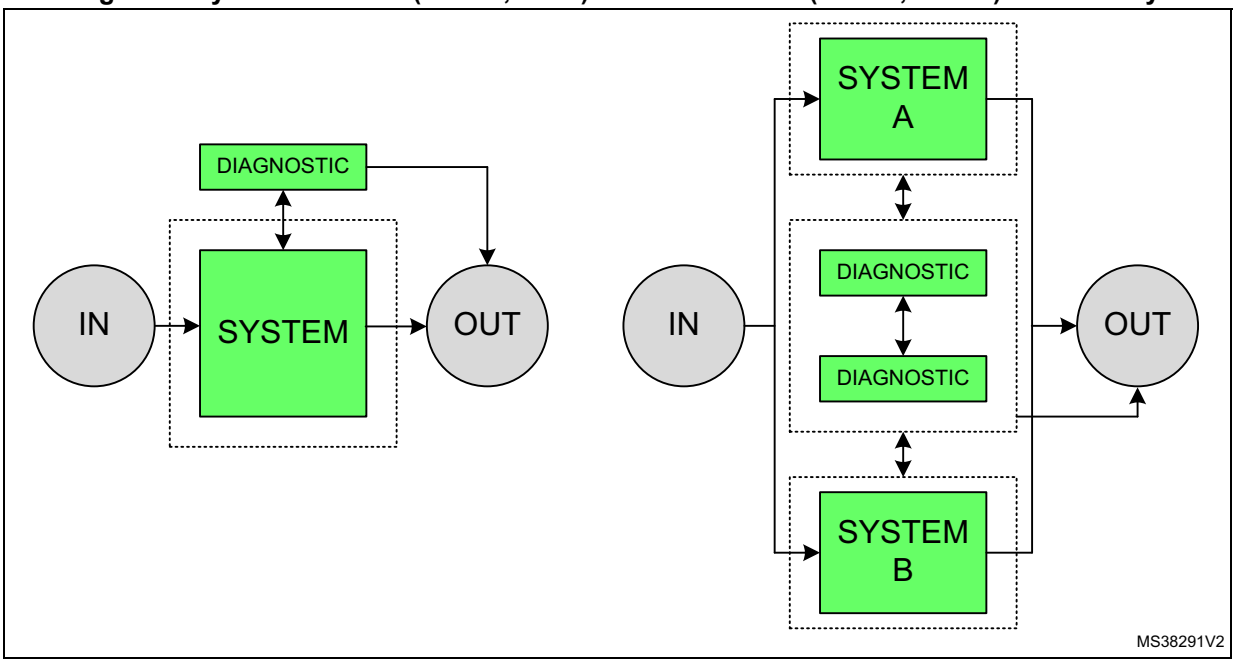
Compensation methods usually allow system to continue normal operation.

Compensation can be provided by error correction or masking. In correction mode, the correct result is calculated from an erroneous one using redundant information (e.g. ECC). In masking mode, the erroneous result is ignored and redundant resources are used if available (next channel, input, output or data). The correct result can be determined based on result of sure voting process recognizing just the bad and correct part of the component, or the corrupted and correct data.

Standards recognize Hard Fault Tolerance (HFT – the maximum number of errors which system can absorb while it continues normal operation).

Besides specific functional testing, redundancy is the essential diagnostic principle. Most detection and compensation techniques require a sure level of redundancy to be efficient. Compensation is considerably more demanding than detection, as not only a discrepancy but just the correct state has to be identified additionally. To do so, specific comparison and voting mechanisms and logic has to be applied.

Figure 3. Systems without (HFT=0, 1oo1) and with built-in (HFT=1, 1oo2d) redundancy



Note: NooM notation is used to describe number of outputs and redundant channels available in the system.
Here 1oo2d means one common output of two channels with built-in diagnostic (voting system).
Similarly, a 2oo2 system can be used when two independent actuators are used instead of the common one directly connected to System A and B outputs (e.g. two vents in series closing a pipe flow, or two independent circuit breakers connected in series interrupting a signal or a power line).

4 Applicable redundancy techniques

The required level of redundancy can be achieved by wide range of different methods and techniques. Such techniques can be achieved with both hardware and software, or a combination of them to maximize efficiency and diagnostic coverage. Added comparison of outputs and specific voting mechanisms between correct and failure path are integral parts of each redundancy system implementation when compensation is required.

4.1 Structural redundancy

Structural redundancy is based on parallel identical (symmetrical) structures performing the same task contemporary.

At system level, dual registers, memories, CPUs or even built-in entire microcontrollers can be used. At application level, some critical inputs or outputs can be doubled or tripled (both digital and analog), pair of cooperating microcontrollers can be used, doubled sensors, actuators, communication buses can be applied etc.

4.2 Functional redundancy

Functional redundancy is based on parallel diverse (asymmetrical) HW structures or different SW methods which are applied for a single task.

For example, CPU behavior can be monitored by specific built-in HW checker, or a 32-bit microcontroller performing the main task can be monitored by external simpler 8-bit one checking some interleaving results or guard some specific aspects of the main system behavior while comparing its inputs and outputs. Software can recalculate and recheck result of some complex flow point calculation by some rough simplified algorithm using e.g. integer calculation with fixed point or result from some complex logical decision scheme to be compared with simplified ones.

4.3 Temporal redundancy

Temporal redundancy is implemented when the same method is applied more times by the same HW or SW in different time slots.

Typical example is lock step dual core microcontrollers where each core performs the same task but in different phases of the core clock pulses. At application level, software can repeat the same calculation or logical task more times sequentially and compare the results.

4.4 Informational redundancy

Informational redundancy is the most common one when added information is implemented at data and evaluated for compliancy by HW or SW (e.g. implementation of parity bit, ECC, CRC check pattern, data protocoling, data copies etc.).

5 Applied methods

5.1 Vendor focus

From a safety point of view, a microcontroller is a complex programmable electronic component which has to satisfy specific requirements coming from the related standards.

When supporting safety for a microcontroller, a vendor considers the product as a component out of context as its final application and safety task are not known in advance. The effort is always to cover the component overall reliability and fulfill the overall budget of diagnostic coverage defined by the standards for a given safety integrity level targeted by the application.

Complex components like microcontrollers can be considered like partial components involved in the safety task. Each part contributes to the overall component safety budget with its own diagnostic coverage and weight.

An effective way to keep the required overall safety budget is to focus on crucial and generic modules of the microcontroller, especially those with the largest contribution and the ones more commonly used by most of the applications. Any small safety improvement on these modules brings the biggest gain for the overall safety budget of the component due to its big weight and importance, thus benefiting all applications.

For soft errors, the most effective section is that used by the SRAM, which takes significant part of the die entire area and it's used by each and every application. That is why this application note takes a specific care about methods applicable on volatile memories.

According to safety standards, definition of soft errors requires recoverable capability. From this point of view, we can consider as an example, data retention failures in programmable NVMs. After detection of any of these failures (e.g. by built in ECC or by a software check) a correct pattern can be programmed back into the memory (if it is known) during a proper recovery process driven by software.

Besides hardware safety features focused on memories (because of their significant contribution), many other supporting features are implemented in STM32 microcontrollers.

An overview is given in *Table 1*. More details can be found in dedicated reference and safety manuals.

5.2 User focus

Once the user includes microcontroller into an application design and the safety task is specified then the safety support can be spread out in much more efficient way to cover the specific sections involved in the required safety case.

Many efficient methods can be applied based on detailed knowledge of the application requirements, its design, the process and the equipment under control. Redundancy and knowledge of the system behavior are the crucial principles, to be applied either separately or in a common way. Inputs and outputs can be multiplied or checked by feedback loops, tested for logical states, values or expected responses in trends or time intervals. Communicated data can be secured by redundant information or even doubled. The processes can be monitored for their correct timing and flow order to check correct and complete sequences of control. Correct decisions can be taken based on comparison of results coming from redundant and independent flows, analysis, calculations or data.

5.3 Overview of methods applicable to handling soft errors

Table 1 gives an overview of direct or indirect methods applicable by both hardware and software to handling soft errors in STM32 microcontrollers, and their relation to IEC 60355 standard requirements. The overview is informative only. Most of the listed items and methods have an overlay, as they comply either directly or indirectly with different diagnostic purposes.

Table 1. Applicable methods to handle soft errors supported by STM32 HW features

HW feature or SW method	Addressed goals	Methods	IEC 60355 - Class B references
ARM Cortex® core exceptions / periodical core self-test	Permanent faults affecting the core functionality unpredictable software or system behavior or malfunction capture	Handling system exception interrupts, testing sequence of instructions manipulating with core registers	IEC 60335 R.1.1 – H.2.16.5 – H.2.16.6 – H.2.19.6
Checking application interrupt system	Expected and unexpected interrupt check	Checking missing or too frequent application interrupts	IEC 60335 R.1.2 – H.2.16.5 – H.2.18.10.4
Clock measurement and check	Detection of wrong or missing frequency, specific XTAL checks	Clock security system CSS control, independent watchdog, internal clock cross reference measurement	IEC 60335 R.1.3 – H.2.18.10.1 – H.2.18.10.4
ECC on Flash memory/ invariable memory self-test (self test supported by built-in CRC module)	Permanent faults affecting the system Flash memory cells and address decoder	ECC or CRC signature computation at block of memory performed regularly within DTI	IEC 60335 R.1.4.1 – H.2.19.3.1 – H.2.19.3.2 – H.2.19.8.2
Parity bit at SRAM / variable memory self-test	Permanent faults affecting the system SRAM memory cells and address decoder	Single bit per 8-bit word checked at the byte reading or periodic Walkpath test performed regularly within DTI	IEC 60335 R.1.4.2, 1.4.3 – H.2.19.6 – H.2.19.8.2
Control flow monitoring Independent & Window watchdogs	Proper software timing program counter loss of control or hang-up	Run-time control of the application software flow and application related timing, handling the watchdogs timeouts	IEC 60335 R.1.6.3 – H.2.18.10.2 – H.2.18.10.4 – H.2.18.18
Doubled GPIO digital inputs or outputs, loopback scheme for digital outputs	Permanent and transient faults on GPIO lines used as digital inputs and outputs	Application design	IEC 60335 R.1.7 – H.2.18.13
Analog inputs and outputs range/plausibility check	Permanent faults affecting ADC and multiplexers	Application design	IEC 60335 R.1.7.2 – H.2.18.13

Table 1. Applicable methods to handle soft errors supported by STM32 HW features (continued)

HW feature or SW method	Addressed goals	Methods	IEC 60355 - Class B references
Communication hardening HW & SW techniques, handling protocol errors	Errors in data transaction	Protocoling, doubled channels, CRC pattern, repeating messages	IEC 60335 R.1.6 – H.2.19.8.1 – H.2.19.4.1 – H.2.18.2.2 – H.2.18.14
Information redundancy for safety critical data Stack hardening techniques	Redundancy for transactions within the MCU, volatile memory keeping safety critical information	Verification of safety variables Plausibility of passed parameters	IEC 60335 R.1.5 – H.2.19.8.2
Power supply supervisors (POR, PDR, BOR) Internal temperature monitoring Option bytes protection SRAM protection Configuration lock / Periodical read back of configuration registers (including unused peripherals)(1)	Safe conditions to ensure correct function of all parts of the system	Interrupt to call emergency shutdown task or keeping the device under reset, verification of all the safety critical system configurations,	Not available

1. The reduction of the probability of cross-interference between peripherals that can potentially conflict on the same output pins (latent faults).

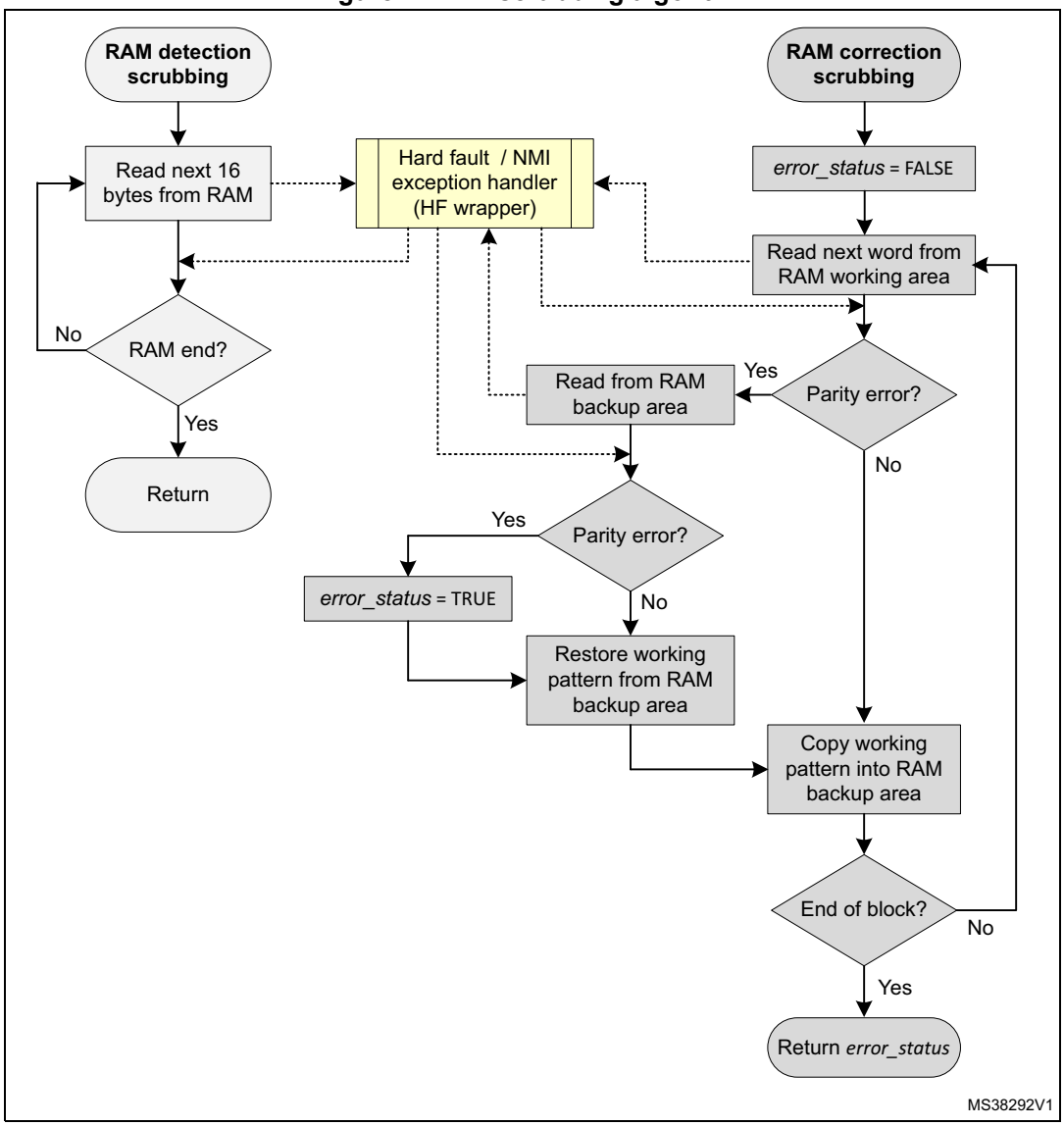
5.4 Specific hardware methods applied to SRAM

One of the most efficient methods is dual RAM structure with comparators. The RAM is divided in two blocks and all the data are mirrored. Each reading is preceded by hardware comparison of the values from both areas. A bit less efficient is protection of the words by EDC or ECC (error detection or correction codes). Standardly, a Hamming code is applied with distance greater than 3 (sure number of checking bits has to be added to each 32-bit word) which involves SEC-DED at minimum (a single error correction and double error detection within a single word).

Some STM32 microcontrollers use parity bit (one bit per 8-bit word) applied either to the whole or to a part of the RAM which is sufficient to recognize single bit or odd bits errors at the protected word. As not all the multiple bit errors are covered, the method is efficient when distributed design of bits collected at single word is applied (design multiplex factor – physical distance of two columns carrying two logically neighbored bits - should be kept greater than 4). In this case, probability of having contemporary multiply bits error during a single flux or EMI attack is relatively low.

Cumulative bit errors perform another case when two single bit errors can occur in the same word by two consequent attacks appearing at different times. That is why it is suggested to combine hardware parity check with regular scrubbing of not frequently used areas of the SRAM especially done by software which can prevent such cumulative errors by detecting single bit errors in time, an example is shown in *Figure 4*.

Figure 4. RAM scrubbing algorithm



In addition to preventing multiple bit errors, such a scrubbing method can be useful to detect latent errors (and then potential data discrepancy) in the SRAM in advance, e.g. before executing a safety critical procedure.

Simple example code performing such testing on Nucleo F030R8 board is associated to this application note. The source files of this example are available upon request, users should contact the local Field Application Engineer.

The example uses STM32F0 but it can be adapted easily to other products featuring hardware parity on SRAM. User can simply include the attached files into template directory

of Cube HAL (EWARM project). The scrubbing procedures are written as optimized code in IAR assembler (see *stm32f0xx_pchck_IAR.s* file). In *main.h* header file user can control configuration. Software can perform detection-only or add compensation method in dependency of conditional compilation parameters defined there. In case of compensation, the algorithm performs backup copy of the working data during RAM working area scrubbing. This method is suitable when data in the working area aren't changed frequently, so it cannot be applied to the area dedicated to stack. When a parity error is detected in the working part of RAM, correct data is restored from its backup copy.

The parity error detection raises hard fault exception. Its handling is a bit tricky, as the correct return address has to be restored by specific wrapper before returning from the handler together with the pointer of the tested address, else the test flow is interrupted and no more able to continue correctly.

The wrapper is simplified just for case of applied LDMIA or LDR instructions and for the memory access done by CPU (access done by DMA is a different case). The built-in LED on Nucleo board flashes quickly at normal run while it slows down when an error appears (PEF flag is found set).

User can split the area under check into slices (by applying proper parameters – intervals - when the scrubbing function is called), and ensure the control of the area in separated steps. When doing compensation in such case, user has to respect corresponding location of backup area because the procedure also executes an automatic refresh of the backup content (during the scrubbing).

To prevent scrubbing process failure the area under check has to be initialized by whatever content after power on. This is done by calling *RAM_parity_init():* this function overwrites the overall RAM content including the stack. It should be called from startup in the best case. When no initialization of the RAM area is included before the scrubbing procedures starts the HW parity error event can be simulated for some cells, as the whole RAM content is coincident.

5.5 Specific software methods applied to SRAM

Parity feature is not supported in all STM32 microcontrollers, or for all the available memory areas. Furthermore, except testing the memory content, standards requires to check correct function of data paths as well.

As there is no specific HW check implemented on busses for most of STM32 devices, it is suggested to perform Walkpath test either at application initialization or periodically. This test is carried out in a reasonable time, with limited performance overhead. Abraham or Galpat tests with considerably higher diagnostic coverage could be applied, however they need considerably higher number of steps and so they result in larger performance overhead and long execution period.

ST firmware uses destructive and transparent Walkpath test. The first one is used at start up test when the whole RAM is tested, while the second one is used at run time just on the area where critical part of data is stored, and it's performed step by step, testing small inter-overlaid blocks of the memory cells collecting subsequent addresses. Anyway, the Walkpath test ensures coverage for the SRAMs not supported by parity feature.

Additionally, redundancy principle can be combined here when just the critical part of data is double stored by complementary values (each critical value can be stored at a pair of variables placed at non adjacent memory addresses keeping the value and its complement). Software then must ensure that a a valid backup copy of the critical part of

data is kept. It must handle every write and read operation in this area with specific procedures performing correct data mirroring (in case of write) and checking consistency (when reading). If no Walkpath test is carried out during run time, some regular “scrubbing” (comparison of the areas content) can be performed in background, as it is done in the parity case to detect latent errors affecting the critical data in advance.

Stack hardening techniques can be applied too, when all critical parameters are passed redundantly and compared, and/or when their plausibility is checked at entry of called subroutines. Stack area boundaries can be regularly checked for corruption of specific patterns stored there to ensure that the stack pointer never crosses its dedicated area.

6 Conclusion

Soft errors can be critical, especially for microcontrollers featuring large memory areas, parity or CRC checks are valid hardware techniquest for their detection (but not correction). The most efficient hardware method for correction of protected critical data is ECC. To learn more about these and other implemented indirect supporting features, user should study reference and safety manuals of the specific product used in the application.

In addition to these detection and correction mechanisms (already ensuring a good protection level for the STM32 MCUs), STMicroelectronics implements mitigation techniques at design level to ensure a higher level of protection, along with demanding technology shrink.

A large number of additional hardware and software methods can be applied to handle soft errors in final applications. These methods are mostly based on sure level of redundancy, their complexity depends on the level of safety integrity required by the system, and on discriminating if detection only or correction is the required method to dealing with the errors.

Some methods can be applied in both cases, however the most effective diagnostic can be defined only with detailed knowledge of the application and definition of the safety task to target. It’s then possible to identify the specific parts of the microcontroller to be covered.

When starting the design a safety system, it is for sure valuable to discuss it with expert companies, or directly with the certification authorities, to avoid inefficient development efforts.

7 Revision history

Table 2. Document revision history

Date	Revision	Changes
01-Sep-2015	1	Initial release.
06-Oct-2015	2	Updated *Introduction*, *Section 2: Random failures methodology*, *Section 5.1: Vendor focus*, *Section 5.4: Specific hardware methods applied to SRAM*, *Section 5.5: Specific software methods applied to SRAM* and *Section 6: Conclusion*. Updated *Figure 2: Failure rate distribution example* and *Figure 3: Systems without (HFT=0, 1oo1) and with built-in (HFT=1, 1oo2d) redundancy*. Removed former *Figure 3: FIT rate vs. technology*. Updated *Table 1: Applicable methods to handle soft errors supported by STM32 HW features*.