AN4247
Application note

Safety Manual for SPC570S family

Introduction

This document is the SPC570S50E1/SPC570S50E3 Safety Manual. It provides the conditions of use for the SPC570Sxx in ISO 26262 ASIL D applications.

Contents

List of tables

List of figures

1 Preface

The SPC570S family has been specified to provide the safety integrity required by the most demanding safety critical systems. The device architecture and specification has been deeply reviewed in order to integrate safety measures whenever needed in order to reduce and control the risk related to the usage of advanced semiconductor technology devices in safety critical applications. The resulting exhaustive safety framework allows to claim -under proper usage- a residual risk linked to the usage of the device to a level compatible with the ISO26262 ASILD metrics.

This document provides the condition of use conditioning the ASILD metrics claim.

The SEooC[a] concept has been used in order to provide an application independent framework. Nevertheless each application may make use of all or only a subset of the devices features and therefore do not apply all provided recommendations described in this document which is related to the complete SEooC.

The SEooC elements can be classified into mainly 3 categories:

- Major part is composed by the device subsystem that has been made safe with on-chip HW measures. This part is the core part of the SEooC as it exclusively deals with the idea of providing a safe element/device independently from the application
- The SEooC provides 2 further important extensions with the aim to provide external safety support
  - Support to application redundant acquisitions and/or actuators
  - Support to SW safety

[SAG_VE512K_1_0] This document provides guidelines for the proper use of the SPC570Sxx Microcontroller Unit (MCU) in ASIL D applications [end]. It helps the user with the steps necessary to integrate the SPC570Sxx into their application and provide margin wrt ASIL-D target FIT to allow flexibility in FIT budgeting at system assembly.

[SAG_VE512K_1_1] The SPC570Sxx is used as a component within a safety related application. To allow an analysis of the MCU's capability to reach the required safety level, assumptions have been made (following the concept of SEooC described in the ISO26262). These assumptions are on the scope of the MCU (for example, including external components interacting with the MCU) and on its usage by application software. The FMEDA provided with the SPC570Sxx was conducted under inclusion of these assumptions [end].

This document considers:

- The system assembly that contains the SPC570Sxx MCU
- The “Safety Element out of Context” section in the “Road vehicles - Functional safety - Part 10: Guideline [ISO/DIS 26262-10]” standard
- Certain assumptions about the assembly's functional safety needs based on that standard determine whether a measure is an assumption or not based on these factors.

a. The “Safety Element out of Context” section in the “Road vehicles - Functional safety - Part 10: Guideline [ISO/DIS 26262-10]” standard

These guidelines are considered to be a useful approach for the specific topics under discussion, but are not mandatory. The user needs to use discretion in deciding whether these measures are appropriate for their applications.
What this means for designers using the SPC570Sxx is that if they don't fulfill a specific Safety Manual (SM) assumption they have to show that their alternative solution is similarly efficient concerning the safety requirement in question (for example, it provides the same coverage, avoids Common Cause Failure (CCF) as effectively, and so on), it shows that the particular issue is irrelevant for their application (e.g., the module is not used), or estimate how much the failure rate increases and the failure metrics (SPFM/LFM) decrease due to the deviation. Otherwise, the FMEDA provided with the SPC570Sxx is not valid.

This document also contains guidelines on how to configure and operate the SPC570Sxx for ASIL D applications. These guidelines are preceded by one of the following bold text statements:

- Assumption
- Recommended
- Implementation hint
- Rationale

Note: Other information about safety configuration and operation can be found in the SPC570Sxx Reference Manual's "Functional Safety" chapter (see Section 6.1: Document management).

This document is only valid under the assumption that,

- [SAG_VE512K_1_3] The SPC570Sxx is used in automotive applications for use cases requiring a fail-silent or a fail-indicate MCU. [end]
- the environmental conditions given in the *SPC570Sxx Data Sheet* are maintained.

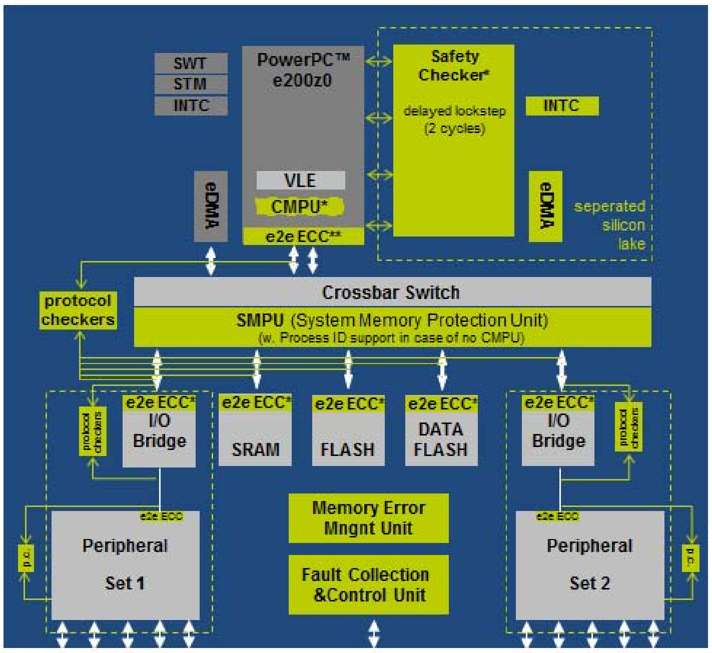
As for all devices, device errata must be taken into account during system design and implementation. For a safety-related device such as the SPC570Sxx, this also concerns safety-related activities such as system safety concept development. The FMEDA and Safety Concept are valid if the listed assumptions in the text are covered.

General failure rate, and FMEDA (Failure Modes, Effects & Diagnostic Analysis) report, are available upon request when covered by an STMicroelectronics NDA (contact your STMicroelectronics representative).

1.1 SPC57S ASILD safety concept

1.1.1 Overview of on-chip measures again random HW failures

Figure 1. Block diagram of SPC570S family on-chip HW Safety measures



The enhanced ASILD SEooC consists of the following measures:

- Processing platform
  - Core, INTC and DMA in delayed-lockstep
  - Memories protected by ECC
- Path down to memories / IO periph. (incl. address & ctlr)
  - End-to-end ECC[b] extended with protocol checkers
- Dedicated measures on peripherals
  - e.g. ADC supervisor, CRC unit for serial communications
- Independant FCCU unit (fault collection & control unit) for failure collection and reaction

Diagnostic measures availability guaranteed by dedicated techniques (e.g HW-BIST, EDC after ECC,.).
Dependant fault mitigation by diverse methods: checker diversity, supply monitoring low & over voltage monitors (incl. pre-warnings), backup clock and clock monitoring, temperature sensor

b. e2e EDC on SPC570S

SEooC has been extended with the following measures to support “external safety” requirements:

- Dual Channel Architecture
  - Specific safety measures as consequence of the ECU/system safety architecture/concept: Redundant IO path to support redundant inputs/outputs (e.g. redundant sensors) MCU does not introduce dependent faults
- SW Safety Support
  - SW access protections at all levels of the architecture (CMPU(c), SMPU, AIPS, selective register protection)

c. On SPC570S CMPU replaced by process ID support in SMPU.

2 General information

2.1 Mission profile

Lifetime for a SPC570Sxx is 20 years which is equivalent to 20000 hours of active operation for the MCU. The assumed mission profile is:

- Lifetime: 20 years
- Total operating hours: 20000 hours
- [SAG_VE512K_2_0] Trip time (driving cycle): 12 hours[end]
  - This is the maximum time of operation of the SPC570Sxx without a start-up reset.
- [SAG_VE512K_2_1] Fault-Tolerant Time Interval (FTTI, also known as Process Safety Time, PST) = 10 ms[end]
  - FTTI is the time the controlled system, it does not pass to a hazardous state, despite the SPC570Sxx failing.

Note: *This is a conservative rule of thumb since the actual number depends on MCU application (See Section 2.6: Failure indication time, for exact calculation instructions).*

Table 1 provides information about temperature profiles for packaged devices.

Note: *The temperature profile is an assumption of the SPC570Sxx safety analysis. If the SPC570Sxx is used with a different temperature profile, the failure rate as calculated with that analysis is not the same as in this document.*

Table 1. Temperature Profile for Packaged Device

Driving (ECU active): 12500 hours	
Temperature mounting base (Celsius degree)	**Duration (hours)**
-40	250
10	1250
53	3000
90	4250
110	3000
130	625
140	125
Charging (ECU active): 30000 hours (Optional)	
Temperature mounting base (Celsius degree)	**Duration (hours)**
70	12723
90	13017
110	3279
130	747
140	234

Table 1. Temperature Profile for Packaged Device (continued)

Driving (ECU active): 12500 hours	
Temperature mounting base (Celsius degree)	**Duration (hours)**
Parking (ECU passive): 88900 hours	
Temperature mounting base (Celsius degree)	**Duration (hours)**
-10	355
20	65342
40	23203
Storage Time	
Temperature mounting base (Celsius degree)	**Duration**
-50 <= Tamb <= -20	50 hours
-20 <= Tamb <= +50	5 years
+50 <= Tamb <= 140	50 hours

Assumption: [SAG_VE512K_2_2] The device is to be handled according to JEDEC standards J-STD-020 and J-STD-033.[end]

2.2 Functional safety – ISO 26262 compliance

[SAG_VE512K_2_3] The SPC570Sxx MCU was developed in accordance with ISO 26262 as a Safety Element out of Context (SEooC).[end]

[SAG_VE512K_2_4] The development process of the SPC570Sxx fulfills ASIL D requirements of ISO 26262. [end].

The SPC570Sxx is suitable to be used in safety-relevant applications including systems that are classified as ISO 26262 ASIL A, ASIL B, ASIL C or ASIL D.

2.3 Safety goal

The safety goal of the MCU is defined as follows:

- [SAG_VE512K_2_5] The **safety goal** is that the SPC570Sxx does not leave its safe states for intervals equal to or longer than the FTTI (10 ms) unless configured by the application software to do so.[end]

The ASIL for the safety goal is D.

2.3.1 Safe state

[SAG_VE512K_2_7] The safety goal is achieved by passing or holding the SPC570Sxx in the following safe states:[end]

- [SAG_VE512K_2_8] Completely unpowered [end]
- [SAG_VE512K_2_9] In reset [end]
- [SAG_VE512K_2_10] Operating correctly [end]
- [SAG_VE512K_2_11] Explicitly indicating an internal error [end]

If the SPC570Sxx continuously switches between a standard operating state and the reset state, without any device shutdown, the MCU is not considered to be in a Safe state (see *Section 3.2.6: Reset Generation Module (MC_RGM)* for details).

Assumption: [SAG_VE512K_2_12] The application identifies and signals such switching as a failure condition.[end]

If the SPC570Sxx signals an internal failure via its error out signals (FCCU_F0, FCCU_F1), the surrounding subsystem should no longer use the SPC570Sxx outputs for safety functions since these signals are no longer considered reliable. This means that if an error is indicated, the system must be able to remain in a Safe state without any additional actions of the MCU. Depending on its configuration, the system may disable, or reset, the MCU as a reaction to the error signal.

Assumption: [SAG_VE512K_2_13] It is assumed that the system reacts safely to the SPC570Sxx being in or entering all safe states shown in *Section 2.3.1: Safe state*.[end]

Since safe state depends on the availability of safety mechanisms SPC570S50Ex, the system is able to detect the permanent unavailability of any safety mechanism that is necessary to reach the safety goal. This is done at least once per driving cycle (i.e. 12 hrs).

2.4 Correct operation

[SAG_VE512K_2_14] Correct operation of the SPC570Sxx is defined as Safety relevant modules are operating according to specification.[end]

2.5 Failure indication signaling

The Fault Collection and Control Unit (FCCU) offers a hardware channel to collect errors and to bring the device to a safe state when a failure is present in the device. The FCCU provides two error output pins (EOUT pins interface) FCCU_F0 and FCCU_F1 as a failure indication to the external world.

Error signalling on pins can be controlled automatically via HW by the FCCU or via SW.

The two control modes are mutually exclusive.

Different protocols for the error output pins are supported:

- Dual rail protocol
- Time switching protocol
- Bi-stable protocol

In case the control is made via SW, the EOUT pin state is controlled by the application SW as long as the FCCU is in NORMAL or ALARM state.

- As soon as a the FCCU enters the FAULT state, both the EOUT pins are forced to the same level, which can be 0-0 or 1-1 depending on the polarity configuration.
- As long as the FCCU is in FAULT state, the SW has no control over the EOUT pins.
- Once the error condition is recovered, the FCCU returns to the NORMAL state and the SW can take back the control of the EOUT pins.

If bi-stable protocol is selected it is possible to use only one of the two error output pins on the SPC570Sxx.

An error status flag can be read to verify if the FCCU is in an error state. The flag can be written by software to either 1 to indicate fault or 0 to indicate operational state. After start-up reset, the error output pins go to the operational state only on software request.

At least one of the failure indication pins must be high to indicate operational state. If a two pin, bi-stable protocol is selected, the application software can configure which pin is the operational high pin configuring the FCCU_CFG[PS] bit.

Refer to *Section 4.5: Error Out Monitor (ERRM)* for details on specific requirements for connecting FCCU_F0 and FCCU_F1 to an external device.

2.6 Failure indication time

Failure indication time is the time it takes from the occurrence of a failure to when the indication of that failure is visible by driving the error out pins or by assertion of reset.

The failure indication time of the SPC570Sxx is finite. It must be taken into account when determining application safety strategies since failure indication time plus reaction time to this indication by the system must be less than the FTTI.

[SAG_VE512K_2_18] Failure indication time has three components, two of which are influenced by certain configuration choices:
Failure indication time = **detection time** + **processing time** + **signalling time**.[end]

Each component of failure indication time is briefly described as follows:

- **Detection time** is the longest detection time required by all involved safety mechanisms. The two mechanisms with the longest times are:
  - Recognition time related to the FMPLL loss of clock: This time depends on the FMPLL configuration. This time is approximately 20 µs.
  - Diagnostic cycle time of software self-tests. This time depends closely on the software implemented.
- **Internal processing time** lasts a maximum 10 IRCOSC clock cycles (nominal frequency of IRCOSC is 16 MHz).
- **Indication time** is the time to notify an observer about the failure. This time depends on the indication protocol in the FCCU:
  - Dual Rail protocol and time switching protocol

    FCCU configured as “fast switching mode”: indication delay is maximum 64 µs. As soon as FCCU receives a fault signal, FCCU reports the failure to the outside world via output pin (if properly configured).

    FCCU configured as “slow switching mode”: an indication delay could occur. The maximum delay is equal to period of the error out signal. This parameter must be configured equal to its minimum which is 128 µs.
  - Bi-stable protocol: indication delay is maximum 64 µs. As soon as FCCU receives a fault signal, FCCU reports the failure to the outside world via output pin (if properly configured).

If the configured reaction to a fault is an interrupt, an additional delay (interrupt latency) can occur until the interrupt handler is able to start executing (for example, higher priority IRQs, XBAR contention, register saving, and so on).

Assumption: [SAG_VE512K_2_19] The overall failure indication time is less than the FTTI of the application (assumed FTTI shown in *Section 2.1: Mission profile*).[end]

2.6.1 Minimum failure indication time

When a failure event occurs, one or both error output signals (F*n*) are set to show an error condition for a minimum time (T_min), even if software attempts to reset the state of the error out signals. The external failure indication stays in failure mode for a configurable minimum time as shown in *Equation 1*. For bi-stable protocol the time DELTA_T is configurable by software up to a maximum of 10 ms by configuring FCCU_DELTA_T[DELTA_T].

Equation 1

T_min = 250 µs + FCCU_DELTA_T[DELTA_T] µs

In case another failure event happens within T_min after the first failure event, the timer measuring T_min is restarted.

2.7 Failure handling

The FCCU is responsible for reacting to internal failures. A different reaction can be configured for each failure source.

Failure sources include:

- All failure indication signals from the modules within the MCU
- Control logic and signals monitored by the FCCU itself
- Software-initiated failure indications

The different failure sources, as represented by the FCCU failure inputs.

Note: See the "Failure handling" chapter of the SPC570Sxx Reference Manual for details (see Section 6.1: Document management).

Available failure reactions include:

- Assertion of an interrupt (maskable or non-maskable)
- Reset
- External failure indication (changing the state of the failure indication pins)
- No reaction

Additionally, the transmission capabilities of the communication controllers can be disabled when the FCCU transitions to the fault state in conjunction with changing the state of the failure indication pins.

Sources of error are collected, and reported, by the FCCU. Each source of error has an associated status flag, refer to the FCCU_RF_S[0:3] registers in the Reference Manual, which can be accessed by the software to gather details about collected errors.

As FCCU is not affected by a functional reset, the status of the sources of error can be read either before or after such a reset.

Software can also clean the fault by clearing the error status flag in the FCCU.(d) Even if the software clears an error status, the external failure indication stays in failure mode for a configurable amount of time (see Equation 1).

When the source of the failure is determined, software can decide the next step to be taken to handle the error. The software then may decide to do nothing, cause resets of certain modules, reset the whole MCU or change the value of the error out pins.

Error handling can be split into two categories:

- Handling of errors during run-time
- Handling of errors during boot-time (for example, LBIST, MBIST)

Assumption: [SAG_VE512K_2_20] Run-time errors are handled in a time shorter than the FTTI.[end]

Assumption: [SAG_VE512K_2_21] Boot-time failures are handled before the safety function starts execution. Typically, the reaction is not to let the safety function start and give a failure indication to the user.[end]

d. Some error status can be cleared just via the FCCU programming interface while others require additional actions to clear also the source of error indication.

3 Functional safety requirements for application software

This section gives an overview of the necessary or recommended measures when using SPC570Sxx in safety related applications. If a module in SPC570Sxx is used without following the required actions there is a risk that the safety analysis for the entire MCU may be invalidated, or other modules if the failure interferes with the correct operation of the other modules.

It is nevertheless possible to replace the required/recommended measures presented in this section with alternative measures capable of covering the same failure modes with the same effectiveness.

Modules not explicitly covered by this document do not require any safety specific software measures.

To allow continuous improvement of STMicroelectronics products it is recommended to report field failures. Management of field returns is required independently of the measures adopted. The subsequent analysis determines the cause to the failure, which can be the violation of the recommendations in the Safety Manual in accordance with ISO 26262-7 Chapter 6.4.2.1.

3.1 Disabled modes of operation

The system and application software must ensure that the functions described in this section are not activated while running safety-relevant operations.

3.1.1 Debug mode

The debugging facilities of the SPC570Sxx pose a possible source of failures in case they activate during the operation of safety-relevant applications. They can halt the cores, cause breakpoints to hit, write to core registers and the address space, and activate boundary scan. The MCU must therefore not enter debug mode to avoid interference with the normal operation of the application software.

Assumption: [SAG_VE512K_3_0] Debugging is disabled in the field while the device is being used for safety-relevant functions.[end]

Assumption: [SAG_VE512K_3_1] In the field, while running a safety application, software needs to configure the Software Watchdog Timer (SWT), clock generation, and FCCU to continue execution in case the device is transitioned into debug mode (to not ‘freeze’ operation when in debug mode) (for example, SWT_CR[FRZ] = 0).[end]

3.1.2 Test mode

Several mechanisms of the SPC570Sxx can be circumvented in test mode which endangers the safety concept. Test mode is used for comprehensive factory testing and should not be used in normal operating mode.

The TEST pin is for testing purposes only and should be tied to GND in normal operating mode. The system must ensure the TEST pin is not asserted during boot to enable test mode. The activation of test mode is supervised by the FCCU and signals a fault condition when test mode is entered.

Assumption: [SAG_VE512K_3_2] Test mode is disabled in the field while the device is being used for safety-relevant functions.[end]

To avoid unwanted activation of the testing circuitry the Design For Testability (DFT*n)* FCCU error inputs must be enabled even if they are not needed by the application. The FCCU channels for DFT[1:4] are 46 to 49, respectively. These error inputs are enabled by setting the appropriate bits in the following registers:

- FCCU_RF_CFG
- FCCU_RFS_CFG
- FCCU_RF_TOE
- FCCU_RF_E
- FCCU_IRQ_ALARM_EN
- FCCU_NMI_EN
- FCCU_EOUT_SIG_EN

3.2 Initial checks and configurations

Assumption: [SAG_VE512K_1_4] All relevant hardware safety mechanisms are enabled and configured correctly.
After start-up, the application software must ensure the conditions described in this section are satisfied before safety-relevant functions are enabled. Additional configuration needed to ensure freedom from interference between concurrent software tasks are described in *Section 3.4: Operational interference protection*).

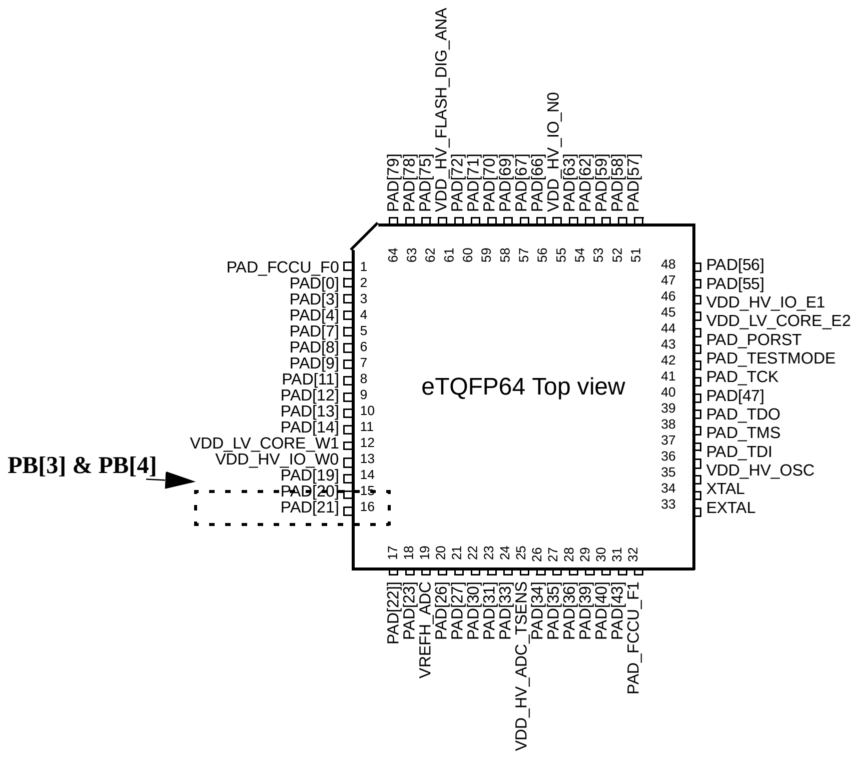
3.2.1 I/O pin configuration

Assumption: [SAG_VE512K_3_3] The user avoids configurations that place redundant signals on neighboring pads or pins.[end]

[SAG_VE512K_3_4] Whether two functions on two package pins are adjacent to each other can easily be determined by looking at the mechanical drawings of the packages (see the SPC570Sxx Data Sheet) together with the pin number information of the packages as seen in the SPC570Sxx Reference Manuals “System Integration Unit Lite2 (SIUL2)” section and the table “Pin Muxing description” (see chapter 4.2.3 of the SPC570Sxx Reference Manual)*(Section 6.1: Document management).* [end]

The internal die pad sequence can be derived from the package pin sequence of the eTQFP64/eTQFP100 pin package shown in the *SPC570Sxx Data Sheet*.

Figure 2. eTQFP 64-pin configuration

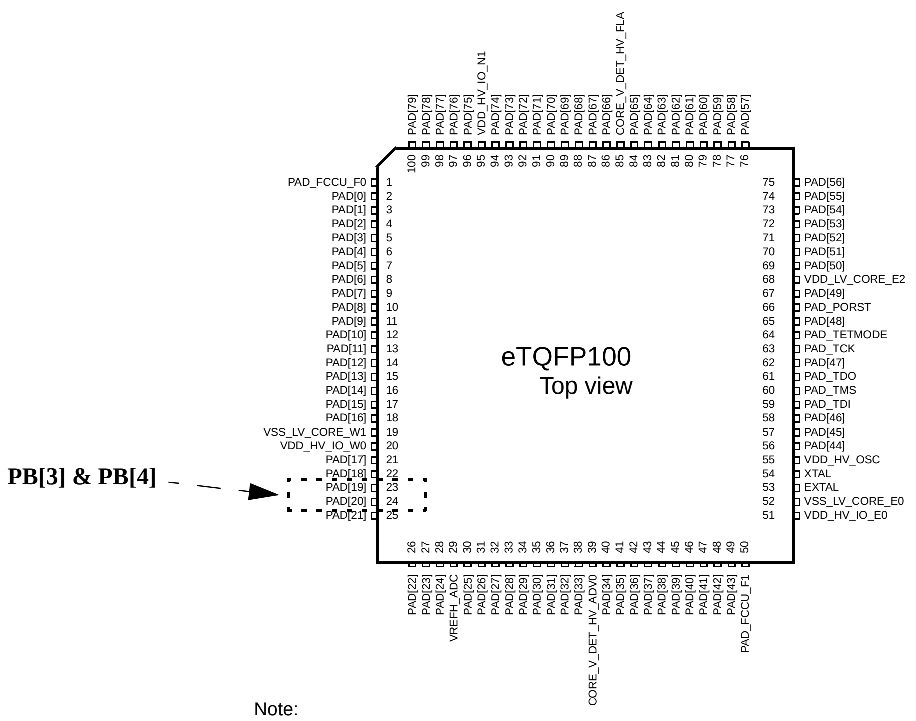


Note:

Availability of port pin alternate functions depends on product selection.

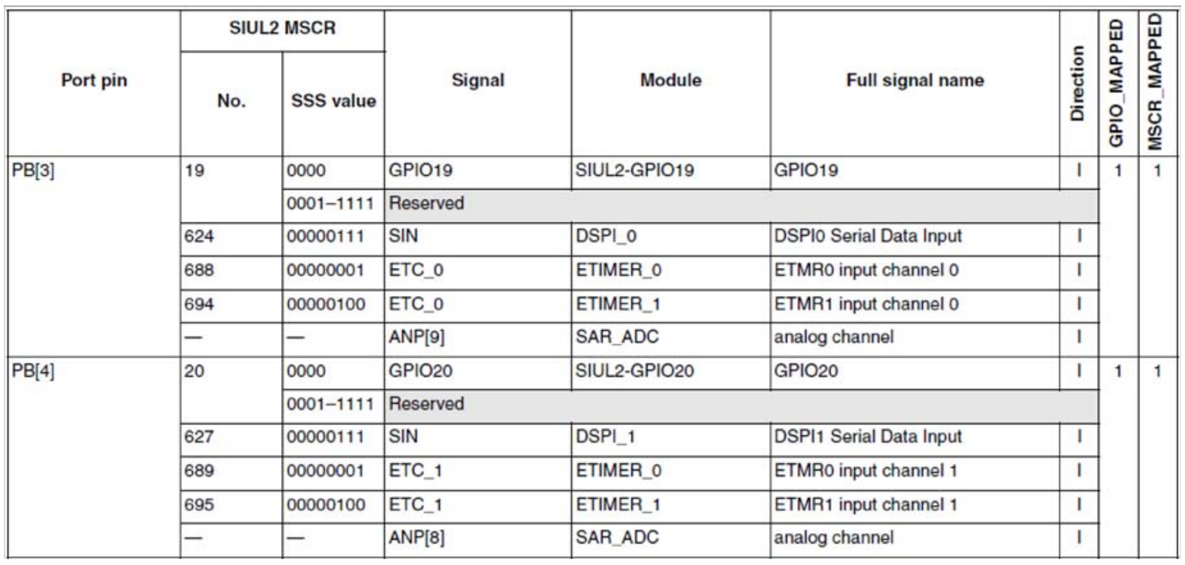
Note: All eTQFP64 information is indicative and must be confirmed during silicon validation.

Figure 3. eTQFP 100-pin configuration



Note:
Availability of port pin alternate functions depends on product selection.

Figure 4. Example of eTQFP64/eTQFP100 pin adjacency



Port pin	SIUL2 MSCR		Signal	Module	Full signal name	Direction	GPIO_MAPPED	MSCR_MAPPED
	No.	SSS value						
PB[3]	19	0000	GPIO19	SIUL2-GPIO19	GPIO19	I	1	1
		0001–1111	Reserved					
	624	00000111	SIN	DSPI_0	DSPI0 Serial Data Input	I		
	688	00000001	ETC_0	ETIMER_0	ETMR0 input channel 0	I		
	694	00000100	ETC_0	ETIMER_1	ETMR1 input channel 0	I		
	—	—	ANP[9]	SAR_ADC	analog channel	I		
PB[4]	20	0000	GPIO20	SIUL2-GPIO20	GPIO20	I	1	1
		0001–1111	Reserved					
	627	00000111	SIN	DSPI_1	DSPI1 Serial Data Input	I		
	689	00000001	ETC_1	ETIMER_0	ETMR0 input channel 1	I		
	695	00000100	ETC_1	ETIMER_1	ETMR1 input channel 1	I		
	—	—	ANP[8]	SAR_ADC	analog channel	I		

For example, the internal die pads supporting the functionality described in *Figure 4* are referred to by “Port Pin” in the first column. From this figure it can be seen that the port pins are PB[3] and PB[4]. Since these two port pins are in sequential order on the same port (Port B) the die pads are adjacent to each other. The corresponding two eTQFP64/eTQFP100 package pin numbers are directly adjacent to each other, pin 14 and 15 for eTQFP64 and pin 23 and 24 for eTQFP100. In general, the internal die pads follow the same sequence as the corresponding package pins for eTQFP64/eTQFP100 packages. If pins on the eTQFP64/eTQFP100 pins are adjacent to each other, the corresponding internal die pads are also adjacent. Likewise, if package pins are not adjacent to each other the corresponding die pads are also not adjacent.

3.2.2 MCU configuration

Depending on the specific usage the user can program the device to start with some specific configurations. User saves such configurations in a dedicated NVM called UTest sector.

See the “Device Configuration Format (DCF) Records” chapter in the Reference Manual *see Section 6.1: Document management).*

After reset, the hardware initializes the devices by moving such data into the relevant internal register.

Assumption: [SAG_VE512K_3_5] Safety software running on the Safety core checks correct initialization of the SPC570Sxx before activating the safety-relevant functionality.[end]

The MCU memory configuration and the JTAG Part ID number can be read in the SSCM_MEMCONFIG register (JTAG Part ID = SSCM_MEMCONFIG[JPIN]).

This information is normally used for debugging purposes, and is not necessary for the safety function.

Assumption: [SAG_VE512K_3_6] Application software does not use the JTAG Part ID, nor does it affect safety critical operations.[end]

With the System Status and Configuration Module (SSCM), it is possible to configure different MCU behaviors.

Assumption: [SAG_VE512K_3_7] Application software checks the configuration of the SCCM once after boot.[end]

Assumption: [SAG_VE512K_3_8] SSCM is configured to trigger an exception in case of any access to a peripheral slot not used on the device (SSCM_ERROR register).[end]

Assumption: [SAG_VE512K_3_9] Once after the boot, application performs an intended access to an unimplemented memory space and check for the expected abort to occur.[end]

3.2.3 Mode Entry (MC_ME)

To overcome faults in the wakeup and interrupt inputs to the MC_ME, the following is assumed if the application uses Low Power mode (LP):

- **Assumption:** [SAG_VE512K_3_10] The duration in LP mode is monitored. If the system does not wake up within a specified time frame, the system is reset by the monitor (for example, SWT can provide the time monitoring).[end]
- **Assumption:** [SAG_VE512K_3_11] Software performs a test of entry and exit to and from LP mode at startup.[end]

An incorrect clock source as the system clock could be selected due to faults, resulting in multiple faults. In order to improve detection, of such faults and the effect, by the clock monitors:

- **Assumption:** [SAG_VE512K_3_12] It is assumed that the different clock sources that can be selected as the system clock have different frequencies.[end]

The mode configuration registers of MC_ME take effect only when the mode transition request is initiated. Thus, instead of the configuration registers the global status register should be CRCed (if configuration register CRCing is done) as that represents the current state.

Assumption: [SAG_VE512K_3_13] The intended value of the target mode configuration registers is verified before mode transition request is issued.[end]

Assumption: [SAG_VE512K_3_14] In order to check that a mode transition has been correctly executed, software, after initiating mode transition request, verify the mode transition status within the expected completion delay. This does not apply in case target of mode transition is LP mode.[end]

Note: The MC_ME implements a register to request a mode transition and registers that report the status of the transition (for example, ME_GS, ME_IMTS, ME_DMTS registers).

3.2.4 Dual core lockstep mode

The SPC570Sxx device operates in delayed lockstep mode (LSM) to allow the highest safety level to be reached. Replicated peripherals are Core, interrupt controller (INTC), DMA and DMAMux.

The checker core receives all inputs delayed by two clock cycles. Outputs of the checker core is compared with outputs of the master core. Any differences are flagged as an error which is processed by the FCCU. This kind of behavior is valid also for INTC and DMA.

For safety operation the appropriate configuration in the flash memory must not be programmed to disable LSM. If the LSM is disabled, the checker core and the Redundancy Checker Control Units (RCCUs) are disabled. This triggers a fault indication to the FCCU. The checker core does not work independently from the master core. No dynamic switching is possible between LSM on and LSM off (any change to the respective bit in Flash only takes effect after the next reset).

Assumption: [SAG_VE512K_3_18] Before starting safety-relevant operations, the application software checks that lockstep mode is enabled (in the Mode Entry Module (MC_ME)) and configure the FCCU to react to lockstep disablement (MC_ME_CS[S_CORE1] = 1 (checker) and MC_ME_CS[S_CORE2] = 1 (master)).[end]

3.2.5 FCCU fault reaction configuration

The Fault Collection and Control Unit (FCCU) collects faults and manages the reaction to these faults. A mechanism is usually provided to allow software to check the integrity of the different error paths. Most reactions are disabled at boot time so software configuration is required. Refer to *Section 2.7: Failure handling* for the valid FCCU fault reactions.

Assumption: [SAG_VE512K_3_19] Application software checks the FCCU configuration once after programming.[end]

The FCCU is checked by the FCCU Output Supervision Unit (FOSU) which provides a secondary path for the failure indication and reports to the Reset Generation Module (MC_RGM). The FOSU only causes a reset when the FCCU fails to react to an incoming and enabled fault within a fixed time interval (1000 IRCOSC cycles). The FOSU does not require software configuration.

While FCCU is in CONFIG state, FOSU does not monitor FCCU for asserted faults and the resulting reaction.

Assumption: [SAG_VE512K_3_20] Application SW checks and clear any pending faults when it moves FCCU out of CONFIG state.[end]

Assumption: [SAG_VE512K_3_21] Before starting safety-relevant operations, software must configure the fault reactions to each fault that is safety-relevant for the application.[end]

Note: ***Implementation hint:*** *Software must program the FCCU NCFS Configuration Registers (FCCU_RFS_CFGx) to configure the fault reaction of each fault. These registers are writable only if the FCCU is in the CONFIG state.*

Assumption: [SAG_VE512K_3_22] To detect failures affecting the FCCU error reaction paths, specific software tests should be run. [end]

The FCCU comes out of reset with most of the failure inputs disabled. Failures occurring during boot is for the most part captured by the FCCU without triggering any error reaction. To check whether such errors have occurred, SW can read the FCCU failure status registers for any latched error and act on the status of those bits accordingly (FCCU_RF_S[0:3]).

Note: *See the “Failure handling” chapter of the SPC570Sxx Reference Manual for details (see Section 6.1: Document management).*

The error indication on pins, FCCU_F0 and FCCU_F1, are controlled by the SIUL2 and FCCU. The FCCU_CFG register is used to configure other FCCU_F*n* options like signal polarity, switching mode, software control, and so on.

Since the status of the FCCU is maintained during a functional reset, FCCU together with INTC can lead to cyclic reset. For example consider the following situation:

1. Error indication arrives at FCCU
2. FCCU triggers IRQ
3. SW analyzes fault and causes a functional reset
4. MCU comes out of reset and hands over to SW
5. SW configures INTC
6. SW gets the same IRQ again (because FCCU still holds the IRQ line), analyzes fault and causes a reset ad infinitum (or rather till the reset escalator engages and causes a destructive reset).

To avoid this situation the following assumption is considered.

Assumption: [SAG_VE512K_3_23] It is assumed that FCCU pending faults status should be cleared before INTC is configured.[end]

Unwanted activation of LBIST/MBIST causes a violation of the safety goal.

Assumption: [SAG_VE512K_3_36] Software always enables FCCU reactions to error events indicating unexpected STCU2 activations.[end]

3.2.6 Reset Generation Module (MC_RGM)

The MC_RGM is the central point for resetting the MCU. One of its tasks is to prevent reset cycling. It also can transition to SAFE mode[e]. The SAFE mode has not been considered a safe state during safety analysis (Safe state MCU is defined on *Section 2.3.1: Safe state*).

Permanent cycling through otherwise safe states or permanent cycling between a safe state and an unsafe state is considered a violation of the safety goal. Specifically, this scenario relates to a continuous Reset – Start, Operation – Reset or Reset – Self-test – Reset sequence. Allowing such cycles would be problematic as it would allow an unlimited number of attempts of the test that is causing the cycle which could possibly endanger its ability to detect device failures.

To detect a loop of continuous functional resets, the SPC570Sxx supports functional reset escalation which can be used to generate a destructive reset if the number of functional resets reaches the programmed value. Once the functional reset escalation is enabled, the Reset Generation Module (MC_RGM) increments a counter for each functional reset that occurs. When the number of functional resets reaches the programmed value in the MC_RGM_FRET, the MC_RGM initiates a destructive reset. The counter can be cleared by software, destructive reset or start-up reset.

A similar mechanism to detect a loop of continuous destructive resets is implemented in the MC_RGM. When the destructive reset counter reaches the programmed value, the MCU is held in reset until the next power-on reset. The destructive reset counter can be cleared by software or by a power-on reset.

Assumption:[SAG_VE512K_3_24] Safety software resets the functional and destructive reset counters every time it has finished checking its environment (for example, before making the FCCU_F0 and FCCU_F1 pins active).[end] The MC_RGM_FRET (functional reset counter) and MC_RGM_DRET (destructive reset counter) registers allow the user to select the number of functional and destructive resets that can occur before action is taken (see “Reset Generation Module (MC_RGM)” in the *SPC570Sxx Reference Manual* for details, *see Section 6.1: Document management).*

Assumption: [SAG_VE512K_3_26] Functional and destructive reset escalation are enabled by default. To avoid condition when multiple resets occur consecutively, software does not disable them.

Note: Functional reset escalation is enabled by writing a non-zero value to the MC_RGM_FRET register (see the SPC570Sxx Reference Manual’s ‘Reset Generation Module (MC_RGM)) (see Section 6.1: Document management).

The time between reset loops can be so short that the application software doesn’t have time to take any action to manage them. On the other hand, longer reset cycles must be managed by application software.

Assumption: [SAG_VE512K_3_27] Before resetting the reset counters of the escalation mechanism, the safety software checks for the occurrence of long reset cycles.[end]

Note: Different implementations of this requirement can be thought. One example is that the safety software, before clearing the reset counters, reads (and saves) the FCCU error status indication (if any fault has been observed) and compare it with the last few seen. If too many resets due to detected faults are observed, software can react by triggering a destructive reset.

e. SAFE mode is one mode of the MC_ME module

For some events, the MC_RGM can be configured to react not with a functional reset, but with a transition to the SAFE mode (see SPC570S50Lx Reference Manuals "Functional Event Reset Disable Register (RGM_FERD)" section for details) (see*Section 6.1: Document management*). In such a case, one watchdog is kept enabled. In case of its timeout, the FCCU moves the MCU into one of its safe state.

Assumption: [SAG_VE512K_3_28] If the MC_RGM is configured to react with a transition into SAFE mode, at least one watchdog timer remains enabled. The FCCU is configured to react to a timeout of this watchdog with a long functional reset or driving the error out signals to a fault condition.[end]

Assumption: [SAG_VE512K_3_29] Software read the reset status after boot ensuring that the reset cause is indicated. Then software clears the register, and read back the value verifying that it is actually cleared.[end]

Assumption: [SAG_VE512K_3_30] Resets during normal operation is executed only as a reaction to an error, not as a functional measure. This avoids undetected faults due to interrupts that are not being generated.[end]

3.2.7 Self-test completion

To ensure absence of latent faults, the Self Test Control Unit executes both a Logic Built-In Self-Test (LBIST) and a Memory Built-In Self-Test (MBIST)[f] during boot while the device is still under reset (off line).

The overall control of the LBISTs and MBISTs is provided by the Self-Test Control Unit (STCU2).
The STCU2 executes automatically after a power-on reset[g] (POR) destructive reset. The SPC570Sxx logic is grouped into 3 LBIST partitions used for both production testing and self-test.

Note: *The SPC570Sxx Reference Manual's "Self-Test Control Unit (STCU2)" chapter and "Use cases and limitations" section discusses details on how to correctly execute off line self-tests (see Section 6.1: Document management).*

The section "Online Logical BIST (LBIST)" of the SPC570Sxx Reference Manual's "Functional safety chapter" shows tables of the module groupings of each LBIST partition (see Section 6.1: Document management).

f. This does not include flash memory.

g. The user must enable the self-test in the shadow sector of the flash memory since the factory default configuration does not run the self-test.

Table 2. LBIST partitions

LBIST Partition	Instance Name
LBIST 0 (platform)	CPU0
	XBAR
	DMA0
	DBG
	FCCU
	INTC0
	DMAMUX0
	LINFLEX1
	SRAMC
	FLASHC
	MBIST-COLLARS
	CMU1-2
	ETMR2-3
	DSPI2-3
	AIPS1
	AIC0
LBIST 1 (channel checker)	CPU1
	INTC1
	DMA1
	RCCUx
	DMAMUX1

Table 2. LBIST partitions

LBIST Partition	Instance Name
LBIST 2 (Peripherals)	AIPS0
	AIC1
	DSPI0-1
	CMU0
	MEMU
	CTU
	FLEXCAN
	LINFLEX0
	ETMR0-1
	PIT
	sarADCdig
	sarADCdig
	CRC
	SIUL
	FLASHdig
	MBIST
	MBIST-COLLARS

Table 3. Not in LBIST pertitions

Functional Group	Instance Name
System	MC_CGM
	MC_ME
	MC_PCU
	MC_RGM
	SSCM
	WKPU
	SIUL2
	IOMUX
	STCU2
Clocking	PLLs
	CMUs
	XOSC
	IRCOSC

Table 3. Not in LBIST pertitions

Functional Group	Instance Name
Power	PMC
	ADC bandgap reference
	TSENS
Debug	JTAGC
	SPU
	NEXUS
	DPS
	NAR
	DCI
	JTAGM

Assumption: [SAG_VE512K_3_31] If there is an LBIST failure, or MBIST detects uncorrectable failures, the STCU2 cause a destructive reset, causing execution of the self-test again. This is to ensure that a self-test, which fails only due to a transient error, does not block device usage. If several self-tests fail in a row, the destructive reset escalations activates and hold the MCU in reset.[end]

On the other hand, if MBIST detects correctable failures, software must decide whether to continue or halt execution. In fact, the MBIST may detect and report two (or more) Single Bit Errors (SBEs) occurring in multiple test passes instead of one Multiple Bit Error (MBE).

Assumption: [SAG_VE512K_3_33] After start up (and more in general, always after the execution of MBISTs), software cross check MBIST status in the STCU2 (pass or fail) with the content of MEMU MBIST buffer (same as system RAM) to detect failures affecting the reporting of MBIST errors. This can be due to faults affecting the reporting path for MEMU or STCU2 logic. (notice that STCU2 is not part of any LBIST partition and only a pass/fail flag is available).[end]

Assumption: [SAG_VE512K_3_34] After start-up and before the safety application starts, application software confirms that all LBISTs and MBISTs finished successfully, MISR registers contain the expected value and no critical failure is flagged. The critical failures may include LBIST failures, MBIST MBEs, MBIST SBEs exceeding the maximum tolerated number (<= 8 due to MEMU buffer size) and self-test failures.[end]

Note: See the "Off-Line Self-Test Sequence" section in the SPC570Sxx Reference Manual for details about test sequencing and completion validation (see Section 6.1: Document management).

The STCU2, as well as LBIST and MBIST controllers, are themselves subject to failures, which may prevent self-tests from executing correctly (for example, no self-test execution, or execution of the wrong algorithm) or from correctly reporting error indications to FCCU and MEMU. For latent faults affecting LBIST execution and error reporting, checking the MISR register upon LBIST completion is considered sufficient. For MBIST only a pass/fail flag is provided (besides the collection of detected MBIST errors in the MEMU).

The following must be followed to improve the detection of latent faults, particularly those affecting correct MBIST execution:

- **Recommendation:** LBIST should be scheduled before MBIST since LBISTs also cover the logic running memory self-tests and the MEMU BIST error collection logic/buffers; this helps to detect latent faults responsible for the wrong or incomplete execution of memory self tests or wrong reporting of their results.
- **Recommendation:** The STCU2 CRC feature should be enabled to check that the signals exchanged between the STCU2 and MBIST/LBIST controllers are correct (for example, STCU2 commands and LBIST/MBIST responses).

Note: The expected signature depends on the sequence of tests. User can determine the expected signature by running the desired sequence of tests and reading the resulting CRC upon test completion. One signature must be computed for each test sequence (for example, one for the start-up test sequence).

As far as the STCU2 error reaction path is concerned, the following is given:

- **Assumption:** [SAG_VE512K_3_35] SW checks the integrity of the STCU2 Unrecoverable Fault/Recoverable Fault (UF/RF) error lines that signal the FCCU and the MC_RGM (UF only) via the fake error injection register interface provided by STCU2. Before running the test, FCCU and MC_RGM is configured in order not to cause undesired reaction.[end]
- **Recommendation:** During the execution of the safety function, software should disable the FCCU and MC_RGM reactions to STCU2 UF/RF error indications to avoid false trip to the safe state or interference in case of unexpected error indications.

The STCU2 provides a key-based mechanism to prevent unauthorized write accesses to its register interface. The integrity of such protection mechanism can be checked by running the following test: [end]

- **Assumption:** [SAG_VE512K_3_37] SW performs a write access to one of the STCU2 registers without providing the requested key pair and check for the generation of the expected transfer error.[end]

The STCU2 allows the execution of logic and memory BIST also during run-time upon SW request.

Note: During start-up, no safety function is executed and the start up time is supervised by the external WDT. The internal prescaler feeding both the STCU2 WDT and core logic can be checked by running an on-line test and checking its execution time.

- [SAG_VE512K_3_38] On exiting from a functional reset, software checks the status of the STCU2 to verify there are no running BISTs nor any hardware aborted tests.[end]

Note: BISTs still running after a functional reset are the result of incorrectly handled hardware abort requests by the STCU2 that occurred while on-line BISTs were executing.

- [SAG_VE512K_3_39] If STCU2 interrupt capabilities are used to notify end of test session execution, application handles the case of missing interrupt(s) (for example, by supervising test execution time or periodically polling STCU2 status (checking STCU2_RUNSW[RUNSW], or STCU2_INT_FLG[MBIFLG] (for MBIST) and STCU2_INT_FLG[LBIFLG] (for LBIST)).[end]

3.2.8 MEMU initial checks

MBISTs report detected faults to the same MEMU reporting block used for System RAM ECC failures. Errors are in general distinguished between single-bit and multi-bit. However, it is not guaranteed that single-bit errors found in different steps on the same address are reported as multi-bit errors

Assumption: [SAG_VE512K_3_40] Software checks after MBIST execution whether two reported SBEs belong to the same address, which constitutes an MBE.[end]

Recommended: The application software can write known error addresses into the MEMU reporting table to prevent reporting of those errors to the FCCU in case the addresses are accessed again.

3.2.9 Flash memory configuration and tests

SPC570Sxx provides up to 512 KB of programmable non-volatile (NVM) flash memory with ECC which can be used for instruction and/or data storage.

Assumption: [SAG_VE512K_3_41] To detect failures where a wrong or multiple selection targets a different block while programming, application SW configures flash memory blocks as read only when not the target of a write operation.[end]

Note: See the “Program software locking” section in the “Embedded Flash Memory” chapter of the SPC570Sxx Reference Manual for details (see Section 6.1: Document management).

The flash memory array integrity self check detects possible latent faults affecting the flash memory array, including potential data retention issues, or the logic involved in read operations (for example, sense amplifiers, column mux’s, address decoder, voltage/timing references). It calculates an MISR signature over the array content and thus validates the content of the array as well as the decoder logic. The calculated MISR value is dependent on the array content and must be validated by software.

AI check does not bypass ECC flash logic. In case of SBEs occurring during the execution of the AI check the test still completes with the correct result (signature) being the SBEs corrected by the ECC logic.

Single bit correction reporting still occurs in the FLASH_MCR[SBC] bit and the FLASH_ADR during AI if FLASH_UT0[SBCE] = 1.

The AI breakpoint feature allows to break the Array Integrity Check execution if an error event is a single bit correction. Array Integrity Check can be resumed by the application after verifying the source of the error and clearing the respective status bit (for example, MCR[SBC] or MCR[EER]).

Assumption: [SAG_VE512K_3_42] The application software runs the flash memory AI at start-up to detect possible latent faults.[end]

Note: See the “Array Integrity Self Check” section in the SPC570Sxx Reference Manual for details (see Section 6.1: Document management).

In the event of a user-detected single-bit correction through user reads or an array integrity check, a margin read may be done to check for a possible second bit failing within the selected margin levels.

Margin reads are completed at voltages that differ from the normal read voltage and life expectancy of the flash memory bitcells are impacted by the execution of margin reads. Doing margin reads repetitively results in deterioration of the flash memory array, and shorten expected life when compared to normal read levels.

Assumption: [SAG_VE512K_3_43] A margin read test should be executed after a new single-bit error correction has occurred in flash memory. The margin read test does not need immediate execution, but it needs to be run within the next few trip cycles. Multiple single-bit errors can be the first indications of a data retention problem that could have the potential of causing multi-bit errors. [end] The MEMU can be used to store the address of the location reporting the error event.

Note: ***Implementation hint****: Refer to the SPC570Sxx Reference Manual's "User margin read" section of the "Embedded Flash Memory (c55fmc)" chapter for details (see Section 6.1: Document management).*

3.2.10 Voltage monitor configuration

To assist in maintaining functional safety, the Power Management Controller (PMC) monitors various supply voltages of the SPC570Sxx device. The "POR and voltage monitors description" table in the "Power management" chapter of the *SPC570Sxx Reference Manual (see Section 6.1: Document management)* shows a detailed list of the LVDs and HVDs embedded in the SPC570Sxx.

Apart from the self-test, the use of the PMC for ASIL D applications is transparent to the user because the operation of the PMC is automatic (see SAG_VE512K_3_44).

The PMC BISTs automatically execute during start-up, but the LVDs and HVDs are disabled until after the testing has completed.

PMC failures primarily report to the MC_RGM. Since safety-relevant voltages have the potential to disable the failure indication mechanisms of the SPC570Sxx (the FCCU and its error out signals), their error indication directly causes a transition into a safe state by reset. Additionally, LVDs and HVDs also report errors to the FCCU, but under the recommended configuration (MCU reset by MC_RGM enabled) this is irrelevant.

Assumption: Software does not disable the direct transition into a safe state due to an over- or under-voltage indication.

The user can, at their own risk, disable the direct triggering of resets by the MC_RGM and rely on FCCU reactions to over-voltage and under-voltage, even allowing IRQs as the reaction. In general, the FCCU reaction (clocked by the IRCOSC) takes more time than the MC_RGM reaction (asynchronous), so there is an increased probability that a fast voltage drop can cause a brownout condition on the device before a reaction can occur. Especially if IRQs are selected as the reaction there is no guarantee that any Diagnostic Coverage (DC) for over-voltage or under-voltage detection exists, and the safety analysis (FMEDA) of the MCU, is not valid with respect to this failure mode.

To check the LVDs and HVDs for latent fault which could influence their correct triggering in under-/overvoltage situations, the 2 tests are assumed to be executed by software during startup.

Assumption: [SAG_VE512K_3_44] Reference voltages, and input voltages of LVDs/HVDs, are acquired using the ADC. The measured values are compared with the expected ADC values. The application software is tested.[end]

Note: *This is to check that the voltage supervised by the LVD/HVD is actually the correct one and not influenced by opens or shorts in such a way that it never could cross the LVD/HVD threshold.*
See the SPC570Sxx Reference Manual's "Analog-to-Digital Converters (ADC) Configuration" chapter and the "SARADC_B analog test channel assignment" table for details. The LVDs/HVDs are monitored by 'SARADC_B input channels' 104, 105, 109, 120, 121 and 124 to 127 (see Section 6.1: Document management).

Assumption: [SAG_VE512K_3_45] Software initiates a self-test of LVD/HVD comparator.[end]

Note: This is to check that an LVD/HVD triggers at the approximately correct value.
See the SPC570Sxx Reference Manual's "Power Management Controller digital interface (PMC_dig)" chapter section "Voltage Detect User Mode Test Register (VD_UTST) (see Section 6.1: Document management).

3.2.11 Temperature monitoring configuration

The SPC570Sxx supports a temperature sensor to detect over-temperature conditions. The temperature sensor can be configured to signal over-temperature faults digitally to the FCCU. It can also provide an analog measurement of the temperature using SARB input channel 120.

To set a proper threshold the user must consider the maximum operating junction temperature (see the *SPC570Sxx Data Sheet* for the temperature sensor accuracy and maximum junction temperatures).

Assumption: [SAG_VE512K_3_46] Application software configures the FCCU to react to over-temperature faults indicated by the temperature sensor.[end]

3.2.12 Clock monitoring configuration

Clocks are supervised by the Clock Monitoring Units (CMUs). The CMUs are driven by the 16 MHz internal reference clock oscillator (IRCOSC) to ensure independence from the monitored clocks. The CMUs flag errors associated with conditions due to clocks being out of programmable bounds, and loss of reference clock. If a supervised clock leaves the specified range for the device, an error signal is sent to the FCCU. SPC570Sxx includes the CMUs as shown in the "Clocking" chapter of the *SPC570Sxx Reference Manual*. It is (see *Section 6.1: Document management)* the responsibility of the software to verify that the IRCOSC and XOSC are valid before starting the CMUs.

Assumption: [SAG_VE512K_3_47] The CMU frequency thresholds is configured for high and low limits which are used to compare with the MCU operating frequency.[end]

Assumption: [SAG_VE512K_3_48] The potential exactness (or the required inexactness) of the CMU thresholds are taken into account for both the IRCOSC and clock, or clocks, being monitored.[end]

*Note: **Implementation hint:** For example, for the upper threshold user should target CLKnominal_freq + CLKacc% and convert it into the number of IRC cycles in the worst case (slowest possible IRCOSC), for example IRC_freq = IRCnominal_freq – IRCacc%. The opposite applies to the lower threshold.*

Assumption: [SAG_VE512K_3_49] For ASIL D applications, the use of the CMUs is mandatory. If the related modules are used by the application safety function, the user verifies that the CMUs are not disabled and their faults are managed by the FCCU. The FCCU's default configuration does not manage the CMU faults, so it is configured accordingly.[end]

Assumption: [SAG_VE512K_3_50] Application software is checked by the configuration of the CMU once after programming.[end]

Assumption: [SAG_VE512K_3_51] Once after the boot, the application measures the CLKMT0_RMN frequency (IRCOSC) with CLKMN0_RMT (XOSC) as reference clock exploiting the CMU frequency measurement feature. To detect failure of the IRCOSC, the application software utilizes the CMU's frequency meter to read the IRCOSC frequency and compare it against the expected value of 16 MHz[h][end]

Assumption: [SAG_VE512K_3_52] After start-up, the CMU reaction path is tested by modifying the CMU threshold.[end]

Assumption: [SAG_VE512K_3_53] To detect delays in clock mode switching and lost clock switching, each time a new frequency or clock is selected, the software ensures the CMU is reprogrammed with the new expected clock frequency minimum and maximum values within the FTTI.[end]

Note: The frequency range of the CMU must be increased before switching clock modes. The requirement is to program the CMU with the correct minimum and maximum values for the new frequency soon after the switch.

Recommendation: The application may perform a test once per FTTI to verify proper IRCOSC operation.

3.2.13 System clock availability

At start-up, the CMUs are not initialized and the IRCOSC is the default system clock. Stuck-at faults on the external oscillator (XOSC) are not detected by the CMUs at start-up since the monitoring units are not initialized and the SPC570Sxx is still running on the IRCOSC.

Assumption: [SAG_VE512K_3_54] The software verifies that the clocks are valid by checking the state of the following:[end]

1. MC_ME_GS[S_XOSC] = 1, verifies valid XOSC
2. MC_ME_GS[S_IRC] = 1, verifies valid IRCOSC
3. The quality of the IRCOSC frequency is determined by clock metering and measuring the IRCOSC against the XOSC (see the *SPC570Sxx Reference Manual's* "Clock Monitoring Unit (CMU)" chapter for details see *Section 6.1: Document management*.
4. Based on measurement from 3, software updates the user trim bits of the internal oscillator (IRCOSC_CTL[USER_TRIM]).
5. Enable CMUs since we have valid XOSC and IRCOSC
6. MC_ME_GS[S_PLL0] = 1 and MC_ME_GS[S_PLL1] = 1, verifies valid PLL0 and PLL1 outputs

Assumption: [SAG_VE512K_3_55] Software checks that the system clock is available, and sourced by the FMPLL (PLL1), before running any safety element function or setting the FCCU into the operational state. [end]

3.2.14 Clock Generation Module (MC_CGM)

The CMUs are the main mechanism used to check the integrity of MCU clocks, but other indirect measures like delayed lockstep, fault tolerant communication protocols and replicated usage of peripherals may also be used. The following assumptions are necessary to cover the clock failures that escape these safety mechanisms which can potentially lead to the failure of specific modules.

h.Nominal frequency of the IRCOSC is 16 MHz, but a post trim accuracy of $\pm$6% over voltage and temperature must be taken into account.

Assumption: [SAG_VE512K_3_56] The sample time for the SARADC is at least one clock cycle longer than the minimum time required. This avoids clock glitches on the SAR clock from affecting sampling.[end]

Assumption: [SAG_VE512K_3_57] Detecting CLKOUT failures is demanded to the application and it is application dependent.[end]

Assumption: [SAG_VE512K_3_58] To detect PSI5 reception failures due to a clock glitch, PSI5 use the three bit CRC included in the protocol.[end]

3.2.15 FMPLL generated clocking

SPC570Sxx provides dual frequency modulated PLLs (FMPLL) for separate system and peripheral clocks.

Each FMPLL provides a glitch-free and fast clock to the SPC570Sxx and provides a loss of lock signal that is routed to the FCCU.

To reduce the impact of glitches stemming from the XOSC, the FMPLL should be used as the system clock.

Assumption: [SAG_VE512K_3_60] Application software ensures that the system is using the FMPLL (PLL1) clock as the system clock before running any safety functions, or before FCCU status is set to fault-free. [end]

Assumption: [SAG_VE512K_3_61] Application configures the FCCU to react to loss of lock of both FMPLLs. [end]

CAN features mode in which it is directly clocked from the XOSC. For applications targeting ASIL D, using these clocking modes increases the risk of communication failures.

Assumption: [SAG_VE512K_3_62] Application software does not use CAN modules directly clocked by the XOSC, or the used fault-tolerant communication layer is capable of handling failures induced by clock glitches (for example, timing errors, sampling errors and complete failure of logic due to setup/hold time violations). [end]

3.2.16 Wake-Up Unit (WKPU) / External NMI

Assumption: [SAG_VE512K_3_63]NMI is only used for error notifications or other uses where all dangerous failures are latent failures.[end]

Assumption: [SAG_VE512K_3_64]To test the analog filter of the WKPU for external NMIs, the user configures the NMI during startup to cause only a critical interrupt, then triggers the external NMI and checks that the critical interrupt occurred.[end]

3.3 Run-time checks

During the execution of the safety function, application software is assumed to perform a set of tasks to support the detection of random hardware failures and transition the device to a safe state in case of a failure. This section collects the assumptions software has to fulfill during run-time.

3.3.1 General requirements

The safety concept does not protect against spurious subtle timing changes (for example, due to the XBAR not parking on the safety relevant master due to other accesses). Thus, such subtle timing must not be relied on.

Assumption: [SAG_VE512K_3_65] During the development of safety-relevant software, counting clock cycles is not used (for example, relying on the execution time of core assembler instructions to measure time).[end]

Assumption: [SAG_VE512K_3_66] If independent data paths to or from any, software uses them redundantly to read or write safety related data.[end]

An independent data path exists to access the two PBRIDGEs and application software should use the peripheral set redundantly. In this case, failures in one of the data paths is detected by application level checks (for example, by comparing data provided by two redundantly used peripherals when each is attached to a different PBRIDGE).

The data path from any master to the AIPS (including it) is also covered by dedicated HW safety mechanisms consisting of AIC and E2E EDC. This is particularly important for peripherals mapped to one AIPS only, particularly INTC and DMA. The only component not covered is the peripheral IO Mux on the read data path as the ECC computation is done north of it. However, permanent faults in this component can be detected when executing e.g. registers CRC. Transient faults require executing reads twice and performing readback after each write to safety critical registers.

Note: The "Block diagram" figure in the SPC570Sxx Reference Manual shows the peripheral split between the two peripheral bridges (PBRIDGEA, PBRIDGEB) (see Section 6.1: Document management).
Section 3.3.14: I/O and Peripheral Bridge gives additional detail about using safety relevant I/O.

Assumption: [SAG_VE512K_3_67] A safety mechanism is implemented at application level to detect critical timing failures leading to violation of application timeouts.[end]

*Note: **Implementation Hint**: SWT can be used to satisfy this requirement.*

Machine check exceptions of the safety core are directly forwarded to the FCCU's "Safety Core Exception" input.

Recommendation: Due to the more comprehensive information available in the exception handler it is recommended to handle machine check exceptions in the exception handler and not use the FCCU mechanism.

Assumption: [SAG_VE512K_3_68] Other exceptions, which are not directly forwarded to the FCCU (for example, Data Storage, Alignment), must be handled by the core itself. This assumption is considered only for exception considered safety relevant by the application.[end]

Note: SPC570Sxx Reference Manual's "Core e200z0hn2p Description" chapters and the "Exceptions" sections of each chapter for details on core exceptions (see Section 6.1: Document management).

3.3.2 CRC of configuration registers

The CRC unit offloads the core in computing a CRC checksum. There is one set of CRC registers to allow CRC computations in the SPC570Sxx device. The CRC unit should be used to detect accidental modifications of data in configuration registers by calculating its CRC signature and comparing it against a pre-calculated CRC.

Note: Some configuration registers, as those for clock and MCU mode configuration, are copied to the corresponding internal registers only when an event (for example, mode change) is triggered. The values of those configuration registers themselves have no effect. Additional measures are needed, along with CRCing, to ensure correct operation of the MCU.

Assumption: [SAG_VE512K_3_69] A periodic scan of the safety relevant configuration registers is executed once per FTTI. The CRC include configuration registers for which no other safety mechanisms exists, to ensure that the configuration has not changed due to a bit flip or a permanent fault.[end] A list of safety relevant registers is provided in the *SPC570Sxx Reference Manual's* “Functional Safety” chapter and the “Register safety classifications” table *(see Section 6.1: Document management).*

*Note: **Implementation hint:** The CRC of the configuration registers of the modules involved with the safety function should be calculated off line.*

At run time, the same CRC value must be calculated by the CRC module within the safety process time. To avoid overloading a core, the DMA module can be used to support the data transfer from the registers under check to the CRC module.

The result of the runtime computation is then compared to the off line one.

Assumption: [SAG_VE512K_3_70] Even though faults affecting CMU registers cannot lead directly to the violation of the safety goal, the configuration registers of the CMU used to monitor the clock source of the safety core (CMU1) is included into the periodic register CRCing scheme. This helps to reduce the number of possible CCFs in the safety relevant configuration registers.[end]

Assumption: [SAG_VE512K_3_71] The two CRC units are not specifically designed to be used redundantly (for example, to check each other); in fact no independency constraint has been imposed on the design. [end]

3.3.3 XBAR usage

The application software must check the XBAR configuration once after programming but it must also detect failures of the XBAR when safety-relevant functions are running.

Assumption: [SAG_VE512K_3_72] Application software checks the configuration of XBAR every FTTI.[end]

Assumption: [SAG_VE512K_3_73] Within the FTTI, application software detects failures of the XBAR configuration affecting system performance. [end]

The detection of failures of the XBAR configuration can be achieved as a combination of periodic read-back of the configuration registers and control flow monitoring using the SWT. The SWT is needed to cover those failure conditions leading to a complete lock-out of XBAR masters. The need for periodic configuration read-back depends on how stringent the control flow monitoring is implemented.

XBAR data and address lines are covered by E2E EDC. Some failures, particularly those affecting muxing logic, might introduce multi-bit errors on data and addresses.

Assumption: [SAG_VE512K_3_74] Safety analysis assumes that at least two transactions are affected (for example, at least two accesses are made by or to the safety relevant XBAR master(s) or slave(s) within each FTTI).[end]

3.3.4 System Memory Protection Unit (SMPU)

The SMPU provides memory protection at the XBAR. A failure in the SMPU can change the behavior of the SMPU (for example, enabling or disabling the protection), resulting in unauthorized access to the protected data, or leading to unexpected access violations and data storage exceptions.

The protection against such failures is given by the read-back of the configuration registers, together with the exception handler and the SWT if the exception handler cannot execute. SWT and exception handlers are assumed to cover cases when accesses to system resources are unexpectedly prevented to authorized masters. On the other hand, SMPU failures resulting in missing memory protection are considered critical only if coupled with other failures causing the undesired access to protected data (notice that systematic failures, besides random HW failures, can lead to this scenario).

Safety analyses are performed under the following assumptions:

- **Assumption:** [SAG_VE512K_3_76] FMEDA assumes that 90% of region descriptors are usually used during the execution of safety tasks.[end]
- **Assumption:** [SAG_VE512K_3_77] SMPU is enabled approximately 99% of the time during the execution of safety tasks.[end]

3.3.5 Platform flash memory controller (PFLASHC)

The PFLASHC configuration controls aspects of read wait states, port arbitration, prefetching policy, master access.

Some of these failures only cause performance reductions, so they can be covered by the SWT while others might have adverse effects on the data.

Assumption: [SAG_VE512K_3_78] Wrong wait state configuration can cause bad data reads, but it can be covered by ECC assuming at least four reads within the FTTI.[end]

Other configuration failures, such as master access and safe remapping, only cause MultiPoint Failures (MPF), so one readback for trip time is sufficient.

Assumption: [SAG_VE512K_3_79] After configuring the PFLASH controller, the application reads back the PFLASH controller registers and compare them with the expected values every FTTI.[end]

3.3.6 Flash memory

Flash memory program and erase

Flash memory program/erase operations are stopped in the event of a fault event (for example, no flash sector selected, or elevated current draw).

Assumption: [SAG_VE512K_3_80] For program operations, only the address specified by an interlock write determines the partition being written. An interlock sequence is used to prevent accidental programming of flash memory.[end]

Assumption: [SAG_VE512K_3_81] A software safety mechanism is implemented to ensure the correct termination of any program/write operation of the flash memory.[end]

Even when flash memory signals the correct termination of programming operations, there is still the chance that flash memory content is incorrect due to failures of the flash memory write path and programming logic.

Assumption: [SAG_VE512K_3_82] To ensure that the content of a write operation to flash memory is correct, software read back the data that was written and compare it with the expected data. This checks the integrity of the programmed data. This test should execute after every program or erase operation.[end]

Note: In addition, this test prevents the return of stale data from the PFLASHC minicache.

Flash memory multi-bit error

Non-volatile flash memory is protected with a single-bit error correction/double-bit error detection (SEC/DED) ECC scheme.

If ECC correction occurs, a software test must be run to verify that the ECC correction was done on a word affected by an SBE rather than an MBE pattern (for example, random word). MBE can be due to control logic, which may be global or shared by the blocks of each Read-While-Write (RWW) partition. If the MBE is due to global control logic it is sufficient to read-back patterns from any flash location. If the MBE is due to shared control logic it is necessary to exercise the affected RWW partition (or test each RWW partition).

Assumption: [SAG_VE512K_3_83] According to the specific flash memory usage by application software, a software test should be implemented to discriminate between SBE and multibit errors introduced by permanent failures in Flash control logic (for example, read pump, read timing, V_{ref}, and so on). This test consists on a read-back pattern to be run in case of SBE event.[end]

Assumption: [SAG_VE512K_3_84] The read-back patterns can be compared with the expected values, or the application can just force the read of a number of patterns sufficient to trigger other ECC error corrections/detections revealing the actual nature of the fault (in fact permanent control logic failures typically affect multiple read operations).[end]

EEPROM emulation

The SPC570Sxx provides four blocks (4 x 8 KB) of the flash memory for EEPROM emulation. ECC events detected on accesses to the EEPROM flash memory blocks are not reported to the MEMU. SBEs are corrected, but not signaled to the MEMU. MBEs are replaced by a fixed word (for example, an illegal instruction) and are also not forwarded to the MEMU.

Assumption: [SAG_VE512K_3_85] Software using EEPROM for storage of information uses its own information redundancy (for example, CRCs) to detect incorrect data returned from the EEPROM emulation.[end]

3.3.7 PRAMC configuration

PRAMC provides a certain level of configurability on accessing the RAM. Faults in the PRAMC configuration registers may change PRAMC port priority and, most important, read wait states. Changes in port priorities, or more wait states, can have an impact on the overall performance, which is typically covered by using the SWT. However, fewer wait states can lead to multi-bit errors that are detected by E2E ECC with only low to medium effectiveness.

Assumption: []SAG_VE512K_3_86] Application software checks the configuration of the PRAMC every FTTI.[end]

The periodic check of PRAMC configuration can be avoided if the following assumption is satisfied:

- **Assumption:** [SAG_VE512K_3_87] Application software performs at least two accesses to the SRAM within the FTTI, or a simple test could be performed (for example, a test based on write and read-back performed every FTTI).[end]

3.3.8 RAM

RAM is protected from failures by ECC logic with various hardware mechanisms. To increase the coverage against MBE certain SW measures have been assumed.

Assumption: [SAG_VE512K_3_88] Data in RAM that is supposed to survive reset needs to be CRC encoded after each modification. This allows the RAM content to be verified after each reset.[end]

Error Correcting Code (ECC)

Some failure modes of the RAM, for example failures affecting common part of the RAM structure, cause data to be read from all location as All-0 or All-1(i).

In such a case if a data is read out of the RAM, an event should be detected by the ECC which perceives All-X code-word as invalid code.
Anyway RAM address is included in the ECC calculation to increase the diagnostic coverage in case of addressing failures (see “End to end (E2E) protection on the data path and memory ECC” section in the “Functional Safety” chapter of the SPC570S50Lx Reference Manual (see *Section 6.1: Document management*), so, due to this hardware architecture, there are address locations out of which reading All-X word is valid and no ECC event is triggered(j). Approximately there is one of these addresses out of 256 ones.

Assumption: [SAG_VE512K_3_130] There is no special handling of All-X words for ECC that includes addresses in the ECC code-bit calculation. They can be valid, correctable or uncorrectable words depending on the address. [end]

Consider a permanent All-X failure mode:

- If there is a transaction to a “normal address”, this failure mode is detected by the ECC which perceives the All-X code-word as invalid (either single bit error or correction is triggered).
- If there is a transaction to an “All-X address”, this failures mode is not detected by the ECC or other means (All-X code-word is valid for “All-X address”).

The permanent All-X failure mode is not detected by the ECC if an application reads only All-X addresses of the RAM. This is a pure theoretical case because a real application doesn't read only such All-X addresses, but reads different parts of the RAM including (and mainly) normal addresses.

Assumption: [SAG_VE512K_3_131] Within the FTTI, application software reads, in each RAM block, two addresses that are known to cause an uncorrectable error in case the memory globally shows an All-0 or an All-1 state. [end]

i. All-0 and All-1 are referenced generically as All-X.

j. For the sake of simplicity, let us call “All-X addresses” the ones which perceive All-X as valid and “Normal addresses” the remaining ones.). Approximately there is one of these addresses out of 256 ones.

Real application continuously reads several RAM locations (both normal and All-X addresses) in different RAM blocks. Some of these transactions are directed to addresses which trigger uncorrectable error in case of All-X events.

As a consequence, in most of the cases, the assumption above is satisfied by the application accessing RAM during the normal code execution without any additional overhead in terms of both coding and timing.

The program, in *Section 5: Candidate address for testing All-X issue* (see *Section 6: Further information*), calculates the list of addresses, which trigger uncorrectable errors if an All-0 (or an All-1) failure occurs, by a linear search from a start address. These locations are used to verify the presence of a global All-0 (or All-1) error. The user can verify that the application software reads, once per FTTI, at least one location to detect a global All-0 errors and one location to detect for All-1. In that case additional readings of previous assumption are not necessary.

Repair logic

Memory repair faults can cause a partial shift of the word. This failure mode can affect one word or an entire column depending on the type of failure in the repaired column (single bit in array or column periphery). If a read operation is performed first, this results in a MBE (white noise model). In the event a write operation to a specific address was executed first after this error resulted, any subsequent read of that same address either correct or result in a SBE.

Assumption: [SAG_VE512K_3_89] To guarantee coverage for MBEs, it is assumed at least four reads on different addresses per RAM block within FTTI occurs. This provides sufficient diagnostic coverage for column repair with ECC.[end]

Error reporting

The MEMU collects and reports error events associated with ECC logic used on system RAM, peripheral RAM and flash memory. The MEMU stores the addresses where ECC errors occurred. The MEMU also reports whether the error is correctable vs. uncorrectable. Uncorrectable errors cause a report to the FCCU.

Correctable errors include:

- Single-bit error in the data part that is detected via ECC for a system RAM, peripheral RAM or flash memory
- Single-bit error in the data part that is detected via MBIST on any RAM

Uncorrectable errors include:

- Multi-bit error that is detected via ECC for a system RAM, peripheral RAM or flash memory
- Multi-bit error that is detected via MBIST on any SRAM
- Addressing errors and unused data bit errors detected by ECC logic

Assumption:[SAG_VE512K_3_200] A software test shall be implemented that will fetch the error address of corrected errors from the MEMU and assess the nature of the fault by writing and reading back a few patterns (for example, two's-complemented patterns) to the faulty location. In case the fault is permanent, the device shall be reset to execute MBIST. The software test shall be executed within the FTTI after a new ECC error correction in RAM is reported.[end]

Assumption: [SAG_VE512K_3_132] During operation, if the MEMU contains two entries for the same address of which is part of a memory for which ECC syndromes are reported, SW checks whether one of the two syndromes is FFh. If so, this entry should be deleted. [end]

3.3.9 Interrupt management

Since the INTC is a replicated module, no software action is needed to detect faults inside this module.

No specific hardware protection is provided against spurious or missing interrupt requests caused by Electromagnetic Interference (EMI) on the interrupt lines or bit flips in the interrupt registers of the peripherals. Applications not resilient against such errors must include detection or protection measures.

The following software measure needs to be considered:

- **Assumption:** [SAG_VE512K_3_93] Check for functional effects of wrong IRQ processing. For this a functional safety application can periodically check for effects such as buffer over/underflow and communication errors. This check should be run periodically within the required fault tolerant time interval (FTTI).[end]
- **Assumption:** [SAG_VE512K_3_98] Interrupts with known periodicity are checked by the SWT triggering routine for the event occurrence.[end]

*Note: **Implementation Hint:** This can be achieved, for example, by the ISR setting a flag which can be checked and reset by the SWT triggering software routine*

- **Assumption:** [SAG_VE512K_3_97] Unused interrupt vectors points, or jumps, to an address which is illegal to execute, contains an illegal instruction, or in some other way causes detection of their execution.[end]

3.3.10 Reset Generation Module (MC_RGM)

MC_RGM can trigger interrupts or request a transition to SAFE mode, if the MC_RGM is configured to do so.

Assumption: [SAG_VE512K_3_99] To detect spurious interrupts or transition requests to SAFE mode, the interrupt handler, for IRQs coming from the MC_RGM, check that the shown source is one which was configured to cause such a request from the MC_RGM (for example, compare the MC_RGM_FES register to the expected MC_RGM_FERD and MC_RGM_FEAR register contents). [end]

Note: If the MC_RGM triggers a transition request to SAFE mode, no interrupt is triggered by the MC_RGM. An interrupt is triggered by the MC_ME.

3.3.11 Detection of unwanted resets

MC_RGM allows triggering individual resets for MCU modules (for example, using the Peripheral Reset Registers (RGM_PRST[*n*]). This can be prohibited by using the access control mechanisms of the SPC570Sxx such as the MPU or PAC. In case those are not used, and also to detect spurious resets caused by SEE, the following describes how such spurious resets can be detected.

Assumption: [SAG_VE512K_3_100] To detect unwanted reset of these modules, software and hardware counter measures can be applied. [end]

3.3.12 Periodic interrupt timer (PIT)

If the PIT is used by the application software, a checksum of its configuration registers using the CRC must be calculated and compared with the expected CRC to verify proper PIT configuration. The PIT could be used for various safety relevant timing operations. Since the safety function could be impacted by a PIT failure the integrity of the PIT needs to be verified.

Assumption: [SAG_VE512K_3_101] PIT operations (for example, the number of periodic triggers) are checked and compared against the expected values every FTTI. This includes comparing the number of generated triggers within a given time period using another timer as reference and, if not covered by the previously mentioned check, reading back, or CRCing, PIT configuration (check for enabled channels, load values, so on). To avoid CCFs it is important that the software check be triggered by an independent (from PIT) timer or supervised by a watchdog clocked by an independent time source (for example, SWT).[end]

3.3.13 System timer module (STM) usage

[SAG_VE512K_3_102] Since a failure in the System Timer Module (STM) can cause a violation of the safety goal, one of the two assumptions below is satisfied.[end]

Assumption: [SAG_VE512K_3_103] At every STM interrupt, the IRQ handler compares the elapsed time since the previous interrupt versus a free running counter to check whether the interrupt time is consistent with the STM setting, or[end]

Assumption: [SAG_VE512K_3_104] The STM IRQ handler is under SWT protection.[end]

A second timer may be used to measure STM interrupts and compare with the STM measured time.

3.3.14 I/O and Peripheral Bridge

Peripheral Bridge module is protected by E2E EDC (see *Section 3.4.2: AIPS protection mechanism*).

Assumption: [SAG_VE512K_3_105] The integrity of safety relevant periphery is mainly ensured by application-level measures (for example, connecting one sensor to different I/O modules, sensor validation by sensor fusion, and so on) which are supported at the hardware level by a replication of all potentially safety-relevant I/O with appropriate freedom of interference but no hardware checkers. This replication starts at the I/O bridges (including them).[end]

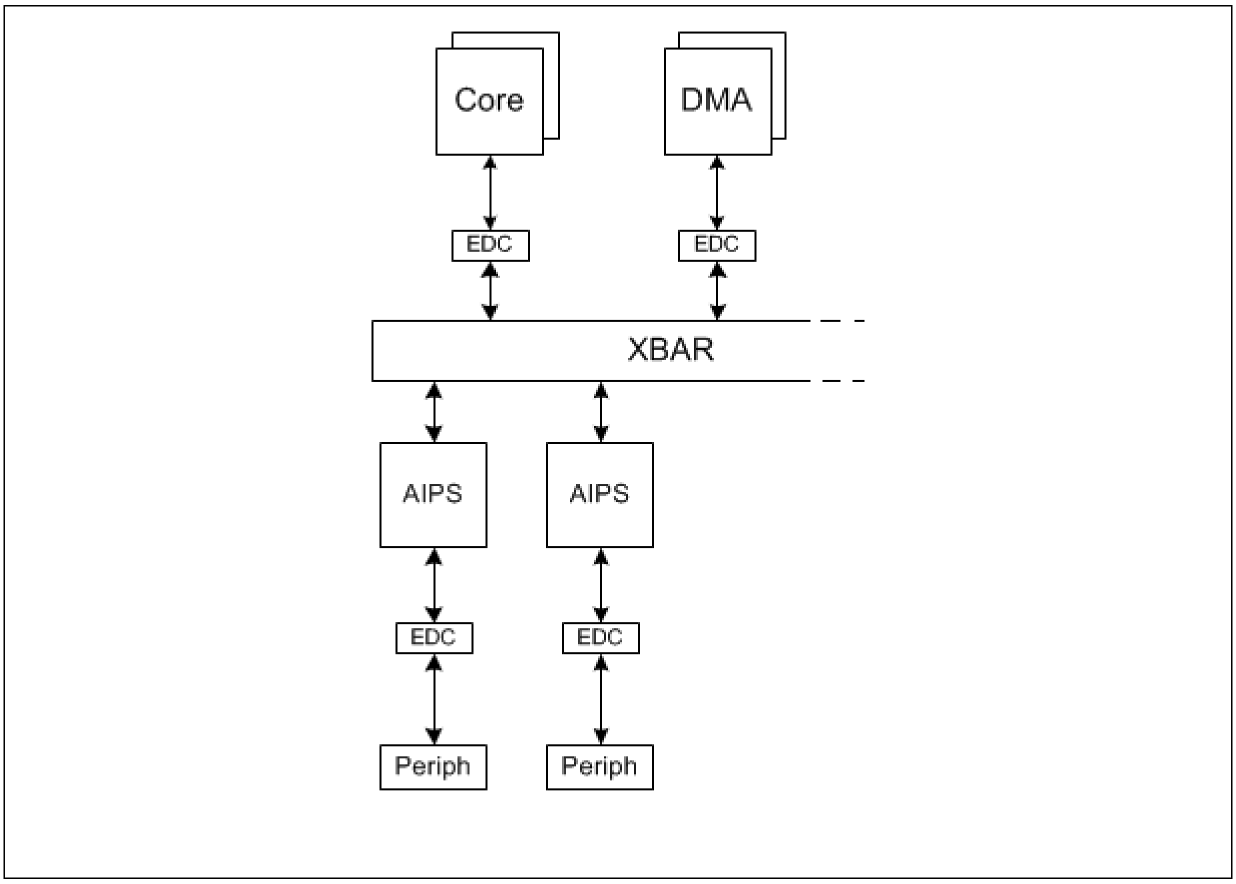
Safety relevant peripherals are assumed to be used redundantly in some way. Different approaches can be used, for example by getting replicated input, (for example, connect one sensor to two DSPIs) or by cross-checking some I/O operations with different operations (for example, using sensor values of different quantities to check for validity).

Users can choose the approach that better fits their needs.

To allow a safety application to make redundant use of all I/O peripherals, the peripherals have two instances and each instance is connected to a different peripheral bridge (PBRIDGE). The arrangement of the I/O peripherals onto two PBRIDGEs allows redundant use of peripherals while limiting CCFs.

The SPC570Sxx architecture allows making redundant use of the communication peripherals like LINFlexD, DSPI. *Figure 5* shows the distribution of the SPC570Sxx peripherals.

Figure 5. SPC570Sxx peripheral allocation



As a usage example, if an application needs to use LIN communications protocol, two different LINFlexD modules may be used; one connected to the PBRIDGEA and one the PBRIDGEB.

Note: In case of high bandwidth The safety concept for high bandwidth communication controllers (for example, FlexRay, FlexCAN and FEC (Ethernet)) is not based on the redundant usage of multiple modules but rather on the implementation of a fault tolerant protocol. This is the reason why they are typically not split over the AIPSs. Section 3.3.18: Communication peripherals discusses in more detail the usage of these types of communication controllers.

Assumption: [SAG_VE512K_3_106] Comparison of redundant operation is the responsibility of the application software. [end]

Note: Additional details can be found in the “IO safety” section in the “Functional Safety” chapter of the SPC570Sxx Reference Manual (see Section 6.1: Document management).

There are modules, particularly on-platform peripherals such as INTC and eDMA, which are connected to a single PBRIDGE. For these modules, the integrity of accesses to their register interface is not guaranteed by the PBRIDGE replication but the e2e EDC protocol and by the AIC.[end]

Assumption: [SAG_VE512K_3_108] To ensure safe usage of modules which do not exist redundantly and are connected to only one PBRIDGE[k], one of the following is true for each user-visible (via the PBRIDGE*n*) register:[end]

- The register is not relevant for the safety goal of the application.
- The register has a constant value (typically a configuration register) which is periodically checked for correct value (for example, by CRCing).
- Wrong values written into the register are detected by other safety measures.

Furthermore, for reading such registers the following is obviously true (due to the single nature of the data source):

- Values read from such registers are not guaranteed to be free of SPFs

Assumption: [SAG_VE512K_3_111] When software reads a safety-relevant value from a peripheral, that value is read twice in a row, then the two read values are compared. This helps detect transient errors in the IO Mux for non-redundant peripherals.[end]

3.3.15 System Integration Unit Lite (SIUL2)

Since the SIUL2 PBRIDGE interface is unique, its failure, and particularly of its register protection module, may impact redundant I/O functionalities leading to CCF.

Assumption: [SAG_VE512K_3_112] When the SIUL2 is used for the implementation of safety related I/O functionalities, application level redundancy is such that it covers at least 60% of failures introduced by the REG_PROT unit that can result in blocked writes (lost updates) to non-locked registers (mostly GPO data registers). An additional software test runs to detect such a failures mode.[end]

Assumption: [SAG_VE512K_3_113] To detect wrong or multiple addressing failures, the startup read-back of SIUL configuration registers is executed after all SIUL2 registers have been written. [end]

Assumption: [SAG_VE512K_3_114] If the SIUL2 is used to perform a redundant digital input or output (read two GPIs or write two GPOs), the application executes a periodic CRC of configuration registers that is used to detect decoder hard faults that lead to CCF on data reads or writes. [end]

3.3.16 Reading analog inputs

Acquisition of safety related analog inputs can be performed using two independent ADC modules redundantly.

The *dual read analog input* uses two analog input channels provided by two separate ADC modules to acquire a replicated analog input signal. Both ADC units acquire and digitize the two copies of a redundant analog signal connected to the inputs. In this configuration (if applied to all possible analog inputs), only half of the analog inputs are available to the application.

Assumption: [SAG_VE512K_3_116] Software compares the value sampled by the two ADCs and decide on their consistency (comparison has to take into account conversion differences and tolerances). [end]

k. e.g. SIUL

3.3.17 Software Watchdog Timer (SWT) usage

The objective of the Software Watchdog Timer (SWT) is to detect a defective program sequence when individual elements of a program are processed in the wrong sequence or period of time. Once the SWT is enabled, it requires periodic and timely execution of the watchdog servicing procedure. The service procedure must be performed within the configured time window, before the service timeout expires.

Assumption: [SAG_VE512K_3_117] These requirements apply to the SWT for ASIL D applications: [end]

- The SWT is enabled and configuration registers have to be protected against undesired accesses using one or multiple hardware mechanism implemented (e.g. SMPU, REG_PROT).
- The SWT time window settings is set to a value less than the FTTI. Detection latency is smaller than the FTTI.
- Before the safety function is executed, the software verifies that the SWT is enabled by reading the SWT control register (SWT_CR).

It is in general to be expected that software uses the software watchdog timer (SWT) to detect lost clocks or significantly slow clocks. Using the SWT to detect clock issues is a secondary measure since there are primary means for checking the clock integrity (for example, CMU).

SPC570Sxx provides the hardware support (SWT) to implement both control flow and temporal monitoring methods. If Windowed mode and Keyed Service mode (two pseudo random key values used to service the watchdog) are enabled, it is possible to reach a high effective temporal flow monitoring.

Assumption: [SAG_VE512K_3_118] It is the responsibility of the application software to insert the control-flow checkpoints with the required granularity according to application needs.[end]

Assumption: [SAG_VE512K_3_119] The SWT service procedure reload the down-counter with the expected time-out period.[end]

SWT can be configured to stop, or continue, running when the MCU is in STOP mode by configuring SWT_CR[STP]. If this SWT feature doesn't work as expected due to a fault, the safety function could be impaired (for example, SWT could trigger an unwanted reset while the device is in STOP mode).

Assumption: [SAG_VE512K_3_120] Current FMEDA assumes that STOP mode is not used in normal operations.[end]

3.3.18 Communication peripherals

Assumption: [SAG_VE512K_3_121] Communication over FlexCAN interfaces is protected by a fault-tolerant communication protocol (implemented by the operating system or the application]

Note: *Typically such a layer would contain an E2E CRC, a sequence counter, a sender ID and (if transmission loss needs to be prevented) an acknowledgment mechanism.*

Assumption: [SAG_VE512K_3_122] If safety relevant, FlexCAN are not clocked directly by the XOSC.[end]

Note: *Using the XOSC directly can expose the FlexRay or FlexCAN engine to glitches that would otherwise be filtered by the PLL. The User can still use the XOSC provided they implement* *safety mechanisms to detect the effects of glitches. These mechanisms can be part of the FT protocol itself.*

An appropriate safety software protocol should be utilized for any communication peripheral employed to meet ASIL D application requirements.

FlexCAN doesn’t have special safety mechanisms other than what is included into them by their protocol specs. The application software or operating system needs to provide the safety measures on top of the IP modules to meet safety requirements.

3.3.19 Temperature sensor

Assumption: [SAG_VE512K_3_124] During run-time, software reads temperature sensor voltage value from the ADC every FTTI. In case the measurement indicates an over-temperature condition, software triggers the appropriate reaction.[end]

3.4 Operational interference protection

To handle concurrent operation of software tasks with different or lower ASIL, the SPC57S50Lx provides safety mechanisms to prevent interference.

3.4.1 System Memory Protection Unit (SMPU)

The System Memory Protection Unit (SMPU) provides hardware access control for all memory references generated in a device. Using pre-programmed region descriptors (up to16 different regions) define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions (including those initiated by the eDMA controller) and evaluates the appropriateness of each transfer.

Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The SMPU implements a set of program-visible region descriptors that monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes represent the second dimension.

Assumption: [SAG_VE512K_3_125] The SMPU should be used to ensure that only authorized software routines can configure modules and all other bus masters (except the Safety core) can access only their allocated resources according to their access rights. [end]

3.4.2 AIPS protection mechanism

The peripheral bridges (PBRIDGE*n*) translate accesses on the switched AMBA bus (XBAR) to point-to-point accesses to the majority of peripherals on the SPC570Sxx. The peripherals connected to the PBRIDGEs are PBRIDGE slaves.

The PBRIDGEs implement an additional protection mechanism to support the requirement that all masters do not interfere with one another. The protection mechanism allows for protection of each slave from master accesses (for example, read/write or supervisor/user access).

Assumption: [SAG_VE512K_3_126] The application software configures the PBRIDGEs to define the access permissions for each slave module that requires access protection, unless protected by the mechanisms in sections *Section 3.4.1: System Memory Protection Unit (SMPU)* or *Section 3.4.3: Register protection (REG_PROT)*. [end]

PBRIDGE*n* should be configured to prevent any write access to the entire MC_RGM address space for all masters except the safety core and the DMA which are in lock step configuration.

3.4.3 Register protection (REG_PROT)

Accidental writes to configuration registers can affect the execution of the MCU's safety function and disable the safety mechanism due to their change. Register protection offers a mechanism to protect defined memory mapped address locations in a module against writes. The address locations that can be protected are module specific.

Register protection includes these distinctive features:

- Register protection restricts write accesses for the module under protection to supervisor mode only. This access restriction is in addition to any access restrictions imposed by the protected module.
- A register cannot be written once Soft Lock Protection is set. Soft Lock Protection can be cleared by software or system reset.
- A register cannot be written once Hard Lock Protection is set. Hard Lock Protection can only be cleared by system reset.

Recommended: It is recommended that only hardware related software (OS, drivers) runs in supervisor mode.

Assumption: [SAG_VE512K_3_128] For ASIL D applications, all configuration registers, and registers that aren't modified during application execution, are protected with a Hard Lock Protection. Configuration registers, and registers which have limited writes every trip time, are protected with soft-lock protection. [end]

Assumption: [SAG_VE512K_3_129] FMEDA assumes that configuration registers are locked at least 90% of the time, either by Soft Lock or Hard Lock Protection, to prevent unwanted modifications. [end]

Note: ***Implementation hint:*** *Most of the off-platform peripherals have their own REG_PROT. Each peripheral that can be protected through the REG_PROT has a Set Soft Lock bit in the Register Protection space. This bit should be asserted to enable the protection of the related peripheral.*

Each peripheral register that can be protected through register protection has a Set Soft Lock bit reserved in the Register Protection address space. This bit should be asserted to enable the protection of the related peripheral registers. Moreover, the Hard Lock Bit (REG_PROT_GCR[HLB] = 1) should be set for best write protection.

4 Functions of external devices for ASIL D applications

This section describes the external components needed to use with SPC570Sxx in a system for ASIL D applications. It is assumed that the system reacts safely to SPC570Sxx being in or entering all safe states.

It should be noted that the failure rates of external services are not included in the FMEDA of SPC570Sxx and have to be included in the system FMEDA by the user himself.

4.1 External reset output

SPC570Sxx has pin named external reset output (ESR0). The signal on this pin can be used as input to one, or more, external devices. the SPC570Sxx can thus reset the external devices.

Simplified working flow is:

1. The SPC570Sxx power-on
2. The SPC570Sxx resets the external device via the ESR0
3. The SPC570Sxx initializes itself (including the start-up tests)
4. The SPC570Sxx initializes the external devices
5. The safety function starts to execute

It is possible that an unwanted or spurious assertion of ESR0 may not be detected by the SPC570Sxx. This could cause the external device to get reset without any automatic detection by SPC570Sxx. Assumption is that countermeasures against this failure mode must be considered at system level.

Assumption: [SAG_VE512K_4_0] System level I/O safety measures have at least 99% DC against joint spurious reset of all external devices. [end]

4.2 High impedance outputs

System-level countermeasures have to be placed in order to bring the safety-critical outputs to their safe state (for example, by pull-up or pull-down resistors) when an output high-impedance is not considered safe. Normally this requirement is fulfilled by ensuring the error out pin(s) are pulled to the failure state. Additionally, users may drive pins (for example, CAN Tx pins) to levels that prevent interference with other parts of the system that are assumed to be independent.

Assumption: [SAG_VE512K_4_1] If a high impedance state on an output pin is not safe, pull-up or pull-down resistors need to be added to outputs that are safety-critical depending on application requirements for the SPC570Sxx during unpowered or reset conditions. [end]

4.3 External Watchdog (EXWD)

An external device, acting as supervisor of the operations, must provide a watchdog to cover common-cause failures of SPC570Sxx for ASIL D applications.

Assumption: [SAG_VE512K_4_2] An external watchdog exists to detect failures completely disabling the SPC570Sxx, including its safety mechanisms. [end]

The external watchdog detects CCFs, such as failure of the power supply. If a failure is detected, the external watchdog should move the system to a safe state within the FTTI.

Assumption: [SAG_VE512K_4_3] The EXWD is triggered periodically, either by the software providing the safety function on the SPC570Sxx or by a toggling protocol on the error output pin(s). [end]

Implementation of the watchdog communication between SPC570Sxx and the external device is up to the user (for example, communication via serial link, ethernet, via toggling pin, or via the FCCU error out signals).

Assumption: [SAG_VE512K_4_4] To avoid undetected reset cycling under rare circumstances the external watchdog is not reset by the MCU reset output. [end]

Note: There must be a signalling path from the safety software to the external system through which the software can confirm correct initialization. This is not automatically guaranteed by the FCCU_F[n] signals which communicate the status of the device independently from software. On the other hand, a different communications interface (such as a serial link) can be used to detect incorrect software initialization.

4.4 Power supply

Assumption: [SAG_VE512K_4_5] The device has been developed with the assumption that an external power supply of appropriate voltage is supplied (see the *SPC570Sxx Data Sheet's* for operating voltage specifications). All internal and external supplies are considered safety critical and are monitored for deviations beyond predefined thresholds.[end]

Assumption: [SAG_VE512K_4_6] External power supply is supervised for high voltage deviations as shown in *Table 4*. Required monitors for each power supply can be found in column “External Monitor Required”. [end]

Note: Voltage monitors are provided on the SPC570Sxx to monitor the internal supplies of the device. See table “POR and voltage monitors description” in the “Power management” chapter of the SPC570Sxx Reference Manual for a list of monitored supplies (see Section 6.1: Document management).

External LVD can be removed if the failure can be detected by other means, but the external HVD is necessary to prevent physical damage.

External supervision hold the system in its safe state if the external voltage is outside the allowed range.

Table 4. SPC570Sxx required external monitors

Name	Description	External Monitor Required
VDD_HV_ADC	High voltage external power supply for ADC module	HVD
VDD_HV	High voltage external power supply for I/Os, JTAG, and most analog modules	HVD

Assumption: [SAG_VE512K_4_7] External supervision hold the SPC570Sxx in a safe state if the external voltage is outside specification, and the MCU is protected against voltages over the maximum survivable level of the technology. [end]

Recommended: It is recommended to protect the MCU against voltage above the maximum survivable level of the technology (for example, using a Zener diode) to prevent destruction of the MCU.

Note: See the SPC570Sxx Data Sheet's "Absolute maximum ratings" and "DC electrical specifications" sections for power supplies requirements.

[SAG_VE512K_4_8] The SPC570Sxx embeds two types of voltage supervisors, Low Voltage Detect (LVD) and High Voltage Detect (HVD) monitors. Safety relevant voltages are supervised for voltages that are out of these ranges. Since safety relevant voltages have the potential to disable the failure indication mechanisms of the SPC570Sxx (such as FCCU, Pads, and so on) their error indication directly causes a transition into the safe state (for example, reset assertion). [end]

Some LVDs and HVDs are configurable and enabled by default. Application should not disable safety relevant LVDs or HVDs. See the "POR and voltage monitors description" table in the "Power management" chapter of the *SPC570Sxx Reference Manual* for the list of configurable LVDs and HVDs *(see Section 6.1: Document management)*.

4.5 Error Out Monitor (ERRM)

The FCCU has two external pins: FCCU_F0 and FCCU_F1. An external device must be connected to the FCCU pins (FCCU_F0 and FCCU_F1) to continually monitor the error output pins of the FCCU.

Recommended: [SAG_VE512K_4_9] The overall system needs to include measures to monitor the error output pin(s) of the SPC570Sxx and move the system into a safe state when an error is indicated.[end]

Recommended: [SAG_VE512K_4_10] If the SPC570Sxx is signalling a failure via its error output pin(s), other output pins can not be relied on. [end]

4.5.1 Both FCCU pins connected to external device

Depending on user selection, there are two different ways to interface to the FCCU:

- Both FCCU pins connected to the external device
- Only a single FCCU pin connected to the external device

The user can choose between these two FCCU configurations, depending on which best fits the hardware and software system.

Assumption: [SAG_VE512K_4_11] If both error out pins are connected to the external device, the external device itself checks both signals, taking into account that FCCU_F0 = $\overline{\text{FCCU\_F1}}$ for the fault free state.[end]

In this configuration the external device continuously monitors the output of the FCCU. Thus it can detect if the error out pins are not working properly. The advantage of the two pin configuration with respect to the single pin option is that it does not require any dedicated software.

*Note: **Implementation hint:** Monitoring the error output pins through a combinatorial logic (for example, XOR port) can generate some glitches. Oversampling these pins reduces the possibility that the glitches occur.*

4.5.2 Single FCCU pin connected to external device

In this configuration, only one of the FCCU pins is connected to the external device. If a fault occurs, the FCCU communicates it to the external device through the FCCU_F0 (or FCCU_F1)pin.

The functionality of FCCU_F0 (or FCCU_F1) can be verified in two ways on a system level by testing that FCCU_F0 (or FCCU_F1) can trigger the safe state of the system.
If only FCCU_F0 (or FCCU_F1) needs to be tested (without the rest of the shutdown path), this can be accomplished in the following way:

- FCCU_F0 (or FCCU_F1) output connected externally to a normal GPIO.

Assumption: [SAG_VE512K_4_12] After boot, but before executing the safety function, the functionality of FCCU_F0 (or FCCU_F1) pin is verified(l). [end]

As access to FCCU_F0 (or FCCU_F1) is shared between the FCCU and GPIO there is a failure mode where the multiplexing fails and FCCU_F0 (or FCCU_F1) becomes controlled by GPIO. To make this detectable, the respective GPIO needs to be configured to drive FCCU_F0 (or FCCU_F1) into the fail state.

Assumption: [SAG_VE512K_4_13] If an error indication protocol is selected which makes use of only one error out pin (for example, FCCU_F0), then SW configures any I/O MUXed to that pin to drive low before switching the pin to FCCU control [end] (see the *SPC570Sxx Reference Manual's* “Signal Description” chapter and the “I/O Signal Description table” for pin MUXing specifics) *(see Section 6.1: Document management)*.

The advantage of this configuration with respect to the configuration where two error indication signals are used, is that it can use an external device that does not compare the two signals.

Assumption: [SAG_VE512K_4_14] If the system is using the SPC570Sxx in a single error output pin mode, the application software configures the pins and pads neighboring the FCCU_F0 (or FCCU_F1) to use a lower drive strength. [end]

Using a lower drive strength on the pins near FCCU_F0 (or FCCU_F1) reduces the effects of simultaneously switching outputs on signal integrity. Software must configure the slew rate for the relevant pads in the Pad Configuration Register.

l. Since FCCU is a monitor, it is sufficient to verify the FCCU_F0 and/or FCCU_F1 signals only at start-up in order to avoid latent faults.

5 Candidate address for testing All-X issue

This section describes a Perl script which can be used for finding a candidate address for testing All-X in the RAMs. Some examples of usage of the script are provided.

```
--- start Perl script ---:
: # -*- perl -*-
eval 'exec perl -w -S $0 ${1+"$@"}'
     if 0;

use strict;

my $base = hex($ARGV[0]);
my $num_to_find = ($#ARGV > 0) ? $ARGV[1] : 1;
my $all0_found = 0;
my $all1_found = 0;
my $guesses = 0;
my $addr = $base;
my $ecc;
my $bit_count;

printf "RAM base address = 0x%08x\n", $base;
printf "  All 0s - Addresses with two bits set in the address ECC
contribution:\n";

while(($guesses < 131072) && ($all0_found < $num_to_find)) {
      $ecc = get_ecc($addr, 0, 0);
      $bit_count = count_ones($ecc);
      if($bit_count == 2) {
      $all0_found++;
      printf "    (%d) addr = 0x%08x, addr_ecc = 0x%02x\n", $all0_found,
      $addr, $ecc;
      }
$addr += 8;
$guesses++;
}

printf "\n  All 1s - Addresses with two bits cleared in the address ECC
contribution:\n";

$addr = $base;
while(($guesses < 131072) && ($all1_found < $num_to_find)) {
      $ecc = get_ecc($addr, 0xffffffff, 0xffffffff);
      $bit_count = count_zeroes($ecc);
      if($bit_count == 2) {
            $all1_found++;
            printf "   (%d) addr = 0x%08x, addr_ecc = 0x%02x\n", $all1_found,
            $addr, $ecc;
      }
```

```
        $addr += 8;
        $guesses++;
    }

    sub count_ones {
        my $string = sprintf("%08b", shift);
        my $count = 0;
        my $i;
        for($i=0; $i<8; $i++) {
            if(substr($string, $i, 1) eq "1") {
                $count++;
            }
        }
        return($count);
    }

    sub count_zeroes {
        my $string = sprintf("%08b", shift);
        my $count = 0;
        my $i;
        for($i=0; $i<8; $i++) {
            if(substr($string, $i, 1) eq "0") {
                $count++;
            }
        }
        return($count);
    }

    sub get_ecc {
        my $addr = shift;
        my $data_be0 = shift;
        my $data_be1 = shift;

        my @addrx8;
        my @data_bex8;
        my @data_lex8;
        my $i;
        my $j;
        my $bit;

        for($i=3; $i<32; $i++) {
            $bit = ($addr >> $i) & 1;
            $addrx8[$i]  = $bit;
            $addrx8[$i] |= $bit << 1;
            $addrx8[$i] |= $bit << 2;
            $addrx8[$i] |= $bit << 3;
            $addrx8[$i] |= $bit << 4;
            $addrx8[$i] |= $bit << 5;
            $addrx8[$i] |= $bit << 6;
            $addrx8[$i] |= $bit << 7;
        }

        for($i=0; $i<64; $i++) {
            if($i < 32) {
            $bit = ($data_be1 >> $i) & 1;
            } else {
                $bit = ($data_be0 >> ($i-32)) & 1;
            }

            $data_bex8[$i]  = $bit;
            $data_bex8[$i] |= $bit << 1;
            $data_bex8[$i] |= $bit << 2;
            $data_bex8[$i] |= $bit << 3;
            $data_bex8[$i] |= $bit << 4;
            $data_bex8[$i] |= $bit << 5;
            $data_bex8[$i] |= $bit << 6;
            $data_bex8[$i] |= $bit << 7;
        }

        for($i=0; $i<8; $i++) {
            for($j=0; $j<8; $j++) {
                $data_lex8[$i*8+$j] = $data_bex8[(7-$i)*8+$j];
            }
        }



        my $addr_ecc
            = (0x1f & $addrx8[31])
            ^ (0xf4 & $addrx8[30])
            ^ (0x3b & $addrx8[29])
            ^ (0xe3 & $addrx8[28])
            ^ (0x5d & $addrx8[27])
            ^ (0xda & $addrx8[26])
            ^ (0x6e & $addrx8[25])
            ^ (0xb5 & $addrx8[24])
            ^ (0x8f & $addrx8[23])
            ^ (0xd6 & $addrx8[22])
            ^ (0x79 & $addrx8[21])
            ^ (0xba & $addrx8[20])
            ^ (0x9b & $addrx8[19])
            ^ (0xe5 & $addrx8[18])
            ^ (0x57 & $addrx8[17])
            ^ (0xec & $addrx8[16])
            ^ (0xc7 & $addrx8[15])
            ^ (0xae & $addrx8[14])
            ^ (0x67 & $addrx8[13])
            ^ (0x9d & $addrx8[12])
            ^ (0x5b & $addrx8[11])
            ^ (0xe6 & $addrx8[10])
            ^ (0x3e & $addrx8[9])
            ^ (0xf1 & $addrx8[8])
            ^ (0xdc & $addrx8[7])
            ^ (0xe9 & $addrx8[6])
            ^ (0x3d & $addrx8[5])
            ^ (0xf2 & $addrx8[4])
            ^ (0x2f & $addrx8[3]);

        my $addr_ecc_tcm
            = (0x1f & $addrx8[31])
            ^ (0xf4 & $addrx8[30])
            ^ (0x3b & $addrx8[29])
```

```
          ^ (0xe3 & $addrx8[28])
          ^ (0x5d & $addrx8[27])
          ^ (0xda & $addrx8[26])
          ^ (0x6e & $addrx8[25])
          ^ (0xb5 & $addrx8[24])
          ^ (0x8f & $addrx8[23])
          ^ (0xd6 & $addrx8[22])
          ^ (0x79 & $addrx8[21])
          ^ (0xba & $addrx8[20])
          ^ (0x9b & $addrx8[19])
          ^ (0xe5 & $addrx8[18])
          ^ (0x57 & $addrx8[17])
          ^ (0xec & $addrx8[16]);

my $ecc_tcm_fix
          = (0xc7 & $addrx8[15])
          ^ (0xae & $addrx8[14])
          ^ (0x67 & $addrx8[13])
          ^ (0x9d & $addrx8[12])
          ^ (0x5b & $addrx8[11])
          ^ (0xe6 & $addrx8[10])
          ^ (0x3e & $addrx8[9])
          ^ (0xf1 & $addrx8[8])
          ^ (0xdc & $addrx8[7])
          ^ (0xe9 & $addrx8[6])
          ^ (0x3d & $addrx8[5])
          ^ (0xf2 & $addrx8[4])
          ^ (0x2f & $addrx8[3]);

my $data_ecc
          = (0xb0 & $data_lex8[63])
          ^ (0x23 & $data_lex8[62])
          ^ (0x70 & $data_lex8[61])
          ^ (0x62 & $data_lex8[60])
          ^ (0x85 & $data_lex8[59])
          ^ (0x13 & $data_lex8[58])
          ^ (0x45 & $data_lex8[57])
          ^ (0x52 & $data_lex8[56])

          ^ (0x2a & $data_lex8[55])
          ^ (0x8a & $data_lex8[54])
          ^ (0x0b & $data_lex8[53])
          ^ (0x0e & $data_lex8[52])
          ^ (0xf8 & $data_lex8[51])
          ^ (0x25 & $data_lex8[50])
          ^ (0xd9 & $data_lex8[49])
          ^ (0xa1 & $data_lex8[48])

          ^ (0x54 & $data_lex8[47])
          ^ (0xa7 & $data_lex8[46])
          ^ (0xa8 & $data_lex8[45])
          ^ (0x92 & $data_lex8[44])
          ^ (0xc8 & $data_lex8[43])
          ^ (0x07 & $data_lex8[42])
          ^ (0x34 & $data_lex8[41])
          ^ (0x32 & $data_lex8[40])
```

```
                    ^ (0x68 & $data_lex8[39])
                    ^ (0x89 & $data_lex8[38])
                    ^ (0x98 & $data_lex8[37])
                    ^ (0x49 & $data_lex8[36])
                    ^ (0x61 & $data_lex8[35])
                    ^ (0x86 & $data_lex8[34])
                    ^ (0x91 & $data_lex8[33])
                    ^ (0x46 & $data_lex8[32])

                    ^ (0x58 & $data_lex8[31])
                    ^ (0x4f & $data_lex8[30])
                    ^ (0x38 & $data_lex8[29])
                    ^ (0x75 & $data_lex8[28])
                    ^ (0xc4 & $data_lex8[27])
                    ^ (0x0d & $data_lex8[26])
                    ^ (0xa4 & $data_lex8[25])
                    ^ (0x37 & $data_lex8[24])

                    ^ (0x64 & $data_lex8[23])
                    ^ (0x16 & $data_lex8[22])
                    ^ (0x94 & $data_lex8[21])
                    ^ (0x29 & $data_lex8[20])
                    ^ (0xea & $data_lex8[19])
                    ^ (0x26 & $data_lex8[18])
                    ^ (0x1a & $data_lex8[17])
                    ^ (0x19 & $data_lex8[16])

                    ^ (0xd0 & $data_lex8[15])
                    ^ (0xc2 & $data_lex8[14])
                    ^ (0x2c & $data_lex8[13])
                    ^ (0x51 & $data_lex8[12])
                    ^ (0xe0 & $data_lex8[11])
                    ^ (0xa2 & $data_lex8[10])
                    ^ (0x1c & $data_lex8[9])
                    ^ (0x31 & $data_lex8[8])


                    ^ (0x8c & $data_lex8[7])
                    ^ (0x4a & $data_lex8[6])
                    ^ (0x4c & $data_lex8[5])
                    ^ (0x15 & $data_lex8[4])
                    ^ (0x83 & $data_lex8[3])
                    ^ (0x9e & $data_lex8[2])
                    ^ (0x43 & $data_lex8[1])
                    ^ (0xc1 & $data_lex8[0]);

     my $ecc       = $data_ecc ^ $addr_ecc;
     my $ecc_tcm   = $data_ecc ^ $addr_ecc ^ $addr_ecc_tcm ^ 0x55;
     my $ecc_flash = $data_ecc ^ 0xff;
     return($ecc);
}

##printf "addr         = 0x%08x\n", $addr;
##printf "data_be      = 0x%08x_%08x\n", $data_be0, $data_be1;
##printf "addr_ecc     = 0x%02x\n", $addr_ecc;
```

```
##printf "data_ecc    = 0x%02x\n", $data_ecc;
##printf "ecc         = 0x%02x\n", $ecc;
##printf "ecc_tcm     = 0x%02x\n", $ecc_tcm;
##printf "ecc_tcm_fix = 0x%02x\n", $ecc_tcm_fix;
##printf "ecc_flash   = 0x%02x\n", $ecc_flash;
----- end per script -----
```

This script finds the first N addresses with 2 bits set and 2 bits cleared in the address ECC contribution. Usage is as follows:

- find_allx_addr address [number]
- address – starting address to start searching from
- number – number of addresses to find, default is 1

Example:

1. Find the first address of each type for system RAM:
   - ./find_allx_addr 40000000

RAM base address = 0x40000000

All 0s - Addresses with two bits set in the address ECC contribution:

- addr = 0x40000010, addr_ecc = 0x06

All 1s - Addresses with two bits cleared in the address ECC contribution:

1. addr = 0x40000008, addr_ecc = 0xdb
2. Find the first 5 addresses of each type for system RAM:
   - ./find_allx_addr 40000000 5

RAM base address = 0x40000000

All 0s - Addresses with two bits set in the address ECC contribution:

1. addr = 0x40000010, addr_ecc = 0x06
2. addr = 0x40000038, addr_ecc = 0x14
3. addr = 0x40000058, addr_ecc = 0xc0
4. addr = 0x40000080, addr_ecc = 0x28
5. addr = 0x400000f8, addr_ecc = 0x21

All 1s - Addresses with two bits cleared in the address ECC contribution:

1. addr = 0x40000008, addr_ecc = 0xdb
2. addr = 0x40000098, addr_ecc = 0xf5
3. addr = 0x400000b0, addr_ecc = 0xe7
4. addr = 0x400000c8, addr_ecc = 0xee
5. addr = 0x400000e0, addr_ecc = 0xfc

6 Further information

6.1 Document management

- *32-bit Power Architecture® microcontroller for automotive ASILD Chassis & Safety applications* (RM0349, DOC ID 024507)
- *Safety application guide for SPC57EM80x family*

6.2 Conventions and terminology

Table 5 shows the list of conventions for this document.

Table 5. List of conventions and terminology

Convention	Description
error	Discrepancy between a computed, observed, or measured value or condition and the true, specified or theoretically correct value or condition.
fault	Abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function.
failure	The termination of the ability of a functional unit to perform a required function.

6.3 Acronyms and abbreviations

A short list of acronyms and abbreviations used in this document is shown in *Table 6*.

Table 6. Acronyms and abbreviations

Terms	Meanings
ADC	Analog to Digital Converter
BAF	Boot Assist Flash
BAR	Boot Assist ROM
CCF	Common Cause Failure
CMU	Clock Monitor Unit
CRC	Cyclic Redundancy Check
CTU	Cross-Triggering Unit
DC	Diagnostic Coverage
ECC	Error Correcting Code
DMA	Direct Memory Access
ERRM	Error Out Monitor function
EXWD	External Watchdog function
FCCU	Fault Collection and Control Unit
FMEDA	Failure Modes, Effects & Diagnostic Analysis

Table 6. Acronyms and abbreviations (continued)

Terms	Meanings
FMPLL	Frequency-Modulated Phase-Locked Loop
GPIO	General Purpose Input/Output
LBIST	Logic Built-In Self-test
MBIST	Memory Built-In Self-test
MC_CGM	Clock Generation Module
MC_ME	Mode Entry
MCU	Microcontroller Unit
MPU	Memory Protection Unit
NCF	Non-Critical Fault
NMI	Non-Maskable Interrupt
NVM	Non-Volatile Memory
PMU	Power Management Unit
PSM	Power Supply and Monitor function
PST	Process Safety Time
RCCU	Redundancy Control Checking Unit
MC_RGM	Reset Generation Module
SM	Safety Manual
SIL	Safety Integrity Level
SSCM	System Status and Control Module
SWG	Sine Wave Generator
SWT	Software Watchdog Timer

7 Revision history

Table 7. Revision history

Date	Revision	Changes
02-May-2013	1	Initial release
21-Sep-2013	2	Updated Disclaimer.
23-Jul-2015	3	Changed RPN from SPC570S50L1/SPC570S50L3 in SPC570S50E1/SPC570S50E3. Updated: – *Chapter 1: Preface* – *Section 2.2: Functional safety – ISO 26262 compliance* – *Section 2.3: Safety goal* – *Chapter 3: Functional safety requirements for application software* Added in *Section 3.3.8: RAM* the **Assumption**:[SAG_VE512K_3_200] Updated Disclaimer.