Safety Manual for SAC57D5xx

Document Number: SAC57D54HSM
Rev. 2.1, 08/2016

Contents

Section number	Title	Page

Chapter 1
Preface

1.1 Overview

This document discusses requirements for the integration and use of the SAC57D5xx Microcontroller Unit (MCU) in safety-related systems. It is intended to support safety system developers in building their safety-related systems using the safety mechanisms of the SAC57D5xx, and describes the system level hardware or software safety measures that should be implemented to achieve the desired system level functional safety integrity level. The SAC57D5xx is developed according to ISO 26262 and has an integrated safety concept.

1.2 Safety manual assumptions

During the development of the SAC57D5xx, assumptions were made on the system level safety requirements with regards to the MCU. During the system level development, the safety system developer is required to establish the validity of the MCU assumptions in the context of the specific safety-related system. To enable this, all relevant MCU assumptions are published in the Safety Manual and can be identified as follows:

- **Assumption:** An assumption that is relevant for functional safety in the specific safety system. It is assumed that the safety system developer fulfills an assumption in the design.
- **Assumption under certain conditions:** An assumption that is relevant under certain conditions. If the associated condition is met, it is assumed that the safety system developer fulfills the assumption in the design.

Example: **Assumption:** It is assumed that the system is designed to go into a safe state (Safe state$_{\text{system}}$) when the safe state of the MCU (Safe state$_{\text{MCU}}$) is entered.

Example: **Assumption under certain conditions:** If a high impedance state on an output is not safe, pull-up or pull-down resistors shall be added to safety-critical outputs. The need for this will be application dependent for the unpowered or reset condition (tristated I/O) of the SAC57D5xx.

The safety system developer will need to use discretion in deciding whether these assumptions are valid for their particular safety-related system. In the case where an MCU assumption does not hold true, the safety system developer should initiate a change management activity beginning with impact analysis. For example, if a specific assumption is not fulfilled, an alternate implementation should be shown to be similarly effective at meeting the functional safety requirement in question (for example, the same level of diagnostic coverage is achieved, the likelihood of dependent failures are similarly low, and so on). If the alternative implementation is shown to be not as effective, the estimation of an increased failure rate and reduced metrics (SFF: Safe Failure Fraction, SPFM: Single-Point Fault Metrics, LFM: Latent Fault Metric) due to the deviation must be specified. The FMEDA can be used to help make this analysis.

1.3 Safety manual guidelines

This document also contains guidelines on how to configure and operate the SAC57D5xx in safety-related systems. These guidelines are preceded by one of the following text statements:

- **Recommendation:** A recommendation is either a proposal for the implementation of an assumption, or a reasonable measure which is recommended to be applied, if there is no assumption in place. The safety system developer has the choice whether or not to adhere to the recommendation.
- **Rationale:** The motivation for a specific assumption and/or recommendation.
- **Implementation hint:** An implementation hint gives specific details on the implementation of an assumption and/or recommendation on the SAC57D5xx. The safety system developer has an option to follow the implementation hint.

The safety system developer will need to use discretion in deciding whether these guidelines are appropriate for their particular safety-related system.

1.4 Functional safety standards

It is assumed that the user of this document is familiar with the functional safety standards *ISO 26262 Road vehicles - Functional safety* and *IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems.* The

SAC57D5xx is a component as seen in the context of ISO 26262 and in this case its development is completely decoupled from the development of an item or system. Therefore the development of the SAC57D5xx is considered a Safety Element out of Context (SEooC) development, as described in *ISO 26262-10.9 Safety element out of context* and more specifically detailed in *ISO 26262-10.9.2.3 Development of a hardware component as a safety element out of context* and *ISO 26262-10 Annex A ISO 26262 and microcontrollers*.

1.5 Related documentation

The SAC57D5xx is developed according to ISO 26262 and has an integrated safety concept targeting safety-related systems requiring high safety integrity levels. In order to support the integration of the SAC57D5xx into safety-related systems, the following documentation will be available:

- Reference Manual (SAC57D5xxRM) - Describes the SAC57D5xx functionality
- Data Sheet (SAC57D5xxDS) - Describes the SAC57D5xx operating conditions
- Safety Manual (SAC57D5xxSM) - Describes the SAC57D5xx safety concept and possible safety mechanisms (integrated in SAC57D5xx, system level hardware or system level software), as well as measures to reduce dependent failures
- FMEDA - Inductive analysis enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and β-factor β_{IC})
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The FMEDA and FMEDA report are available upon request. The SAC57D5xx is a SafeAssure solution; for further information regarding functional safety at NXP, visit www.nxp.com/safeassure.

1.6 Other considerations

When developing a safety-related system using the SAC57D5xx, the following information should be considered:

- The SAC57D5xx is handled in accordance with JEDEC standards J-STD-020 and J-STD-033.
- The operating conditions given in the SAC57D5xx Data Sheet.
- If applicable, any published SAC57D5xx errata.

- The recommended production conditions given in the SAC57D5xx quality agreement.
- The safety system developer is required to report all field failures of the SAC57D5xx to NXP.

As with any technical documentation, it is the reader’s responsibility to ensure he or she is using the most recent version of the documentation.

Chapter 2
MCU safety context

2.1 Overview

The SAC57D5xx family is the next generation platform of devices specifically targeted to the instrument cluster market using single and dual high resolution displays. Leveraging the highly successful MPC56xxS product family, NXP is introducing a multi-core architecture powered by ARM Cortex-M (for real time) and Cortex-A processors (for applications and HMI), coupled with 2-D Graphics Accelerators (GPU), Heads Up Display (HUD) Warping Engine, Dual TFT display drive, integrated Stepper Motor Drivers and a powerful I/O Processor, that will offer leading edge performance and scalability for cost-effective applications.

Powered by ARM Cortex-M and Cortex-A processors, coupled with 2D Graphics Accelerators, HUD Warping Engine, Dual TFT display drive, integrated Stepper Motor Controller and a powerful I/O Processor, these products offer leading performance and scalability for cost-effective applications.

The family supports up to 2 WVGA resolution displays, including one with in-line Heads Up Display (HUD) warping hardware. Graphics content is generated using a powerful Vivante GPU supporting OpenVG1.1, and the 2D Animation and Composition Engine (2D-ACE), which significantly reduces memory footprint for content creation. Embedded memories include up to 4 MB Flash, 1 MB SRAM with ECC and up to 1.3 MB of graphics SRAM without ECC. Memory expansion is available through DDR2 and SDR DRAM interfaces while two flexible QuadSPI modules provide SDR and DDR serial flash expansion. In response to the growing desire for security and safety, the SAC57D5/4xx family integrates NXP's latest SHE-compliant CSE2 engine and delivers support for ISO26262 ASIL-B functional safety compliance.

2.2 Safety integrity level

The SAC57D5xx is designed to be used in automotive, or industrial, applications which need to fulfill functional safety requirements as defined by functional safety integrity levels (for example, ASIL B of ISO 26262 or SIL 2 of IEC 61508). The SAC57D5xx is a component as seen in the context of ISO 26262 and in this case its development is completely decoupled from the development of an item or system. Therefore the development of the SAC57D5xx is considered a Safety Element out of Context (SEooC) development.

The SAC57D5xx is seen as a Type B subsystem in the context of IEC 61508 (“complex,” see IEC 61508-2, section 7.4.4.1.3) with a HFT = 0 (Hardware Fault Tolerance) and may be used in any mode of operation (see IEC 61508-4, section 3.5.16).

2.3 Safety function

2.3.1 MCU safety functions

Given the application independent nature of the SAC57D5xx, no specific safety function can be specified. Therefore, during the SEooC development of the SAC57D5xx, MCU safety functions were assumed. During the development of the safety-related system, the MCU safety functions are mapped to the specific system safety functions (application dependent). The assumed MCU safety functions are:

- **Reverse Camera:** Image frozen. Continuous image update faults (frozen image display) require safety integrity of ASIL B. All other faults are perceived by driver (no image, image distortion, color change, contrast change...). These systems have the risk that they tend to replace the usage of mirrors etc. therefore no longer only assisting driver as the display distracts the attention of the driver no longer able to look back or look into mirror. The VIDEO MCU safety function is mapped to this system level safety function. Please see the attachment "Module Classification.xlsx" for details of the MCU modules involved in this MCU safety function.
- **Warning indications relating to safety critical systems:** Wrong information displayed or missing trigger. False negative faults require safety integrity of ASIL B. Warning indications may be shown on the panel, or with an LED or be audio warning signals. These are latent faults. Driver perception would be improved if an audio warning signal is used in addition to the display. False positive faults do not endanger safety. The DISPLAY, LED and AUDIO MCU safety functions are mapped to this system level safety function. Please see the attachment "Module

Classification.xlsx" for details of the MCU modules involved in these MCU safety functions.

- **Speed gauge:** Wrong speed displayed, frozen or corrupt image when screen is used to display speed. False negative and false positive (wrong speed displayed) faults require the safety integrity of ASIL B. The DISPLAY and MOTOR MCU safety functions are mapped to this system level safety function. Please see the attachment "Module Classification.xlsx" for details of the MCU modules involved in these MCU safety functions.
- **Gear selection display:** Gear displayed does not match gear selected. Driver would perceive false negative faults. False positive faults (wrong gear displayed) require safety integrity of ASIL B. The DISPLAY MCU safety function is mapped to this system level safety function. Please see the attachment “Module Classification.xlsx” for details of the MCU modules involved in this MCU safety function.

Moreover, the following approach is assumed for Input / Output related functions and debug functions:

- **Input / Output Functions (Application dependent)**: Input / Output functions of the SAC57D5xx have a high application dependency. **Functional safety will be primarily achieved by system level safety measures**.
- **Not Safety Related Functions**: It is assumed that some functions are **Not Safety Related (e.g. debug)**.

Please see the Module classification section for further details.

2.3.2 Correct operation

Correct operation of the SAC57D5xx is defined as:

- **MCU Safety Function** and **Safety Mechanism** modules are operating according to specification.
- **Peripheral** modules are usable by qualifying data with system level safety measures or by using modules redundantly. Qualification should have a low risk of dependent failures. In general, **Peripheral** module safety measures are implemented in system level software.
- **Not Safety Related** modules are not interfering with the operation of other modules.

2.4 Safe states

A safe state of the system is named Safe state$_{\text{system}}$, whereas a safe state of the SAC57D5xx is named Safe state$_{\text{MCU}}$. A Safe state$_{\text{system}}$ is an operating mode without an unreasonable probability of occurrence of physical injury or damage to the health of any persons. A Safe state$_{\text{system}}$ may be the intended operating mode or a mode where the system has been disabled.

Assumption: [SM_200] It is assumed that the system is designed to go into a safe state (Safe state$_{\text{system}}$) when the safe state of the MCU (Safe state$_{\text{MCU}}$) is entered. [end]

2.4.1 MCU Safe state

The safe states (Safe state$_{\text{MCU}}$) of the SAC57D5xx are:

- Operating correctly (see Figure 2-1 and section Correct operation)
- Explicitly indicating an internal error (indication on EOUT[*n*], Figure 2-1)
- In reset (see Figure 2-1)
- Completely unpowered (see Figure 2-1)
- Safe mode (see Figure 2-1)

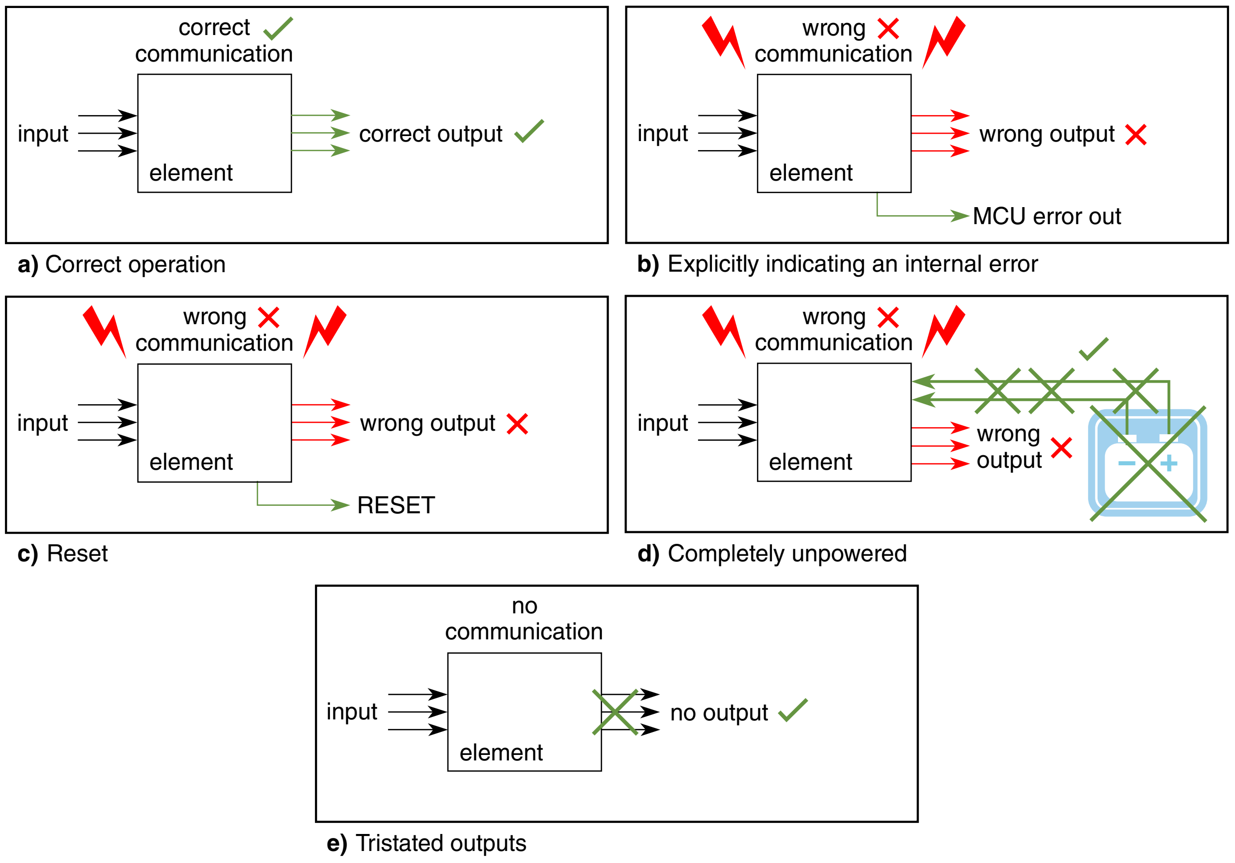

Figure 2-1. Safe state$_{\text{MCU}}$ of SAC57D5xx

2.4.2 Transitions to Safe state$_{\text{system}}$

Assumption: [SM_015] The system transitions itself to a Safe state$_{\text{system}}$ when the MCU explicitly indicates an internal error (as shown on EOUT[0] or EOUT[1]). [end]

Implementation hint: If the SAC57D5xx signals an internal failure via its error out signals (EOUT[*n*]), the surrounding subsystem shall no longer use the SAC57D5xx outputs for safety functions since these signals can no longer be considered reliable. If an error is indicated, the system shall be able to remain in a Safe state$_{\text{system}}$ without any additional action by the SAC57D5xx. Depending on the configuration, the system may disable, or reset, the SAC57D5xx as a reaction to the error signal.

Assumption: [SM_016] The system transitions itself to a Safe state$_{\text{system}}$ when the MCU is in a reset state. [end]

Assumption: [SM_017] The system transitions itself to a Safe state$_{\text{system}}$ when the MCU is unpowered. [end]

Assumption: [SM_018] The system transitions itself to a Safe state$_{\text{system}}$ when the MCU has no active output (for example, tristate). [end]

2.4.3 Continuous reset transitions

If a system continuously switches between a standard operating state and the reset state, without any device shutdown, it is not considered to be in a Safe state.

Assumption: [SM_019] It is assumed that the application identifies, and signals, continuous switching between reset and standard operating mode as a failure condition. [end]

2.5 Faults and failures

2.5.1 Failure types

Failures are the main detrimental impact to functional safety:

- A systematic failure is manifested in a deterministic way to a certain cause (systematic fault), that can only be eliminated by a change of the design process, manufacturing process, operational procedures, documentation, or other relevant factors. Thus, measures against systematic faults can reduce systematic failures (for example, implementing and following adequate processes).
- A random hardware failure can occur unpredictably during the lifetime of a hardware element and follows a probability distribution. A reduction in the inherent failure rate of the hardware will reduce the likelihood of random hardware faults to occur. Detection and control will mitigate the effects of random hardware faults when they do occur. A random hardware failure is caused by a permanent fault (for example, physical damage), an intermittent fault, or a transient fault. Permanent faults are unrecoverable. Intermittent faults are, for example, faults linked to specific operational conditions, or noise. Transient faults are, for example, particles (alpha, neutron) or EMI-radiation. An affected configuration register can be recovered by setting the desired value or by power cycling. Due to a transient fault, an element may be switched into a self destructive state (for example, single event latch up), and therefore may cause permanent destruction.

2.5.2 Faults

The following random faults may generate failures, which may lead to the violation of a functional safety goal. Citations are according to ISO 26262-1. Random hardware faults occur at a random time, which results from one or more of the possible degradation mechanisms in the hardware.

- **Single-Point Fault (SPF):** A fault in an element that is not covered by a safety mechanism, and results in a single-point failure. This leads directly to the violation of a safety goal. 'a' in the Figure 2-2 shows a SPF inside an element, which generates a wrong output. The equivalent in IEC 61508 to Single-Point Fault is a **Random fault**. Whenever a SPF is mentioned in this document, it is to be read as a random fault for IEC 61508 applications.
- **Latent Fault (LF):** A fault whose presence is not detected by a safety mechanism nor perceived by the automobile driver. A LF is a fault that does not violate the functional safety goal(s) itself, but leads to a dual-point or multiple-point failure when combined with at least one additional independent fault, which then leads directly to the violation of a functional safety goal. 'b' in the Figure 2-2 shows a LF inside an element, which still generates a correct output. No equivalent in IEC 61508 to LF is named.
- **Dual-Point Fault (DPF):** An individual fault that, in combination with another independent fault, leads to a dual-point failure. This leads directly to the violation of a functional safety goal. 'd' in the Figure 2-2 shows two LFs inside an element, which generate a wrong output.
- **Multiple-Point Fault (MPF):** An individual fault that, in combination with other independent faults, leads to a multiple-point failure. This leads directly to the violation of a functional safety goal. Unless otherwise stated, multiple-point faults are considered safe faults and are not covered in the functional safety concept of SAC57D5xx.
- **Residual Fault (RF):** A portion of a fault that independently leads to the violation of a functional safety goal, where that portion of the fault is not covered by a functional safety mechanism. 'c' in the Figure 2-2 shows a RF inside an element, which – although a functional safety mechanism is set in place – generates a wrong output, as this particular fault is not covered by the functional safety mechanism.
- **Safe Fault (SF):** A fault whose occurrence will not significantly increase the probability of violation of a functional safety goal. Safe faults are not covered in this document. SPFs, RFs or DPFs are not safe faults.

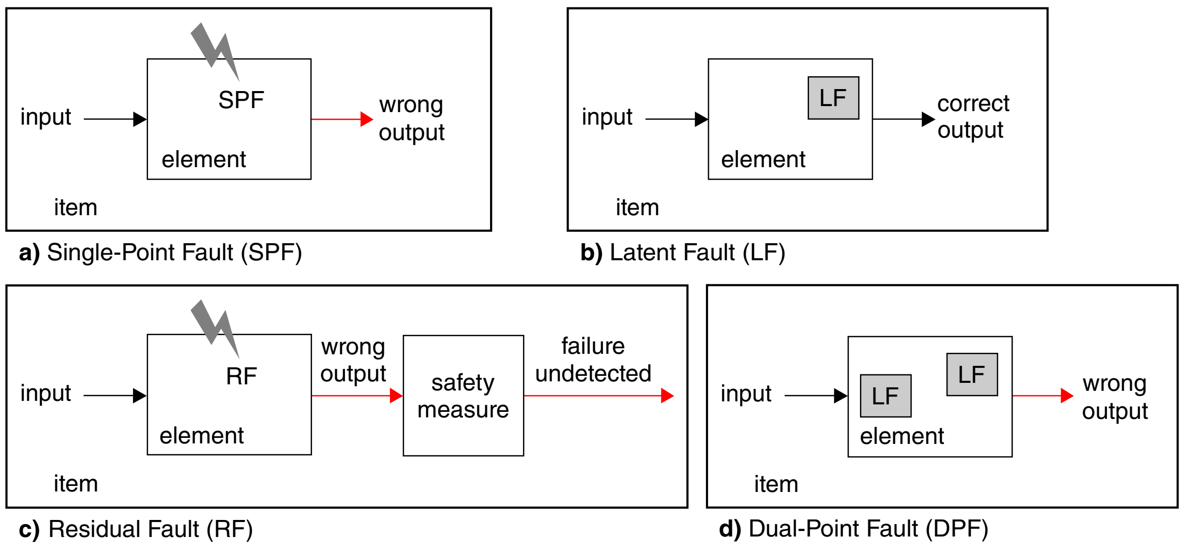

Figure 2-2. Faults

SPFs should be detected within the Fault Tolerant Time Interval (FTTI). LFs (DPFs) should be detected within the Latent-Fault Tolerant Time Interval (L-FTTI). In automotive applications, L-FTTI is generally accepted to occur once per typical automotive T_{trip} and potential faults are typically detected by safety mechanisms which are executed during system testing at startup. Detecting DPFs once per T_{trip} reduces the accumulation time of latent faults in T_{life} of the product, to a maximum time period of T_{trip}.

2.5.3 Dependent failures

- **Common cause failure (CCF):** Subset of dependent failures in which two or more component fault states exist at the same time, or within a short time interval, as a result of a shared cause (see Figure 2-3).

  A CCF is the coincidence of random failure states of two or more elements on separate channels of a redundancy element which lead to the failure of the defined element to perform its intended safety function, resulting from a single event or root cause (chance cause, non-assignable cause, noise, natural pattern, and so on). A CCF causes the probability of multiple channels (N) to have a failure rate larger than $\lambda_{\text{single channel}}^N$ ($\lambda_{\text{redundant element}} > \lambda_{\text{single channel}}^N$).

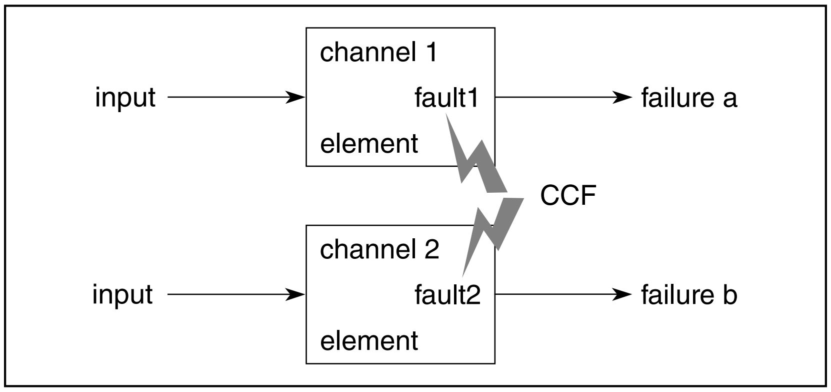

Figure 2-3. Common Cause Failures

- **Common mode failure (CMF):**A single root cause leads to similar coincidental erroneous behavior (with respect to the safety function) of two or more (not necessarily identical) elements in redundant channels, resulting in the inability to detect the failures. Figure 2-4 shows three elements within two redundant channels. One single root cause (CMFA or CMFB) leads to undetected failures in the primary channel and in one of the elements of the redundant channel.

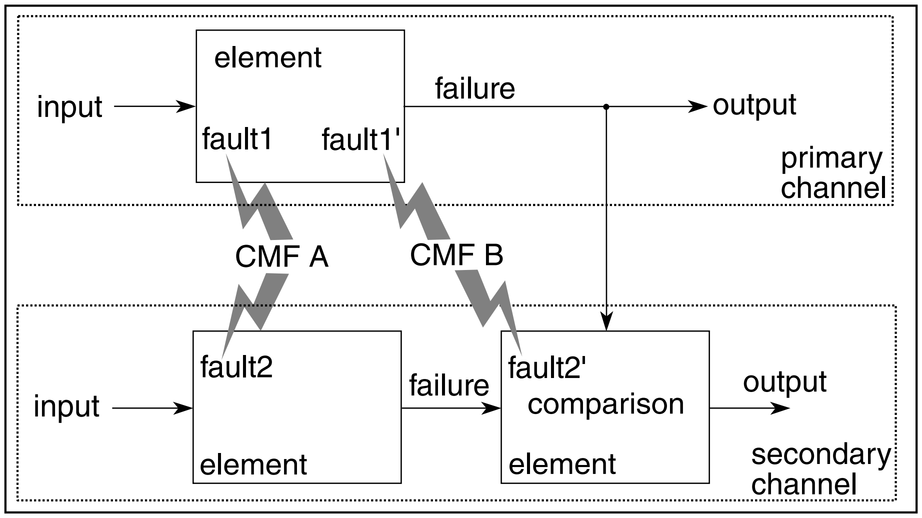

Figure 2-4. Common Mode failures

- **Cascading failure (CF):** CFs occur when local faults of an element in a system ripple through interconnected elements causing another element or elements of the same system and within the same channel to fail. Cascading failures are dependent failures that are not common cause failures. Figure 2-5 shows two elements within a single channel, in which a single root cause leads to a fault (fault 1) in one element resulting in a failure (failure a). This failure then cascades to the second element, causing a second fault (fault 2) that leads to a failure (failure b).

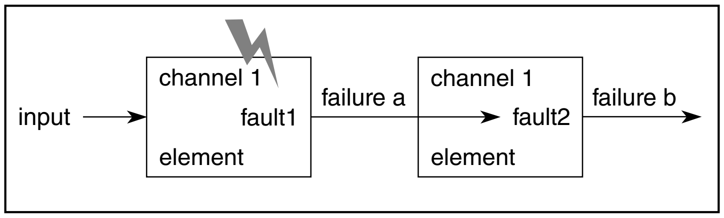

Figure 2-5. Cascading failures

2.6 Single-point fault tolerant time interval and process safety time

The single-point Fault Tolerant Time Interval (FTTI)/Process Safety Time (PST) is the time span between a failure that has the potential to give rise to a hazardous event and the time by which counteraction has to be completed to prevent the hazardous event from occurring.

Figure 2-6 shows the FTTI for a system:

- Normal MCU operation (a).
- With an appropriate functional safety mechanism to manage the fault (b).
- Without any suitable functional safety mechanism, a hazard may appear after the FTTI has elapsed (c).

The equivalent in IEC 61508 to FTTI is Process Safety Time (PST). Whenever single-point fault tolerant time interval or FTTI is mentioned in this document, it shall be read as PST for IEC 61508 applications.

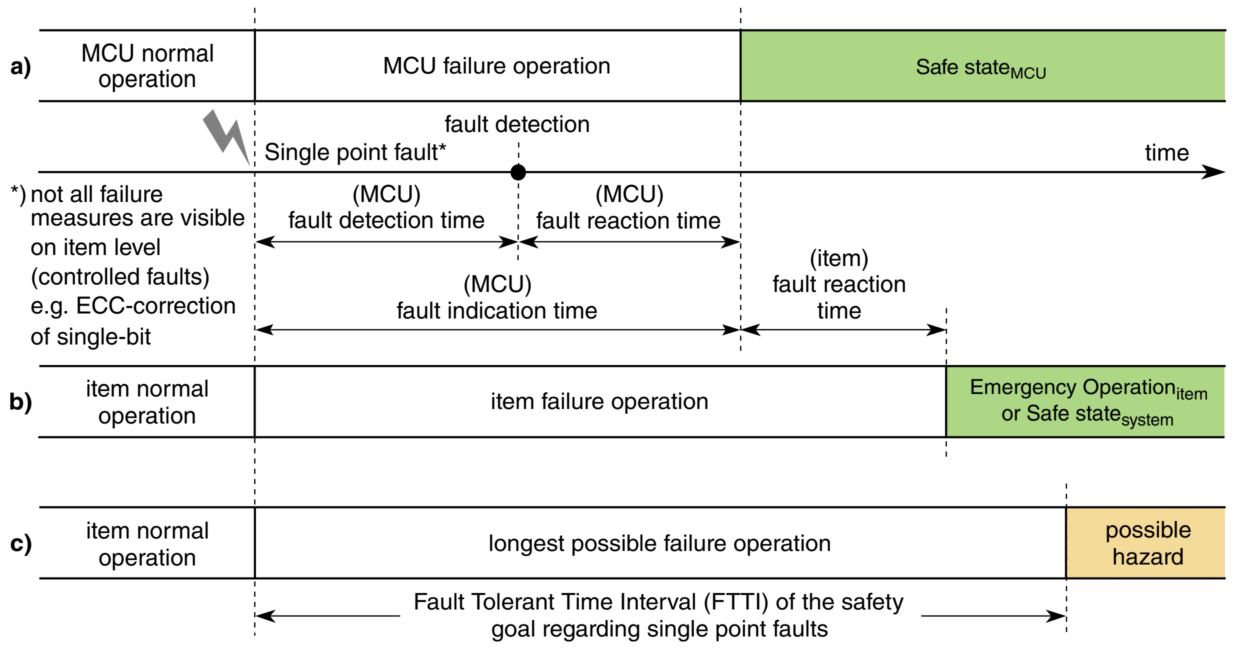
a)
MCU normal operation
MCU failure operation
Safe state$_{MCU}$
fault detection
Single point fault*
time
*) not all failure measures are visible on item level (controlled faults) e.g. ECC-correction of single-bit
(MCU) fault detection time
(MCU) fault reaction time
(MCU) fault indication time
(item) fault reaction time
b)
item normal operation
item failure operation
Emergency Operation$_{item}$ or Safe state$_{system}$
c)
item normal operation
longest possible failure operation
possible hazard
Fault Tolerant Time Interval (FTTI) of the safety goal regarding single point faults

Figure 2-6. Fault tolerant time interval for single point faults

Fault indication time is the time from the occurrence of a fault to when the SAC57D5xx is switched into a Safe state$_{MCU}$ (for example, indication of that failure by driving the error out pins, forcing outputs of the SAC57D5xx to a high impedance state, or by assertion of reset).

2.6.1 MCU fault indication time

Fault indication time is the sum of **Fault detection time** and **Fault reaction time**.

- **Fault detection time** (Diagnostic test interval + Recognition time) is the maximum time for detection of a fault and consists of:
  - **Diagnostic test interval** is the interval between online tests (for example, software based self-test) to detect faults in a functional safety-related system. This time depends closely on the system implementation (for example, software).
    - Software cycle time of software based functional safety mechanisms. This time depends closely on the software implementation.
  - **Recognition time** is the maximum of the recognition time of all involved functional safety mechanisms. The mechanisms with the longest time are:

    - Recognition time related to the FMPLL loss of clock: it depends on how the FMPLL is configured. It is approximately 20 μs.
    - Software execution time of software based functional safety mechanisms. This time depends closely on the software implementation.
- **Fault reaction time** (Internal processing time + External processing time) is the maximum of the reaction time of all involved functional safety mechanisms consisting of internal processing time and external indication time:
  - **Internal processing time** to communicate the fault to the Fault Collection and Control Unit (FCCU), and can take up to a maximum of 10 FastInternal RC Oscillator (FIRC) clock cycles (nominal frequency of 16 MHz).
  - **External indication time** to notify an observer about a failure external to the SAC57D5xx. This time depends on the indication protocol configured in the Fault Collection and Control Unit (FCCU):
    - Dual Rail protocol and time switching protocol:
      - **FCCU configured as "fast switching mode":** indication delay is a maximum of 64 μs. As soon as the FCCU receives a fault signal, it reports the failure to the system.
      - **FCCU configured as "slow switching mode":** an indication delay could occur. The maximum delay is equal to the duration of the semiperiod of the error out (EOUT[*n*]) frequency. With an IRCOSC frequency of 16 MHz, the error out frequency is 61Hz. Therefore, the maximum indication delay is 8.2 ms.
    - **Bi-stable protocol:** indication delay is a maximum of 64 μs. As soon as the FCCU receives a fault signal, it reports the failure to the system.

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

The sum of the SAC57D5xx fault indication time and system fault reaction time should be less than the FTTI of the functional safety goal.

2.7 Latent-fault tolerant time interval for latent faults

The Latent-fault tolerant time interval (L-FTTI) is the time span between a latent fault, that has the potential to coincide along with other latent faults and give rise to a hazardous multiple-point event, and the time at which counteraction has to be completed to prevent the hazardous event from occurring. L-FTTI defines the sum of the respective worst case fault indication time and the time for execution of the corresponding countermeasure. Figure 2-7 shows the L-FTTI for multiple-point faults in a system.

There is no equivalent to L-FTTI in IEC 61508.

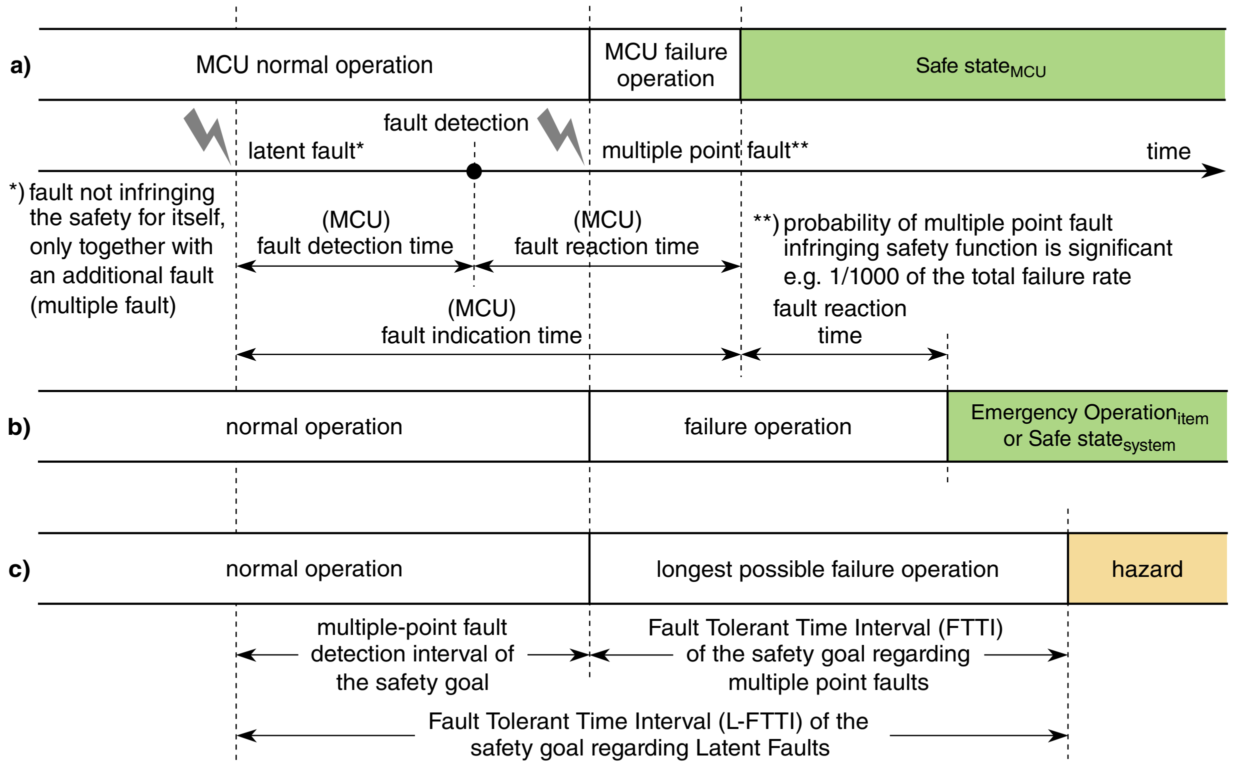
a)
MCU normal operation
MCU failure operation
Safe state$_{MCU}$
latent fault*
fault detection
multiple point fault**
time
*) fault not infringing the safety for itself, only together with an additional fault (multiple fault)
(MCU) fault detection time
(MCU) fault reaction time
**) probability of multiple point fault infringing safety function is significant e.g. 1/1000 of the total failure rate
(MCU) fault indication time
fault reaction time
b)
normal operation
failure operation
Emergency Operation$_{item}$ or Safe state$_{system}$
c)
normal operation
longest possible failure operation
hazard
multiple-point fault detection interval of the safety goal
Fault Tolerant Time Interval (FTTI) of the safety goal regarding multiple point faults
Fault Tolerant Time Interval (L-FTTI) of the safety goal regarding Latent Faults

Figure 2-7. Fault Tolerant Time Interval for latent faults

Latent fault indication time is the time it takes from the occurrence of a multiple-point failure to when the indication of that failure is driven on EOUT[*n*], forcing the outputs of the SAC57D5xx to a high impedance state (Safe Mode) or by assertion of reset.

Assumption:[SM_212] It is assumed that the MCU will go through a complete power-up/power-down cycle, or enter and exit standby mode, within the L-FTTI. When exiting standby mode, all safety relevant volatile memories shall be initialized.[end]

Rationale: To remove the effect of any transient faults.

2.7.1 MCU fault indication time

Fault indication time is the sum of **Fault detection time** and **Fault reaction time**. In general, the Fault detection time and Fault reaction time are negligible for multiple-point failures since the L-FTTI is significantly larger (hours, rather than seconds) than typical safety mechanism detection and reaction times. Typically the safety mechanisms to detect latent faults are executed during start-up, shut-down or periodically as required by the diagnostic test interval of the safety system.

The sum of latent fault indication time and latent and multiple point fault reaction time should be less than the L-FTTI of the functional safety goal.

Note

Detection and handling of a latent fault by a latent fault detection mechanism must be completed within the Multi-Point Fault (MPF) detection interval. Afterwards, it is assumed that the fault caused a multi-point failure, and latent fault detection is no longer guaranteed to work properly.

2.8 MCU failure indication

2.8.1 Failure handling

Failure handling can be split into two categories:

- Handling of failures before enabling the system level safety function (for example, during/following the SAC57D5xx initialization). These failures are required to be handled before the system enables the safety function, or in a time shorter than the respective FTTI or L-FTTI after enabling the safety function.
- Handling of failures during runtime with repetitive supervision while the safety function is enabled. These errors are to be handled in a time shorter than the respective FTTI or L-FTTI.

Assumption:[SM_022] It is assumed that single-point and latent fault diagnostic measures complete operations (including fault reaction time) in a time shorter than the respective FTTI or L-FTTI when the safety function is enabled. [end]

Recommendation: It is recommended to identify startup failures before enabling system level safety functions.

A typical failure reaction, with regards to power-up/start-up diagnostic measures, is to not initialize and start the safety function, but instead provide failure indication to the user.

Software can read the failure source that caused a FCCU fault, and can do so either before or after a functional reset. Software can also reset the failure, but the external failure indication will stay in failure mode for a configurable amount of time. If necessary, software can also reset the SAC57D5xx.

2.8.2 Failure indication signaling

The FCCU offers a hardware channel to collect errors and bring the device to a Safe state$_{MCU}$ when a failure is present in the SAC57D5xx. The FCCU provides two error output signals (EOUT[0] and EOUT[1]) used for external failure indication.

Different protocols for the error output pins are supported:

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

After power-on reset, the EOUT[*n*] outputs are either high-impedance or they are in a state that indicates an error. An error status flag can be read to indicate if the FCCU is in an error state. The flag can be written by software to 1, to indicate a fault, or 0, to indicate operational state. The EOUT[*n*] outputs will transition to the operational state only by software request.

At least one of the EOUT[*n*] outputs will be high to indicate that the device is in the operational state. If a two-pin bi-stable protocol with differential outputs is implemented (for example, EOUT[0] = 0 and EOUT[1] = 1 and vice-versa), the application software can configure that EOUT[*n*] signal that will be high to indicate the operational state (see Error Out Monitor (ERRM) for details on requirements for connecting EOUT[*n*] to external devices).

Chapter 3
MCU Safety Concept

3.1 General concept

The integrity of the Cortex M4 and A5 cores, memories, data path, and closely related periphery used in each safety function shall be ensured with the following methods:

- Structural core self test can be used to ensure the integrity of the Cortex M4 and A5 cores, along with platform elements used within the test, for example system RAM, flash and crossbar (see section Structural software based self-test (SBST)).
- The 2D-ACE has two safety layers to ensure that safety relevant content is driven to the display regardless of the setting of other layers, and without any pixel manipulation. These safety layers also have the highest priority (see section Display Controller (2D-ACE)).
- A frozen image detection feature is provided on the VIU. This detects if vsync or hsync fail to toggle or toggle incorrectly by counting valid pixels per line and valid lines per frame (see section Video Interface Unit (VIU4)).
- ECC / parity schemes, including flexECC on graphics RAM, can be used to ensure the integrity of data stored in system memories. For details of which memories are protected by ECC/parity see section ECC).
- A SWT is provided for each core to enable detection of a defective program sequence (see section Software Watchdog Timer).
- CRC provides the ability to calculate a checksum independently of the CPU (see section Cyclic Redundancy Checker Unit).
- Clock and power, generation and distribution, are supervised by dedicated monitors (see section Clock and power monitoring)
- The safety of the periphery is ensured by application-level measures (such as connecting one sensor to different I/O modules, sensor validation by sensor fusion, and so on) (see sections I/O functions and Communication controllers)
- MBISTs are provided to avoid the accumulation of latent faults in memories (see section BIST during boot). Dedicated mechanisms are provided to check the availability of safety mechanisms and the functionality of each error reaction path

(such as by fake fault injection) (see the "Error Injection Module (EIM)" chapter of the SAC57D5xx Reference Manual for details).

- The Fault Collection and Control Unit is responsible for collecting and reacting to failure notifications (see section FCCU and failure monitoring)
- Common Cause Failures (CCFs) are dealt with by a set of measures for both control and avoidance of CCFs (such as clock and power monitoring) (see section Common Cause Failure measures)
- Operational interference protection is ensured via a memory protection schema allowing concurrent execution of software with different (lower) ASIL (see section Operational interference protection)

3.2 ECC

Error correcting codes are used for protection of system RAM and flash, as well as for individual protection of peripheral RAMs.

3.2.1 ECC for storage

The majority of storage used in normal operation is protected by ECC with SEC/DED (Single Error Correct and Double Error Detect) and some are protected by EDC (Error Detection Code). The list showing the implementation of RAMs with ECC (including address protection) is shown in Table 3-1.

Table 3-1. ECC RAM implementations

Module	Memory	Memory column muxing factor	ECC	Address in ECC
Cortex M4	I-Cache	16	EDC	No
Cortex M4	D-Cache	16	EDC	No
Cortex M4	TCM	16	EDC	No
Cortex A5	I-Cache	8	No	-
Cortex A5	D-Cache	8	No	-
Cortex M0+	M0+ SRAM	16	SEC/DED	No
System RAM	System RAM	16	SEC/DED	Yes
Graphics RAM	Graphics RAM	16	SEC/DED (FlexECC)	No
DCU_0 and DCU_1	Palette/Tile/CLUT	16	No	-
DCU_0 and DCU_1	Cursor Memory	4	No	-
DCU_0 and DCU_1	Gamma	4	No	-
DCU_0 and DCU_1	Input Buffer	4	No	-
DCU_0 and DCU_1	Layer Memory	16	No	-

Table continues on the next page...

Table 3-1. ECC RAM implementations (continued)

Module	Memory	Memory column muxing factor	ECC	Address in ECC
DCU_1	HUD Line Buffer RAM	16	No	-
VIU4	Y/U/V RAM	16	No	-
VIU4	B/C RAM 0/1/1	4	No	-
VIU4	FIFO RAM	4	No	-
GPU RAM	GPU RAM	4	No	-
CSE RAM	CSE RAM	16	SEC/DED	No
DMA	DMA	4	No	-
FlexCAN FD	FlexCAN FD	4	SEC/DED	No
MLB50	MLB50	16	No	-
ENET	ENET	4	No	-
QuadSPI	QuadSPI	4	No	-

Assumption:[SM_113] The Cortext A5 I-Cache and D-Cache are not protected by ECC or EDC, and therefore should not be used for storing safety relevant instructions or data.[end]

Assumption:[SM_109] GRAM has FlexECC available, where a portion of the RAM can be sacrificed to enable ECC if required. FlexECC shall be enabled if GRAM is to be used to store safety relevant information.[end]

Some memories, particularly system storage, use an ECC computed over data and address to detect data and addressing faults (no/wrong/multiple selection). This is different for storage related to peripheral modules, which in general use ECC without address error protection.

Assumption under certain conditions:[SM_110] If external nonvolatile memory (FLASH) does not support ECC/parity and is used to store safety relevant code or data, then the safety relevant code or data shall be copied from external flash to main memory and a CRC check performed.[end]

Assumption under certain conditions:[SM_111] If External DDR DRAM does not support ECC/parity and is used to store safety relevant information, application software protection methods must be used.[end]

3.2.2 All-X words and ECC

There is a special case for legal ECC values in the SAC57D5xx. Memory entries that are all zeros (All-0) or all ones (All-1), including the ECC parity bits, are not legal for memory that is checked by ECC. The flash memory allows All-1, corresponding to the status of an erase block, as a valid codeword.

Memories that include addresses in the ECC calculation do not specifically protect against All-0 or All-1. This means that for some addresses All-0 or All-1 may be legal.

All-0 and All-1 memory content is indicated in different ways. For memories that do not include address into the ECC calculation, All-0 and All-1 will be uncorrectable errors. For all memories that include address into the ECC code-bit calculation, since the ECC checkbits depend on the address, it is not possible to generate an uncorrectable error indication for all the possible addresses. Therefore, an All-x content may result in a correctable error.

Notice that for flash memory, additional dedicated safety mechanisms exist to detect failures that have the potential of leading to an All-1 word (see the "Flash Memory Controller (PFLASH)" chapter in the *SAC57D5xx Reference Manual* for more details on flash memory safety mechanisms).

3.2.3 ECC failure handling

Single-bit and double-bit errors (correctable and uncorrectable errors) are signaled to the FCCU.

3.3 Clock and power monitoring

3.3.1 Clock

Clocks in the SAC57D5xx are supervised by Clock Monitor Unit (CMU). The CMU is driven by the FIRC (16 MHz internal oscillator) for independent operation from the monitored clocks. If a supervised clock exceeds or falls below its specified frequency range on the chip, the supervising CMU flags an error that sends a signal to the FCCU.

Clocks supervised by CMU instance are as follows:

- CMU: PLL, FXOSC

NOTE

The CMU is not initialized after reset. SW must check to be sure that the clock is locked at the PLLDIG module and that the CMU is initialized before running any safety functions.

3.3.2 Power

There are two types of voltage supervisors on the SAC57D5xx, Low Voltage Detect (LVD) and High Voltage Detect (HVD) monitors. Safety relevant voltages, (recommended operating voltages), are supervised for voltages that are out of these ranges. Since any voltage running outside of the safety relevant range has the potential to disable the failure indication mechanisms of the MCU (such as FCCU, pads, and so on) the indication of these errors can be used to cause a direct transition of the MCU into the safe state (reset assertion) (see Power Management Controller Digital Interface (PMCDIG) for details on power monitoring).

3.4 Communication controllers

Communication controllers provide the ability to exchange information with external components and therefore fall under the same safety reasoning as I/O peripherals. Yet we assume that for high bandwidth communication controllers additional software measures are employed that do not require redundant communication peripherals.

The following communication controllers do not contain special safety mechanisms (above what is included in them by their protocol specifications):

- CAN
- ENET
- MLB

Typically, software measures for the communication controllers (also called fault-tolerant communication layer) could contain sender identification, sequence numbering, and an acknowledgement mechanism.

3.4.1 Disabling of communication controllers

In the event of a dangerous failure, the MCU offers the capability of disabling transmission of individual channels of communication controllers such as CAN. Such disabling prevents the transmission of erroneous messages while preserving the capability of communicating over the diagnostic bus. Disabling outputs is controlled by resetting

SIUL2_MSCR*n*[SMC] for the pins that are associated with communication controllers where this feature is needed. See "Pin Muxing" details for mapping between functionality, pins, and MSCR*n*, and see the SIUL2 Multiplexed Signal Configuration Register in the SIUL2 chapter for a description of SIUL2_MSCR*n* fields, both of which reside in this chip's Reference Manual.

The FCCU intends to drive EOUT[n] to a fault state whenever FCCU FSM is in fault state or FCCU_CFG[FCCU_SET_CLEAR] is 01b. When the FCCU intends to drive EOUT[n] to a fault condition, the SIUL2 disables the output buffer of such pins for which SIUL2_MSCR*n*[SMC] is cleared and thus disables transmission of erroneous messages until FCCU intends to drive FCCU_F[0] to a non-fault condition. After a communication controller transmission port is disabled, it remains in the same state as long as the FCCU drives EOUT[n] to a non-fault condition. During this mode, the state of weak pull-up/ pull-down remain unchanged.

The application should configure SIUL2_MSCR*n*[SMC] for pins that have active mapping of CAN functionality and ensure those pins do not remain in an undriven state.

3.5 Built-In Self Tests (BIST)

The term BIST indicates the set of built-in hardware mechanisms that can be used (typically at startup) to avoid the accumulation of latent faults. BIST is a mechanism that permits a device to test itself. On the SAC57D5xx, BIST is the main means to meet the requirement on latent faults as defined by the ISO 26262 standard. MBIST for memories is implemented on this MCU. MBIST execution is managed by the STCU2 (see the "Self-Test Control Unit (STCU2)" chapter in this chip's Reference Manual).

3.5.1 BIST during boot

A device BIST is performed every time the device boots. BIST is performed transparently for the application while the device is still under reset. In case the BIST fails, the device is held in reset, but the BIST continues running, if possible, and informs the system of the failure. Application software can start executing when the BIST finishes successfully without detecting a fault. The boot time BIST includes memory BIST for all RAMs and BAR ROM.

3.6 FCCU and failure monitoring

The FCCU offers a hardware mechanism to aggregate error notifications and a configurable means to bring the device to a safe state. No CPU intervention is required for collection and control operation. Error indications are passed from the individual hardware components to the FCCU where the appropriate action is decided (according to the FCCU configuration).

3.6.1 External error indication

Failure of the MCU is signaled to one or two pins, EOUT[0] and EOUT[1]. EOUT[0] can also serve as an error input mechanism (see Fault Collection and Control Unit (FCCU) and the FCCU configuration section in the *SAC57D5xx Reference Manual* for details on the fault output signals).

If only one error pin is used it is recommended that this pin is EOUT[1]. These pins (and corresponding die pads) are not adjacent to VDD pins (or die pads) and therefore the chances of a short circuit masking the error out signal are reduced.

The error indication on pins EOUT[0] and EOUT[1] is controlled by the FCCU.

The error status flag (FCCU_STAT[ESTAT]) can be read to determine whether the FCCU is in an error state. This flag can be written by software to either a 1 (fault) or 0 (operational) when the FCCU is in operational state. Another flag, FCCU_STAT[PhysicErrorPin], is accessible through the register interface. It mirrors the physical state of the FCCU_F[n] external pin's value, though this might differ from the logical state if a toggling protocol is used.

3.6.2 Failure handling

The FCCU is an autonomous module that is responsible for reacting to failure indicators. A different reaction can be configured for each failure source. Overall failure reaction time requires time for detecting, processing, and indicating the error. During this time, the SAC57D5xx could provide incorrect results to the system.

Failure sources include:

- All failure indication signals from modules within the MCU
- Control logic and signals monitored by the FCCU itself. See Fault Collection and Control Unit (FCCU) for details.

- Software-initiated failure indications. For example, software signals the FCCU that it has evidence of a failure. Keep in mind that software can also directly influence the state of the EOUT[*n*] pins.
- External failure input

Available failure reactions are:

- Assertion of an interrupt (maskable or non-maskable)
- Resetting the chip
- Changing the state of the failure indication pins, EOUT[*n*]
- Disabling the transmission capabilities of communication controllers (for example, FlexCAN, LINFlexD) (note: possible only in conjunction with changing the state of the failure indication pins)
- No reaction

Software can read the failure source that caused a fault, and can do so either before, or after, a functional reset (the condition indicators are not volatile). Software can also reset the failure, but the external failure indication will stay in failure mode for a configurable minimum time. If necessary, software can also reset the MCU.

3.6.3 Fault inputs

The table "FCCU Non-Critical Faults Mapping" in the "Fault Collection and Control Unit" chapter" of the *SAC57D5xx Reference Manual* shows the source of the fault signals and the type of fault input to which these signals are connected at the FCCU.

3.7 Common Cause Failure measures

Various measures are included to prevent CCFs from endangering the effectiveness of the replication of the Safety Core or of the peripherals. These measures include supervision of clock, power, temperature, test and debug signals. In general these measures are independent from the software.

Also, there are several functional configuration registers throughout the MCU where, if they erroneously change, they can affect the execution of the MCU's safety function and, at the same time, disable the respective safety mechanism. These registers in particular are either protected against bit flips or those flips are detected by independent measures. These same registers are also protected against accidental software writes by employing as well the register protection safety feature.

3.8 Operational interference protection

Being a multi-master system, SAC57D5xx provides safety mechanisms to prevent non-safety masters from interfering with the operation of the masters performing safety-relevant tasks, as well as mechanisms to handle the concurrent execution of software with different (lower) ASIL. Interference freedom is guaranteed via a hierarchical memory protection schema including:

- MPUs
- PBRIDGEs
- Register protection.

There are two Memory Protection Unit levels included in the SAC57D5xx. The Core Memory Protection Unit (CMPU), present on the ARM Cortex M4, is a mechanism included to protect address ranges against access by software developed according to lower ASIL. It will typically be used by the operating system to ensure inter-task interference protection.

The second memory protection level is provided by the Extended Resource Domain Controller (XRDC). It will prevent access of different bus masters to address ranges and will typically be used by the safety application to prevent non-safety modules from accessing the application's safety-relevant resources.

Furthermore, the PBRIDGE can restrict read and write access to individual I/O modules based on the origin of the access and its state (user mode/supervisor mode).

Finally, the register protection included allows individual registers to be "locked" against any manipulation without unlocking.

These safety mechanisms are further described in the "Extended Resource Domain Controller (XRDC)" chapter in this chip's Reference Manual.

Chapter 4
Hardware requirements

4.1 Hardware requirements on system level

This section describes the system level hardware safety measures needed to complement the integrated safety mechanisms of the SAC57D5xx.

The SAC57D5xx integrated safety concept enables SPFs and latent failures to be detected with high diagnostic coverage. However, not all CMFs may be detected. In order to detect failures which may not be detected by the SAC57D5xx, it is assumed that there will be some separate means to bring the system into Safe state$_{\text{system}}$.

Figure 4-1 depicts a simplified application schematic for a functional safety-relevant application in conjunction with an external IC (only functional safety related elements shown). The supplies generated from the external IC should be protected against voltage over the absolute maximum rating of the device (as documented in the SAC57D5xx Data Sheet in section "Absolute maximum ratings").

The external circuit will also monitor the EOUT[*n*] signals. Through a digital interface (for example, SPI), the SAC57D5xx repetitively triggers the watchdog of the external IC. If there is a recognized failure (for example, watchdog not being serviced, assertion of EOUT[*n*]), the reset output of the external IC will be asserted to reset the SAC57D5xx. A fail-safe output is also available to control or deactivate any fail-safe circuitry (for example, power switch).

There is no requirement that these external measures are provided in one IC or even in the specific way as described (for example, the external watchdog functionality can be provided by another component of the system that can recognize that the chip stopped sending periodic packets on a communication network).

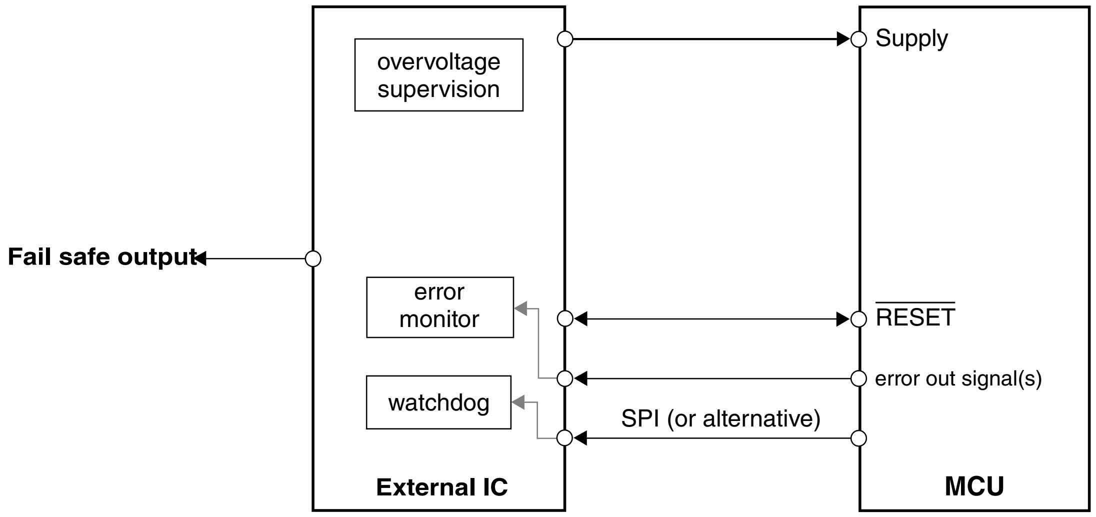

Figure 4-1. Functional safety related connection to external circuitry

4.1.1 Assumed functions by separate circuitry

This section describes external components used in a system in conjunction with the SAC57D5xx for safety-related systems.

It should be noted that failure modes of external services are only partially considered in the FMEDA of the SAC57D5xx (for example, clock(s), power supply), and must be fully analyzed in the system FMEDA by the safety system developer.

4.1.1.1 High impedance outputs

If the SAC57D5xx is considered to be in a Safe state$_{MCU}$ (for example, unpowered and outputs tristated), the system containing the SAC57D5xx may not be compliant with the Safe state$_{system}$. A possible system level safety measure to achieve Safe state$_{system}$ may be to place pull-up or pull-down resistors on I/O when the high-impedance state is not considered safe.

Assumption: [SM_038] If a high-impedance state on an output pin is not safe, pull-up or pull-down resistors shall be added to safety-related outputs. The need for this will be application dependent for the unpowered or reset (tristated I/O) SAC57D5xx.[end]

Rationale: In order to bring the safety-related outputs to such a level, that a Safe state$_{system}$ is achieved.

4.1.1.2 External Watchdog (EXWD)

An external device, acting as an independent timeout functionality (for example, External Watchdog (EXWD)), should be used to cover Common Mode Failures (CMF) of the SAC57D5xx for safety-related systems.

The trigger may be a discrete signal(s) or message object(s). If within a defined timeout period the EXWD is not triggered, a failure will be considered to have occurred which would then switch the system to a Safe state$_{system}$ within the FTTI (for example, the EXWD disconnects the SAC57D5xx from the power supply, or communication messages are invalidated by disabling the physical layer driver).

Assumption under certain conditions: [SM_041] Timeout functionality (for example, EXWD) external to the MCU may improve Common Mode Failure (CMF) robustness. If a failure is detected, the external timeout function must switch the system to a Safe state$_{system}$ within the FTTI.[end]

The implementation of the communication between the SAC57D5xx and the EXWD can be chosen by the user as warranted by the application. Examples of different mechanisms that can be used to trigger the EXWD can include any of the following:

- Serial link (SPI)
- Toggling I/O (GPIO)
- Periodic message frames (CAN)

4.1.1.3 Power Supply Monitor (PSM)

Supply voltages outside of the specified operational ranges may cause permanent damage to the SAC57D5xx, even if it is held in reset.

Assumption: [SM_042] It is assumed that safety measures on system level maintain the Safe state$_{system}$ during and after any supply voltage above the specified operational range. [end]

The *SAC57D5xx Microcontroller Data Sheet* provides specific operating voltage ranges that must be maintained.

Assumption: [SM_087] It is assumed that the external power is supervised for high and low deviations where no supervision is provided on the MCU. [end]

Assumption: [SM_088] It is assumed that the MCU is kept in reset if the external voltage is outside specification and is protected against voltage over the absolute maximum rating of the device (as documented in Data Sheet in section "Absolute maximum ratings"). [end]

If the power supply is out of range, *SAC57D5xx* shall be kept in reset or unpowered, or other measures must possibly be used to keep the system in a safe state. Overvoltage outside the specified range of the technology may cause permanent damage to the *SAC57D5xx* even if kept in reset.

Implementation hint: An external and independent device may provide an over voltage monitor for the external SAC57D5xx supplies. If the supplied voltage supply is above the recommended operating voltage range of the SAC57D5xx, the SAC57D5xx should be maintained with no power. The external power supply monitor will switch the system to a Safe $state_{system}$ within the FTTI, and maintain it in Safe $state_{system}$ (for example, over-voltage protection with functional safety shut-off, or a switch-over to a second power supply unit).

If the SAC57D5xx power supply can be designed to avoid any potential of over-voltage, the external voltage monitoring can be excluded from the system design.

Over-voltage on some supplies will be detected by the SAC57D5xx itself, but system level measures might be required to maintain the Safe $state_{system}$ in case an over-voltage situation may cause damage to the SAC57D5xx.

4.1.1.4 Error Out Monitor (ERRM)

If the SAC57D5xx signals an internal failure on its error out signals (EOUT[0], and/or EOUT[1]), the system may no longer rely on the integrity of the other SAC57D5xx outputs for safety functions. If an error is indicated, the system has to switch to, and remain in, Safe $state_{system}$ without relying on the SAC57D5xx. Depending on its functionality, the system might disable or reset the device as a reaction to the error indication (see **Assumptions** in Safe states).

The safety system developer can choose between two different methods of interfacing to the FCCU:

- Both FCCU signals connected to an external device
- Only a single FCCU signal connected to an external device

Assumption: [SM_043] The overall system needs to include measures to monitor EOUT[*n*] of the MCU and move the system to a Safe $state_{system}$ when an error is indicated. [end]

4.1.1.4.1 Both FCCU signals connected to separate device

In this configuration the separate device continuously monitors the outputs of the FCCU. Thus, it can determine if the FCCU is not working properly.

This configuration does not require any dedicated software support.

Assumption: [SM_201] If both error out signals are connected to an external device, the external device shall check both signals, taking into account the behavior of the two pins. [end]

NOTE

See “EOUT interface” section in the “Fault Collection and Control Unit (FCCU)” chapter of the *SAC57D5xx Reference Manual* for details.

Rationale: To check the integrity of the FCCU, and FCCU signal routing on the system level

Implementation hint: Monitoring the error output signals with combinatorial logic (for example, XOR gate) can generate glitches. Oversampling these signals reduces the possibility that glitches will occur.

4.1.1.4.2 Single FCCU signal connected to separate device

A single signal, EOUT[0] (or EOUT[1]), is connected to a separate device.

If a fault occurs, the FCCU communicates the fault to the separate device through the EOUT[0] (or EOUT[1]).

The functionality of EOUT[0] (or EOUT[1]) can be checked in the following manner:

- EOUT[0] (or EOUT[1]) read back internally.
- EOUT[0] (or EOUT[1]) connected externally to a GPIO.
- EOUT[0] (or EOUT[1]) uses time domain coding (for example, is active for a deterministic time interval).
- Test the ability of EOUT[0] (or EOUT[1]) to disable system functionality (for example, measure voltage available at a motor if EOUT[0] (or EOUT[1]) is expected to disable its power supply).

The system integrator chooses which solution best fits the system level functional safety requirements.

The advantage of a single EOUT[*n*] signal being used instead of using both EOUT[*n*] signals as in the previous section, is the lack of need for the separate device to compare the EOUT[*n*] signals.

4.1.1.4.2.1 Single FCCU signal connected to separate device using voltage domain coding

Recommendation: If EOUT[0], or EOUT[1], is connected to a device not using time domain coding, verification is needed that the EOUT[*n*] signal(s) are operating correctly before execution of any safety function can start.

Rationale: To check the integrity of EOUT[0], or EOUT[1]

To verify the functionality of a EOUT[*n*] signal, a fault may be injected into one of the EOUT[*n*] signals. The behavior of the signal can then be verified by the other EOUT[*n*] signal, or GPIO. Additionally, the fault output mode can be configured to one of the test modes to control one EOUT[*n*] as an output while the other EOUT[*n*] pin is an input or output. For example, TEST0 mode configures EOUT[0] as an input and EOUT[1] as an output. This test mode can be used to check the state of the EOUT[0] input by reading FCCU_EINOUT[EIN0]. Likewise, the user can control the EOUT[1] output by modifying FCCU_EINOUT[EOUT1].

Since the FCCU will be monitoring the system, it is sufficient to check EOUT[0] (or EOUT[1]) within the L-FTTI (for example, at power-up) to help reduce the risk of latent faults. It is recommended that EOUT[*n*] be checked once before the system begins performing any safety-relevant function.

Assumption: [SM_170] If the system is using the MCU in a single error output configuration, the application software will need to configure the signals, and pads, adjacent to EOUT[0] (or EOUT[1]) to have a lower drive strength, and the error output signal is configured with highest drive strength. [end]

Using a lower drive strength on the GPIO near EOUT[0] (or EOUT[1]) will result in the higher drive strength of EOUT[*n*] to effect the logic level of the neighboring GPIO in the event of a short circuit. Software may configure the slew rate for the relevant GPIO in the Multiplexed Signal Configuration Register (SIUL2_MSCR*n*) and Input Multiplexed Signal Configuration Register (SIUL2_IMCR*n*).

4.1.1.4.2.2 Single FCCU signal connected to separate device using time domain coding

Rationale: Decode the time domain coding

Implementation hint: If a single FCCU signal (EOUT[0], or EOUT[1]), is connected to a separate device applying time domain coding (for example, a decoder), a window timeout or windowed watchdog function, is good practice.

Since the FCCU is a safety mechanism, it is sufficient to implement a time domain interval in the range of the L-FTTI.

4.1.2 Optional hardware measures on system level

As input/output operations are highly application dependant, functional safety of input/output modules and peripherals should be assessed on a system level. The following sections provide examples of possible functional safety mechanisms regarding input/output operations.

4.1.2.1 External communication

Assumption under certain conditions: [SM_044] When data communication is used in the implementation of a safety function, then system level functional safety mechanisms are required to achieve the necessary functional safety integrity of communication processes. [end]

Recommendation: System level measures to detect or avoid transmission errors, transmission repetitions, message deletion, message insertion, message resequencing, message corruption, communication delay and message masquerade improves the robustness of communication channels.

4.1.2.2 PWM output monitor

The SAC57D5xx timer modules may require system-level safety measures in order to achieve high functional safety integrity levels.

Assumption under certain conditions: [SM_045] When PWM outputs are used in the implementation of a safety function, suitable system level functional safety integrity measures are assumed to monitor these signals. [end]

Rationale: System level measures to detect or avoid erroneous PWM output signals improves the safety integrity of PWM channels.

Monitoring can be implemented explicitly by monitoring the PWM signal directly with an external device. The PWM signal may be monitored implicitly, by implementing an indirect PWM feedback loop (for example, measuring average current flow of a full bridge driver). This approach may use diverse implementations of input modules (for example, the analog to digital converter).

The specific PWM features that are to be managed by system level safety measures are:

- Dead-time may need to always be positive, and greater than the maximum value of T_{ON} or T_{OFF} of the inverter switches.
- Open GPIO, and shorts to supply or ground, may need to be detected. This can be accomplished, for example, by an external feedback mechanism to a timer module of the SAC57D5xx capable of performing input capture functionality.

The system must be switched to Safe $state_{system}$ if the SAC57D5xx detects an error.

To reduce the likelihood of erroneous control (for example, a motor control application with dead-time requirements to reduce the likelihood of short circuits destroying the motor) in functional safety applications using I/O to control an actuator with a short FTTI, functional safety requires system level supervision if the maximum fault indication time and fault reaction time of SAC57D5xx exceeds the FTTI of the actuators.

If the PWM signals drive switches of a power stage (for example, bridge driver), the timer may not be fast enough to detect a dead-time fault because its fault indication time is often greater than the time required to avoid destruction of the power stage.

4.2 PowerSBC

The system basis chips MC33907 and MC33908 (PowerSBC) from NXP are ideally suited to be used in combination with SAC57D5xx to serve as a separate device as mentioned in Assumed functions by separate circuitry.

The MC33907/08 is a multi-output power supply integrated circuit including enhanced functional safety features.

Figure 4-2 depicts a simplified application schematic for a safety-related system in conjunction with the SAC57D5xx.

Out of a single battery supply with a wide voltage range (V_{SUP}, 3.5 V…28 V), the MC33907/08 generates 5 V (V_{CCA}) and 3.3 V (V_{CORE}) to supply the SAC57D5xx as well as an auxiliary voltage (V_{AUX}) to supply other devices (for example, sensors or separate

ICs). The 1.2 V for digital core supply is generated by an external ballast transistor from V_{CORE}. All voltages generated in the MC33907/08 are independently monitored for under and over voltage.

The MC33907/08 also monitors the state of the error out pins EOUT[0] and EOUT[1], using the bistable protocol. Via SPI, the SAC57D5xx repetitively triggers the windowed watchdog of the MC33907/08 with a valid answer. A dedicated fail safe state machine is implemented to bring and maintain the application in Safe $state_{system}$. In case of a failure (for example, the watchdog is not serviced correctly), RSTb is asserted low to reset the SAC57D5xx. A fail-safe output (FS0b) is available to control or deactivate any fail-safe circuitry (a power switch, for example). Another fail-safe output is available with PWM encoding for error indication (a warning lamp, for example). MC33907/08 also includes hardware Built-In Self-Tests (BIST).

An interrupt output (INTb) is connected to an IRQ input of the SAC57D5xx.

By a connection of the signal MUX_OUT to an ADC input of SAC57D5xx, further diagnostic measures are possible (for example, reading temperature or measuring V_{BATT}). Digital inputs (IO_4, IO_5) may be used for monitoring error signal handling of other devices. Additionally, MC33907/08 may act as a physical interface to connect the SAC57D5xx directly with a CAN or LIN bus.

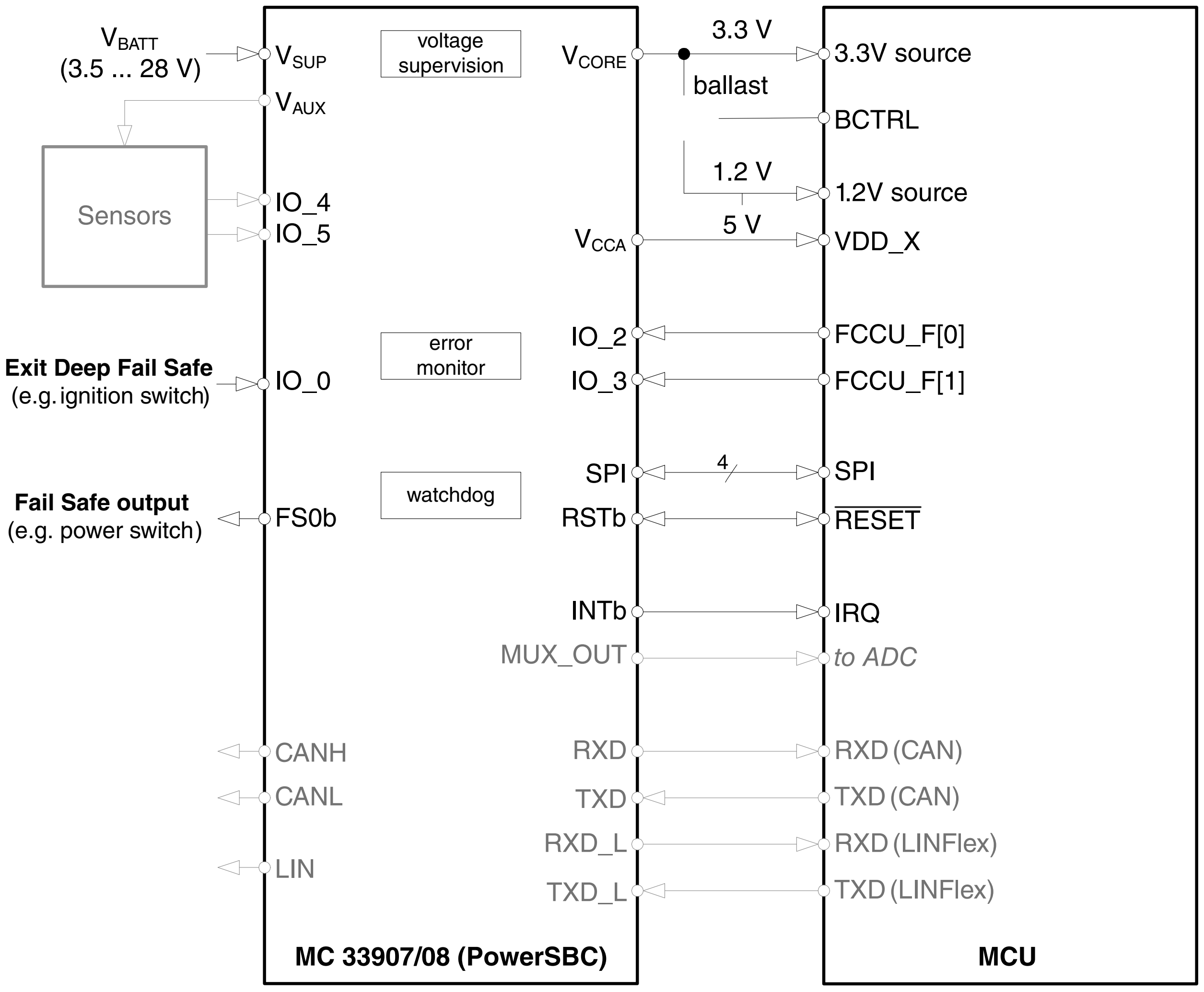

Figure 4-2. Functional safety application with PowerSBC

NOTE

Please see the Data Sheet for the full list of supply names.

Chapter 5
Software requirements

5.1 Software requirements on system level

This section lists required, or recommended, safety measures which should be in place when using the SAC57D5xx in safety systems.

The SAC57D5xx on-chip modules not explicitly mentioned here do not require specific safety measures to be used in safety systems.

5.1.1 Disabled modes of operation

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

5.1.1.1 Debug mode

The debugging facilities of the SAC57D5xx pose a possible source of failures if they are activated during the operation of functional safety-relevant applications. They can halt the cores, cause breakpoints to hit, write to core registers and the address space, activate boundary scan, and so on. To reduce the likelihood of interference with the normal operation of the application software, the SAC57D5xx may not enter debug mode . The state of the JCOMP signal determines whether the system is being debugged or whether the system operates in normal operating mode. When JCOMP is logic low, the JTAGC TAP controller is kept in reset for normal operating mode. When it is logic high, the JTAGC TAP controller is enabled to enter debug mode. During boot, measures must be taken to ensure that JCOMP is not be asserted by external sources so entering debug mode can be avoided.

Assumption: [SM_047] Debugging will be disabled in the field while the device is being used for safety-relevant functions. [end]

Assumption under certain conditions: [SM_048] If modules like the Software Watchdog Timer (SWT), System Timer Module (STM), Serial Peripheral Interface (SPI), Periodic Interrupt Timer (PIT), Fault Collection Control Unit (FCCU), FlexCAN, or in general any modules which can be frozen in debug mode, are functional safety-relevant, it is required that application software configure these modules to continue execution during debug mode, and not freeze the module operation if debug mode is entered. [end]

Rationale: To improve resilience against erroneous activation of debug mode

Implementation hint: In debug mode, the FRZ bit in the SWT_CR register controls operation of the SWT. If the SWT_CR[FRZ] = 0, the SWT counter continues to run in debug mode.

In debug mode, STM_CR[FRZ] controls operation of the STM counter. If the STM_CR[FRZ] = 0, the counter continues to run in debug mode.

When FTM_CONF[BDMODE] = 11b, the FlexTimer (FTM) continues to run while the device is in debug mode.

The SPI_MCR[FRZ] controls SPI behavior in the debug mode. If SPI_MCR[FRZ] = 0, the SPI continues all active serial transfers when the device in the debug mode.

CAN_ MCR[FRZ] controls FlexCAN Module behavior in the debug mode. If the CAN_ MCR[FRZ] = 0, the FlexCAN Module continues communication (not affected by debug mode) when the device in the debug mode.

In debug mode, PIT_MCR[FRZ] controls operation of the PIT counter. If the PIT_MCR[FRZ] = 0, the counter continues to run in debug mode.

RTC_CTRL[FRZEN] controls the Real Time Clock behaviour in debug mode. If RTC_CTRL[FRZEN] = 0, the timers will continue when the device is in debug mode.

If DMA_CR[EDBG] = 0, the eDMA continues to operate in debug mode.

I2C_IBDBG[IPG_DEBUG_EN] controls the I2C module behavior in debug mode. If I2C_IBDBG[IPG_DEBUG_EN] = 0, the module will continue normal operation when the device is in debug mode, however bus idle interrupts will be disabled.

ENET_ECR[DBGEN] controls the Ethernet MAC behavior in debug mode. If ENET_ECR[DBGEN] = 0, the module will continue when the device is in debug mode.

5.1.1.2 Test mode

Several mechanisms of the SAC57D5xx can be circumvented during test mode which endangers the functional safety integrity.

Assumption: [SM_049] Test mode is used for comprehensive factory testing and is not valid for normal operation. Test mode may not be used during normal operating mode without an explicit agreement from NXP Semiconductors. [end]

Implementation hint: The VPP_TEST pin is for test purposes only, and must be tied to GND during normal operating mode. From a system level point of view, measures must ensure that the VPP_TEST pin is not connected to V_{DD} during boot to avoid entering test mode.

5.2 MCU modules

5.2.1 Video Interface Unit (VIU4)

The ability to detect a frozen video stream is one of the safety functions defined for SAC57D5xx. If a video stream is used for a reverse camera application a frozen image may not be obvious to a driver moving the vehicle at slow speed, so could prevent an obstacle from being seen. The following methods can be used to detect if the video stream has frozen.

5.2.1.1 VIU frozen image detect

The VIU features two registers for the purposes of monitoring the input video stream:

- Detected input Video Pixel and Line Count (DINVSZ)
- Detected Input Video Frame Length (DINVFL)

DINVSZ captures the detected number of active lines in each video frame and the number of active pixels in each input video line. The values captured in DINVSZ are pixel clock based. DINVFL captures the detected total number of lines of the input video frame and the number of clock cycles of each input video line. The values captured in DINVFL are based on the VIU module clock rather than the pixel clock. Detection is based on the original hsync and vsync. If hsync or vsync are frozen the counter will increment to its maximum value which can be detected by the application software and indicates a frozen frame.

Recommendation: Use application software to scan these registers regularly to determine if image stream has frozen.

5.2.1.2 Hidden pattern inclusion

An application software method can also be used to detect a frozen video stream by adding a small pattern to the video frame in buffer RAM. The manipulating core can add a small overlay pattern to the video frame stored in RAM (for example, 3 pixels in different colors). A subsequent video frame would overwrite the pattern in RAM. Therefore the manipulating core can check that the pattern has been overwritten prior to re-writing the pattern. If no pattern is present, the image has been overwritten and can be used. If the pattern is still present, no new image has been provided and the existing image can be considered stale.

Assumption: Application software will either ensure that the image is removed or provide an indication to the driver to no longer rely upon the image displayed.

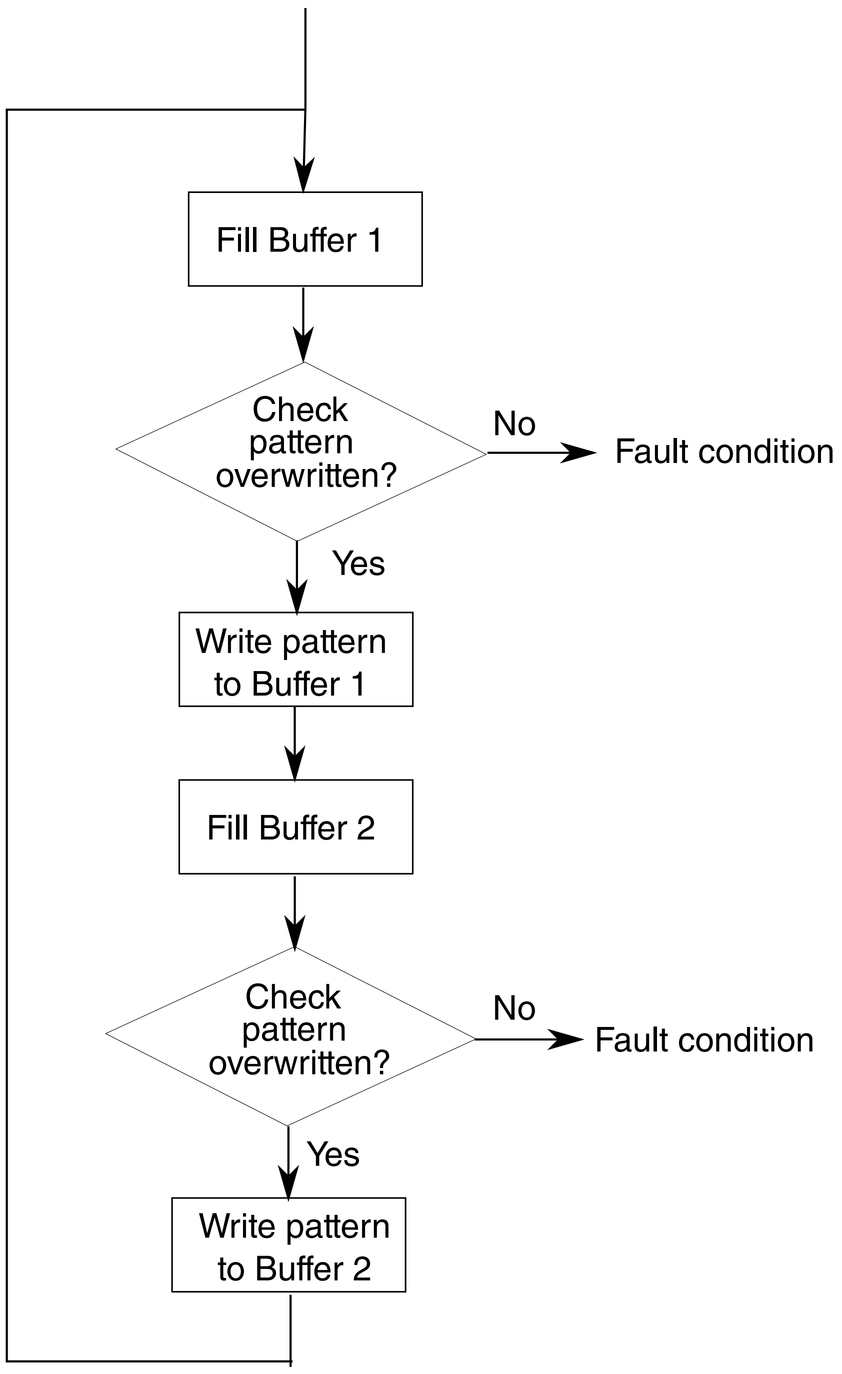
Fill Buffer 1
Check pattern overwritten?
No
Fault condition
Yes
Write pattern to Buffer 1
Fill Buffer 2
Check pattern overwritten?
No
Fault condition
Yes
Write pattern to Buffer 2

Figure 5-1. Hidden pattern inclusion diagram

5.2.2 Display Controller (2D-ACE)

The 2D-ACE supports a write-back mode where the final output image is written back into system memory.

Implementation hint: This mode can be used as a way of self testing the 2D-ACE. For example if a 6 plane blend was performed on 6 overlapping layers the resulting output image could be written back in system memory while also being displayed on the TFT. A CRC (or CPU comparison) of the write-back data could then be compared with a pre-calculated CRC, providing a check that the 2D-ACE is operating correctly. This method of self check is supported on both 2D-ACE modules (see section "Write-back Operation" in the "Display Controller (2D-ACE)" chapter of the SAC57D5xx Reference Manual for details).

5.2.3 Fault Collection and Control Unit (FCCU)

The FCCU uses a hardware fail safe interface which collects faults and brings the device to a Safe state$_{MCU}$ when a failure is recognized.

All faults detected by hardware measures are reported to the FCCU. The FCCU monitors critical control signals and collects all errors. Depending on the type of fault, the FCCU places the device into an appropriately configured Safe state$_{MCU}$. To achieve this, application software only has to configure the FCCU appropriately. No CPU intervention is required for collection and control operation, unless the FCCU is specifically configured to cause software intervention (by triggering IRQs or NMIs).

The FCCU offers a systematic approach to fault collection and control. It is possible to configure the reaction for each fault source separately. The distinctive features of the FCCU are:

- Collection of error information from modules whose behavior is essential to the functional safety goal
- Configurable and graded fault control:
  - Internal reactions
    - No reset reaction
    - IRQ
    - Functional Reset
  - External reaction (external failure reporting using EOUT[*n*])

The table "FCCU non-critical fault (NCF) mapping" in the "Chip-specific FCCU information" section of the *SAC57D5xx Reference Manual* shows the source of the fault signals and the type of fault input to which these signals are connected at the FCCU.

The FCCU has two external signals, EOUT[0] and EOUT[1], through which critical failures are reported. When the device is in reset or unpowered, these outputs are tristated.

EOUT[*n*] are intended to be connected to an independent device which continuously monitors the signal(s). If a failure is detected, the separate device switches to and maintains the system to a Safe state$_{system}$ condition within the FTTI (for example, the separate device disconnects the SAC57D5xx device or an actuator from the power supply).

5.2.3.1 Initial checks and configurations

Besides the possible initial configuration, no intervention from the SAC57D5xx is necessary for fault collection and reaction.

Assumption: [SM_053] Before starting safety-relevant operations, software must ensure that the fault reaction to each safety-relevant fault are configured. [end]

Rationale: Maintain the device in the Safe state$_{system}$ in case of failure

Implementation hint: The FCCU fault path is enabled by configuring FCCU registers (for example, FCCU_NCF_CFG0, FCCU_NCFS_CFG0, FCCU_NCF_TOE0, and so on). These registers are writable only if the FCCU is in the CONFIG state.

If a CMU monitors a FMPLL generated clock, and that clock is not used or is not used for functional safety critical modules, error masking and limited internal reaction of the module using that clock is acceptable.

External reaction of the FCCU is always enabled and can not be disabled.

Assumption under certain conditions: [SM_054] If the outputs of the system I/O need to be forced to a high impedance state upon entering safe mode, MC_ME_SAFE_MC[PDO] = 1 needs to be written. [end]

Assumption:[SM_272] If the SAC57D5xx signals an internal failure via its error out signals (FCCU_F[*n*]), the system can no longer safely use the SAC57D5xx safety function outputs. If an error is indicated, the system has to be able to remain in Safe state$_{system}$ without any additional action from the SAC57D5xx. Depending on its functionality, the system might disable or reset the SAC57D5xx as a reaction to the indicated error. [end]

5.2.3.2 Runtime checks

Assumption:[SM_155] If the SAC57D5xx is continuously switching between a standard operating state and reset, or fault state, without a device shutdown, system level measures should be implemented to ensure that the system meets the Safe state$_{system}$ criteria. [end]

Implementation hint: Software may be implemented to reduce the likelihood of cycling between a functional and fault states. For example, in the case of periodic non-critical faults, the software could clean the respective status and periodically move the device from a fault state to normal state. This procedure may help avoid the possible looping between functional and fault states.

To prevent permanent cycling between a functional state and a fault state, software will need to keep track of cleaned faults, stop cleaning the faults and stay in a Safe state$_{MCU}$. An exception to this would be if there was an unacceptably high occurence of necessary fault cleaning. The limit for the number and frequency of cleaned faults is application dependent. This may only be relevant if continuous switching between a normal operating state and a reset state (as the failure reaction) is not a Safe state$_{system}$.

Assumption: [SM_148] Before resetting the functional and destructive reset counters, the application software shall ensure that it can detect longer reset cycles caused by faults in normal operation. [end]

NOTE

Longer reset cycles refers to length of time since the previous reset.

Implementation Hint: Before the safety application clears the reset counters it reads and saves the FCCU error status indication (if any faults were found) and compares the status with the previous saved versions. If several consecutive resets are caused by the same FCCU fault, or if too many resets due to faults are observed, software can take action, such as causing a destructive reset.

5.2.4 Reset Generation Module (MC_RGM)

5.2.4.1 Initial checks and configurations

Implementation hint: It is good practice to configure a second failure notification channel to communicate redundant critical application faults.

Implementation hint: To enable critical events to trigger a reset sequence, MC_RGM_FERD = 0 should be written. If particular events are excluded, MC_RGM_FEAR shall be configured to generate an alternate request in these cases.

To trigger a reset of the device by software, the MC_ME_MCTL[TARGET_MODE] shall be used. Writing MC_ME_MCTL[TARGET_MODE] = 0000b causes a functional reset where writing MC_ME_MCTL[TARGET_MODE] = 1111b causes destructive reset (see section "Reset Generation Module (MC_RGM)" of the *SAC57D5xx Reference Manual* for details).

5.2.4.1.1 Consecutive resets

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 sequence. Allowing such cycles would be problematic as it would allow an unlimited number of attempts.

To detect a loop of resets, the SAC57D5xx 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 between writes to the MC_RGM_FRET register. 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 power-on reset.

Assumption: [SM_059] The application software should reset the functional reset counter every time it has finished checking its environment during startup. [end]

Assumption: [SM_279] Safety Software shall clear the destructive reset counter every time after a reset, after it has been established that the MCU is working correctly. [end]

Assumption: [SM_060] Since the default setting for the destructive reset counter is disabled, the SW must enable the counter by writing a non zero value to the MC_RGM_DRET register. The functional reset counter is enabled at reset. [end]

5.2.5 Self Test Control Unit (STCU2)

The STCU2 executes built-in self-test (MBIST) and gives reaction to detected faults by signaling faults to either the MC_RGM or to the FCCU (see "Self-Test Control Unit (STCU2)" in the *SAC57D5xx Reference Manual* for details).

5.2.5.1 Initial checks and configurations

The STCU2 does not require any configuration performed by application software.

Assumption under certain conditions: [SM_062] When built-in self-test circuits of the SAC57D5xx are used as functional safety integrity measure (for example, to detect random faults, latent fault detection, and single-point fault detection) in a functional safety system, functional safety integrity measures on system level shall be implemented ensuring STCU2 integrity during/after STCU2 initialization but before executing a safety function. [end]

Rationale: The STCU2's correct behavior shall be verified by checking the expected results by software.

Implementation hint: Software shall confirm that all MBISTs finish successfully with no additional errors flagged.

This software confirmation prevents a fault within the STCU2 itself from incorrectly indicating that the built in self-test passed.

This is an additional functional safety layer since the STCU2 signals faults to the MC_RGM or FCCU. So, reading STCU2_MBSL, STCU2_MBSH*n*, STCU2_MBEL, STCU2_MBEH*n*, STCU2_ERR_STAT, and STCU2_ERR_FM registers help increase the STCU2 self-test coverage.

Implementation hint: The STCU2 shall be configured (in test flash memory) to execute the MBIST before activating the application safety function (see section "STCU2 Configuration Register (STCU2_CFG)" in the "Self-Test Control Unit (STCU2)" chapter of the *SAC57D5xx Reference Manual*).

5.2.6 Software Watchdog Timer

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 in an excessive 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. When a timeout occurs, a trigger to the FCCU can be generated immediately, or the SWT can first generate an interrupt and load the down-counter with the timeout period. If the service sequence is not written before the second consecutive timeout, the SWT drives its FCCU channel to trigger a fault (see "FCCU non-critical fault (NCF) mapping" table shown in section "Chip-specific FCCU information" of the *SAC57D5xx Reference Manual*).

Assumption: [SM_067] Before the safety function is executed, the SWT must be enabled and configuration registers hard-locked against modification. [end]

Assumption: [SM_202] The SWT time window settings must be set to a value less than the FTTI/PST. Detection latency shall be smaller than the FTTI/PST. [end]

Assumption: [SM_203] Before the safety function is executed, software must verify that the SWT is enabled by reading the SWT control register (SWT_CR). [end]

Implementation hint: To enable the SWT and to hard-lock the configuration registers, the SWT control register flags SWT_CR[WEN] and SWT_CR[HLK] need to be asserted. The timeout register (SWT_TO) should contain a 32-bit value that represents a timeout less than the FTTI/PST.

In general, it is expected that the SWT helps to detect lost or significantly slow clocks. Thus, the SWT needs to be used to also detect hardware faults, not only to detect software faults. Using the SWT to detect clock issues is a secondary measure since there are primary means for checking clock integrity (for example, by CMUs).

The SAC57D5xx provides the hardware support (SWT) to implement both control flow and temporal monitoring methods. If Windowed mode and Keyed Service mode (two pseudorandom key values used to service the watchdog) are enabled, it is possible to reach a high effective temporal flow monitoring.

Assumption: [SM_069] It is the responsibility of the application software to insert control flow checkpoints with the required granularity as required by the application. [end]

Two service procedures are available:

- A fixed service sequence represented by a write of two fixed values (A602h, B480h) to the SWT service register. Writing the service sequence reloads the internal down counter with the timeout period.
- The second is based on a pseudo-random key computed by the SWT every time it is serviced and which is written by the software on the successive write to the service register. The watchdog can be refreshed only if the key calculated in hardware by the watchdog is equal to the key provided by software which may calculate the key in one or more procedure/tasks (so called signature watchdog). The 16-bit key is computed as $SK_{(n+1)} = (17 \times (SK_n + 3) \bmod 2^{16}$.

The SWT down counter is always driven by the FIRC clock.

The SAC57D5xx has three instances of the SWT module which can be assigned to each of the three cores.

5.2.6.1 Run-time checks

Implementation hint: Control flow monitoring can be implemented using the SWT. However, other control flow monitoring approaches that do not use the SWT may also be used. When using the SWT, the SWT shall be enabled and its configuration registers shall be hard-locked to prohibit modification by application software.

5.2.7 Cyclic Redundancy Checker Unit

The Cyclic Redundancy Checker Unit (CRC) offloads the CPU in computing a CRC checksum. The CRC has the capability to process two interleaved CRC calculations. The CRC module may be used to detect erroneous corruption of data during transmission or storage. The CRC takes as its input a data stream of any length and calculates a 32-bit output value (signature). There are three sets of CRC registers to allow concurrent CRC computations in the SAC57D5xx.

The contents of the configuration registers of the functional safety related modules shall be checked within the FTTI. The CRC unit should be used to detect accidental alteration of data in configuration registers by calculating its CRC signature and comparing it against a previously calculated CRC.

5.2.7.1 Runtime checks

Parts of the SAC57D5xx configuration registers do not provide the functional safety integrity IEC 61508 series and ISO 26262 requires for high functional safety integrity targets on their own. This relates to systematic faults (for example, application software incorrectly overwriting registers), as well as random hardware faults (bit flipping in registers).

Assumption: [SM_070] The CRC calculation shall be executed at least once per FTTI to verify the content of the safety-relevant configuration registers. [end]

Implementation hint: The CRC of the configuration registers of the modules involved with the safety function should be calculated offline. Online CRC calculation (for example, if some registers are dynamically modified) is possible if an independent source for the expected register content is available.

At run time, the value calculated by the CRC module needs to be identical to the offline value. To avoid overloading the core, the eDMA module can be used to support the data transfer from the registers under check to the CRC module.

Note

For some configuration registers (specifically clock and MCU mode configurations) CRCing is insufficient since the registers are unavailable until an event is triggered. In those instances, additional measures to check correct initial configuration are necessary (for example, clocks checked by the CMUs).

Implementation hint: The CRC module offloads the CPU in computing a CRC checksum. The CRC has the capability to process two different CRC calculations at the same time. To verify the content of the SAC57D5xx configuration registers of the modules involved with the safety function, the CRC module may be used to calculate a signature of the content of the registers and compare this signature with a value calculated during development.

Alternatively, the CPU could be used instead of the CRC module to check that the value of the configuration registers has not been modified. However, using the CRC module is more effective.

Implementation hint: The CRC module could be used to detect data corruption during transmission or storage. The CRC takes as its input a data stream of any length and calculates a 32-bit signature value.

Implementation hint: When the safety function is active (application run time), the same CRC value shall be calculated by the CRC module within the FTTI. To unload the CPU, the eDMA module can be used to support the data transfer from the registers being checked by the CRC module. The result of the runtime computation is then compared to the predetermined value.

The application shall include detection, or protection measures, against possible faults of the CRC module only if the CRC module is used as safety integrity measure or within the safety function.

Implementation hint: An alternative approach would be to use the eDMA to reinitialize the content of the configuration registers of the modules involved with the safety function within the respective FTTI when the safety function is active (application runtime). This approach may require additional measures to detect permanent failures (not fixed by reinitialization). It also needs measures against transfer errors and ignores the fact that some configuration registers cannot be changed except by a mode change.

5.2.7.1.1 Implementation details

The eDMA and CRC modules should be used to implement these safety integrity measures to unload the CPU.

Note

Caution: The signature of the configuration registers is computed in a correct way only if these registers do not contain any volatile status bit.

5.2.8 Fast Internal RC Oscillator

The Fast Internal RC Oscillator (FIRC) has a nominal frequency of 16 MHz, but frequency accuracy over the full voltage and temperature range has to be taken into account (see the *SAC57D5xx Data Sheet*). Functional safety-related modules which use the clock generated by the FIRC are:

- FCCU
- CMU
- SWT

In the rare case of an FIRC clock failure, these modules will stop functioning.

5.2.8.1 Initial checks and configurations

The CMU frequency meter shall be used to check the availability and frequency of the internal FIRC. This feature allows measurement of the FIRC frequency using the FXOSC as the reference (IRC_SW_CHECK).

Assumption: [SM_073] The FIRC frequency is measured and compared to the expected frequency of 16 MHz. This test is performed after power-on, but before executing any safety function. Software writes CMU_CSR[SFM] = 1 to start the frequency measurement, and the status of the measurement is checked by reading this same field. When as CMU_CSR[SFM] = 0 the frequency measurement has completed (see "Frequency meter" section in the "Clock Monitor Unit (CMU)" chapter of the *SAC57D5xx Reference Manual* for details.). [end]

Rationale: To check the integrity of the FIRC.

Note

If the FIRC is not operating due to a fault, the measurement of the FIRC frequency will never complete and the CMU_CSR[SFM] flag will remain set. The application may need to manage detecting this condition. For example, implementing a software watchdog which monitors the CMU_CSR[SFM] flag status.

5.2.8.2 Runtime checks

CMU frequency metering shall be used to verify the availability and frequency of the FIRC. This feature allows measurement of the FIRC frequency using the FXOSC as the clock source.

Assumption: [SM_074] To detect failure of the FIRC, the application software shall utilize CMU frequency metering to read the FIRC frequency and compare it against the expected value of 16 MHz[1]. [end]

Implementation hint: See the **Assumption:** in Initial checks and configurations for an explanation on how to use the CMU to check the FIRC.

If the measured FIRC frequency does not match the expected value, there exists the possibility of a complete failure of all safety measures. Software should then bring the system to a Safe state$_{system}$ without relying on the modules driven by the FIRC (for example, FCCU, CMU and SWT).

Recommendation: To increase the fault detection, this functional safety integrity measure should be executed once per FTTI.

5.2.9 Fast External Oscillator (FXOSC)

FlexCAN features a mode in which it is directly clocked from the FXOSC.

5.2.9.1 Initial checks and configurations

Assumption: [SM_075] The FlexCAN features modes to be clocked directly by the FXOSC, but it should not make use of these modes during normal operation unless the effects of clock glitches are sufficiently detected by the applied FT-COM layer.[end]

5.2.9.2 Runtime checks

Assumption: [SM_076] Software shall check that the system clock is available, and sourced by the FXOSC, before running any safety element function or enabling the FCCU into the operational state. [end]

1. Nominal frequency of the FIRC is 16 MHz, but a post trim accuracy over voltage and temperature must be taken into account (see the *SAC57D5xx Data Sheet*).

5.2.10 Dual PLL Digital Interface (PLLDIG)

The SAC57D5xx consists of four FMPLLs used to generate high speed clocks. Each PLL can be configured to enable frequency modulation. The FMPLL provides a loss of lock error indication that is routed to the MC_RGM and the FCCU (NCF[20:23]). If there is no PLL lock, the system clock can be driven by the FIRC. Glitches which may appear on the crystal clock are filtered (low-pass filter) by the FMPLL. FMPLL0 or FMPLL1 can be selected as the system clock and is distributed to most of the SAC57D5xx modules. Frequency modulation can optionally be enabled to reduce EMI.

Other peripherals which require unique or precise timing (for example, FlexTimer, 2D_ACE, QSPI) can be supplied by other FMPLLs which would typically have frequency modulation disabled.

Implementation hint: PLLDIG_PLL*n*SR[LOLF] field indicates that a loss of lock event occurred. The PLLDIG_PLL*n*CR[LOLIE] field can be set to enable an interrupt request upon loss of lock.

5.2.10.1 Initial checks and configurations

After system reset, the external crystal oscillator is powered down and the PLLs are deactivated. Software shall enable the oscillator. Out of reset, the SAC57D5xx uses the Fast Internal RC oscillator (FIRC) as the default clock (see the "Oscillators" chapter in the *SAC57D5xx Reference Manual* and Fast Internal RC Oscillator for details on FIRC configuration).

Assumption: [SM_078] Before executing any safety function, a high quality clock (low noise, low likelihood for glitches) based on an external clock source shall be configured as the system clock of the SAC57D5xx. [end]

Rationale: Since the FIRC is used by the CMUs as reference to monitor the output of the two PLLs, it cannot be used as input of these PLLs.

Implementation hint: The PLLs can be configured to use the Fast External Oscillator (FXOSC) as a clock reference, or an externally provided clock reference. In general MC_CGM_AC0_SC[SELCTL] shall be set to 1, selecting the FXOSC as the source.

Assumption under certain conditions: [SM_079] When clock glitches endanger the system level functional safety integrity, functional safety-relevant modules shall be clocked with an FMPLL generated clock signal, as the PLL serves as a filter to reduce the likelihood of clock glitches due to external disturbances. Alternatively a high quality external clock having low noise and low likelihood of clock glitches shall be used.

Modules which are capable of being clocked directly by the XOSC (such as Real Time Clock, FlexTimer, dSPI or Linflex) and are being used for a safety function should instead be clocked by the PLL. [end]

Rationale: To reduce the impact of glitches stemming from the external crystal and its hardware connection to the SAC57D5xx.

Implementation hint: This requirement is fulfilled by appropriately programming the Clock Generation Module (MC_CGM) and Mode Entry Module (MC_ME).

Either during or after initialization, but before executing any safety function, application software shall check that the SAC57D5xx uses the FMPLL clock as system clock.

Implementation hint: Application software can check the current system clock by checking the MC_ME_GS[S_SYSCLK] flag. MC_ME_GS[S_SYSCLK] = 4 or 5 indicates that an FMPLL clock is being used as the system clock.

5.2.11 Clock Monitor Unit (CMU)

At startup, the CMUs are not initialized and the FIRC is the default system clock. Stuck-at faults on the FXOSC are not detected by the CMUs at power-on since the monitoring units are not initialized and the SAC57D5xx is still running on the FIRC.

The CMUs are driven by the 16 MHz (FIRC) to ensure independence from the monitored clocks. CMUs flag errors associated with conditions due to clock out of a 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. SAC57D5xx includes a CMU which monitors the FXOSC, FIRC, SXOSC and System Clock.

The CMU uses the FIRC (16 MHz internal oscillator) as the reference clock for independent operation from the monitored clocks. The purpose is to check for error conditions due to:

- loss of clock from fast external crystal (FXOSC)
- loss of reference (FIRC)
- loss of clock from slow external crystal (FXOSC)
- system clock out of programmable frequency range (frequency too high or low)

The CMU supervises the frequency range of various clock sources. In case of abnormal behavior, the information is forwarded to the FCCU as faults (see "FCCU non-critical fault (NCF) mapping" table shown in section "Chip-specific FCCU information" of the *SAC57D5xx Reference Manual*).

Assumption: [SM_080] For safety-relevant applications, use of the CMU is mandatory. If the modules that the CMU monitors are used by the application safety function, the user shall verify that the CMU is not disabled and its faults are managed by the FCCU. The FCCU's default condition does not manage the CMU faults, so it must be configured accordingly. [end]

5.2.11.1 Initial checks and configurations

Assumption: [SM_081] The following supervisor functions are required: [end]

- Loss of external clock
- System FMPLL frequency higher than the (programmable) upper frequency reference
- System FMPLL frequency lower than the (programmable) lower frequency reference

Rationale: To monitor the integrity of the clock signals.

Recommendation: The CMU should be used for each clock that is being monitored and used by a functional safety-relevant module. Application software shall check that the CMU is enabled and its faults are managed by the FCCU.

Implementation hint: In general, the following two application-dependent configurations shall be executed before CMU monitoring can be enabled.

- The first configuration is related to the crystal oscillator clock (FXOSC) monitor. Software configures CMU_CSR[RCDIV] to select an FIRC divider. The divided FIRC frequency is compared with the FXOSC.
- The second configuration is related to other clock signals being monitored. The high frequency (CMU_HFREFR[HFREF]) and low frequency references (CMU_LFREFR[LFREF]) are configured.

Once the CMU is configured, clock monitoring will be enabled when software writes CMU_CSR[CME] = 1.

5.2.12 Power Management Controller Digital Interface (PMCDIG)

The PMCDIG manages the supply voltages for all modules on the device. This unit includes the internal regulator for the logic power supply (1.25 V) and a set of voltage monitors. Particularly, it embeds low voltage detectors (LVD) and high voltage detectors (HVD). If one of the monitored voltages goes below (LVD) or above (HVD) a given threshold, a destructive reset is initiated to control erroneous voltages before these cause a CMF (for correct operating voltage ranges please see the *SAC57D5xx Data Sheet*).

To ensure functional safety, the PMCDIG monitors various supply voltages of the SAC57D5xx device (as seen in Table 5-1).

Assumption: [SM_204] It is assumed that the ADC's are used to monitor the bandgap reference voltage of the PMC. [end]

Apart from the ADC monitoring of the bandgap reference voltage, the use of the PMC for safety-relevant applications is transparent to the user.

Undervoltage and overvoltage conditions are primarily reported to the MC_RGM, where they directly cause a transition into a safe state by a reset. This solution was chosen because safety-relevant voltages have the potential to disable the failure indication mechanisms of the SAC57D5xx (the FCCU). The LVDs and HVDs also report errors to the FCCU, but since the LVD and HVD errors are handled by the MC_RGM, the FCCU error reporting is not utilized.

Note

Only for development purposes, different fault reactions can be programmed in the PMC for LVD and HVD error reporting to the FCCU and the MC_RGM reset be disabled.

Assumption: [SM_085] Software must not disable the direct transition by the MC_RGM into a safe state due to an overvoltage or undervoltage indication. [end]

Table 5-1. PMC monitored supplies

Detector Type	Detector Name	Voltage Monitored
LVD_HV_A	Low Voltage Detector - Low Range	5 V or 3.3 V supply to I/O VDDE_A
LVD_HV_AHi	Low Voltage Detector - High Range	5 V of 3.3 V I/O supply to I/O VDDE_A
LVD_FLASH	Low Voltage Detector	3.3 V flash memory supply (VDD_HV_FLA)
LVD_LV_x	Various Low Voltage Detectors	1.2 V core supply (VDD12)
HVD_LV	High Voltage Detector	1.2 V core supply (VDD12)

Over voltage of any 3.3 V or 5 V supply shall be monitored externally as described in Power Supply Monitor (PSM).

5.2.12.1 1.25 V supply supervision

Voltage detectors LVD_LV and HVD_LV monitor the digital (1.25 V) core supply voltage for over and under voltage in relation to a reference voltage. The figure below depicts the logic scheme of the voltage detectors. In case the core main voltage detector detects over or under voltage during normal operation of the SAC57D5xx, a destructive reset is triggered.

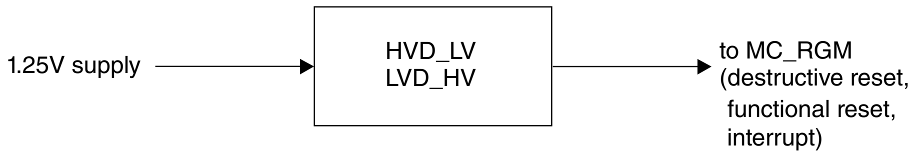

Figure 5-2. Logic scheme of the core voltage detectors

Assumption under certain conditions: [SM_089] When the system requires robustness regarding 1.25 V over voltage failures, the external VREG mode is preferably selected. The internal VREG mode uses a single pass transistor and, therefore, over voltage cannot be shut off redundantly. [end]

Rationale: To enable system level measures to detect or shut down the supply voltage in case of a destructive (multiple point faults) 1.25 V over voltage incident.

Implementation hint: The digital (1.25 V) core supply voltage may be monitored externally and the power supply shut down in case of an overvoltage. An external 1.25 V HVD may detect overvoltage and shut down the 3.3 V supply voltage.

5.2.12.2 3.3 V supply supervision

Voltage detectors LVD_HV_A, LVD_HV_AHi and LVD_FLASH monitor the power supplies for under voltage in relation to a reference voltage. The figure below depicts the logic scheme of the voltage detectors. In case a single LVD detects under voltage during normal operation of the SAC57D5xx, a destructive reset is triggered.

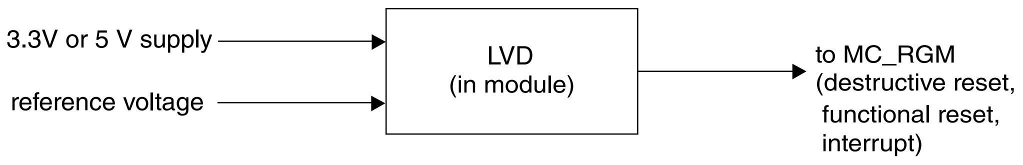

Figure 5-3. Logic scheme of the 3.3 V or 5 V voltage detectors

5.2.13 Extended Resource Domain Controller

As a multimaster, concurrent bus system, the SAC57D5xx provides safety mechanisms to prevent masters executing non-safety relevant code from interfering with the operation of safety-relevant parts of the system. The Extended Resource Domain Controller (XRDC) provides access control, memory protection and peripheral isolation. It allows the access rights of each master to be limited to the resources and modules they have been allocated. This allows safety relevant software to operate concurrently with software with a lower or no ASIL classification.

Recommendation: For safety-relevant applications, the XRDC should be used to ensure that only authorized software tasks can configure modules and all bus masters can access only their allocated resources according to their access rights.

Access restriction at the XRDC level provides protection against unwanted read/write accesses to predefined memory-mapped address locations by unauthorized software routines.

Assumption:[SM_094] The XRDC shall only be programmed by a safety task with the appropriate ASIL. This software shall use the XRDC to block write access to the XRDC configuration registers for all other masters. [end]

5.2.14 PBRIDGE protection

The PBRIDGE access protection can be used to restrict read and write access to individual peripheral modules and restrict access based on the master's access attributes.

- Master privilege level – The access privilege level associated with each master is configurable. Each master can be configured to be trusted for read and write accesses.
- Peripheral access level – The access level of each on-platform and off-platform peripheral is configurable. The peripheral can be configured to require the master accessing the peripheral to have supervisor access attribute. Furthermore, if the peripheral write protection is enabled, write accesses to the peripheral are terminated. The peripheral can also be configured to block accesses from an untrusted master.

Recommendation: Using application software, periodically check the contents of configuration registers (more than 10 registers containing different values) of modules attached to the PBRIDGE to help detect faults in the PBRIDGE.

5.2.14.1 Initial checks and configurations

The application software should configure the PBRIDGEs to define the access permissions for each slave module that requires access protection.

Application software should configure the PBRIDGE to prevent write accesses to the MC_RGM address space for all masters except the core.

5.2.15 Built-in Hardware Self-Tests (BIST)

Built-in hardware self-test (BIST) or built-in test (BIT) is a mechanism that permits circuitry to test itself. Hardware supported BIST is used to speed-up self-test and reduce the CPU load. As hardware assisted BIST is often destructive, it shall be executed ahead or after a reset (destructive reset or external reset).

To ensure absence of latent faults, the self-test executes Memory Built-In Self Test (MBIST) during boot while the device is still under reset (offline). The boot time BIST includes MBIST to test all RAMs and ROMs.[2]

The overall control of the MBISTs is provided by the Self-Test Control Unit (STCU2). The STCU2 will execute automatically after a power-on-reset, external reset and destructive reset, and it will also execute when initiated by software (online).

If MBIST detects uncorrectable failures, the HW will prevent further execution. On the other hand, if MBIST detects correctable failures SW must decide whether to continue or halt execution. This is true even if several of the correctable failures combined to create an uncorrectable failure.

Assumption: [SM_097] After startup and before the safety application starts, application software shall confirm all MBISTs finished successfully and no further errors are flagged. [end]

Note

Implementation hint: Software can read the following registers to check the BIST results:

- STCU_MBSL, STCU_MBSM and STCU_MBSH to determine which offline MBISTs failed
- STCU_MBEL, STCU_MBEM and STCU_MBEH to determine which offline MBISTs did not finish

2. This does not include flash memory.

- STCU_MBSLSW, STCU_MBSMSW and STCU_MBSHSW to determine which online MBISTs failed
- STCU_MBELSW, STCU_MBEMSW and STCU_MBEHSW – To determine which online MBISTs did not finish
- STCU_ERR_STAT – To check for internal STCU failure

Not every fault expresses itself immediately. For example, a fault may remain unnoticed if a component is not used or the context is not causing an error or the error is masked.

If faults are not detected over a long time (latent faults), they can pile up once they propagate. ISO 26262 requires 90% latent-fault metric for ASIL D, 80% for ASIL C, and 60% for ASIL B. Typically hardware assisted BIST is therefore used as safety integrity measure to detect latent faults.

The SAC57D5xx is equipped with a Built-in hardware self-test:

- System SRAM (MBIST, executed at boot-time, latent failure measure)
- Flash memory integrity self check (executed at least once per drive cycle, latent failure measure)
- Flash memory margin read (executed at least once per drive cycle, latent failure measure)

Boot-time test (MBIST) are performed after the occurrence of a destructive or external reset, unless they are disabled. All boot-time tests are executed before application software starts executing. If failed, chip will remain in Safe state$_{MCU}$.

All tests may be performed without dedicated external test hardware.

The following safety integrity measure validates the ECC fault signalling and is executed by software to detect single-point faults, although no built-in hardware support is used:

- Flash memory: ECC Fault Report Check: Software can read from the flash memory a set of test patterns (provided by NXP) to test the integrity of faults reported by the ECC logic and captured in the FCCU (shall be performed at startup).

5.2.15.1 MBIST

The SRAM BIST (MBIST) runs during initialization (during boot) and can be run during shutdown, if configured appropriately and triggered by software (see Self Test Control Unit (STCU2)).

NOTE

In principle MBIST can be run at any time, but the MCU will execute a reset after MBIST completes.

5.2.15.2 Flash memory array integrity self check

The flash memory array integrity self check runs in flash memory user test mode and is initiated by software. When the check has completed, software verifies the result (see Flash memory).

5.2.15.3 Flash memory margin read

The flash memory margin reads may be activated to increase the sensitivity of the array integrity self check. It may be enabled in flash memory user test mode and is initiated by software.

5.2.15.4 Flash memory ECC fault report check

The flash memory ECC fault report check is executed in software (see section Flash memory).

5.2.16 Interrupt controller

The CA5 core uses the ARM Generic Interrupts Controller (GIC). The CM4 and CMO+ use the Nested Vectored Interrupt Controller (NVIC). All peripheral interrupts on the device will be directed to a specific interrupt controller (GIC or NVIC) through a centralized Interrupt Router module. In addition to the shared peripheral interrupts, there are CPU-to-CPU interrupts that will pass through the Interrupt Router module. No specific hardware protection is provided to reduce the likelihood of spurious or missing interrupt requests, caused by faults before the IRQ, such as by Electromagnetic Interference (EMI) on the interrupt lines, bit flips in the interrupt registers of the peripherals, or a fault in the peripherals. The interrupt controllers can drop, delay or create spurious interrupts.

5.2.16.1 Runtime checks

Assumption under certain conditions: [SM_100] Applications that are not resilient against spurious or missing interrupt requests may need to include detection or protection measures on the system level. [end]

Rationale: To manage spurious or missing interrupt requests.

Implementation hint: A possible way to detect spurious interrupts is to check corresponding interrupt status in the interrupt status register (polling) of the related peripheral before executing the Interrupt Service Routine (ISR) service code.

5.2.17 Enhanced Direct Memory Access (eDMA)

The eDMA provides the capability to perform data transfers with minimal intervention from the core. It supports programmable source and destination addresses and transfer size.

Failures outside of the eDMA can lead to faulty eDMA behavior, these failures shall be detected by software.

5.2.17.1 Runtime checks

Assumption: [SM_101] The eDMA will be supervised by software which detects spurious, too often, or constant activation. [end]

Rationale: Prevent the eDMA from stealing transfer bandwidth on the System Bus, as well as prevent it from copying data at a wrong point in time

Implementation hint: A possible implementation to protect against spurious or missing interrupts is to have software count the number of eDMA transfers triggered inside a control period and compare this value to the expected value.

Assumption: The eDMA will be supervised by software to detect data corruption during the transfer, and to check that a transfer has completed correctly.

Rationale: A fault occurring in the eDMA may cause an error in the transfer of data. This may result in data being corrupted during a transfer, or an incorrect source or destination address being used, or the DMA failing to complete a transfer correctly.

Implementation hint: A checksum may be calculated on safety critical data prior to transfer at the source, then recalculated at the destination.

Implementation hint: A signature may be written to the destination address prior to transfer, then a check that it has been overwritten may be performed after the transfer has completed.

Implementation hint: Channel to channel linking (also known as chained eDMA transfers) may be used to chain a notification of execution at the end of the transfer.

Assumption under certain conditions: [SM_102] Applications that are not resilient to spurious, or missing functional safety-relevant, eDMA requests can not use the PIT module to trigger functional safety-relevant eDMA transfer requests. [end]

Rationale: To reduce the likelihood of a faulty PIT from triggering an unexpected eDMA transfer

In cases where the eDMA is used to transfer data to non-replicated peripherals such as GPIO or replicated peripherals which are not used redundantly, additional software measures are needed since both halves of the eDMA Channel Mux will not implicitly supervise each other.

Assumption:[SM_104] If safety-relevant software is using the eDMA to transfer data to a non-replicated peripheral or within the RAM, the following holds: "Always on" channels of the eDMA Channel Mux should not be used. Instead, the eDMA should be triggered by software. If "always on" channels are used, their failure has to be detected by software. In this case, software must ensure that the eDMA transfer was triggered as expected at the correct rate and the correct number of times. This test should detect unexpected, spurious interrupts. [end]

5.2.18 System Timer Module (STM)

5.2.18.1 Runtime checks

If a failure in the System Timer Modules (STM) causes a violation of the safety goal, one of the two conditions below shall be satisfied when the STMs are used in the application software.

Assumption: [SM_105] At every STM interrupt, the IRQ handler shall compare the elapsed time since the previous interrupt versus a free running counter to check whether the interrupt time is consistent with the STM setting. [end]

Assumption: [SM_106] The STM IRQ handler shall be under SWT protection. [end]

Implementation Hint: In the first option, the SWT can be used to measure time between the STM interrupts by reading the SWT counter on consecutive interrupts and comparing the difference with the STM measured time. In the second option, the application can set the SWT to a time just greater than the STM measured time and use the STM IRQ to service the SWT.

5.2.19 Periodic Interrupt Timer (PIT)

5.2.19.1 Runtime checks

Assumption: [SM_107] When using PIT module, it should be used in such a way that a possible functional safety-relevant failure is detected by the Software Watchdog Timer (SWT). [end]

Rationale: To catch possible PIT failures

Recommendation under certain conditions: [SM_108] If the PIT is used by the application software in a safety function, a checksum of its configuration registers using the CRC must be calculated and compared with the expected one to verify that the PIT configuration is correct. [end]

The application software shall invoke this test once per FTTI/PST.

Rationale: To check that the PIT remains at its expected configuration

5.2.20 Mode Entry (MC_ME)

Assumption under certain conditions: [SM_082] If the application uses Low Power (LP) mode as part of a safety function, it is required to monitor the duration of LP mode. If the system does not wakeup within a specified period, the system will be reset by the monitoring circuitry. [end]

Implementation hint: The SWT may provide the time monitoring.

Rationale: To overcome faults in the wakeup and interrupt inputs to the MC_ME if the application uses Low Power mode

5.2.21 System Status and Configuration Module (SSCM)

5.2.21.1 Initial checks and configurations

Recommendation: Since the software integrated in the BAF has not been developed in an ISO 26262 or IEC 61508 compliant development process, system level measure must be taken to ensure system integrity or disable use of the BAF.

Rationale: Since BAF code was neither developed nor qualified according to the IEC 61508-4 or ISO 26262-6, any execution of the BAF, or part of it, needs to be inhibited or validated by appropriate measures.

Implementation hint: Execution of BAF code is inhibited by setting the BAF_DIS field in the flash controller's PFCR3 register. Once this field is set, attempted instruction accesses targeting the BAF region are aborted and terminated with a system bus error. This field is set by the BAF code immediately before jumping to application code, thus preventing the CPU from executing the BAF again.

5.2.22 Flash memory

The SAC57D5xx provides programmable non-volatile flash memory (NVM) with ECC which can be used for instruction and/or data storage.

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. Array Integrity self-check calculates a MISR signature over the array content and thus validates the content of the array as well as the decoder logic. The calculated MISR value depends on the array content and must be validated by application software.

Implementation hint: The array integrity self check must be executed on each program flash memory block used.

5.2.22.1 EEPROM

The SAC57D5xx provides eight blocks (2 – 64 kB and 6 – 16 kB) of the flash memory for EEPROM emulation.

Assumption: [SM_114] Software using the flash memory blocks designated for EEPROM emulation will perform checks to detect incorrect data returned from flash memory. [end]

Typically, a CRC will be stored to validate the data.

5.2.22.2 Initial checks and configurations

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 (e.g. sense amplifiers, column mux's, address decoder, voltage/timing references). It calculates a 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.

Assumption:[SM_112] Before executing any safety function, a flash memory array integrity self check should be executed. The calculated MISR value is dependent on the array content and therefore has to be validated by system level application software. [end]

Rationale: To check the integrity of the flash memory array content

Implementation hint: This test may be started by application software: its result may be validated by reading the corresponding registers in the flash memory controller after it has been finished (see "Array integrity self check" section in the "Embedded Flash Memory (c55fmc)" chapter of the *SAC57D5xx Reference Manual*).

Implementation hint: The programming of flash memory may be validated by checking the value of C55FMC_MCR[PEG]. Furthermore, the data written may be read back, then checked by software to verify it matches the intended data. The data read back may be executed in Margin Read Enable mode (C55FMC_UT0[MRE] = '1'). This enables validation of the programmed data using read margins that are more sensitive to weak program or erase status.

5.2.22.3 Runtime checks

The application software checks the status and contents of the programmed sector at the end of a programming operation. The safety mechanism can be based on a read-back scheme, where the written word is read back and compared with the intended value. Alternatively, a CRC check can also be implemented to validate the data.

Assumption: [SM_116] A software test should be implemented to check for potential multi-bit errors introduced by permanent failures in the flash memory control logic.[end]

Assumption: [SM_117] A software safety mechanism shall be implemented to ensure the correctness of any write operation to the flash memory.[end]

Rationale: To check that the written data is coherent with the expected data

This test should be performed after every write operation or after a series of write operations to the flash memory.

Assumption: [SM_119] The Flash memory ECC failure reporting path should be checked to validate that detected ECC faults are correctly reported. [end]

Rationale: The intention of this test is to assure that failure detection is correctly reported.

Implementation hint: The flash memory ECC fault report check is executed in software. The test consists of software reading from the flash memory UTest area (see the "UTEST flash memory map" table in the "Memory map" attachment of the SAC57Dxx Reference Manual). It is a set of test patterns to test the integrity of the ECC logic fault reporting path to the FCCU (executed at start-up, latent failure measure).

5.2.23 Error reporting path tests

It is possible to use fake fault injection to check the correct operation of several reporting paths from supervisors. The "FCCU non-critical fault (NCF) mapping" table in the "Fault Collection and Control Unit (FCCU)" chapter specifically lists those inputs.

Other measures in that column can also be used for a full error reporting path check if so desired.

These fake faults can also be used during development to test whether software programmed to handled such faults works correctly.

Additionally, ECC errors can be injected into FlexCAN SRAM and System SRAM/local RAMs/Caches to check the reporting of such errors to the FCCU.

A multiple cell failure caused, for example, by a neutron or alpha particle or a short circuit between cells may cause three or more bits to be corrupted in an ECC-protected word. As a result, either the availability of the device may be reduced or the ECC logic may perform an additional data corruption labeled as single-bit correction. This is prevented within the design of SAC57D5xx by the use of bit scrambling (column multiplexing). Physically neighboring columns of the RAM array do not contain bits of the same logical word but the same bit of neighboring logical words. Thus the information is logically spread over several words causing only single-bit faults in each word which can be correctly corrected by the ECC. SAC57D5xx has a multiplexor factor of 16 for its system RAM multiplexing adjacent analog bit lines to an analog sense amplifier. It is always enabled and needs no configuration.

5.2.24 Wake-Up Unit (WKPU) / External NMI

Assumption under certain conditions:[SM_226] If external NMI and Wake-up are used as a safety mechanism, especially if waking up within a certain timespan or at all is considered safety-relevant, it is required to implement corresponding system level measures to detect latent faults in the WKPU. [end]

Rationale: To test the analog filter of the WKPU for external NMIs and wakeup events.

Implementation hint: To test the analog filter of the WKPU for external NMIs, application software may configure the NMI during startup to cause only a critical interrupt, then trigger the external NMI and check that the critical interrupt occurred.

5.2.25 Glitch filter

The WKPU supports analog glitch filters. The Wakeup/Interrupt Filter Enable Register (WKPU_WIFER) is used to enable an analog filter on the corresponding interrupt pads to filter out glitches on the inputs.

The SIUL2 supports 1 to 30 external interrupts. Each of these external interrupt lines can have a digital glitch filter applied to them. These filters are used to reduce noise and transient spikes in order to reduce the likelihood of unintended activation of the interrupt inputs. The glitch filters need a running internal oscillator clock to work. If no such clock is available, external interrupts will be effectively disabled when the glitch filter is enabled on a interrupt line.

5.2.26 Register Protection module (REG_PROT)

The ARM architecture allows processors to operate in various modes: User, Fast Interrupt, Interrupt, Supervisor, Abort and System. Of the multiple modes listed, User mode is the single non-privileged mode, while the others collectively form the privileged mode. Only privileged modes have access to all resources, and the execution of all the instructions. It is intended that most parts of the software be executed in User mode so that the SAC57D5xx is protected from errant register changes made by other User mode processes.

In addition, all peripherals, processing modules and other configurable IP is protected by a REG_PROT module, which offers a mechanism to protect individual address locations in a module under protection from being written (for example, to handle the concurrent operation of software tasks with different or lower functional safety integrity level). It includes the following levels of access restriction:

- A register cannot be written once Soft Lock Protection is set. The lock can be cleared by software or by a system reset.
- A register cannot be written once Hard Lock Protection is set. The lock can only be cleared by a system reset.
- If neither Soft Lock nor Hard Lock is set, the Register Protection module may restrict write accesses for a module under protection to supervisor mode only.

Recommendation: Only hardware related software (OS, drivers) should run in privileged mode.

Assumption: [SM_125] For safety-relevant applications, all configuration registers, and registers that aren't modified during application execution, must be protected with a Hard Lock. [end]

5.2.26.1 Runtime checks

Recommendation: All configuration registers, and registers that are not modified during application execution, are to be protected with a Hard Lock.

Rationale: Hard Lock is the last access protection against unwanted writes to some predefined memory mapped address locations.

Implementation hint: Most of the off-platform peripherals have their own Register Protection module. Register Protection address space is inside the memory space reserved for the peripherals (please, refer to the "SAC57D5xx register protection" section of the SAC57D5xx Reference Manual). Each peripheral register that can be protected through the Register Protection module has a Set Soft Lock bit reserved in the Register Protection address space. This bit is 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.

5.2.27 Crossbar Switch (AXBS)

The multi-port AXBS switch allows concurrent transactions from any master (for example, core, eDMA, and so on) to any slave (for example, memories, peripheral bridge, and so on). The AXBS module includes a set of configuration registers for arbitration parameters, including priority, parking and arbitration algorithm. Faults in the configuration registers affect slave arbitration, and thereby potentially software execution times, so software countermeasures must detect these faults.

Assumption: [SM_127] Masters of the AXBS which are not safety related shall have a lower arbitration priority on the AXBS than safety relevant masters. [end]

In cases where it is not possible to set the AXBS arbitration appropriately, a failure probability shall be estimated for such cases.

5.2.27.1 Runtime checks

The application software shall check the AXBS configuration at least once after programming, but it must also detect failures of the AXBS during safety-relevant function execution.

The detection of failures of the AXBS configuration can be achieved as a combination of periodic readback 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 AXBS masters. The need for periodic configuration readback depends on how stringent the control flow monitoring is implemented.

The application software shall detect AXBS configuration failures once per FTTI/PST.

Assumption: [SM_128] Within the FTTI, application software shall detect failures of the AXBS configuration that affects system performance by using the configuration readback and SWT monitoring as described above.[end]

5.2.28 Cores

The ARM Cortex-M processors and the Cortex-A processor may be used to execute non safety-critical tasks as well as safety-critical tasks. When executing safety critical tasks, it may be required to add software-based safety measures in order to achieve the desired level of safety integrity.

5.2.28.1 Runtime checks

5.2.28.1.1 Structural software based self-test (SBST)

During execution of safety relevant code on the ARM Cortex-M4 processor or the Cortex-A5 processor, processing unit self test software must be executed to detect failures in the processing unit. The execution frequency is application-dependent. NXP has developed core self test software; please contact NXP to discuss its availability and manual. Within the FTTI, the processing unit shall periodically run code that tests the functionality of the processing unit (structural test). The result of this SBST is calculated offline during development and compared with the real-time value. Preferably, the processing unit should not be the only method used to compare these two values (selftest). Additionally, independent hardware should check that the result is correct. Independent hardware could be:

- Signature watchdog: The software watchdog is triggered by writing a data pattern to a specific address. The SBST software result could be converted (add an offset) to equal the software watchdog data pattern. The watchdog is used to check the correctness of the SBST execution result.
- Address Decoder: The SBST software result could be converted (by adding an offset) to generate a safety relevant address. The erroneous SBST result could be converted into an unpopulated address space, thus triggering an illegal address trap. Alternatively, a time domain life signal could be generated using address based on the result of the the SBST. When the address is incorrect, the life signal is not generated correctly in the time domain (a trigger is missing) and this indicates to the system to switch into a Safe state$_{system}$.
- Message Objects: The result of the SBST software may be communicated in message objects (for example of a multiplexed communication network like FlexCAN or Ethernet) to validate a message. The communication could be performed explicitly by adding it to the payload or implicitly by including the result in, for example, a payload CRC (for example a safety communication protocol).

Assumption:[SM_316] It is assumed that a software based self-test shall be executed within the FTTI on each core executing safety critical tasks independently. [end]

5.2.28.1.2 Reciprocal comparison

If the ARM Cortex-M0 processor is used to execute safety-critical tasks, it may be required to perform a reciprocal comparison in order to achieve the desired level of safety integrity. This is performed by executing application software on two processing units and exchanging data (including results, intermediate results and test data) reciprocally. A comparison of the data is carried out using software in each unit and detected differences lead to a failure message.

5.2.29 Analog to Digital Converter (ADC)

Parts of the Successive Approximation Register (SAR) Analog-to-Digital Converters (ADCs) of the SAC57D5xx do not provide the functional safety integrity to achieve high functional safety integrity targets. Therefore, system level safety measures are required.

Assumption under certain conditions:[SM_130] When the Analog-to-Digital Converters (ADC) of the SAC57D5xx are used in a safety function, suitable system level functional safety integrity measures must be implemented after reset (external reset or destructive reset) before starting the respective safety function to ensure ADC integrity. [end]

Rationale: To check the integrity of the ADC modules against latent failures.

Implementation hint: After reset (external or destructive reset), but before executing any safety function, perform the calibration of the ADC using application software to detect latent faults.

5.3 I/O functions

Assumption: [SM_232] The integrity of functional safety-relevant periphery will mainly be ensured by application level measures (for example, connecting one sensor to different I/O). [end]

Functional safety-relevant peripherals are assumed to be used redundantly in some way. Different approaches can be used, for example, by implementing replicated input (for example, connect one sensor to two SPIs or even connect two sensors measuring the same quantity to two ADCs) or by crosschecking some I/O operations with different operations (for example, using sensor values of different quantities to check for validity). Also, intelligent self-checking sensors are possible if the data transmitted from the sensors contains redundant information in the form of a checksum, for example. Preferably, the replicated modules generate or receive the replicated data using different coding styles (for example, inverted in the voltage domain or using voltage and time domain coding for redundant channels). Safety system developers may choose the approach that best fits their needs.

Assumption: [SM_133] Comparison of redundant operation of I/O modules is the responsibility of the application software, as no hardware mechanism is provided for this. [end]

Implementation hint: Possible measures could use different coding schemes within each redundant I/O channel (for example, inverted signals, different time periods).

Implementation hint: Possible measures could be using different peripherals (for example, FTM0 and FTM2) to implement multiple independent and different channels.

5.3.1 Digital inputs

Assumption under certain conditions:[SM_137] When safety functions use digital input, system level functional safety mechanisms have to be implemented to achieve required functional safety integrity.[end]

5.3.1.1 Hardware

Implementation hint: Functional safety digital inputs may need to be acquired redundantly. To reduce the risk of CMFs, the redundant channels may not use GPIO adjacent to each other (see Causes of dependent failures).

- Double read operation of a digital input is implemented by two general purpose inputs (GPI) of the SIUL2 unit. A comparison (by software) between the double reads (for example, reads from both GPIOs) detects an error (please refer to Figure 5-4).
- A double read PWM input is implemented by using two modules as two channels. The functional safety integrity is achieved by double reads and a software comparison. One channel is provided by one instance of the FlexTimer while another channel is provided by a different instance. For example, one channel is provided by FTM0 and the other is provided by FTM1, FTM2, or FTM3. Read PWM input means any input read related to signal transitions (rise or fall). This may also include the time that the signal was high, low or both (see Figure 5-4).

For each signal of a double read, the SIUL2 can provide additional channels to support interrupt-based reading for each signal (see Figure 5-5).

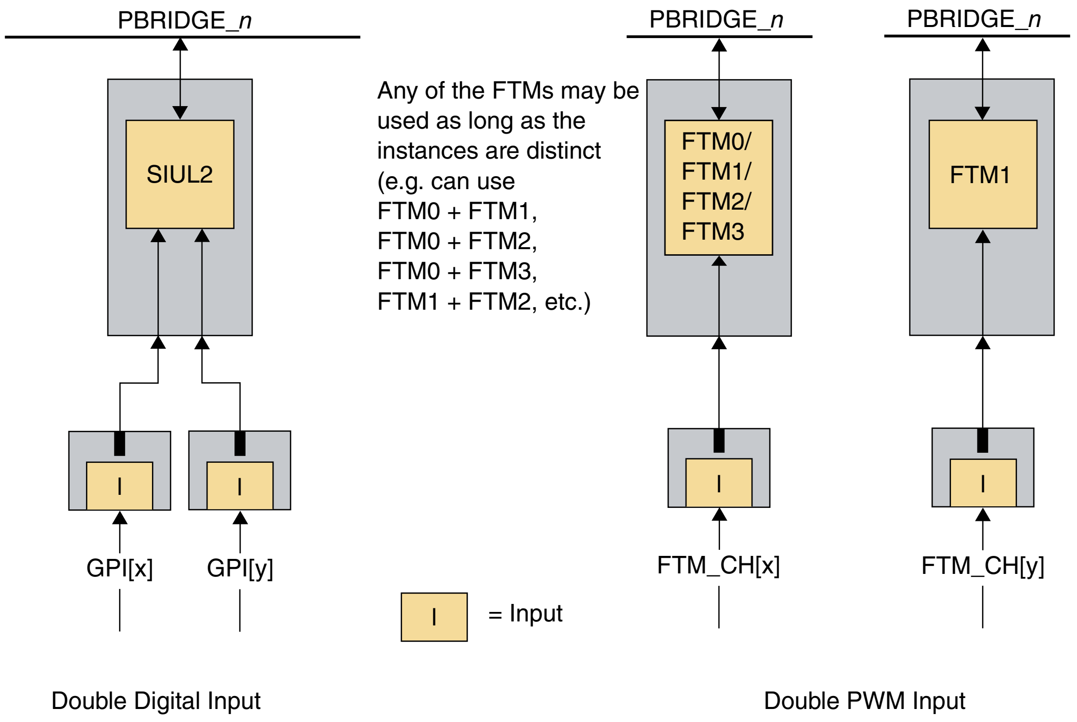

Figure 5-4. Double Digital input and Double PWM input

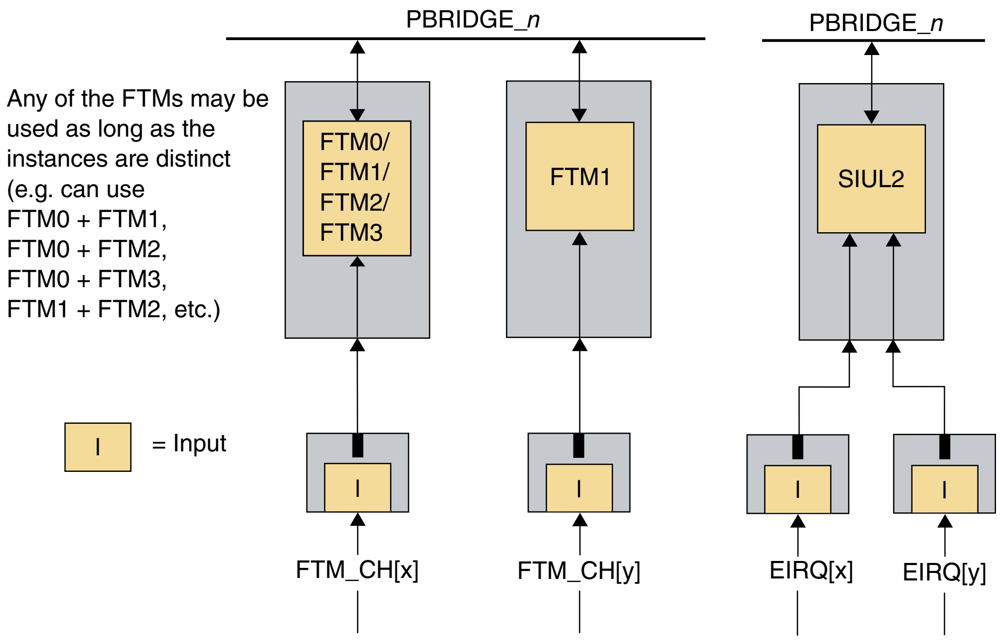
PBRIDGE_*n*
Any of the FTMs may be used as long as the instances are distinct (e.g. can use FTM0 + FTM1, FTM0 + FTM2, FTM0 + FTM3, FTM1 + FTM2, etc.)
I = Input
FTM0/ FTM1/ FTM2/ FTM3
FTM1
I
I
FTM_CH[x]
FTM_CH[y]
PBRIDGE_*n*
SIUL2
I
I
EIRQ[x]
EIRQ[y]

Figure 5-5. Double Read Encoder Input (IRQ triggered)

Implementation hint: If sufficient diagnostic coverage can be obtained by a plausibility check on a single acquisition for a specific application, that check can replace a redundant acquisition.

5.3.2 Digital outputs

Functional safety digital outputs are always assumed to be written either redundantly or with read back. In case of single output with read back, the feedback loop should be as large as possible to cover faults on system level also. The figure below depicts the connection of two (functional safety critical) actuators connected to the SAC57D5xx. Actuator 1 is connected to an output peripheral, for example, a motor is connected to a PWM output (output peripheral 3). The signal generated by the output peripheral 3 can be input to an input peripheral, for example, a FlexTimer. This measure is to confirm, that the generated output signal is correct. This read back may be internally of the SAC57D5xx (internal read back) or externally (external read back). The external read back covers more types of failures (for example, corrupt wire bonds or solder joints) than the internal read back, but still does not guarantee, that the actuator really behaves as desired. This is achieved by including the actuator and sensor into the read back loop. An alternative solution is to redundantly output a signal. For example, actuator 2 consists of two relays in series to switch off a functional safety-relevant supply voltage. The selection of the suited output connection is part of the I/O functional safety concept on system level.

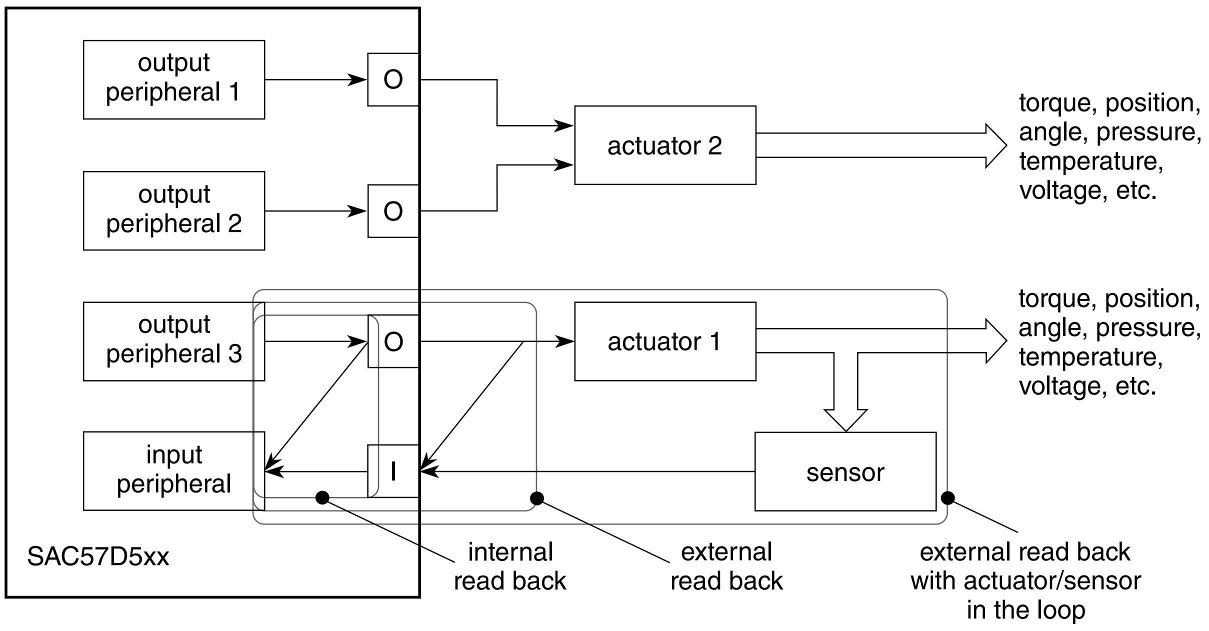
output peripheral 1
O
output peripheral 2
O
actuator 2
torque, position, angle, pressure, temperature, voltage, etc.
output peripheral 3
O
actuator 1
torque, position, angle, pressure, temperature, voltage, etc.
input peripheral
I
sensor
SAC57D5xx
internal read back
external read back
external read back with actuator/sensor in the loop

Figure 5-6. Digital Outputs with redundancy and read back

Implementation hint: Each of the PWM outputs of the Stepper Motor Controller (SMC) are equipped with an internal short-circuit detection feature. This feature consists of a timer to measure the time during which the PWM signal from the motor controller and the signal at the pad (after sampling and synchronization) are not equal. If this time is greater than or equal to the time represented by MCSDTO[TOUT] then a short circuit is assumed. For further details see the Stepper Motor Controller chapter of the *SAC57D5xxReference Manual*.

Implementation hint: If a sufficient diagnostic coverage can be reached by a plausibility check on a single output channel for a specific application, that check can replace a redundant write or read-back. This hint is a special case of deviating from **Assumptions** as described in the preface.

5.3.2.1 Hardware

5.3.2.1.1 Single Write Digital Output

- Single Write Digital Output with external read-back (Figure 5-7, left):

  A comparison between the desired output values and the value read back via external read-back configuration is done. After writing the output value, the status of the digital input is evaluated.

- Single Write Digital Output with internal read-back (Figure 5-7, right):

  A comparison between the desired output values and the value read back via internal read-back configuration. After writing the output value, the internal read-back status is evaluated.

- Single Write PWM Output with external read-back (Figure 5-8, left):

  This procedure output compare the PWM read-back provided by a single channel of the FTM0 (or FTM1, FTM2, FTM3) with the expected values that have been written to the external pad of a different FTM instance’s output channel.

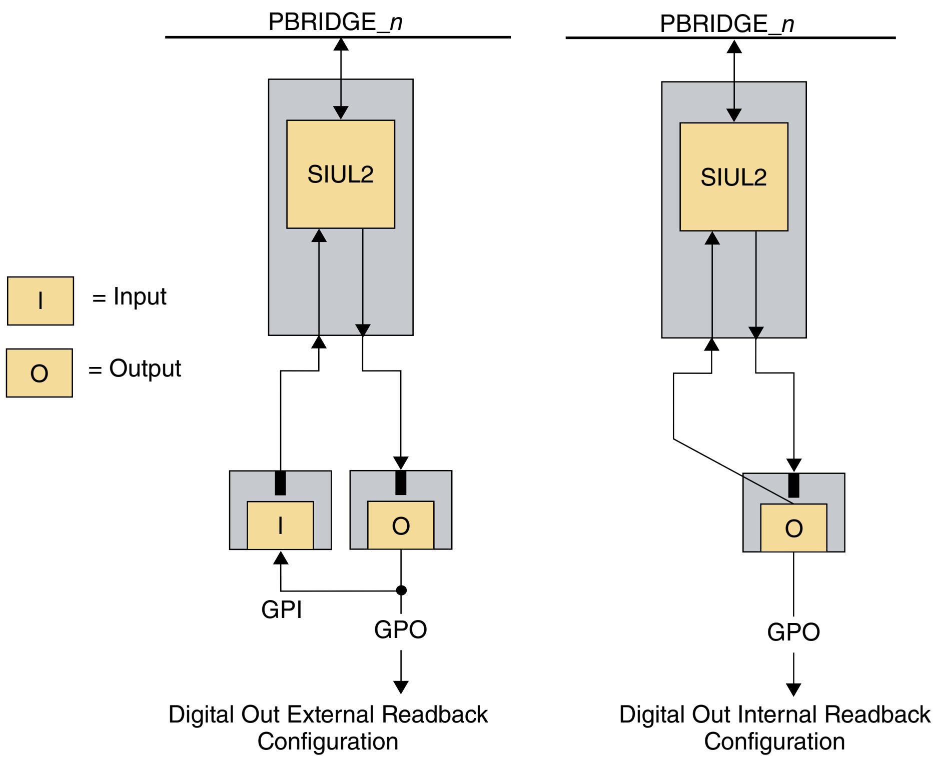
PBRIDGE_n
SIUL2
I = Input
O = Output
I
O
GPI
GPO
Digital Out External Readback Configuration
PBRIDGE_n
SIUL2
O
GPO
Digital Out Internal Readback Configuration

Figure 5-7. Single Write Digital Output With Read-Back

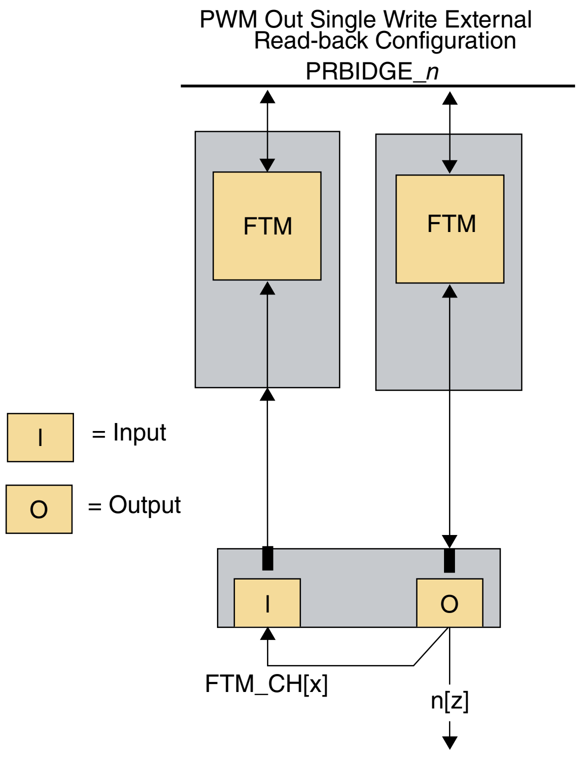

Figure 5-8. Single Write PWM Output With Read-Back

5.3.2.1.2 Double Write Digital Output

- Double Write Digital Output

  The SIUL2 hardware element is used to perform a double-write digital output.

- Double Write PWM Output
  - The hardware elements are used to perform a double write PWM output:
    - FTM0 and FTM1
    - FTM2 and FTM3

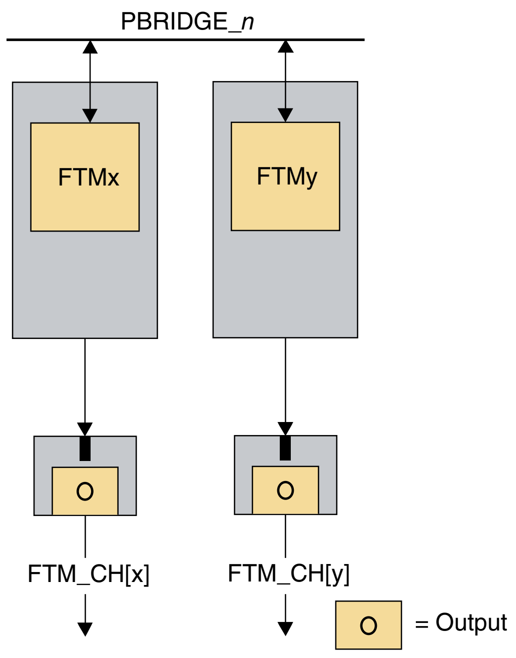

Figure 5-9. Double Write PWM Output

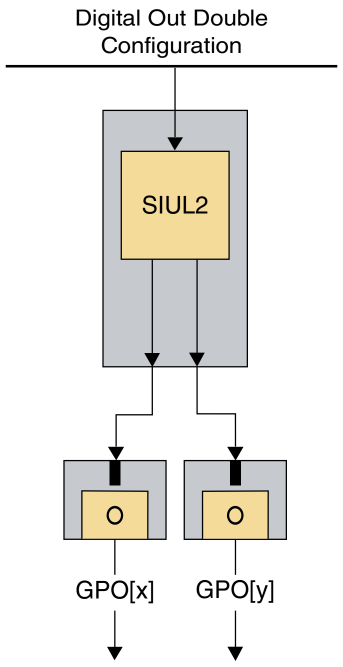

Figure 5-10. Double Write Digital Output

5.3.3 Analog inputs

5.3.3.1 Hardware

Two options for reading analog inputs exist:

- Single Read Analog Inputs
- Double Read Analog Inputs

Additional tests may be implemented in software as described in section Single Read Analog Inputs and Double Read Analog Inputs.

Oversampling can be used to detect transient faults affecting the ADC channel during normal operation.

5.3.3.1.1 Single Read Analog Inputs

The single-read analog input uses a single-analog-input channel either of ADC_0, ADC_1, ADC_2, or ADC_3 to acquire an analog voltage signal (see the figure below).

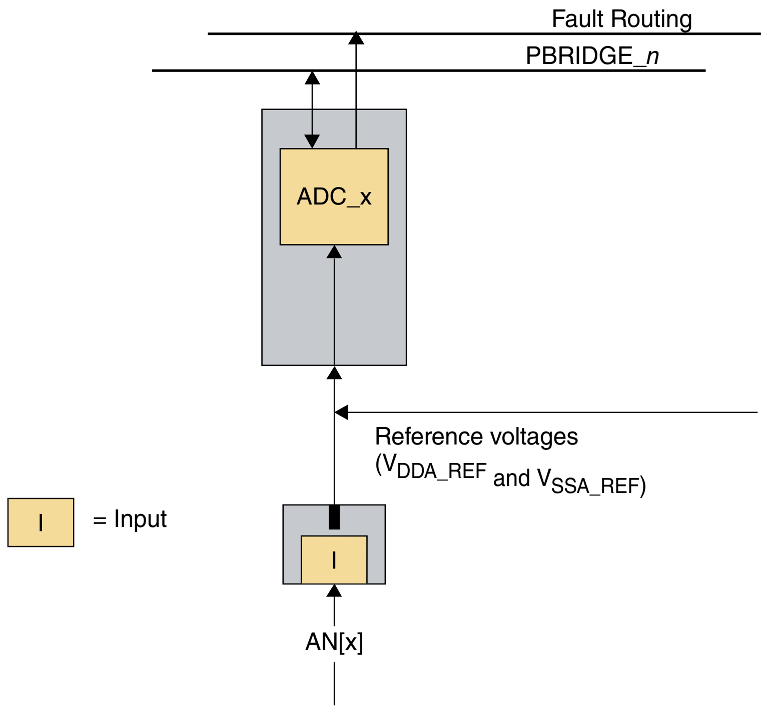

Figure 5-11. Single Read Analog Input configuration

5.3.3.1.2 Double Read Analog Inputs

The Double Read Analog Input uses two analog input channels to acquire a replicated analog input signal. Two ADC units acquire and digitize the two copies of a redundant analog signal connected to the inputs. In this configuration only a portion of the analog inputs are available. The channels that are used for the comparison need to reside on different PBRIDGEs (ADC0 and ADC2 on PBRIDGE_B, ADC1 and ADC3 on PBRIDGE_A). The comparison of the results is performed by the system level application software (see the figure below).

The following is the list of ADC channels that may be used with the Double Read Analog Inputs function:

- ADC0/1 AN[11:14]
- ADC1/3 AN[4:8]
- ADC2/3 AN[0:2]

Rationale: ADC_0 and ADC_1 share input channels (AN[11:14]), ADC_1 and ADC_3 (AN[4:8]), ADC_2 and ADC_3 (AN[0:2]). Using double reads on these shared channels is a possible source of CCFs.

Implementation hint: One shared ADC-channel (AN[11:14]) may not be used for both inputs of the Double Read Analog Input function.

The possible combinations of double read ADC inputs are as follows:

- ADC0_AN[0:8] – ADC1_AN[0:8,11:14], ADC3_AN[0:7]
- ADC1_AN[0:8] – ADC0_AN[0:8,11:14]
- ADC2_AN[0:2] – ADC1_AN[0:8,11:14], ADC3_AN[3:7]
- ADC3_AN[3:7] – ADC0_AN[0:8,11:14], ADC2_AN[0:2]
- ADC3_AN[0] – ADC0_AN[0:8,11:14], ADC2_AN[1:2]
- ADC3_AN[1] – ADC0_AN[0:8,11:14], ADC2_AN[0,2]
- ADC3_AN[2] – ADC0_AN[0:8,11:14], ADC2_AN[0:1]

The functional safety integrity is achieved by replicated acquisition with separated analog input channels and software comparison by the processing function (see the figure below).

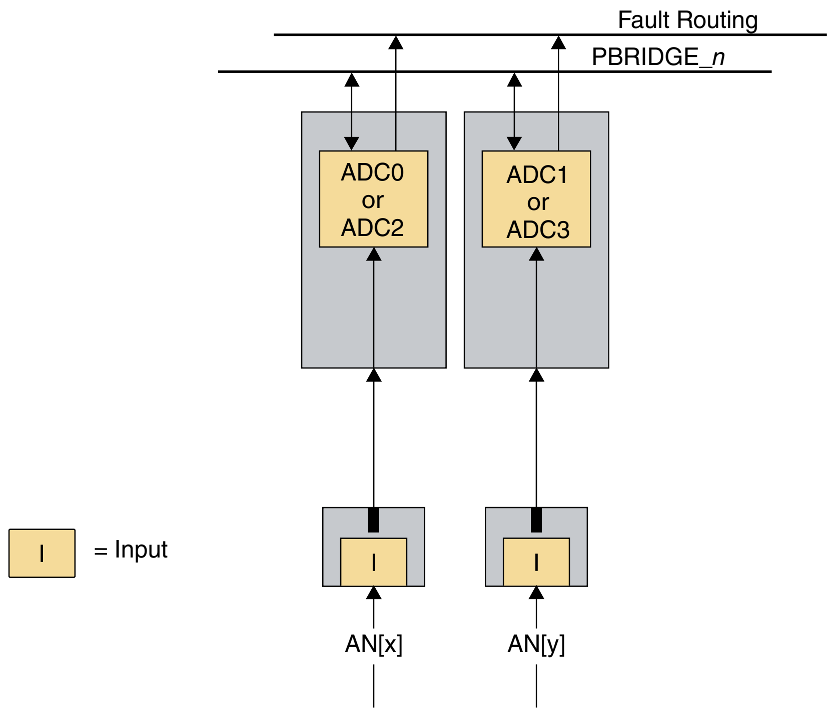

Figure 5-12. Double Read Analog Inputs configuration

5.3.4 Other requirements

Implementation hint:

- When an application needs to access the ADC result FIFO, a 32-bit read access enables the verification of the correct channel number on which the conversion was executed.
- All the FIFO empty interrupt flags should checked when the motor control period interrupt occurs
- In the eTimer module, the capture flag should be used to detect missing eTimer acquisition.
- If the ADC analog watchdog function is used for functional safety-relevant signal, two analog watchdog channels should monitor the same signal.
- If Sine Wave Generator (SWG) is used, the ADC (eventually in conjunction with CTU) should be used to check the output signal.
- If an external temperature sensor is used to validate the accuracy of the internal temperature sensors, the external temperature sensor may not be converted by the same ADC that was used to convert the internal temperature value.

5.4 Communications

An appropriate safety software protocol should be utilized (for example, Fault-Tolerant Communication Layer, FTCOM) for any communication peripheral used in a safety-relevant application.

Recommendation: [SM_051] It is assumed that communication over the following interfaces be protected by a fault-tolerant communication protocol: [end]
- FlexCAN

FlexCAN does not have safety mechanisms other than what is included in its protocol specifications. The application software, or operating system, needs to provide the safety measures for these modules to meet safety requirements.

5.4.1 Redundant communication

Parts of the integrated SPI and LINFlexD communication controllers do not independently provide the functional safety integrity that IEC 61508 and ISO 26262 requires for high functional safety integrity targets. As these communication protocols often deal with low complex slave communication nodes, higher level functional safety protocols as described in Fault-tolerant communication protocol may not be feasible.

Therefore, appropriate communication channel redundancy may be required. Multiple instances of communication controllers may be used to build up a single fault robust communication link.

Implementation hint: If communications over the following interfaces is part of the safety function, redundant instances of the hardware communication controller should be used, preferable using different data coding (for example, inversion):

- Synchronous Serial Communication Controller (SPI)
- LINFlexD Communication Controller

5.4.2 Fault-tolerant communication protocol

Portions of the integrated LINFlexD and FlexCAN communication channels do not independantly provide the functional safety integrity required by IEC 61508 and ISO 26262 for high functional safety-relevant applications.

Implementation hint: If communication over the following interfaces is part of the functional safety function, a software interface with the hardware communication channel, in accordance with the IEC 61784-3 or IEC 62280 series, is required for the following:

- FlexCAN Communication Controller
- Universal Asynchronous Communication Controller (LINFlexD)

FlexCAN and LINFlexD do not have specific functional safety mechanisms other than ECC protection of SRAM arrays and what is included in their protocol specifications. The application software, middleware software, or operating system needs to provide the functional safety mechanisms on the interface of the IP modules to meet functional safety requirements.

Typically mechanisms are:

- end-to-end CRC to detect data corruption
- sequence numbering to detect message repetitions, deletions, insertions, and resequencing
- an acknowledgement mechanism or time domain multiplexing to detect message delay or loss
- sender identification to detect masquerade

As the 'black channel' typically includes the physical layer (for example, communication line driver, wire, connector), the functional safety software protocol layer is an end-to-end functional safety mechanism from message origin to message destination.

An appropriate functional safety software protocol layer (for example, Fault Tolerant Communication Layer, FTCOM, CANopen Safety Protocol) may be necessary to ensure the failure performance of the communication process. Software protocol layer implements a software interface with the hardware communication channel in accordance with the IEC 61784-3 or IEC 62280 series (so-called 'black channel').

An alternative approach to improve the functional safety integrity of FlexCAN may be to use multiple instances of the FlexCAN channels and use an appropriate protocol to redundantly communicate data (for example, using the CANopen Safety protocol). This approach communicates redundant data (for example, one message payload inverted, the other message payload not inverted) using a different communication controller.

Due to the limited bandwidth and the point to point communication architecture for LINFlexD, only a simplified functional safety protocol layer may be required.

5.5 Additional configuration information

5.5.1 SAC57D5xx configuration

Assumption: [SM_140] It is required that application software verifies that the initialization of the SAC57D5xx is correct before activating the safety-relevant functionality. [end]

Assumption: [SM_141] It is required that application software checks the configuration of the SSCM once after boot. [end]

Recommendation: It is recommended that SSCM is configured to trigger an exception in case of any access to a peripheral slot not used on the device.

Rationale: To detect erroneous addressing and fault in address and bus logic.

Recommendation: It is recommended that after the boot, application software perform an intended access to an unimplemented memory space and check for the expected abort to occur.

Rationale: To detect erroneous addressing and fault in address and bus logic.

Recommendation: It is recommended that unused interrupt vectors point, or jump, to an address that is illegal to execute, contains an illegal instruction, or in some other way causes detection of their execution.

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

Rationale: To reduce the risk accidental writes to configuration registers affecting the execution of the SAC57D5xx's safety function or disable the safety mechanism due to their change.

Recommendation: All configurations registers, and registers that are not modified during application execution, should be protected with Hard Lock Protection (if that option is available for the register) or using Peripheral Access Control. Configuration registers, and registers which have limited writes every trip time, should be protected with soft-lock protection.

Rationale: To reduce the risk accidental writes configuration registers affecting the execution of the SAC57D5xx's safety function or disable the safety mechanism due to their change.

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

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

Chapter 6
Failure rates and FMEDA

6.1 Failure rates

In order to analyze and quantify the effectiveness of the SAC57D5xx integrated safety architecture to handle random hardware failures, the inductive analysis method of FMEDA (Failure Modes Effects and Diagnostic Analysis) was performed during the development of the SAC57D5xx. The following methods for deriving the base failure rates of the SAC57D5xx were used as input to the FMEDA:

- Permanent faults (Die & Package): IEC TR 62380 - Reliability data handbook – Universal model for reliability prediction of electronics components, PCBs and equipment
- Transient faults (Die): JEDEC Standard JESD89 - Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices

6.2 FMEDA

In order to support the integration of the SAC57D5xx into safety-related systems and to enable the safety system developer to perform the system level safety analysis, the following documentation is available:

- FMEDA - Inductive analysis of the SAC57D5xx enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and β-factor β_{IC})
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The FMEDA and FMEDA report are available upon request.

6.2.1 Module classification

For calculating the safety metrics for ISO 26262 (Single-Point Failure Metric (SPFM), Latent Failure Metric (LFM) and Probabilistic Metric for random Hardware Failures (PMHF)) and for IEC 61508 (Safe Failure Fraction (SFF) and β_{IC} factor) the modules of the SAC57D5xx are classified as follows:

- **MCU Safety Functions**: All modules which can directly influence the correct operation of the **MCU Safety Functions**.
- **Safety Mechanism**: All modules which detect faults or control failures to achieve or maintain a safe state. These modules cannot independently directly influence the correct operation of one of the safety functions in the case of a single fault.
- **Peripheral**: All modules which are involved in I/O operation. Peripheral modules are usable by qualifying data with system level safety measures or by using modules redundantly. Qualification should have a low risk of dependent failure. In general, **Peripheral** module safety measures are implemented in system level software.
- **Debug Functions**: All modules which are not safety related, i.e. none of their failures can influence the correct operation of one of the safety functions.

The complete module classification for the SAC57D5xx can be found in the attached "SAC57D5xx Module Classification" spreadsheet.

Chapter 7
Dependent failures

7.1 Provisions against dependent failures

7.1.1 Causes of dependent failures

ISO 26262-9 lists the following dependent failures, which are applicable to the SAC57D5xx on chip level:

- Random hardware failures, for example:
    - dependent failures that are able to influence an on-chip function and its respective safety mechanisms
- Environmental conditions, for example:
    - temperature
    - EMI
- Failures of common signals (external resources), for example:
    - clock
    - power-supply
    - non-application control signals (for example, testing, debugging)
    - signals from modules that are not replicated

Additionally, the following topics are mentioned, which are out of scope of this document and not treated here:

- Development faults:
    - development faults are systematic faults which are addressed by design-process
- Manufacturing faults:
    - manufacturing faults are usually systematic faults addressed by design-process and production test
- Installation and repair faults:
    - installation and repair faults need to be considered at system level
- Stress due to specific situations:

- Specific situations may be considered at system level. Additionally, the result of stress (for example, wear and aging due to electro-migration) usually lead to single-point faults and are not considered dependent failures.

7.1.2 Measures against dependent failures

7.1.2.1 Environmental conditions

7.1.2.1.1 Temperature

The SAC57D5xx was designed to work within a maximum operational temperature profile (see the *SAC57D5xx Data Sheet*).

7.1.2.1.2 EMI and I/O

To cope with noise on digital inputs, the I/O circuitry provides input hysteresis on all digital inputs. Moreover, the RESET and NMI inputs contain glitch filtering capabilities, which are described in sections Hardware requirements on system level and "Glitch filter".

To reduce interference due to digital outputs, the I/O circuitry provides signal slope control. An internal weak pull up or pull down structure is also provided to define the input state.

7.1.2.2 Failures of common signals

7.1.2.2.1 Clock

To cover dependent failures caused by clock issues, modules for supervision are implemented which are described in Clock Monitor Unit (CMU). Major failures in the clock system are also detected by the SWT (Software Watchdog Timer).

7.1.2.2.2 Power supply

To cover dependent failures caused by issues with the power supplies, supervision modules are implemented (see Power Management Controller Digital Interface (PMCDIG)). Some dependent failures (for example, loss of power supply) are detected since software will no longer be able to trigger the external watchdog (see External Watchdog (EXWD)).

7.1.2.2.3 Nonapplication control signals

Modules and signals (for example, for scan, test and debug), which are not safety-related should never be able to lead to a safety-related failure. This can be ensured by either not interfering with the safety-related parts of the SAC57D5xx or by detecting such interference. For example, there must be assurance that the system is not debugged (or unintentionally placed in debug mode), or placed in any other special mode different from normal application execution mode (for example, test mode). In addition, an FCCU failure indication is generated if:

- A self-test sequence of the STCU is unintentionally executed during normal operation of the device.

7.1.3 Dependent failure avoidance on system level

It is recommended to not use adjacent input and output signals of peripherals, which are used redundantly, in order to reduce dependent failures. As internal pad position and external pin/ball position do not necessarily correspond to each other, the safety system developer may take the following recommendations into consideration:

- Usage of non-contiguous balls of the package
- Usage of non-contiguous pads of the silicon
- Non-contiguous routing of these signals on the PCB

Assumption under certain conditions: [SM_142] If the system requires robustness regarding dependent failures, configurations that place redundant signals on neighboring pads or pins should be avoided. [end]

Implementation hint: Pad position as well as pin/ball position should be taken into consideration.

The pin/ball assignment for individual peripherals can be extracted from the *SAC57D5xx Microcontroller Data Sheet*. The following section explains how this can be achieved.

7.1.3.1 I/O pin/ball configuration

Whether two functions on two signals are adjacent to each other can be determined by looking at the mechanical drawings of the packages (see the *SAC57D5xx Data Sheet*) together with the ball number information of the packages as seen in the *SAC57D5xx Reference Manuals* "System Integration Unit Lite2 (SIUL2)" section and the "Pin muxing" table.

The layout of the device balls and the order of die pad signals need to both be taken into consideration. Adjacency of the package balls is straight forward since it can be seen in the package layout. It is more difficult to determine adjacency on the die. The Signal Description chapter in the SAC57D5xx Reference Manual can be used in assisting to determine adjacency of signals on the die. To help avoid potential issues, redundant signals cannot be on adjacent balls or on adjacent die pads. Avoiding adjacency limits crosstalk, signal drive strength, and other associated issues.

7.1.3.2 External timeout function

A dependent failure may lead to a state where the SAC57D5xx is not able to signal an internal failure via its EOUT[*n*] signals (error out). With the use of a system level timeout function (for example, watchdog timer), the likelihood that dependent failures affect the functional safety of the system can be reduced significantly.

In general, the external watchdog covers dependent failures which are related to:

- General destruction of internal components (for example, due to non-mitigated overvoltage or a latch-up at redundant input pads). Since these errors do not result in subtle output variations of the SAC57D5xx but typically in a complete failure, a simple watchdog is sufficient.

Additionally, the external watchdog is able to detect failures related to:

- Missing/wrong power
- Missing/wrong clocks
- Errors in mode change (for example, unintentionally entering test or debug mode)

NOTE

All of these are expected to be detected by internal safety mechanisms (CMUs, LVDs/HVDs, signals to the FCCU), so the external watchdog serves as a fallback for unexpected failure effects and dependent failures with wider than expected effects (for example, disabling an on-chip function and its respective safety mechanisms at the same time).

The external watchdog function is in permanent communication with the CPU of SAC57D5xx. As soon as there are no correct communications, the external watchdog function switches the system to Safe state$_{\text{system}}$. Thus, either the SAC57D5xx or external watchdog function can transition the system to Safe state$_{\text{system}}$. The external watchdog function is required to be sufficiently independent of the SAC57D5xx (for example, regarding clock generation, power supply, and so on).

The external watchdog function does not necessarily need to be a dedicated IC, the requirements may also be fulfilled by another MCU (already used in the system) which is capable of detecting a lack of communication (such as via CAN) and moving the system to Safe state$_{\text{system}}$.

7.1.4 β_{IC} considerations

During the development of the SAC57D5xx, the susceptability of the MCU to dependent failures is evaluated by ensuring sufficient independence between on-chip functions and their respective safety mechanisms.

One method to do this for an MCU is to determ the β-factor β_{IC} as defined in annex E of IEC 61508-2. The β_{IC} is calculated based on a checklist of questions with associated scoring. The smaller the β_{IC}, the less susceptible the on-chip function and their respective safety mechanisms are to dependent failures. The final β_{IC} estimate should not exceed 25%. The β_{IC} is calculated multiple times, for each pairing of on-chip function and their respective safety mechamisms.

The FMEDA includes the β_{IC} calculations and is available upon request.

Chapter 8
Additional information

8.1 Testing All-X in RAM

All-0 or All-1 content will be an uncorrectable error only at some addresses in RAMs where address is included in the ECC calculation. This section contains a program which provides these adresses and can thus be used to either determine an address to periorically read or check whether addresses which are periodically read by an application show this desired behaviour.

8.1.1 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 ---:
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 perl script -----
```

This script finds the first N addresses with 2 or 6 bits set and 2 or 6 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 = 40000000h

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

- `addr = 40000010h, addr_ecc = 06h`

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

1. `addr = 40000008h, addr_ecc = DBh`
2. Find the first 5 addresses of each type for system RAM:
    - `./find_allx_addr 40000000 5`

RAM base address = 40000000h

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

1. `addr = 40000010h, addr_ecc = 06h`
2. `addr = 40000038h, addr_ecc = 14h`
3. `addr = 40000058h, addr_ecc = C0h`
4. `addr = 40000080h, addr_ecc = 28h`
5. `addr = 400000f8h, addr_ecc = 21h`

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

1. `addr = 40000008h, addr_ecc = DBh`
2. `addr = 40000098h, addr_ecc = F5h`
3. `addr = 400000b0h, addr_ecc = E7h`
4. `addr = 400000c8h, addr_ecc = EEh`
5. `addr = 400000e0h, addr_ecc = FCh`

8.1.2 ECC checkbit/syndrome coding scheme

The ECC scheme implements a single-error correction, double-error detection (SECDED) code using the so-called Hsiao odd-weight column criteria. These codes are named for M.Y. Hsiao, an IBM researcher who published extensively in the early 1970s on SECDED codes better suited for implementation in protecting (mainframe) computer memories than traditional Hamming codes.

The Hsiao codes are Hamming distance 4 implementations which provide the SECDED capabilities. The minimum odd-weight constraints defined by Hsiao are relatively simple in the resulting implementation of the parity check H matrix which defines the association between the data (and address) bits and the checkbits. They are:

1. There are no all zeroes columns.
2. Every column is distinct.
3. Every column contains an odd number of ones, and hence is "odd weight".

In defining the H-matrix for this family of devices, these requirements from Hsiao were applied. Additionally, there are a variety of ECC code-word requirements associated with specific functional requirements associated with the flash memory that further dictated the specific column definitions. In any case, the resulting ECC is organized based on 64 data bits plus 29 address bits (the upper bits of the 32-bit address field minus the 3 bits which select the byte within 64-bit (8-byte) data field.

The basic H-matrix for this (101, 93) code (93 is the total number of "data" bits, 101 is the total number of data bits (93) plus 8 checkbits) is shown in the table below. A '*' in the table below indicates the corresponding data or address bit is XOR'd to form the final checkbit value on the left. For 64-bit data writes, the table sections corresponding to D[63:32], D[31:0], and A[31:3] are logically summed (output of each table section is XOR'ed) together to the final value driven on the hwchkbit[7:0] outputs. Note that this table uses *the AHB bit numbering convention where bit[0] is the least significant bit.*

Table 8-1. ECC basic H-matrix definition

Checkbits [7:0]	Data Bit																															
	Byte 7								Byte 6								Byte 5								Byte 4							
	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
7	*				*					*			*		*	*		*	*	*	*					*	*			*	*	
6			*	*			*	*					*		*		*				*				*			*	*			*
5	*	*	*	*					*				*	*		*		*	*				*	*	*				*			
4	*		*			*		*					*		*		*			*			*	*			*				*	
3									*	*	*	*	*		*				*		*				*	*	*	*				
2					*		*					*		*			*	*				*	*							*		*
1		*		*		*		*	*	*	*	*						*		*		*		*						*		*
0		*			*	*	*				*			*	*	*		*				*				*		*	*		*	
	Byte 3								Byte 2								Byte 1								Byte 0							
Checkbits [7:0]	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7					*		*				*		*				*	*			*	*			*				*	*		*
6	*	*		*	*				*				*				*	*		*	*					*	*				*	*
5			*	*			*	*	*			*	*	*					*		*	*		*								
4	*		*	*				*		*	*				*	*	*			*			*	*				*		*		
3	*	*	*			*						*	*		*	*			*				*		*	*	*			*		
2		*		*	*	*	*	*	*	*	*			*					*				*		*		*	*		*		
1		*						*		*			*	*	*			*				*				*			*	*	*	
0		*		*		*		*				*				*				*				*				*	*		*	*
Checkbits [7:0]	Address Bit[1]																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3			
7		*		*		*		*	*	*		*	*	*		*	*	*		*		*		*	*	*		*				
6		*		*	*	*	*			*	*			*	*	*	*		*		*	*		*	*	*		*				
5		*	*	*			*	*			*	*		*		*		*	*			*	*	*		*	*	*	*			
4	*	*	*		*	*		*		*	*	*	*		*					*	*		*	*	*		*	*				
3	*		*		*	*	*		*		*	*	*			*		*		*	*		*		*	*	*		*			
2	*	*			*		*	*	*	*				*	*	*	*	*	*	*		*	*		*		*		*			
1	*		*	*		*	*		*	*		*	*		*		*	*	*		*	*	*					*	*			
0	*		*	*	*			*	*		*		*	*	*		*		*	*	*			*		*	*		*			

1. Bit numbering is AHB convention, bit 0 is LSB. D[7:0] corresponds to byte at address 0. D[63:56] corresponds to byte at address 7.

Figure 8-1 shows an alternative representation of the ECC encode process, written as a C language function.

Figure 8-1. C Language encode ECC function description

```
encodeEcc (addr, data_a2_is_zero, data_a2_is_one)
    unsigned int        addr;                   /* 32-bit byte address */
    unsigned int        data_a2_is_zero;        /* 32-bit data lower, a[2]=0 */
    unsigned int        data_a2_is_one;         /* 32-bit data upper, a[2]=1 */

{
    unsigned int        addr_ecc;               /* 8 bits of ecc for address */
    unsigned int        ecc;                    /* 8 bits of ecc codeword */

/* the following equation calculates the 8-bit wide ecc codeword by examining each addr or
data bits and xor'ing the appropriate H-matrix value if the bit = 1 */

    addr_ecc
        = (((addr            >> 31) & 1) ? 0x1f : 0x0)      /* addr[31] */
        ^ (((addr            >> 30) & 1) ? 0xf4 : 0x0)      /* addr[30] */
        ^ (((addr            >> 29) & 1) ? 0x3b : 0x0)      /* addr[29] */
        ^ (((addr            >> 28) & 1) ? 0xe3 : 0x0)      /* addr[28] */
        ^ (((addr            >> 27) & 1) ? 0x5d : 0x0)      /* addr[27] */
        ^ (((addr            >> 26) & 1) ? 0xda : 0x0)      /* addr[26] */
        ^ (((addr            >> 25) & 1) ? 0x6e : 0x0)      /* addr[25] */
        ^ (((addr            >> 24) & 1) ? 0xb5 : 0x0)      /* addr[24] */

        ^ (((addr            >> 23) & 1) ? 0x8f : 0x0)      /* addr[23] */
        ^ (((addr            >> 22) & 1) ? 0xd6 : 0x0)      /* addr[22] */
        ^ (((addr            >> 21) & 1) ? 0x79 : 0x0)      /* addr[21] */
        ^ (((addr            >> 20) & 1) ? 0xba : 0x0)      /* addr[20] */
        ^ (((addr            >> 19) & 1) ? 0x9b : 0x0)      /* addr[19] */
        ^ (((addr            >> 18) & 1) ? 0xe5 : 0x0)      /* addr[18] */
        ^ (((addr            >> 17) & 1) ? 0x57 : 0x0)      /* addr[17] */
        ^ (((addr            >> 16) & 1) ? 0xec : 0x0)      /* addr[16] */

        ^ (((addr            >> 15) & 1) ? 0xc7 : 0x0)      /* addr[15] */
        ^ (((addr            >> 14) & 1) ? 0xae : 0x0)      /* addr[14] */
        ^ (((addr            >> 13) & 1) ? 0x67 : 0x0)      /* addr[13] */
        ^ (((addr            >> 12) & 1) ? 0x9d : 0x0)      /* addr[12] */
        ^ (((addr            >> 11) & 1) ? 0x5b : 0x0)      /* addr[11] */
        ^ (((addr            >> 10) & 1) ? 0xe6 : 0x0)      /* addr[10] */
        ^ (((addr            >>  9) & 1) ? 0x3e : 0x0)      /* addr[ 9] */
        ^ (((addr            >>  8) & 1) ? 0xf1 : 0x0)      /* addr[ 8] */

        ^ (((addr            >>  7) & 1) ? 0xdc : 0x0)      /* addr[ 7] */
        ^ (((addr            >>  6) & 1) ? 0xe9 : 0x0)      /* addr[ 6] */
        ^ (((addr            >>  5) & 1) ? 0x3d : 0x0)      /* addr[ 5] */
        ^ (((addr            >>  4) & 1) ? 0xf2 : 0x0)      /* addr[ 4] */
        ^ (((addr            >>  3) & 1) ? 0x2f : 0x0);     /* addr[ 3] */

    ecc = (((data_a2_is_zero >> 31) & 1) ? 0xb0 : 0x0)      /* data[63] */
        ^ (((data_a2_is_zero >> 30) & 1) ? 0x23 : 0x0)      /* data[62] */
        ^ (((data_a2_is_zero >> 29) & 1) ? 0x70 : 0x0)      /* data[61] */
        ^ (((data_a2_is_zero >> 28) & 1) ? 0x62 : 0x0)      /* data[60] */
        ^ (((data_a2_is_zero >> 27) & 1) ? 0x85 : 0x0)      /* data[59] */
        ^ (((data_a2_is_zero >> 26) & 1) ? 0x13 : 0x0)      /* data[58] */
        ^ (((data_a2_is_zero >> 25) & 1) ? 0x45 : 0x0)      /* data[57] */
        ^ (((data_a2_is_zero >> 24) & 1) ? 0x52 : 0x0)      /* data[56] */

        ^ (((data_a2_is_zero >> 23) & 1) ? 0x2a : 0x0)      /* data[55] */
        ^ (((data_a2_is_zero >> 22) & 1) ? 0x8a : 0x0)      /* data[54] */
        ^ (((data_a2_is_zero >> 21) & 1) ? 0x0b : 0x0)      /* data[53] */
        ^ (((data_a2_is_zero >> 20) & 1) ? 0x0e : 0x0)      /* data[52] */
        ^ (((data_a2_is_zero >> 19) & 1) ? 0xf8 : 0x0)      /* data[51] */
        ^ (((data_a2_is_zero >> 18) & 1) ? 0x25 : 0x0)      /* data[50] */
        ^ (((data_a2_is_zero >> 17) & 1) ? 0xd9 : 0x0)      /* data[49] */
        ^ (((data_a2_is_zero >> 16) & 1) ? 0xa1 : 0x0)      /* data[48] */

        ^ (((data_a2_is_zero >> 15) & 1) ? 0x54 : 0x0)      /* data[47] */
        ^ (((data_a2_is_zero >> 14) & 1) ? 0xa7 : 0x0)      /* data[46] */
        ^ (((data_a2_is_zero >> 13) & 1) ? 0xa8 : 0x0)      /* data[45] */
        ^ (((data_a2_is_zero >> 12) & 1) ? 0x92 : 0x0)      /* data[44] */
        ^ (((data_a2_is_zero >> 11) & 1) ? 0xc8 : 0x0)      /* data[43] */
        ^ (((data_a2_is_zero >> 10) & 1) ? 0x07 : 0x0)      /* data[42] */
        ^ (((data_a2_is_zero >>  9) & 1) ? 0x34 : 0x0)      /* data[41] */
        ^ (((data_a2_is_zero >>  8) & 1) ? 0x32 : 0x0)      /* data[40] */
        ^ (((data_a2_is_zero >>  7) & 1) ? 0x68 : 0x0)       /* data[39] */
        ^ (((data_a2_is_zero >>  6) & 1) ? 0x89 : 0x0)       /* data[38] */
        ^ (((data_a2_is_zero >>  5) & 1) ? 0x98 : 0x0)       /* data[37] */
        ^ (((data_a2_is_zero >>  4) & 1) ? 0x49 : 0x0)       /* data[36] */
        ^ (((data_a2_is_zero >>  3) & 1) ? 0x61 : 0x0)       /* data[35] */
        ^ (((data_a2_is_zero >>  2) & 1) ? 0x86 : 0x0)       /* data[34] */
        ^ (((data_a2_is_zero >>  1) & 1) ? 0x91 : 0x0)       /* data[33] */
        ^  ((data_a2_is_zero        & 1) ? 0x46 : 0x0)       /* data[32] */

        ^ (((data_a2_is_one  >> 31) & 1) ? 0x58 : 0x0)       /* data[31] */
        ^ (((data_a2_is_one  >> 30) & 1) ? 0x4f : 0x0)       /* data[30] */
        ^ (((data_a2_is_one  >> 29) & 1) ? 0x38 : 0x0)       /* data[29] */
        ^ (((data_a2_is_one  >> 28) & 1) ? 0x75 : 0x0)       /* data[28] */
        ^ (((data_a2_is_one  >> 27) & 1) ? 0xc4 : 0x0)       /* data[27] */
        ^ (((data_a2_is_one  >> 26) & 1) ? 0x0d : 0x0)       /* data[26] */
        ^ (((data_a2_is_one  >> 25) & 1) ? 0xa4 : 0x0)       /* data[25] */
        ^ (((data_a2_is_one  >> 24) & 1) ? 0x37 : 0x0)       /* data[24] */

        ^ (((data_a2_is_one  >> 23) & 1) ? 0x64 : 0x0)       /* data[23] */
        ^ (((data_a2_is_one  >> 22) & 1) ? 0x16 : 0x0)       /* data[22] */
        ^ (((data_a2_is_one  >> 21) & 1) ? 0x94 : 0x0)       /* data[21] */
        ^ (((data_a2_is_one  >> 20) & 1) ? 0x29 : 0x0)       /* data[20] */
        ^ (((data_a2_is_one  >> 19) & 1) ? 0xea : 0x0)       /* data[19] */
        ^ (((data_a2_is_one  >> 18) & 1) ? 0x26 : 0x0)       /* data[18] */
        ^ (((data_a2_is_one  >> 17) & 1) ? 0x1a : 0x0)       /* data[17] */
        ^ (((data_a2_is_one  >> 16) & 1) ? 0x19 : 0x0)       /* data[16] */

        ^ (((data_a2_is_one  >> 15) & 1) ? 0xd0 : 0x0)       /* data[15] */
        ^ (((data_a2_is_one  >> 14) & 1) ? 0xc2 : 0x0)       /* data[14] */
        ^ (((data_a2_is_one  >> 13) & 1) ? 0x2c : 0x0)       /* data[13] */
        ^ (((data_a2_is_one  >> 12) & 1) ? 0x51 : 0x0)       /* data[12] */
        ^ (((data_a2_is_one  >> 11) & 1) ? 0xe0 : 0x0)       /* data[11] */
        ^ (((data_a2_is_one  >> 10) & 1) ? 0xa2 : 0x0)       /* data[10] */
        ^ (((data_a2_is_one  >>  9) & 1) ? 0x1c : 0x0)       /* data[ 9] */
        ^ (((data_a2_is_one  >>  8) & 1) ? 0x31 : 0x0)       /* data[ 8] */

        ^ (((data_a2_is_one  >>  7) & 1) ? 0x8c : 0x0)       /* data[ 7] */
        ^ (((data_a2_is_one  >>  6) & 1) ? 0x4a : 0x0)       /* data[ 6] */
        ^ (((data_a2_is_one  >>  5) & 1) ? 0x4c : 0x0)       /* data[ 5] */
        ^ (((data_a2_is_one  >>  4) & 1) ? 0x15 : 0x0)       /* data[ 4] */
        ^ (((data_a2_is_one  >>  3) & 1) ? 0x83 : 0x0)       /* data[ 3] */
        ^ (((data_a2_is_one  >>  2) & 1) ? 0x9e : 0x0)       /* data[ 2] */
        ^ (((data_a2_is_one  >>  1) & 1) ? 0x43 : 0x0)       /* data[ 1] */
        ^  ((data_a2_is_one         & 1) ? 0xc1 : 0x0);      /* data[ 0] */

    ecc = ecc ^ addr_ecc;   /* combine data and addr ecc values */

    return(ecc);
}
```

On a memory read operation, the ECC logic performs the same type of optional adjustment on the read checkbits.

As the ECC syndrome is calculated on a read operation by applying the H-matrix to the data plus the checkbits, an all zero syndrome indicates an error free operation. If the generated syndrome value is non-zero and matches one of the H-matrix values associated with the data or checkbits, it represents a single-bit error correction case and the specific bit is complemented to produce the correct data value. If the syndrome value matches one of the H-matrix values associated with the address bits, or is an even weight value, or represents an unused odd weight value, a non-correctable ECC event has been detected and the appropriate error termination response is initiated.

Chapter 9
Acronyms and abbreviations

9.1 Acronyms and abbreviations

A short list of acronyms and abbreviations used in this document is shown in the table below.

Table 9-1. Acronyms and abbreviations

Terms	Meanings
CCF	Common Cause Failures
CMF	Common Mode Failures
DC	Diagnostic Coverage
DED	Double-Error Detection
DPF	Dual-Point Fault
ECC	Error Correction Code
EDC	Error Detection Code
FMEDA	Failure Modes, Effects & Diagnostic Analysis
LF	Latent Fault
LFM	Latent Fault Metric
MCU	Microcontroller Unit
MPF	Multiple-Point Fault
PMHF	Probabilistic Metric for random Hardware Failures
PST	Process Safety Time
RF	Residual Fault
SEooC	Safety Element out of Context
SEC	Single-Error Correction
SF	Safe Fault
SFF	Safe Failure Fraction
SIL	Safety Integrity Level
SM	Safety Manual
SPF	Single-Point Fault
SPFM	Single-Point Faults Metric
TED	Triple-Error Detection

Appendix A
Release Notes for Revision 2

A.1 General changes in this document

• Editorial changes and improvements throughout the document.
• Changed device number from "MAC57D54H" to "SAC57D5xx". • Removed document subtitle ("Devices Supported" MAC57D5xx"). • Attached updated "Module Classification" spreadsheet. • Removed the "Physical Pin Displacement" spreadsheet.

A.2 Preface changes

• In the Safety manual assumptions section: • Editorial updates.

A.3 MCU safety context changes

• In MCU safety functions :   • Editorial updates. • In Correct operation :   • Changed "Software Execution Function" to "MCU Safety Function".
• In the Faults section: • Editorial changes to the "Latent Fault" bullet for better clarity.
• In the Safety integrity level section: • Editorial updates (changed "ISO26262" to "ISO 26262", and "IEC61508" to "IEC 61508").
• In the MCU safety functions section: • Added the sentence/paragraph "Please see the Module classification section for more details" at the end of the section.
• In Failure types :   • Updated the explanation of random hardware fault reduction and detection for improved clarity.
• In Latent-fault tolerant time interval for latent faults :   • Editorial change.   • Added Assumption SM_212 and Rationale.

- In the MCU safety functions section:
  - Changed the "Debug Functions" bullet to "Not Safety Related functions: It is assumed that some functions are Not Safety Related (e.g. debug)."

- In the MCU safety functions section:
  - Editorial change.

- In the Overview section:
  - Changed "MAC57D5/4xx family" to "SAC57D5xx family".
  - Editorial changes.
- In the MCU fault indication time section:
  - Changed the "FCCU configured as slow switching mode maximum indication delay" from TBD to 8.2 ms.
- Cleaned up Figure 2-1.
- In the Faults and failures section:
  - Modified the hierarchy of this section - no content changes.

- In the MCU fault indication time section:
  - Changed the "FCCU configured as fast switching mode external maximum indication delay" from TBD to 64 μs.
  - Changed the "Bi-stable protocol maximum indication delay" from TBD to 64 μs.

- In the MCU safety functions section:
  - Added missing device number instances in this section.
  - Changed "The following are assumed to be typical system level safety functions..." to "The assumed MCU safety functions are..."

A.4 MCU Safety Concept changes

- In the Disabling of communication controllers section:
  - Added indication that the Pin Muxing and SIUL2 information reside in the chip's Reference Manual.
- In the Built-In Self Tests (BIST) section:
  - Added indication that the STCU2 chapter resides in the chip's Reference Manual.
- In the Operational interference protection section:
  - Added indication that the "Extended Resource Domain Controller (XRDC)" chapter resides in this chip's Reference Manual.

- In the General concept section:
  - Changed the broken cross-reference link to the "I/O peripherals" section to reference the "I/O functions" section instead.

- In the ECC for storage section:
  - Added Assumption SM_113.
  - Added Assumption SM_109.
  - Added Assumption under certain conditions SM_110.
  - Added Assumption under certain conditions SM_111.
- In the Common Cause Failure measures section:
  - In the first paragraph, removed "physical separation of the components on the die, routing restrictions and" from the sentence "These measures include..."

A.5 Hardware requirements changes

- In Power Supply Monitor (PSM) :
  - Removed sentence regarding disabling power or replacing the MCU for an overvoltage event.
  - Removed Assumption [SM_086], "It is assumed that external power of appropriate voltage is supplied".

- In Assumption [SM_088], changed "the maximum survivable voltage of the technology" to "the absolute maximum rating of the device".
- Removed the recommendation to disable the system when an overvoltage occurs.

- In Hardware requirements on system level :
  - Updated the introductory text/description related to the application schematic figure.
  - Updated the "Functional safety related connection to external circuitry" application schematic figure.
- In External Watchdog (EXWD) :
  - Changed "FlexCAN" to "CAN".
- In PowerSBC, in the "Functional safety application with PowerSBC" figure:
  - Changed "FlexCAN" to "CAN".
  - Changed "FlexLIN" to "LINFlex".

- In the Hardware requirements on system level section:
  - Editorial update.
  - Simplified the figure.
  - Removed the Notes.

- In the External Watchdog (EXWD) section:
  - Removed the bullet, "Toggling error out signals... from the FCCU".
  - Removed the Implementation hint.
- In the Power Supply Monitor (PSM) section:
  - Changed sentence regarding over-voltage detection "on the 1.25 V core supply" to "on some supplies".

- In the Power Supply Monitor (PSM) section:
  - Changed the word "powerless" to "unpowered".
  - In the Assumption "It is assumed that the external power is supervised for high and low deviations", added "where no supervision is provided on the MCU."
- In the PWM output monitor section:
  - In the list of features to be managed by system level measures, removed the example at the end of the "Open GPIO" bullet.

- In the Hardware requirements on system level section:
  - Corrected instance of FCCU_F[n] to EOUT[*n*].

- In the PowerSBC section:
  - Editorial update.

- Added the External communication section.

- In the Error Out Monitor (ERRM) section:
  - Changed "EOUT[0], and optionally EOUT[1]" to "EOUT[0] and/or EOUT[1]".

A.6 Software Requirements changes

- In the Test mode section:
  - Removed the "Recommendation" to use system-level software measures to disable test mode.
  - Editorial update.
- In the 1.25 V supply supervision section:
  - Removed the content regarding ballast transistors.
  - Editorial updates.
  - Reworked the "Implementation hint".
- In the Built-in Hardware Self-Tests (BIST) section:
  - Editorial update.
- In the Structural software based self-test (SBST) section:
  - Editorial update.
- In the FCCU Runtime checks section:
  - In the "Note", changed "Longer reset cycles means length of time since the previous reset" to "Longer reset cycles refers to length of time since the previous reset".
- In the PBRIDGE protection section:

    - In the "Recommendation", changed "more than 10 registers" to "more than 10 registers containing different values".
- Moved the "Implementation hint" that mentions using Margin Read Enable Mode from the EEPROM Runtime checks section to the EEPROM Initial checks and configurations section.

- In the Software Watchdog Timer section:
    - Changed the content in Assumption SM_068 to be plain text, i.e. it is no longer an assumption.
- In the CRC Runtime checks section:
    - Removed Assumption SM_071.
- In the Fast External Oscillator (FXOSC) section:
    - Removed the sentence "For safety relevant applications, these clocking modes should not be used."
- In the FXOSC Initial checks and configurations section:
    - Updated Assumption SM_075.
- In the Error reporting path tests section:
    - Removed Assumption SM_124 and its accompanying Rationale.
- In the Crossbar Switch (AXBS) section:
    - Changed "vital system modules or other supporting modules" to "safety related".

- In the FCCU Initial checks and configurations section:
    - Tagged the last paragraph ("If the SAC57D54H signals an internal failure via its error out signals...") in this section as Assumption SM_272.
- In the FCCU Runtime checks section:
    - Tagged the first paragraph ("If the SAC57D54H is continuously switching between...") in this section as Assumption SM_155.
- In the Consecutive resets section:
    - Added Assumption SM_279.
- In the PLLDIG Initial checks and configurations section:
    - The cross-reference to the section named "Internal RC Oscillator" was renamed to "Fast Internal RC Oscillator".
    - Added a sentence to the end of Assumption SM_079 stating that modules that can be clocked directly by the XOSC and are being used for a safety function should instead be clocked by the PLL.
- In the Extended Resource Domain Controller section:
    - Added Assumption SM_094.
- In the eDMA Runtime checks section:
    - Added Assumption SM_104 and the related paragraph preceding it.
- In the Mode Entry (MC_ME) section:
    - Added this new section.
- In the Flash Initial checks and configurations section:
    - Changed the first Implementation hint to be Assumption SM_112.
- In the Wake-Up Unit (WKPU) / External NMI section:
    - Added this new section.
- In the Cores section:
    - Added this new section, which contains the following subsections:
        - "Runtime checks" (new hierarchy).
        - "Structural software based self-test (SBST)" (the previously existing content with the same name).
        - "Reciprocal comparison" (new).
- In the Cores Structural software based self-test (SBST) section:
    - In the first sentence, changed "During code execution, a processing..." to: "During execution of safety relevant code on the ARM Cortex-M4 processor or the Cortex-A5 processor, processing..."
    - Added Assumption SM_316.
- In the Analog to Digital Converter (ADC) section:
    - Added this new section.
- In the I/O functions section:
    - Changed the first paragraph to be Assumption SM_232.

- In the EEPROM Runtime checks section:
    - Replaced Assumption SM_119.
    - Reworked the "Implementation hint".
- In the Double Read Analog Inputs section:
    - Editorial change.

A.7 Failure Rates and FMEDA changes

- In the Module classification section, changed the term "Software Execution Function" to "MCU Safety Functions".

A.8 Dependent Failures changes

- In I/O pin/ball configuration :
    - Changed references to the "Physcial Pin Displacement on Internal Die" attachment to "The Signal Description chapter in the SAC57D5xx Reference Manual".

- Editorial updates throughout chapter.

- In Dependent failure avoidance on system level :
    - Removed the Recommendation as it was redundant.
- In External timeout function :
    - Removed the sentence that immediately followed the note, as it was redundant.

- In the I/O pin/ball configuration section:
    - Editorial update.

- In the Power supply section:
    - Editorial update.

A.9 Additional Information changes

- In the Testing All-X in RAM section:
    - Removed the mention of "e2eECC".
- In the ECC checkbit/syndrome coding scheme section:
    - Changed instances of "e2eECC" to "ECC".

A.10 Acronyms and abbreviations changes

- Updated the list of acronyms.

- Editorial change.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number SAC57D54HSM
Revision 2.1, 08/2016