10 SAFETY RELATED APPLICATION CONDITIONS

This section presents the Safety Related Application Conditions (SRAC).

The user **shall meet all the SRACs** listed in this section otherwise the certificate of the CLEARSY Safety Platform is not valid and none of the properties offered by the CLEARSY Safety Platform can be considered as granted.

For each SRAC, a single section will be used and the structure will be the following:

- Statement: official text of the SRAC
- Verification proposal: Example of how to check if the SRAC is fulfilled or not
- Associated risk: Explanation of the associated hazard if the SRAC is not fulfilled.

The verification proposal has to be understood as a proposal. The end user and the application team is, of course, free of using any means and methods to verify and validate that the SRAC are respected. The proposal is only provided to help the end user to determine how the compliance to the SRACs can be claimed.

In order to ease the browsing of these Safety Related Application Conditions, the following figure shows the breakdown structure of the SRACs between the different fields.

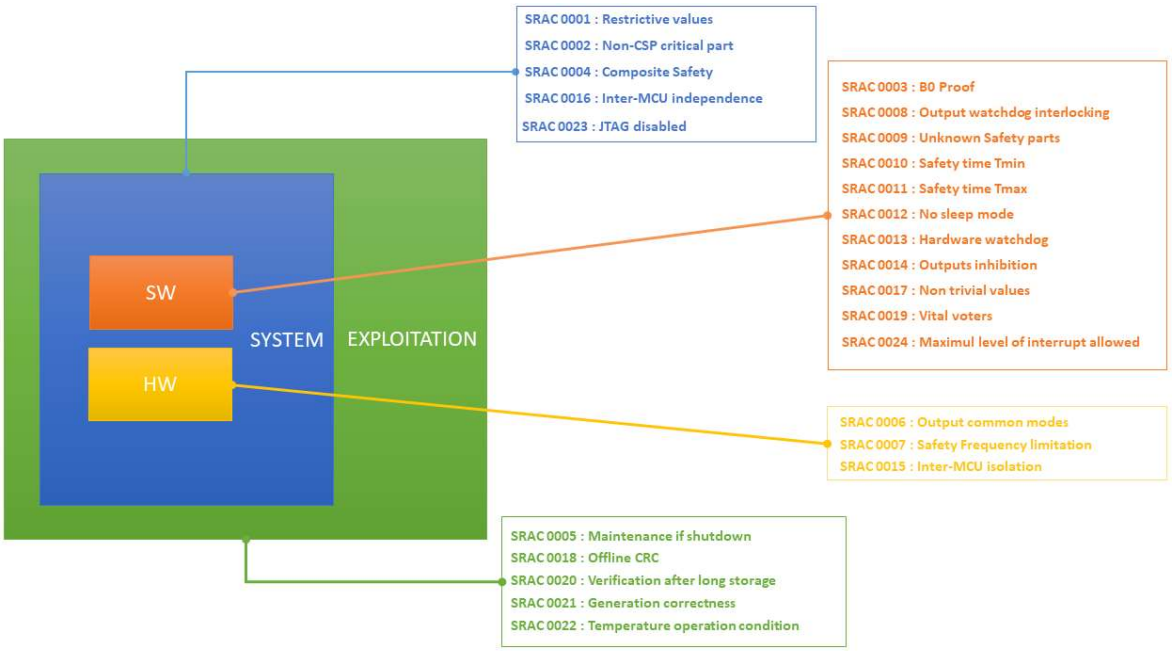

Figure 9 : SRAC breakdown structure

The following lookup table gives the matching between the number of the Safety Related Application Conditions, their full title, and their searchable short name in camel case. This table is intended to help the reader to browse the various documents more efficiently.

ID	TEXTUAL ID	FULL TEXT
0001	RestrictiveValues	Restrictive values
0002	NonCspCriticalParts	Non-CSP critical parts
0003	B0proof	B0 proof
0004	CompositeSafety	Composite safety
0005	MaintenanceIfShutdown	Maintenance if shutdown
0006	CommonModes	Output common modes
0007	SafetyFrequencies	Safety Frequency limitation
0008	OutputWatchdogInterlocking	Output watchdog interlocking
0009	UnkownSafetyParts	Unknown safety parts
0010	SafetyTimeTmin	Safety time Tmin
0011	SafetyTimeTmax	Safety time Tmax
0012	NoSleepMode	No sleep mode
0013	HardWatchdog	Hardware watchdog
0014	OutputInhibition	Output inhibition
0015	InterµCIsolation	Inter-MCU isolation
0016	InterµCIndependence	Inter-MCU independence
0017	NonTrivialValues	Non trivial values
0018	OfflineCRC	Offline CRC
0019	VitalVoters	Vital voters
0020	VerifAfterStorage	Verification after long storage
0021	GenerationCorrectness	Generation correctness
0022	TemperatureCondition	Temperature operation conditions
0023	JTAGDisabled	JTAG disabled
0024	InterruptLevel	Maximum level of interrupt allowed

10.1 SRAC 0001: RESTRICTIVE VALUES

10.1.1 Statement

> Given a product using the CSP for one or several features, it is assumed that the designer and the safety team define a property P that sums up all the product's safety (P is usually a conjunction of conditions). They shall ensure that P holds when all outputs (hardware, messages, …) are restrictive. This implies to define a restrictive state for each output.

10.1.2 Verification method proposal

A critical review of the safety demonstration of the final application could be performed. During this review the safety related guaranteed properties are clearly identified. Then it is sufficient to provide a short explanation, for each of the safety properties, of the reasons why the property is still respected when the CSP outputs are restrictive. Then the review should conclude that if the properties hold then no danger can occur.

10.1.3 Associated risk

In order to achieve the safety properties offered, the CLEARSY Safety Platform needs to define a "*safe state*" (as defined in the EN50129:2018 standard). For the CLEARSY Safety Platform the safe state corresponds to a state where all the outputs are restrictive (no valid message is emitted, no output safety frequency signal is produced, …). When a first failure has occurred on the CLEARSY Safety platform, the built-in mechanisms ensure quickly reaching this safe state. If the safety property of the user is not guaranteed when the system is in safe state then the safety target is impossible to achieve (by definition) and any demonstration will be doomed to fail.

In this context, an unsound safety demonstration, without an explicit P or without restrictive state definitions, may lead for instance in cases where the safety is not ensured if the CSP shuts down. In the worst case, this can lead to death or injury.

10.2 SRAC 0002: NON-CSP CRITICAL PARTS

10.2.1 Statement

Except the following errors:

- Any failure of the B0 replicated software (compiled accordingly to the validated compilation toolchain from CLEARSY) to process in accordance to its theoretical model;
- short circuits between the MCU1 and MCU2 isolation groups on the CS0 board

the end user shall consider all potential failures in its own applicative safety case. Especially any software developed and run outside the proposed framework comes with no safety guarantee. In addition, no guarantees are provided against internal peripheral failures that may occur in the application context of the end user.

Note: The term *theoretical model* used in the text above has to be understood as that the process described by the software (for example *var := var+2*) will be guaranteed to be executed as imagined (i.e. variable *var* receives the value of *var+2*) whatever the failures. In other words, the order of the instruction sequences and the operators will behave has expected by their definition and the instructions will be executed in correct order. This is guaranteed whatever failures happen on the microcontroller (or the other microcontroller shuts down).

Granted that all the SRACs are fulfilled, the user safety case can rely on the fact that the following failure modes are covered by the CLEARSY Safety Platform built-in mechanisms and thus do not need to be taken into account in the safety case of the user application:

- Random transient upsets in the duplicated variables (i.e. the ones prefixed with *rv_* in the user vital software)
- Permanent or transient modifications of Special Function Registers which have been monitored as explained in section Monitoring SFRs
- Permanent or transient modifications of the CP0 register configuration

- Modifications of the flash memory contents (this guarantee is applicable to the full FLASH memory and is not limited to the memory region storing the vital software).
- Damage on the Random Access Memory (RAM)
- Compilation errors on the vital software (the one written in B0)
- Perturbation on the clock frequency and timing variables (provided that the time is used as prescribed in the section Using time variable)
- Incorrect software uploaded on the target
- Damage on the Arithmetic and Logic Unit

Except these failure modes, the user safety case shall consider all the failure modes relevant to the user's own application.

Caution: The variables that are not prefixed by *rv_* (see Defining replicated variable section) come with no guarantee against corruption. All these variables are not crosscompared and thus not guaranteed against random perturbations.

Note: Pay attention that all peripherals of the PIC32MX MCU used in the CLEARSY Safety Computer (CS0) can behave wrongly and are not covered by the generic SIL4 certificate of the CLEARSY Safety Platform. Thus the user may use these peripheral at is own risk and include the relevant failure modes if any to his/her own safety demonstration. A short list of peripherals (not covered by the generic certificate) is provided below:

Not covered peripheral (non-limiting list):

- UART
- I2C
- SPI
- GPIO
- TIMER
- PMP
- DMA
- USB
- RMII
- Analog comparator
- ADC
- RTCC
- Input capture / PWM

10.2.2 Verification method proposal

A critical review of the user safety case could be performed in order to validate that no assumptions stronger than the ones given above are done.

10.2.3 Associated risk

This SRAC is motivated by the experience of misused safety platforms with which the end user wrongly trusts properties that the platforms do not offer. This situation results in unsafe design due to a lack of understanding of the interface requirements between platform and user application. For example death or injury may be caused by the product if using a non-guaranteed feature (e.g., using peripheral devices in CSP PIC32 as safety devices).

10.3 SRAC 0003: B0 PROOF

10.3.1 Statement

The CLEARSY Safety Platform does not provide any mechanism against the systematic design error of the vital duplicated software (the one written in B0). The end user is sole responsible for managing any bug inside the B0 written software. The end user shall demonstrate that the vital software written in B0 is free of any systematic error that may lead to a hazard. In addition, the end user shall validate the units and the value of the associated constants (i.e. data validation is in the user's scope).

Note: The CLEARSY Safety Platform ensures by design that the value defined in the B0 constants is used unchanged in the final software. Nevertheless you shall pay extra attention to ensure that these constants are correctly named, valuated and correct in terms of units.

Caution: If you do not develop your application with the full B formal method, then you have to take care that the INITIALISATION clauses of your B0 components are called recursively following the IMPORTS link starting with the implementation of the csp_user_watchdog machine. In other words, to ensure that an INITIALISATION is called, its component has to be transitively imported by the csp_user_watchdog implementation.

10.3.2 Verification method proposal

CLEARSY strongly encourages to use the B formal method approach to prove that the B0 software is bug free. Other methods may be acceptable but have to be developed and assessed by the end user.

Concerning data validation, it is worth noting that things like correct output assignments, ranges and units may be difficult to assert from the source code only. In this case, type tests could be performed to validate the units and the proper valuation of the constants.

10.3.3 Associated risk

If the user develops a B0 software with systematic errors, the CLEARSY Safety Platform will only ensure its correct execution, but the errors can still cause hazardous states. The correctness of the vital software in term of systematic programming error and correctness of the specification is under the responsibility of the end user. Failures here can lead to death or injury.

10.4 SRAC 0004: COMPOSITE SAFETY

10.4.1 Statement

The end user shall consider that one of the 2 microcontrollers in any CS0 board may encounter a dangerous random failure (i.e. able to cause this microcontroller to wrongly authorize) with a hazard rate of 10^{-7} failure per hour. Whatever the failure, the CSP library and safety mechanisms guarantee that the detection is performed in less than 1 hour. Especially, the healthy microcontroller (i.e. the one not hit by the failure) reaches a safe state with all its outputs in restrictive state in less than 1 hour after the occurrence of the first failure.

10.4.2 Verification method proposal

The user's safety case shall include such failures, hitting one microcontroller and causing it to perform something wrong (whatever it is). For instance, architectures where one microcontroller receives a vital input and passes it to the other one are doomed to being unsafe, as this microcontroller could fake a permissive input. The user shall succeed to prove the safety for his/her target system including such failures, occurring at the indicated maximum failure rate on each microcontroller and causing the affected microcontroller to have any wrong behavior.

10.4.3 Associated risk

Using an architecture where the fault of one CSP microcontroller alone may result in global unsafe behaviors (even with hidden cases) leads to an incorrect safety case regarding the CLEARSY Safety Platform's Safety Related Application Conditions and to potentially fatal hazards. For example, one could wrongly design a system where one CSP microcontroller receives a safety authorization message, decodes it and passes the conclusion to the other one. Even if all outputs are correctly obtained by AND from both microcontrollers, on rare cases (above failure rate) the first one may fail so to have a wrong conclusion about the incoming message while still being able to drive its half of the outputs. The next one is then contaminated and wrong outputs are produced. Such failures are rare, but indeed one microcontroller alone is never safe enough.
The duration of 1 hour is related to the maximum required time for the internal mechanism to detect and react to a failure in order to reach a safe state. This duration cannot be reduced whatever the context of use of the final application. The hazard rate of 10^{-7} failure per hour is a upper bond of the microcontroller failure rate.

10.5 SRAC 0005: MAINTENANCE IF SHUTDOWN

10.5.1 Statement

The designer of the final product shall define a maximum neutralization time T before which the maintenance or the neutralization will occur after the first failure occurrence (and the shutdown of the system). The user shall only rely on restrictive outputs to trigger the neutralization. The 1 hour CSP detection time must be included in T.

The neutralization consists in ensuring by any means that whatever failure accumulation (even with a CSP shutdown) the system cannot trigger a hazard. The term *maintenance* has to be understood (in this SRAC) as a corrective operation which removes the fault permanently.

Tip: A good rule of thumb consists in using one vital output as a health signal of the designed product. Then a falling edge on this interface indicates the start time of the T timeout. In operation, the maintenance supervision device should only poll this output and schedule neutralization/maintenance activity accordingly.

10.5.2 Verification method proposal

To ensure that this Safety Related Application Condition is fulfilled, a critical review of the final application safety case should be performed and indicate what is the T duration (Safe Down Time) selected and check that the THR is calculated accordingly. In addition, a dedicated maintenance procedure should be identified to ensure that the system neutralization is performed in less than T. It shall be verified that this maintenance is triggered using restrictive outputs only.

10.5.3 Associated risk

The non-fulfillment of the Safety Related Application condition may invalidate the final application safety case due to wrong probability assumption. On the field it may also lead to a system on which failures can accumulate for a too long period, leading to hazardous events.

For example, trusting a non-vital alarm system to trigger the maintenance could infinitely delay its start, leaving the system shut down but powered for a period longer than T. Then additional failures can accumulate causing some wrong permissive outputs, injury or death. Not accounting the 1 hour detection time in the return to safe state delay T can lead to wrong THR calculation by the user. For example, for an application where T has been selected equal to 1 hour (i.e. maintenance is performed in less than 1 hour after detection) the THR error will be of 50% if the CLEARSY Safety Platform detection time is omitted from theoretical computation.

10.6 SRAC 0006: OUTPUT COMMON MODES

10.6.1 Statement

The safety analysis of the final product that includes the CLEARSY Safety Platform shall consider the risk of a simultaneous freeze of the state of the pins of the two microcontrollers (the pins can be stuck at low logic level or high logic level). In addition, the failure mode of pins oscillating at a random variable frequency shall also be considered. Note that the common failure mode of the two microcontrollers having pins oscillating at a precise frequency should be considered with the limitation of the SRAC_0007.

10.6.2 Verification method proposal

The user can check by critical design review that its vital outputs are driven by inimitable signals. For instance, signed network telegrams with sequence numbers can be considered as inimitable signals (challenge/response mechanism). Generation of safety frequencies can also be considered as an inimitable signal (with limitations exposed in SRAC 0007)

Caution: Outputs driven by simple ON/OFF signal will not satisfy this Safety Related Application Condition because in case of simultaneous freeze of the two processors (which may occur in case of failure or network flooding, etc.) this will lead to an everlasting permissive output.

10.6.3 Associated risk

The risk is to maintain a permissive output when the CPU is frozen in a given state or if a short circuit between pins pulls a given pin to 0 or 1.

10.7 SRAC 0007: SAFETY FREQUENCY LIMITATION

10.7.1 Statement

The associated hardware which takes a safety frequency as a safe input, shall have a pass band filter with a bandwidth narrower than 10% of the passing frequency (for instance for a 65kHz safety frequency the bandwidth of the filter shall be included in the range [58,5kHz;71,5kHz]. In addition the following frequencies shall not be used or included in the bandwidth previously described (due to internal pre-existing oscillators): 8kHz, 400kHz, 8MHz, 12MHz, 20MHz, 80MHz, 32kHz. The switching frequency of DC/DC converters or external hardware oscillators used on the user's board or the frequency spontaneously generated by non vital layers shall also not be equal or included in the bandwidth of any safety frequencies.

10.7.2 Verification method proposal

A possible method for checking compliance to this Safety Related Application Condition is to perform the following process: 1. List all the frequencies generated by hardware and the non vital software. 1. List all the safety frequencies used in the system and compute their associated bandwidth. 1. Check that the intersection between the two previously defined sets is empty.

10.7.3 Associated risk

If a short circuit inside the CPU or a software issue copies an already existing frequency signal on a safety output interface then a falsely permissive output may be produced. Note that this SRAC is only a safeguard to ensure that the selected safety frequency remains an inimitable value, that a failure cannot generate by chance.

10.8 SRAC 0008: OUTPUT WATCHDOG INTERLOCKING

10.8.1 Statement

Before performing any actual permissive output (i.e. providing a permissive output or sending permissive messages) the vital software shall call the function *watchdogLiveness_get_watchdogIsAlive*. This call ensures that the timing mechanisms are still properly operating and thus the safety built-in mechanisms are running. The output shall be set to permissive only if this function returns STRUE.

Tip: For output based on generation of a safety frequency, it is not necessary to perform the call to *watchdogLiveness_get_watchdogIsAlive* for every rising/falling edges of the signal. In this case the minimum response time of the output (i.e. the minimum duration of the frequency signal required for turning the output into permissive) has to be determined and the call to *watchdogLiveness_get_watchdogIsAlive* has to be performed faster than this duration.

Note: Details about the call to *watchdogLiveness_get_watchdogIsAlive* are provided in section Watchdog Liveness Check

10.8.2 Verification method proposal

The recommended verification method for this Safety Related Application Condition consists in performing a critical code review in order to ensure that any application of a permissive output is prefixed by the call to *watchdogLiveness_get_watchdogIsAlive* (or justified in the case of a safety frequency signal).

10.8.3 Associated risk

If output code is not protected with watchdogIsAlive, it could find itself running alone (after some perturbations) in a permissive state. However this should accumulate on both µCs to actually drive a wrong permissive output. A typical hazardous case is when the source code providing energy to the output is executed from an interrupt service routine (ISR). Under failure scenario it cannot be excluded that the microcontroller keeps looping on the ISR source code and does not execute anything else. In such a failure case, the call to *watchdogLiveness_get_watchdogIsAlive* will shut down the system and prevent the output from remaining permissive infinitely.

10.9 SRAC 0009: UNKNOWN SAFETY PARTS

10.9.1 Statement

The non-vital parts shall not have information about the vital parts implemented in B0 (except the entry points of the software). Especially, the provided library shall not be reversed engineered to access directly the replicated variables or to extract any of the safety keys (for instance precomputed tables).

Note: the user should consider that the B0 entry points can be called by error.

10.9.2 Verification method proposal

To meet this Safety Related Application Condition, you should verify two points:

1. Your custom application does not modify the vital data of the CLEARSY Safety Platform library. This point is easily fulfilled, because CLEARSY does not provide any means for disassembling the library. Thus, as long as you don't perform undocumented actions this point is met.
2. Your custom non-vital software does not access the safety key of your own application. To verify this point, you should list all the safety keys of your application and check through a critical code review that the safety keys of your vital custom-application are not accessed by the non-vital software.

10.9.3 Associated risk

The main risk is to perform permissive outputs within your application or corrupt the safety built-in mechanisms with a non-vital software that has too much knowledge of the structure of the vital software. Such situation may lead to death or injury.

A typical example corresponds to a situation where for some reason the non-vital software contains, for debugging purposes, some keys to produce permissive outputs. In this case, a simple failure may exercise this debug software and produce falsely permissive output.

10.10 SRAC 0010: SAFETY TIME TMIN

10.10.1 Statement

In this SRAC, we name *clock* the value of the *time* as viewed by an external observer and time the representation of elapsed duration since startup as measured by the software (i.e. number of ticks elapsed). We note Δ=105ppm the maximum drift of the internal time counter.

In order to retrieve the current value of the elapsed *time* since startup the software designer can call the function *nr_f_getTime()* which returns a uint64_t corresponding to a non-vital estimate of the *time* since startup expressed in 125µs steps (δ_c=125µs).

Before using this value in the B0-implemented vital software, the designer shall check the accuracy of the value provided by the non-vital layer. This check has to be done by calling the function *vitalClock_checkAccuracy(time,accuracy)* which returns true only if the *time* from the non-vital software is within the range *[now*-accuracy;*now]* where *now* is referring to the current time as known by the vital software.

This SRAC could be used when you want to ensure in safety that a minimum duration t_{min} has elapsed between two events (called E1 and E2). Assuming that the following process is applied:

Process:

- The value of the time, obtained by a call to *nr_f_getTime* performed after E1, is T1 and has been successfully validated with a call to *vitalClock_checkAccuracy(T1, A1)* returning TRUE
- Another call to *nr_f_getTime*, performed after the previous one, returns a time value equal to T2; and the validity of this time T2 is confirmed by a call to *vital-Clock_checkAccuracy(T2, A2)* performed before E2 which returns TRUE.

as soon as that the following inequality is established:

- T1 + A1 + $t_{min}/(\delta_c(1-\Delta)) \leq$ T2

then

- a duration of at least t_{min} has elapsed on the clock (of the external observer) between events E1 and E2.

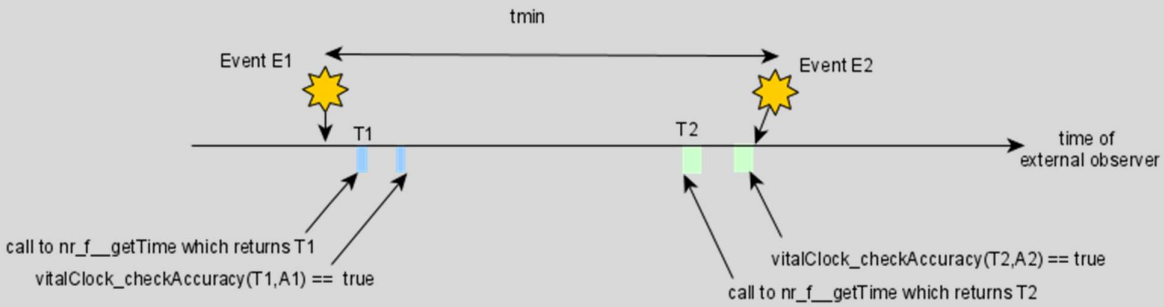

Figure 10 : Process for ensuring a Tmin

Thanks to this equation the designer can define a constant for the minimum clock duration t_{min} he has to respect, and perform periodic polls of the current time (which will provide T2 values) and checks when the above equation becomes true ensuring that the real clock has evolved from a quantity greater than t_{min}.

Tip: A corollary to the method proposed in the SRAC may also help the designer. Between a call to *nr_f_getTime* returning T1 (performed after event E), confirmed successfully by a call to *vitalClock_checkAccuracy(T1, A1)* returning TRUE and a call to *nr_f_getTime* returning T2, confirmed successfully by a call to *vital-Clock_checkAccuracy(T2, A2)* returning TRUE, if T1+A1 is strictly smaller than T2 then the minimum amout of real clock that elapsed between the first call to *nr_f_getTime* and the return of the call to *vitalClock_checkAccuracy(T2, A2)* is equal to (T2-(T1+A1)).δ_c.(1-Δ)

Tip: If the user already has his own source for timing measurement, he is fully allowed to use it for his application. In fact the source of the time is not really critical as long as the value provided to the vital software is not too old (i.e. older than now - accuracy), fulfills this Safety Related Application Condition and passes the *vitalClock_checkAccuracy* test.

Note: The call to the *nr_f_getTime* primitive before the call to each replicated function is required because of the vital variable comparison. Indeed the two vital software replicates need to have exactly the same input data otherwise they may reach different state after execution (due to the execution time of the first replicate). In such cases, the system will shutdown leading to a not available design.

10.10.2 Verification method proposal

In order to comply with this Safety Related Application Condition CLEARSY recommends to:

- Perform a critical review of the B0 software to ensure that in all places where the time is used for vital actions, the accuracy of the clock value is checked.
- Perform a critical verification of the valuated constant to check that the accuracy is taken into consideration in the final timing constant.

10.10.3 Associated risk

The value provided by the *nr_f_getTime()* function is non-vital. Thus no assumption can be made on the returned value. For instance we can imagine a case where the first call the function return t1=10 and a few seconds later the next call return t2=5. In this scenario, the call to *vitalClock_checkAccuracy* will return false at least for the second call. If the user does not comply to the Safety Related Application Condition, he may compute t2-t1 in order to know the elapsed time between two events which will lead to -5 which is not a valid duration (should be strictly positive). Such a negative value may corrupt the execution of the vital-software. Other hazardous scenarios can be imagined by modifying arbitrarily the value returned for t1 and t2.

10.11 SRAC 0011: SAFETY TIME TMAX

10.11.1 Statement

The CLEARSY Safety Platform library provides a mechanism for ensuring that a given piece of code is executed before a given deadline. The end user is free of using this mechanism or not (i.e. he is allowed to develop his own mechanism and use it at his/her own risk). If the user uses the built-in deadline mechanism, then he shall rely only on the two following properties:

- The function *user_watchdogTimer* is guaranteed to be called on each increment of the clock
- The function *watchdogLiveness_testDeadline* will return only if the deadline provided as an argument of the function is in the future compared to the current clock. In any other case this function will trigger a shutdown.

A sufficient implementation for ensuring a deadline should consist of defining a deadline variable that is updated with the new current deadline each time the function to be monitored is executed, and to systematically call the *watchdogLiveness_testDeadline* each time the function *user_watchdogTimer* is executed. By doing so, the deadline variable is checked every clock increment and a shutdown will be triggered if the current clock overruns the saved deadline.

With this implementation, the user can consider that from a system point of view the product will shutdown at the latest at T_{max} if the deadline variable is not refreshed.

T_{max} is defined by the following equation: $T_{max} = T_0 + (T_{inc}+1).(1+\Delta).\delta_c$ with the quantity defined in seconds. In the previous equation:

- T_{max} corresponds to the effective deadline date at which the system will be guaranteed to be in shutdown (i.e. the measured time from the point of view of an external observer)
- T_0 corresponds to real time assessed just before setting the new value of deadline variable. We take the notation T_{now} for the value of the software clock at T_0. The value of T_0 could be retrieved with *nr_f_getTime()* and shall be confirmed to be accurate enough by a call to *vitalClock_checkAccuracy* which will return TRUE only if the provided time is in the range *[now-accuracy;now].*
- The new value of the deadline variable is set to $T_{now} + T_{inc}$, thus if the software overruns the value $T_{now} + T_{inc}$ the system will be shutdown. Note that T_{inc} is expressed in ticks (125µs) and shall be positive or null.
- δ_c=125µs (i.e. clock tick increment)
- Δ = 105 ppm, is for taking the influence of the variation of the quartz due to aging, precision and thermal stress.

Thanks to this mechanism, the user obtains the guarantee that the deadline variable has been refreshed more often than $(T_{inc}+1).(1+\Delta).\delta_c$, implying guarantees about any code in the control flow in between.

Note: In case you do not retrieve the current time (through *nr_f_getTime* and *vitalClock_checkAccuracy*), the value of T_{now} will be equal to 0 (startup of the system). This use case is obviously not prohibited but will lead to bad precision due to the accumulation of the inherent clock drift of the system over long periods.

Tip: More details on how to use the deadline mechanism for guaranteeing maximum timing can be found in the dedicated section: Register a deadline variable

Note: With the previously described implementation, the only way for the user to prevent a shutdown is to push the deadline to a greater date or to stop checking the deadline variable during runtime.

Note: If the user chooses to work with a B model formally proven, the structure of the model of the CLEARSY Safety Platform library enforces that the implementation of the deadline matches the requirement expressed in this Safety Related Application Conditions. The formulation of the requirement is a little bit complex, because the user is free to validate its B0 implementation by its own means, but in this case the formal proof does not ensure that the test deadline is properly called, that is why the SRAC requires to detail what happens under the hood with the *watchdogLiveness_testDeadline* function.

Note: It is possible to apply this mechanism multiple times in order to have several watchdog variables.

10.11.2 Verification method proposal

To comply with this Safety Related Application Condition, CLEARSY recommends to ensure by critical review that the timing execution guarantees taken into consideration in the safety demonstration of the user application are implemented with the deadline mechanism and with correctly valuated values.

10.11.3 Associated risk

The risk associated to this Safety Related Application Condition is to maintain a permissive output longer than the maximum allowed time promised by the safety properties of the developed system.

10.12 SRAC 0012: NO SLEEP MODE

10.12.1 Statement

The user application code shall never voluntarily set or modify any of the bits (and especially the SLPEN bit) in the OSCCON register. In addition, it is forbidden to use peripherals running from the LPRC oscillator or from an external oscillator that may raise a waking interrupt of the CPU. This enforces to never use the power-saving modes of the PIC32MX CPU and also prevents the user application from modifying the clocks settings.

Note: the user is allowed to read the content of this register in order to compute internal baudrate for its own application.

10.12.2 Verification method proposal

In order to comply with this Safety Related Application Condition CLEARSY recommends to perform a simple search for the string “OSCCON”, “SLPEN” in the full source code of the application and to check that the register is never written. You can also list the peripherals running from LPRC or an external clock source and check if they can raise an interrupt or not.

10.12.3 Associated risk

A possible associated risk is if both processors turn into sleep mode and resume afterwards without any trace. Then the sleep delay is not detectable. This could lead to wrong delays, wrong permissive outputs. However, random activation of such sleep modes cannot plausibly cause the hazard.

10.13 SRAC 0013: HARDWARE WATCHDOG

10.13.1 Statement

The user shall not voluntarily disable the hardware watchdog (write to WTDCON register ON bit) nor refresh the hardware watchdog (write to the WDTCLR bit of the WDTCON register)

10.13.2 Verification method proposal

To comply with this Safety Related Application Condition, CLEARSY recommends to perform a simple search for the string “WTDCON”, “WDTCLR” in the full source code of the application and check that this bit is never written.

10.13.3 Associated risk

If the hardware (non-vital) watchdog is voluntarily disabled, the plausibility of undetected temporary safety watchdog stalls raises. However, random failures of the hardware watchdog cannot plausibly cause the hazard.

10.14 SRAC 0014: OUTPUTS INHIBITION

10.14.1 Statement

In all software parts establishing / maintaining permissive outputs, the user shall stop any permissive value if the provided variable OutputInhibition is set.

Caution: This verification shall be performed all the time even for the safety frequency generation (i.e. on every edges)

Note: The state of the *OutputInhibition* can be retrieved by a call to the operation r_f__get_CS0_outputInhibition defined in the file *csp_cs0Lib.mch*.

10.14.2 Verification method proposal

To comply with this Safety Related Application Condition, CLEARSY recommends to perform a critical review to verify that the check of the variable OutputInhibition is performed before any application of a permissive output.

10.14.3 Associated risk

The associated risk is to maintain a permissive output whereas some of the internal built-in vital health test has failed or has not been performed (startup case).

10.15 SRAC 0015 : INTER-MCU ISOLATION

10.15.1 Statement

> Whatever the hardware in which the CS0 is included, the isolation at 2,4kV between the two MCUs shall be maintained and demonstrated.

10.15.2 Verification method proposal

This constraint should be added to specification of the layout of the final application in order to define the proper clearance and creepage distance on the printed circuit board. In addition, it is strongly advised to perform a systematic dielectric test on manufactured unit in order to confirm that the level is reached.

10.15.3 Associated risk

If the isolation between the two microcontrollers is defective then a common mode failure (especially on the power supply) may lead to an hazard (violation of composite safety principle).

10.16 SRAC 0016 : INTER-MCU INDEPENDENCE

10.16.1 Statement

> The primary independence between µC1/µC2 shall be kept by the user's design, according to EN 50129:2018 B.3.2.
>
> Mandatory provisions so that external influences do not break µC1/µC2 independence are not limited to the CS0 board design: the user's design is also concerned.

10.16.2 Verification method proposal

A possible way of covering this Safety Related Application Condition consists in performing the independence analysis as prescribed by the CENELEC standard.

10.16.3 Associated risk

If the inter-MCU independence is not achieved then the generic safety case of the CLEARSY Safety Platform will be invalid.

10.17 SRAC 0017: NON TRIVIAL VALUES

10.17.1 Statement

> The user shall choose the values of the state variables in the B0 program so that setting all variables to a single same value will not create any permissive output.

10.17.2 Verification method proposal

To meet this Safety Related Application Condition, the end user identify the behavior of the system when all replicated variables are cleared to 0 (respectively set to 1). If, in these cases, the output remains restrictive then the requirement can be considered as met.

10.17.3 Associated risk

The associated risk is to perform a permissive action following a full erase of the RAM (failure writing 0 or 1 everywhere).

10.18 SRAC 0018: OFFLINE CRC

10.18.1 Statement

The end user shall verify each time he/she uploads or modify the software flashed on the target, that the offline CRC matches the one computed upfront on the generation computer.

Note: The full system embedding the CLEARSY Safety Platform shall not be run into revenue service during the phase when the software has been uploaded and the Offline CRC has not been checked (verification time).

10.18.2 Verification method proposal

To comply to this Safety Related requirement, the final user manual shall document the method to be used to retrieve the offline CRC through the maintenance link, and explain how to deploy new software and check that it has been correctly uploaded.

10.18.3 Associated risk

The associated risk is to execute corrupted software due to a toolchain compilation bug or wrong upload on the final target.

10.19 SRAC 0019: VITAL VOTERS

10.19.1 Statement

Any permissive output state shall be applied if and only if the two replicated software pieces agreed on the permissive value to be applied. If not the output shall be downgraded to a restrictive state. The CLEARSY Safety Platform library offers vital voter primitives which allow:

- to copy a replicated array into a destination buffer only if the two replicated vital buffers are identical. This primitive offers the guarantee to copy into the destination buffer the bytes provided in the replicas buffer up to the first byte in mismatch or the full payload is copied.
- to perform the bitwise AND (keep only the bits both set to one) between two replicated variables and copy the result at a given address. This primitive offers the guarantee that in the destination buffer set bits exist only if both replicas want to set these bits.

The behavior of these two primitives is guaranteed as safety critical (no extra demonstration by the end user is required).

Note: Details on how to use practically the vital voter primitives can be found in the current document in section "Use of vital voters".

10.19.2 Verification method proposal

The compliance to this Safety Related Application condition can be ensured by a critical code review that identifies these outputs of the vital B0 software and checks that the vital voter routines are properly used.

10.19.3 Associated risk

If this Safety Related Condition is not met then it means that some permissive outputs are not controlled by both replicated instance of the software but only one. In this case the mechanisms against systematic compilation error and memory corruption are bypassed meaning that the safety of the execution is no longer guaranteed.

10.20 SRAC 0020: VERIFICATION AFTER LONG STORAGE

10.20.1 Statement

If the CSP product has been stored in OFF state for more than 10 years, then the offline CRCs (per definition of SRAC 0018) shall be checked again before putting into service.

10.20.2 Verification method proposal

The compliance to this safety related application condition can be ensured by a review of the product's maintenance procedure. This procedure should clearly contain a procedure asking for a verification of Offline CRC, for product stored for more than 10 years.

10.20.3 Associated risk

During the storage the failures may accumulate. This accumulation of failures may disconnect the built-in self test and also mask the failures themselves. Checking the offline CRC ensures that the correct firmware is in the target so the built-in tests are still in place and correct.

10.21 SRAC 0021: GENERATION CORRECTNESS

10.21.1 Statement

Prior to flashing a software on a final product, which is meant to be used in revenue service, the end user shall check that the generation is correct. The generation is correct if and only if:

- all the SRACs defined in the user manual have been fulfilled, and
- the generation report refers to the same source files than the ones on which the verification and validation activity have been performed, and
- the generation report shows that the size of replicated data is not a multiple of 4096 bytes, and
- the generation report contains the same hash as the one listed in the certificate for the CLEARSY Safety platform library object files, and
- the generation cites the same version and hash of the compilation toolchain as the one listed in the CLEARSY Safety platform certificate, and
- the retrieved binary .hex file has the same hash than the one listed in the generation trace.

Note: A detailed checklist of what needs to be done and how these verifications can be performed can be found in the document C_D712 Generation verification procedure.

Tip: The generation needs to be performed only once for a given binary. When the binary has been validated as compliant to this safety related application condition, then it can be flashed on any physical device and as many times as needed. The correctness of the loading is enforced by the SRAC 0018.

10.21.2 Verification method proposal

The compliance to this Safety Related Application condition can be ensured by a review of the final deployment procedure which should require to inspect the result of the checklist defined in the C_D712 Generation verification procedure.

10.21.3 Associated risk

If this Safety Related Condition is not met then a potentially invalid software that does not offer the safety guarantee of the CLEARSY Safety platform can be used on site. For instance, it could be a software compiled with the wrong sources or with an unvalidated toolchain.

10.22 SRAC 0022: TEMPERATURE OPERATION CONDITIONS

10.22.1 Statement

The end user is sole responsible of guaranteeing that the CSP is operated within its allowed temperature range [-40°C;+85°C]. Outside these limits the generic safety certificate is not applicable.

Tip: A non-vital temperature sensor is provided directly on the CS0 hardware for provision of temperature monitoring.

10.22.2 Verification method proposal

The compliance to this Safety Related Application condition can be ensured by a review of the final product datasheet that should clearly state the environmental operating context of the final product. The user design should also take advantages of the temperature sensors to reduce power dissipation in cases where the temperature is close to the upper limit.

10.22.3 Associated risk

If this safety related application condition is not met, then the user may encounter an undefined behavior on both microcontrollers by a common mode overheating/overcooling.

10.23 SRAC 0023: JTAG DISABLED

10.23.1 Statement

> The JTAG feature of the board (i.e. pins TMS, TDI, TDO, TCK, TRCLK, TRD0, TRD1, TRD2, TRD3), shall not be used while the unit is deployed in revenue service. These pins could be used for their other multiplexed functions (GPIO for instance).

10.23.2 Verification method proposal

The compliance to this Safety Related Application condition can be ensured by a review of the software design and hardware architecture to demonstrate that the pins are either not connected or not used for their JTAG capability.

10.23.3 Associated risk

If these JTAG pins are used then the execution of the processor can be arbitrarily modified externally. This may bypass all the safety mechanisms, leading to potential death or injury.

10.24 SRAC 0024: MAXIMUM LEVEL OF INTERRUPT ALLOWED

10.24.1 Statement

> The user application code shall not implement any software or interrupt running with an interrupt level(priority) greater or equal to 7.

10.24.2 Verification method proposal

The compliance to this Safety Related Application condition can be ensured by a critical review of the software source code and by listing each interrupt service routine with its associated priority.

10.24.3 Associated risk

If this precaution is not taken then internal built-in mechanism may not be executed at the correct pace, and thus break the safety guarantees of the platform. This could lead to wrong delays and wrong permissive outputs.